# Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>About the AWS CLI</td>
<td>vii</td>
</tr>
<tr>
<td>About AWS CLI version 1</td>
<td>1</td>
</tr>
<tr>
<td>Maintenance and support for SDK major versions</td>
<td>1</td>
</tr>
<tr>
<td>About Amazon Web Services</td>
<td>2</td>
</tr>
<tr>
<td>Using the examples</td>
<td>2</td>
</tr>
<tr>
<td>Additional documentation and resources</td>
<td>3</td>
</tr>
<tr>
<td>AWS CLI documentation and resources</td>
<td>3</td>
</tr>
<tr>
<td>Other AWS SDKs</td>
<td>3</td>
</tr>
<tr>
<td>Installing the AWS CLI</td>
<td>4</td>
</tr>
<tr>
<td>Python version requirements</td>
<td>4</td>
</tr>
<tr>
<td>Amazon Linux</td>
<td>4</td>
</tr>
<tr>
<td>Prerequisites</td>
<td>5</td>
</tr>
<tr>
<td>pip</td>
<td>5</td>
</tr>
<tr>
<td>yum</td>
<td>6</td>
</tr>
<tr>
<td>Linux</td>
<td>6</td>
</tr>
<tr>
<td>Prerequisites</td>
<td>7</td>
</tr>
<tr>
<td>Install and uninstall using the bundled installer</td>
<td>7</td>
</tr>
<tr>
<td>Install and uninstall using pip</td>
<td>10</td>
</tr>
<tr>
<td>MacOS</td>
<td>14</td>
</tr>
<tr>
<td>Prerequisites</td>
<td>14</td>
</tr>
<tr>
<td>Install and uninstall using the bundled installer</td>
<td>14</td>
</tr>
<tr>
<td>Install and update using pip</td>
<td>18</td>
</tr>
<tr>
<td>Troubleshooting AWS CLI install and uninstall errors</td>
<td>20</td>
</tr>
<tr>
<td>Windows</td>
<td>20</td>
</tr>
<tr>
<td>Install, update, and uninstall using the MSI installer</td>
<td>21</td>
</tr>
<tr>
<td>Install, update, and uninstall using Python and pip</td>
<td>22</td>
</tr>
<tr>
<td>Add the AWS CLI executable to your command line path</td>
<td>23</td>
</tr>
<tr>
<td>Troubleshooting AWS CLI install and uninstall errors</td>
<td>24</td>
</tr>
<tr>
<td>Virtualenv</td>
<td>24</td>
</tr>
<tr>
<td>Prerequisites</td>
<td>25</td>
</tr>
<tr>
<td>Install and update in a virtual environment</td>
<td>25</td>
</tr>
<tr>
<td>Troubleshooting AWS CLI install and uninstall errors</td>
<td>26</td>
</tr>
<tr>
<td>Configuring the AWS CLI</td>
<td>27</td>
</tr>
<tr>
<td>Configuration basics</td>
<td>27</td>
</tr>
<tr>
<td>Quick configuration with aws configure</td>
<td>28</td>
</tr>
<tr>
<td>Access key ID and secret access key</td>
<td>28</td>
</tr>
<tr>
<td>Region</td>
<td>29</td>
</tr>
<tr>
<td>Output format</td>
<td>29</td>
</tr>
<tr>
<td>Profiles</td>
<td>29</td>
</tr>
<tr>
<td>Configuration settings and precedence</td>
<td>30</td>
</tr>
<tr>
<td>Configuration and credential file settings</td>
<td>30</td>
</tr>
<tr>
<td>Where are configuration settings stored?</td>
<td>31</td>
</tr>
<tr>
<td>Set and view configuration settings</td>
<td>31</td>
</tr>
<tr>
<td>Supported config file settings</td>
<td>33</td>
</tr>
<tr>
<td>Named profiles</td>
<td>41</td>
</tr>
<tr>
<td>Creating named profiles</td>
<td>41</td>
</tr>
<tr>
<td>Using named profiles</td>
<td>42</td>
</tr>
<tr>
<td>Environment Variables</td>
<td>43</td>
</tr>
<tr>
<td>How to set environment variables</td>
<td>43</td>
</tr>
<tr>
<td>AWS CLI supported environment variables</td>
<td>44</td>
</tr>
<tr>
<td>Command line options</td>
<td>47</td>
</tr>
<tr>
<td>How to use command line options</td>
<td>47</td>
</tr>
</tbody>
</table>
This documentation is for Version 1 of the AWS CLI only. For documentation related to Version 2 of the AWS CLI, see the Version 2 User Guide.
What is the AWS Command Line Interface version 1?

Note
The AWS CLI version 1 is not the latest version of the AWS CLI. Some features introduced in AWS CLI version 2 are not backported to version 1 and you must upgrade to access those features. There are some “breaking” changes from version 1 that might require you to change your scripts. For a list of breaking changes in version 2, see Breaking changes in the AWS CLI version 2 User Guide.

The AWS Command Line Interface (AWS CLI) is an open source tool that enables you to interact with AWS services using commands in your command-line shell. With minimal configuration, the AWS CLI enables you to start running commands that implement functionality equivalent to that provided by the browser-based AWS Management Console from the command prompt in your terminal program:

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

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

The AWS CLI provides direct access to the public APIs of AWS services. You can explore a service’s capabilities with the AWS CLI, and develop shell scripts to manage your resources. In addition to the low-level, API-equivalent commands, several AWS services provide customizations for the AWS CLI. Customizations can include higher-level commands that simplify using a service with a complex API.

About AWS CLI version 1

The AWS CLI version 1 is the original AWS CLI, and we continue to support it. However, major new features that are introduced in the AWS CLI version 2 might not be backported to the AWS CLI version 1. To use those features, you must install the AWS CLI version 2. The AWS CLI version 1 is built using the SDK for Python, and therefore requires you to install a compatible version of Python.

To install the AWS CLI version 1, see *Installing the AWS CLI* (p. 4).

To check the currently installed version, use the following command:

```
$ aws --version
aws-cli/1.25.55 Python/3.8.8 Linux/4.14.133-113.105.amzn2.x86_64 botocore/1.13
```

For version history, see the AWS CLI version 1 Changelog on GitHub.
Maintenance and support for SDK major versions

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

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

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 AWS Free Tier. To obtain an AWS account, open the AWS home page and then choose Create an AWS Account.

Using the AWS CLI examples

The AWS Command Line Interface (AWS CLI) examples in this guide are formatted using the following conventions:

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

The following `aws configure` command example demonstrates user input, replaceable text, and output:

1. Enter `aws configure` at the command line, and then press Enter.
2. The AWS CLI outputs lines of text, prompting you to enter additional information.
3. Enter each of your access keys in turn, and then press Enter.
4. Then, enter an AWS Region name in the format shown, press Enter, and then press Enter a final time to skip the output format setting.
5. The final Enter command is shown as replaceable text because there is no user input for that line.

```bash
$ aws configure
AWS Access Key ID [None]: akiaiosfodnn7example
AWS Secret Access Key [None]: wjalrxtunfemi/k7mdeng/bpxrficexamplekey
Default region name [None]: us-west-2
```
The following example shows a simple command with output. To use this example, enter the full text of the command (the highlighted text after the prompt), and then press Enter. The name of the security group, my-sg, is replaceable to your desired security group name. 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.

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

**Additional documentation and resources**

**AWS CLI documentation and resources**

In addition to this user guide, the following are valuable online resources for the AWS CLI.

- AWS CLI version 1 reference guide
- AWS CLI code examples repository
- AWS CLI GitHub repository You can view and fork the source code for the AWS CLI on GitHub. Join the community of users on GitHub to provide feedback, request features, and submit your own contributions.
- AWS CLI alias examples repository You can view and fork AWS CLI alias examples on GitHub.
- AWS CLI version 1 Changelog
- AWS CLI version 2 Changelog

**Other AWS SDKs**

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

- AWS Tools for PowerShell
- AWS SDK for Java
- AWS SDK for .NET
- AWS SDK for JavaScript
- AWS SDK for Ruby
- AWS SDK for Python (Boto)
- AWS SDK for PHP
- AWS SDK for Go
- AWS Mobile SDK for iOS
- AWS Mobile SDK for Android
Installing, updating, and uninstalling the AWS CLI

This topic provides links to install, update, and uninstall the original version of the AWS Command Line Interface (AWS CLI). The AWS CLI version 1 is currently supported, but new features added to the AWS CLI version 2 might not be added to the AWS CLI version 1. To use those features, you must install the AWS CLI version 2. For information about how to install version 2, see Installing the AWS CLI version 2.

AWS CLI install, update, and uninstall instructions:
- Python version requirements (p. 4)
- Install, Update, and Uninstall the AWS CLI version 1 on Amazon Linux (p. 4)
- Install, Update, and Uninstall the AWS CLI version 1 on Linux (p. 6)
- Install, Update, and Uninstall the AWS CLI version 1 on macOS (p. 14)
- Install, Update, and Uninstall the AWS CLI version 1 on Windows (p. 20)
- Install and Update the AWS CLI version 1 in a virtual environment (p. 24)

Python version requirements

The AWS CLI version 1 is built using the SDK for Python, and therefore requires you to install a compatible version of Python.

Python version support matrix

<table>
<thead>
<tr>
<th>AWS CLI version</th>
<th>Supported Python version</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.20.0 – current</td>
<td>Python 3.6+</td>
</tr>
<tr>
<td>1.19.0 – 1.19.x</td>
<td>Python 2.7+, Python 3.6+</td>
</tr>
<tr>
<td>1.17 – 1.18.x</td>
<td>Python 2.7+, Python 3.4+</td>
</tr>
<tr>
<td>1.0 – 1.16.x</td>
<td>Python 2.6 and older, Python 3.3 and older</td>
</tr>
</tbody>
</table>

Install, Update, and Uninstall the AWS CLI version 1 on Amazon Linux

The AWS CLI version 1 is preinstalled on Amazon Linux and Amazon Linux 2. Check the currently installed version by using the following command.

```
$ aws --version
aws-cli/1.25.55 Python/3.8.8 Linux/4.14.133-113.105.amzn2.x86_64 botocore/1.13
```

Depending on when you created your Amazon Linux instance, the AWS CLI version 1 is preinstalled using one of the following package managers:

- pip (p. 5)
- yum (p. 6)
Prerequisites

You must have Python 3.6 or later installed. For installation instructions, see the Downloading Python page in Python’s Beginner Guide.

Install, update, or uninstall using pip

Most Amazon Linux instances use pip to preinstall the AWS CLI version 1.

Install or update the AWS CLI version 1 on Amazon Linux using pip

To install the latest version of the AWS CLI version 1 for the current user, use the following instructions.

1. We recommend that if you have Python version 3 or later installed that you use `pip3`. Use `pip3 install` to install or update to the latest version of the AWS CLI version 1. If you run the command from within a Python virtual environment (venv), you don’t need to use the `--user` option.

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

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

```
$ ls -a ~
.  ..  .bash_logout  .bash_profile  .bashrc  Desktop  Documents  Downloads
```
   • Bash – `.bash_profile`, `.profile`, or `.bash_login`
   • Zsh – `.zshrc`
   • Tcsh – `.tcshrc`, `.cshrc` or `.login`
   b. Add an export command at the end of your profile script that’s similar to the following example.

```bash
export PATH=$HOME/.local/bin:$PATH
```
   This command inserts the path, `$HOME/.local/bin` in this example, at the front of the existing `$PATH` variable.
   c. Reload the profile into your current session to put those changes into effect.

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

3. To verify that you're running the new version, use the `aws --version` command.

```bash
$ aws --version
aws-cli/1.25.55 Python/3.8.8 Linux/4.14.133-113.105.amzn2.x86_64 botocore/1.13
```

Uninstall the AWS CLI version 1 using pip

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

```bash
$ pip3 uninstall awscli
```
Install, update, or uninstall using yum

Most Amazon Linux 2 instances use yum to preinstall the AWS CLI version 1.

Install or update the AWS CLI version 1 on Amazon Linux using yum

To install the latest version of the AWS CLI version 1, run the following command.

```bash
$ sudo yum install awscli
```

To update to the latest version of the AWS CLI version 1, run the following command.

```bash
$ sudo yum update awscli
```

To verify that you're running the new version, use the `aws --version` command.

```
$ aws --version
aws-cli/1.25.55 Python/3.8.8 Linux/4.14.133-113.105.amzn2.x86_64 botocore/1.13
```

Uninstall the AWS CLI version 1 using yum

Some newer images of Amazon Linux 2 use yum to install the Amazon Linux 2 you need to uninstall the AWS CLI, use `pip uninstall`.

```bash
$ sudo yum remove awscli
```

Troubleshooting AWS CLI install and uninstall errors

If you come across issues after installing or uninstalling the AWS CLI, see Troubleshooting errors (p. 167) for troubleshooting steps. For the most relevant troubleshooting steps, see the section called “Command not found errors” (p. 172), the section called “The "aws --version" command returns a different version than you installed” (p. 174), and the section called “The "aws --version" command returns a version after uninstalling the AWS CLI” (p. 175).

Install, Update, and Uninstall the AWS CLI version 1 on Linux

You can install the AWS Command Line Interface (AWS CLI) version 1 and its dependencies on most Linux distributions by using the `pip` package manager or the bundled installer.

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

Sections
- Prerequisites (p. 7)
- Install and uninstall the AWS CLI version 1 on Linux using the bundled installer (p. 7)
- Install and uninstall the AWS CLI version 1 using pip (p. 10)
Prerequisites

You must have Python 3.6 or later installed. For installation instructions, see the Downloading Python page in Python's Beginner Guide.

**Warning**

Python 2.7 was deprecated by the Python Software Foundation on January 1, 2020. Starting with AWS CLI version 1.20.0, a minimum version of Python 3.6 is required. In order to use the AWS CLI version 1 with an older version of Python, you need to install an earlier version of the AWS CLI version 1. To view the AWS CLI version 1 Python version support matrix, see the section called "Python version requirements" (p. 4).

Install and uninstall the AWS CLI version 1 on Linux using the bundled installer

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

**Note**

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

**Topics**

- Install the AWS CLI version 1 using the bundled installer with sudo (p. 7)
- Install the AWS CLI version 1 using the bundled installer without sudo (p. 8)
- Uninstall the AWS CLI version 1 bundled installer (p. 10)

Install the AWS CLI version 1 using the bundled installer with sudo

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

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

For the latest version of the AWS CLI, use the following command block:

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

For a specific version of the AWS CLI, append a hyphen and the version number to the filename. For this example the filename for version 1.16.312 would be awscli-bundle-1.16.312.zip resulting in the following command:

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

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

**To install the AWS CLI version 1 using the bundled installer**

1. Download the AWS CLI version 1 bundled installer using one of the the following methods.
• Download using the `curl` command.

For the latest version of the AWS CLI, use the following command block:

```sh
```

For a specific version of the AWS CLI, append a hyphen and the version number to the filename. For this example the filename for version `1.16.312` would be `awscli-bundle-1.16.312.zip` resulting in the following command:

```sh
```

• Download using the direct link.

For the latest version of the AWS CLI: `https://s3.amazonaws.com/aws-cli/awscli-bundle.zip`

For a specific version of the AWS CLI, append a hyphen and the version number to the filename. For this example the filename for version `1.16.312` would be `awscli-bundle-1.16.312.zip` resulting in the following url `https://s3.amazonaws.com/aws-cli/awscli-bundle-1.16.312.zip`

1. Extract the files from the package. If you don’t have `unzip` to extract the files, use your Linux distribution’s built-in package manager to install it.

```sh
# unzip awscli-bundle.zip
```

2. Run the install program. 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 entering `aws` from any directory.

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

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 version to install the AWS CLI, run the install script with that version by absolute path to the Python executable, as follows.

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

3. Verify that the AWS CLI installed correctly.

```sh
$ aws --version
aws-cli/1.25.55 Python/3.8.8 Linux/4.14.133-113.105.amzn2.x86_64 botocore/1.13
```

If you get an error, see Troubleshooting AWS CLI errors (p. 167).

Install the AWS CLI version 1 using the bundled installer without `sudo`

If you don’t have `sudo` permissions or want to install the AWS CLI only for the current user, you can use a modified version of the previous commands. The first two commands are the same.

For the latest version of the AWS CLI, use the following command block:
Install and uninstall using the bundled installer

unzip awscli-bundle.zip
./awscli-bundle/install -b ~/bin/aws

For a specific version of the AWS CLI, append a hyphen and the version number to the filename. For this example the filename for version 1.16.312 would be awscli-bundle-1.16.312.zip resulting in the following command:

unzip awscli-bundle.zip
./awscli-bundle/install -b ~/bin/aws

To install the AWS CLI version 1 for current user

1. Download the AWS CLI version 1 bundled installer in one of the following ways.
   
   • Download using the curl command.
      
      For the latest version of the AWS CLI, use the following command block:


      For a specific version of the AWS CLI, append a hyphen and the version number to the filename. For this example the filename for version 1.16.312 would be awscli-bundle-1.16.312.zip resulting in the following command:


   • Download using the direct link.
      
      For the latest version of the AWS CLI: https://s3.amazonaws.com/aws-cli/awscli-bundle.zip

      For a specific version of the AWS CLI, append a hyphen and the version number to the filename. For this example the filename for version 1.16.312 would be awscli-bundle-1.16.312.zip resulting in the following url https://s3.amazonaws.com/aws-cli/awscli-bundle-1.16.312.zip

2. Extract the files from the package by using unzip. If you don't have unzip, use your Linux distribution's built-in package manager to install it.

   $ unzip awscli-bundle.zip

3. Run the install program. The installer installs the AWS CLI at /usr/local/aws and creates the symlink aws at the /usr/local/bin directory. The command uses the -b parameter to specify the directory where the installer places the aws symlink file. You must have write permissions to the specified folder.

   $ ./awscli-bundle/install -b ~/bin/aws

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

   $ echo $PATH | grep ~/bin // See if $PATH contains ~/bin (output will be empty if it doesn't)
   $ export PATH=~/bin:$PATH // Add ~/bin to $PATH if necessary
4. Ensure the directory that the AWS CLI version 1 is part of your PATH variable.
   a. Find your shell's profile script in your user folder. If you're not sure which shell you have, run `echo $SHELL`.

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

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

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

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

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

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

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

5. Verify that the AWS CLI installed correctly.

   ```bash
   $ aws --version
   aws-cli/1.25.55 Python/3.8.8 Linux/4.14.133-113.105.amzn2.x86_64 botocore/1.13
   ```

   If you get an error, see Troubleshooting AWS CLI errors (p. 167).

### Uninstall the AWS CLI version 1 bundled installer

1. If you installed the AWS CLI using the bundled installer, follow these instructions. The bundled installer doesn't put anything outside of the installation directory except the optional symlink, so uninstalling is as simple as deleting those two items.

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

2. **(Optional)** Remove the shared AWS SDK and AWS CLI settings information in the .aws folder.

   **Warning**
   These configuration and credentials settings are shared across all AWS SDKs and the AWS CLI. If you remove this folder, they cannot be accessed by any AWS SDKs that are still on your system.

   The default location of the .aws folder differs between platforms, by default the folder is located in ~/.aws/. If your user account has write permission to this directory, you don't need to use sudo.

   ```bash
   $ sudo rm ~/.aws/
   ```

---

### Install and uninstall the AWS CLI version 1 using pip

**Topics**
**Install and uninstall using pip**

- **Install pip (p. 11)**
- **Install and update the AWS CLI version 1 using pip (p. 12)**
- **Add the AWS CLI version 1 executable to your command line path (p. 12)**
- **Uninstall the AWS CLI using pip (p. 13)**
- **Troubleshooting AWS CLI install and uninstall errors (p. 13)**

## Install pip

If you don’t already have pip installed, you can install it by using the script that the *Python Packaging Authority* provides. Run `pip --version` to see if your version of Linux already includes Python and pip. We recommend that if you have Python version 3 or later installed, you use the `pip3` command.

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

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

2. Run the script with the `python` or `python3` command to download and install the latest version of pip and other required support packages. When you include the `--user` switch, the script installs pip to the path `~/.local/bin`.

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

3. Ensure the directory that contains pip is part of your PATH variable.

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

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

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

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

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

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

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

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

4. To verify that pip or pip3 is installed correctly, run the following command.

   ```bash
   $ pip3 --version
   pip 19.2.3 from ~/.local/lib/python3.7/site-packages (python 3.7)
   ```
Install and update the AWS CLI version 1 using pip

1. Use the pip or pip3 command to install or update the AWS CLI. We recommend that if you use Python version 3 or later that you use the pip3 command. The --user switch, pip installs the AWS CLI to ~/.local/bin.

   For the latest version of the AWS CLI, use the following command block:

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

   For a specific version of the AWS CLI, append two equals signs == and the version number to the filename. For this example the filename for version 1.16.312 would be awscli==1.16.312 resulting in the following command:

   ```
   $ pip3 install awscli==1.16.312 --upgrade --user
   ```

   Note
   Use appropriate quoting rules for your terminal. In order to use the = character, you might need to use single or double quotes to escape properly. The following example escapes using single quotes:

   ```
   $ pip3 install 'awscli==1.16.312' --upgrade --user
   ```

2. Verify that the AWS CLI installed correctly.

   ```
   $ aws --version
   aws-cli/1.25.55 Python/3.8.8 Linux/4.14.133-113.105.amzn2.x86_64 botocore/1.13
   ```

   If you get an error, see Troubleshooting AWS CLI errors (p. 167).

Add the AWS CLI version 1 executable to your command line path

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

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

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

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

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

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

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

```
$ ls -al /usr/local/bin/python
```
pip installs programs in the same folder that contains the Python application. Add this folder to your PATH variable.

**To modify your PATH variable**

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

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

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

2. Add an export command to your profile script.

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

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

3. Load the updated profile into your current session.

```
# source ~/.bash_profile
```

**Uninstall the AWS CLI using pip**

1. If you installed the AWS CLI using pip or pip3, you need to uninstall the AWS CLI using the same package manager by running one of the following commands.

```
# pip uninstall awscli
```

```
# pip3 uninstall awscli
```

2. **(Optional)** Remove the shared AWS SDK and AWS CLI settings information in the `.aws` folder.

   **Warning**
   These configuration and credentials settings are shared across all AWS SDKs and the AWS CLI. If you remove this folder, they cannot be accessed by any AWS SDKs that are still on your system.

   The default location of the `.aws` folder differs between platforms, by default the folder is located in `~/.aws/`. If your user account has write permission to this directory, you don't need to use `sudo`.

```
# sudo rm ~/.aws/
```

**Troubleshooting AWS CLI install and uninstall errors**

If you come across issues after installing or uninstalling the AWS CLI, see *Troubleshooting errors (p. 167)* for troubleshooting steps. For the most relevant troubleshooting steps, see the section called “Command not found errors” (p. 172), the section called “The “aws --version” command returns a different
version than you installed” (p. 174), and the section called “The "aws --version" command returns a version after uninstalling the AWS CLI” (p. 175).

Install, Update, and Uninstall the AWS CLI version 1 on macOS

You can install the AWS Command Line Interface (AWS CLI) version 1 and its dependencies on macOS by using the bundled installer or `pip`.

Sections
- Prerequisites (p. 14)
- Install, update and uninstall the AWS CLI version 1 on macOS using the bundled installer (p. 14)
- Install, update and uninstall the AWS CLI version 1 using `pip` (p. 18)
- Troubleshooting AWS CLI install and uninstall errors (p. 20)

Prerequisites

Before you can install the AWS CLI version 1 on macOS, be sure you have Python 3.6 or later installed. For installation instructions, see the Downloading Python page in Python’s Beginner Guide.

Warning
Python 2.7 was deprecated by the Python Software Foundation on January 1, 2020. Starting with AWS CLI version 1.20.0, a minimum version of Python 3.6 is required.
In order to use the AWS CLI version 1 with an older version of Python, you need to install an earlier version of the AWS CLI version 1. To view the AWS CLI version 1 Python version support matrix, see the section called “Python version requirements” (p. 4).

Install, update and uninstall the AWS CLI version 1 on macOS using the bundled installer

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

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

Topics
- Install the AWS CLI version 1 using the bundled installer with `sudo` (p. 14)
- Install the AWS CLI version 1 using the bundled installer without `sudo` (p. 16)
- Uninstall the AWS CLI version 1 bundled installer (p. 17)

Install the AWS CLI version 1 using the bundled installer with `sudo`

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

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

For the latest version of the AWS CLI, use the following command block:
Install and uninstall using the bundled installer

1. Download the AWS CLI version 1 bundled installer in one of the following ways:
   - Download using the `curl` command.
     
     For the latest version of the AWS CLI, use the following command block:
     
     ```bash
     unzip awscli-bundle.zip
     sudo ./awscli-bundle/install -i /usr/local/aws -b /usr/local/bin/aws
     
     For a specific version of the AWS CLI, append a hyphen and the version number to the filename. For this example the filename for version **1.16.312** would be `awscli-bundle-1.16.312.zip` resulting in the following command:
     
     ```bash
     unzip awscli-bundle.zip
     sudo ./awscli-bundle/install -i /usr/local/aws -b /usr/local/bin/aws
     
     To install the AWS CLI version 1 using the bundled installer
     ```

     2. Extract (unzip) the files from the package. If you don't have `unzip`, use your macOS distribution's built-in package manager to install it.

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

     ```bash
     $ sudo ./awscli-bundle/install -i /usr/local/aws -b /usr/local/bin/aws
     
     By default, the install script runs under the system default version of Python. If you have installed an alternative version of Python and want to use that to install the AWS CLI, run the install script with that version by absolute path to the Python executable, as follows.

     ```bash
     $ sudo /usr/local/bin/python3.7 awscli-bundle/install -i /usr/local/aws -b /usr/local/bin/aws
     ```
4. Verify that the AWS CLI installed correctly.

```bash
$ aws --version
aws-cli/1.25.55 Python/3.8.8 Linux/4.14.133-113.105.amzn2.x86_64 botocore/1.13
```

If you get an error, see Troubleshooting AWS CLI errors (p. 167).

**Install the AWS CLI version 1 using the bundled installer without sudo**

If you don’t have sudo permissions or want to install the AWS CLI only for the current user, you can use a modified version of the previous commands. The first two commands are the same.

**For the latest version of the AWS CLI,** use the following command block:

```bash
unzip awscli-bundle.zip
./awscli-bundle/install -b ~/bin/aws
```

**For a specific version of the AWS CLI,** append a hyphen and the version number to the filename. For this example the filename for version 1.16.312 would be awscli-bundle-1.16.312.zip resulting in the following command:

```bash
unzip awscli-bundle.zip
./awscli-bundle/install -b ~/bin/aws
```

**To install the AWS CLI version 1 for the current user**

1. Download the AWS CLI version 1 bundled installer using one of the following methods:

   - Download using the `curl` command.
     
     **For the latest version of the AWS CLI,** use the following command block:
     
     ```bash
     ```

     **For a specific version of the AWS CLI,** append a hyphen and the version number to the filename. For this example the filename for version 1.16.312 would be awscli-bundle-1.16.312.zip resulting in the following command:
     
     ```bash
     ```

   - Download using the direct link.

     **For the latest version of the AWS CLI:**
     

     **For a specific version of the AWS CLI,** append a hyphen and the version number to the filename. For this example the filename for version 1.16.312 would be awscli-exe-linux-aarch64-2.0.30.zip resulting in the following url [https://s3.amazonaws.com/aws-cli/awscli-bundle-2.0.30.zip](https://s3.amazonaws.com/aws-cli/awscli-bundle-2.0.30.zip)

2. Extract the files from the package. If you don’t have `unzip`, use your Linux distribution’s built-in package manager to install it.
Install and uninstall using the bundled installer

1. Open a terminal window.

2. Unzip the AWS CLI bundle.

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

3. Run the install program. The installer installs the AWS CLI at `/usr/local/aws` and creates the symlink `aws` at the `/usr/local/bin` directory. The command uses the `-b` parameter to specify the directory where the installer places the `aws` symlink file. You must have write permissions to the specified directory.

   ```
   $ ./awscli-bundle/install -b ~/bin/aws
   ```

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

4. Ensure the folder that the AWS CLI version 1 is installed in is part of your `$PATH` variable.

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

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

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

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

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

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

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

   ```
   $ source ~/bash_profile
   ```

5. Verify that the AWS CLI installed correctly.

   ```
   $ aws --version
   aws-cli/1.25.55 Python/3.8.8 Linux/4.14.133-113.105.amzn2.x86_64 botocore/1.13
   ```

   If you get an error, see Troubleshooting AWS CLI errors (p. 167).

Uninstall the AWS CLI version 1 bundled installer

1. The bundled installer puts everything inside of the installation directory except the optional symlink, so to uninstall, you just need to delete those two items.

   ```
   $ sudo rm -rf /usr/local/aws
   ```
Install and update using pip

2. **(Optional)** Remove the shared AWS SDK and AWS CLI settings information in the `.aws` folder.

   **Warning**
   These configuration and credentials settings are shared across all AWS SDKs and the AWS CLI. If you remove this folder, they cannot be accessed by any AWS SDKs that are still on your system.

   The default location of the `.aws` folder differs between platforms, by default the folder is located in `~/.aws/`. If your user account has write permission to this directory, you don't need to use `sudo`.

   `sudo rm ~/.aws/`

Install, update and uninstall the AWS CLI version 1 using pip

You can use `pip` directly to install the AWS CLI.

**Topics**
- Install pip (p. 18)
- Install and update the AWS CLI using pip (p. 18)
- Add the AWS CLI version 1 executable to your macOS command line path (p. 19)
- Uninstall the AWS CLI using pip (p. 20)

**Install pip**

If you don’t already have `pip` installed, you can install it by using the script that the *Python Packaging Authority* provides. Run `pip --version` to see if your version of Linux already includes Python and `pip`. We recommend that if you have Python version 3 or later installed, you use the `pip3` command.

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

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

2. Run the script with the `python` or `python3` command to download and install the latest version of `pip` and other required support packages. When you include the `--user` switch, the script installs `pip` to the path `~/.local/bin`.

   `python3 get-pip.py --user`

**Install and update the AWS CLI using pip**

1. Use the `pip` or `pip3` command to install the AWS CLI. We recommend that if you use Python version 3 or later, that you use the `pip3` command.

   **For the latest version of the AWS CLI**, use the following command block:

   `pip3 install awscli --upgrade --user`
For a specific version of the AWS CLI, append two equals signs `==` and the version number to the filename. For this example the filename for version `1.16.312` would be `==1.16.312` resulting in the following command:

```
$ pip3 install awscli==1.16.312 --upgrade --user
```

**Note**
Use appropriate quoting rules for your terminal. To use the `=` character, you might need to use single or double quotes to escape properly. The following example escapes using single quotes:

```
$ pip3 install 'awscli==1.16.312' --upgrade --user
```

2. Verify that the AWS CLI is installed correctly.

```
$ aws --version
aws-cli/1.25.55 Python/3.8.8 Darwin/18.7.0 botocore/1.13
```

If the program isn’t found, add it to your command line path (p. 19).

**Add the AWS CLI version 1 executable to your macOS command line path**

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

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

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

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

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

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

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

```
$ ls -al /usr/local/bin/python
~/Library/Python/3.7/bin/python3.7
```

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

**To modify your `PATH` variable**

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

```
$ ls -a ~
```
Troubleshooting AWS CLI install and uninstall errors

If you come across issues after installing or uninstalling the AWS CLI, see Troubleshooting errors (p. 167) for troubleshooting steps. For the most relevant troubleshooting steps, see the section called “Command not found errors” (p. 172), the section called “The "aws --version" command returns a different version than you installed” (p. 174), and the section called “The "aws --version" command returns a version after uninstalling the AWS CLI” (p. 175).

Install, Update, and Uninstall the AWS CLI version 1 on Windows

You can install version 1 of the AWS Command Line Interface (AWS CLI) on Windows by using a standalone installer (recommended) or pip, which is a package manager for Python.

Don't include the prompt symbol (C:\> ) when you type a command. These are included in program listings to differentiate commands that you type from output returned by the AWS CLI. The rest of this guide uses the generic prompt symbol ($), except in cases where a command is Windows-specific.
Install, update, and uninstall the AWS CLI version 1 using the MSI installer

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

Install and update the AWS CLI version 1 using the MSI installer

Check the Releases page on GitHub to see when the latest version was released. When updates are released, you must repeat the installation process to get the latest version of the AWS CLI version 1.

1. Download the appropriate MSI installer:
   - AWS CLI combined setup file for Windows: https://s3.amazonaws.com/aws-cli/AWSCLISetup.exe (includes both the 32-bit and 64-bit MSI installers, and automatically installs the correct version)
2. Run the downloaded MSI installer or the setup file.
3. Follow the on-screen instructions. By default, the AWS CLI version 1 installs to C:\Program Files\Amazon\AWSCLI (64-bit version) or C:\Program Files (x86)\Amazon\AWSCLI (32-bit version).
4. To confirm the installation, use the aws --version command at a command prompt (open the Start menu and search for cmd to start a command prompt).

   C:\> aws --version
   aws-cli/1.25.55 Python/3.8.8 Windows/10 botocore/1.13

   If Windows is unable to find the program, you might need to close and reopen the command prompt to refresh the path, or add the installation directory to your PATH (p. 23) environment variable manually.

Uninstall the AWS CLI version 1

To use the following uninstall instructions, you need to have installed the AWS CLI version 1 with the MSI installer or setup file.

1. Open Programs and Features by doing one of the following:
   - Open the Control Panel, and then choose Programs and Features.
   - Open a command prompt and enter the following command.

   C:\> appwiz.cpl
2. Select the entry named **AWS Command Line Interface**, and then choose **Uninstall** to launch the uninstaller.

3. Confirm that you want to uninstall the AWS CLI.

4. **(Optional)** Remove the shared AWS SDK and AWS CLI settings information in the .aws folder.

   **Warning**
   These configuration and credentials settings are shared across all AWS SDKs and the AWS CLI. If you remove this folder, they cannot be accessed by any AWS SDKs that are still on your system.

   The default location of the .aws folder differs between platforms, by default the folder is located in %UserProfile%\aws.

   ```bash
   $ rmdir %UserProfile%\aws
   ```

### Install, update, and uninstall the AWS CLI version 1 using Python and pip on Windows

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

**Prerequisites**

You must have Python 3.6 or later installed. For installation instructions, see the [Downloading Python page in Python's Beginner Guide](https://docs.python.org/3/tutorial/index.html).

**Warning**
Python 2.7 was deprecated by the Python Software Foundation on January 1, 2020. Starting with AWS CLI version 1.20.0, a minimum version of Python 3.6 is required. In order to use the AWS CLI version 1 with an older version of Python, you need to install an earlier version of the AWS CLI version 1. To view the AWS CLI version 1 Python version support matrix, see the section called “Python version requirements” (p. 4).

### Install and update the AWS CLI version 1 using pip

1. To install the AWS CLI version 1, use the `pip3` command (if you use Python version 3 or later) or the `pip` command.

   **For the latest version of the AWS CLI**, use the following command block:

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

   **For a specific version of the AWS CLI**, append a less-than symbol `<` and the version number to the filename. For this example the filename for version 1.16.312 would be `<1.16.312` resulting in the following command:

   ```bash
   C:\> pip3 install awscli<1.16.312 --upgrade --user
   ```

2. Verify that the AWS CLI version 1 is installed correctly. If there is no response, see the Add the AWS CLI version 1 executable to your command line path (p. 23) section.

   ```bash
   C:\> aws --version
   aws-cli/1.25.55 Python/3.8.8 Windows/10 botocore/1.13
   ```
Uninstall the AWS CLI version 1 using pip

1. If you installed the AWS CLI version 1 using pip, you must also uninstall using pip. If you use Python version 3 or later, we recommend that you use the pip3 command.

   ```
   C:\> pip3 uninstall awscli
   ```

   You might need to restart your command prompt window or your computer to remove all files.

2. (Optional) Remove the shared AWS SDK and AWS CLI settings information in the .aws folder.

   **Warning**
   These configuration and credentials settings are shared across all AWS SDKs and the AWS CLI. If you remove this folder, they cannot be accessed by any AWS SDKs that are still on your system.

   The default location of the .aws folder differs between platforms, by default the folder is located in `%UserProfile%\.aws`.

   ```
   $ rmdir %UserProfile%\.aws
   ```

Add the AWS CLI version 1 executable to your command line path

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

1. Use the where command to find the aws file location. By default, the where command shows where a specified program is found in the system's PATH.

   ```
   C:\> where aws
   ```

   The paths that show up depend on your platform and which method you used to install the AWS CLI. Folder names that include version numbers can vary. These examples reflect the use of Python version 3.7. Replace the version with the version number you're using, as needed. Typical paths include the following:

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

   Use the following steps based on whether a file path is returned.

   A file path is returned

   ```
   C:\> where aws
   C:\Program Files\Amazon\AWSCLI\bin\aws.exe
   ```
You can find where the `aws` program is installed by running the following command.

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

A file path is NOT returned

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

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

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

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

2. Press the Windows key and enter **environment variables**.
3. Choose **Edit environment variables for your account**.
4. Choose **PATH**, and then choose **Edit**.
5. Add the path you found into the **Variable value** field, for example, `C:\Program Files\Amazon\AWSCLI\bin\aws.exe`.
6. Choose **OK** twice to apply the new settings.
7. Close any running command prompts and reopen the command prompt window.

**Troubleshooting AWS CLI install and uninstall errors**

If you come across issues after installing or uninstalling the AWS CLI, see **Troubleshooting errors (p. 167)** for troubleshooting steps. For the most relevant troubleshooting steps, see the section called “Command not found errors” (p. 172), the section called “The "aws --version" command returns a different version than you installed” (p. 174), and the section called “The "aws --version" command returns a version after uninstalling the AWS CLI” (p. 175).

**Install and Update the AWS CLI version 1 in a virtual environment**

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

**Topics**
- Prerequisites (p. 25)
- Install and update the AWS CLI version 1 in a virtual environment (p. 25)
• Troubleshooting AWS CLI install and uninstall errors (p. 26)

Prerequisites

• Python 3.6 or later. For installation instructions, see the Downloading Python page in Python's Beginner Guide.

    Warning
    Python 2.7 was deprecated by the Python Software Foundation on January 1, 2020. Starting with AWS CLI version 1.20.0, a minimum version of Python 3.6 is required. In order to use the AWS CLI version 1 with an older version of Python, you need to install an earlier version of the AWS CLI version 1. To view the AWS CLI version 1 Python version support matrix, see the section called "Python version requirements" (p. 4).

• pip or pip3 is installed.

Install and update the AWS CLI version 1 in a virtual environment

1. Install virtualenv using pip.

    $ pip install --user virtualenv

2. Create a virtual environment and name it.

    $ virtualenv ~/cli-ve

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

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

3. Activate your new virtual environment.

    Linux or macOS

    $ source ~/cli-ve/bin/activate

    Windows

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

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

    (cli-ve)-$

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

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

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

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

**Troubleshooting AWS CLI install and uninstall errors**

If you come across issues after installing or uninstalling the AWS CLI, see [Troubleshooting errors](p. 167) for troubleshooting steps. For the most relevant troubleshooting steps, see the section called “Command not found errors” (p. 172), the section called “The `aws --version` command returns a different version than you installed” (p. 174), and the section called “The `aws --version` command returns a version after uninstalling the AWS CLI” (p. 175).
Configuring the AWS CLI

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

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

Topics
- Configuration basics (p. 27)
- Configuration and credential file settings (p. 30)
- Named profiles for the AWS CLI (p. 41)
- Environment variables to configure the AWS CLI (p. 43)
- Command line options (p. 47)
- Command completion (p. 50)
- AWS CLI retries (p. 54)
- Sourcing credentials with an external process (p. 57)
- Using credentials for Amazon EC2 instance metadata (p. 58)
- Using an HTTP proxy (p. 59)
- Using an IAM role in the AWS CLI (p. 61)

Configuration basics

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

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

Topics
- Quick configuration with aws configure (p. 28)
- Access key ID and secret access key (p. 28)
  - Creating a key pair (p. 28)
- Region (p. 29)
- Output format (p. 29)
- Profiles (p. 29)
- Configuration settings and precedence (p. 30)
Quick configuration with \texttt{aws configure}

For general use, the \texttt{aws configure} command is the fastest way to set up your AWS CLI installation. When you enter this command, the AWS CLI prompts you for four pieces of information:

- Access key ID (p. 28)
- Secret access key (p. 28)
- AWS Region (p. 29)
- Output format (p. 29)

The AWS CLI stores this information in a \textit{profile} (a collection of settings) named \texttt{default} in the \texttt{credentials} file. By default, the information in this profile is used when you run an AWS CLI command that doesn't explicitly specify a profile to use. For more information on the \texttt{credentials} file, see Configuration and credential file settings (p. 30)

The following example shows sample values. Replace them with your own values as described in the following sections.

```
$ 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
```

Access key ID and secret access key

Access keys use an access key ID and secret access key that you use to sign programmatic requests to AWS.

Topics

- Creating a key pair (p. 28)

Creating a key pair

Access keys consist of an access key ID and secret access key, which are used to sign programmatic requests that you make to AWS. If you don't have access keys, you can create them from the AWS Management Console. As a best practice, do not use the AWS account root user access keys for any task where it's not required. Instead, create a new administrator IAM user with access keys for yourself.

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

To create access keys for an IAM user

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

- Access key ID: AKIAIOSFODNN7EXAMPLE
- Secret access key: wJalrXUtnFEMI/K7MDENG/bPxRfiCyXZEXMPQL5Z06GJHBlT/5XGQ==

6. To download the key pair, choose **Download .csv file**. Store the keys in a secure location. You will not have access to the secret access key again after this dialog box closes.

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

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

**Related topics**

- What is IAM? in the **IAM User Guide**
- AWS security credentials in **AWS General Reference**

**Region**

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

**Note**

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

**Output format**

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

- **json** (p. 89) – The output is formatted as a JSON string.
- **text** (p. 90) – The output is formatted as multiple lines of tab-separated string values. This can be useful to pass the output to a text processor, like **grep**, **sed**, or **awk**.
- **table** (p. 92) – The output is formatted as a table using the characters +|- to form the cell borders. It typically presents the information in a “human-friendly” format that is much easier to read than the others, but not as programmatically useful.

**Profiles**

A collection of settings is called a profile. By default, the AWS CLI uses the default profile. You can create and use additional named profiles with varying credentials and settings by specifying the **--profile** option and assigning a name.

The following example creates a profile named produser.

```
$ aws configure --profile produser
```
AWS Command Line Interface User Guide for Version 1
Configuration settings and precedence

AWS Access Key ID [None]: AKIAI44QH8DBHEXAMPLE
AWS Secret Access Key [None]: je7MgGbCbBF/2Zp9Utuk/h3yCo8nvbEXAMPLEKEY
Default region name [None]: us-east-1
Default output format [None]: text

You can then specify a `--profile` profilename and use the credentials and settings stored under that name.

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

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

For more information on named profiles, see Named profiles for the AWS CLI (p. 41).

## Configuration settings and precedence

The AWS CLI uses credentials and configuration settings located in multiple places, such as the system or user environment variables, local AWS configuration files, or explicitly declared on the command line as a parameter. Certain locations take precedence over others. The AWS CLI credentials and configuration settings take precedence in the following order:

1. **Command line options (p. 47)** – Overrides settings in any other location. You can specify `--region`, `--output`, and `--profile` as parameters on the command line.
2. **Environment variables (p. 43)** – You can store values in your system's environment variables.
3. **CLI credentials file (p. 30)** – The credentials and config file are updated when you run the command `aws configure`. The credentials file is located at `~/.aws/credentials` on Linux or macOS, or at `C:\Users\USERNAME\aws\credentials` on Windows. This file can contain the credential details for the default profile and any named profiles.
4. **CLI configuration file (p. 30)** – The credentials and config file are updated when you run the command `aws configure`. The config file is located at `~/.aws/config` on Linux or macOS, or at `C:\Users\USERNAME\aws\config` on Windows. This file contains the configuration settings for the default profile and any named profiles.
5. **Container credentials** – You can associate an IAM role with each of your Amazon Elastic Container Service (Amazon ECS) task definitions. Temporary credentials for that role are then available to that task's containers. For more information, see IAM Roles for Tasks in the Amazon Elastic Container Service Developer Guide.
6. **Amazon EC2 instance profile credentials** – You can associate an IAM role with each of your Amazon Elastic Compute Cloud (Amazon EC2) instances. Temporary credentials for that role are then available to code running in the instance. The credentials are delivered through the Amazon EC2 metadata service. For more information, see IAM Roles for Amazon EC2 in the Amazon EC2 User Guide for Linux Instances and Using Instance Profiles in the IAM User Guide.

## Configuration and credential file settings

You can save your frequently used configuration settings and credentials in files that are maintained by the AWS CLI.

The files are divided into profiles. By default, the AWS CLI uses the settings found in the profile named default. To use alternate settings, you can create and reference additional profiles. For more information on named profiles, see Named profiles for the AWS CLI (p. 41).
You can override an individual setting by either setting one of the supported environment variables, or by using a command line parameter. For more information on configuration setting precedence, see Configuration settings and precedence (p. 30).

Topics
- Where are configuration settings stored? (p. 31)
- Set and view configuration settings (p. 31)
- Supported config file settings (p. 33)

Where are configuration settings stored?

The AWS CLI stores sensitive credential information that you specify with `aws configure` in a local file named `credentials`, in a folder named `.aws` in your home directory. The less sensitive configuration options that you specify with `aws configure` are stored in a local file named `config`, also stored in the `.aws` folder in your home directory.

**Storing credentials in the config file**

You can keep all of your profile settings in a single file as the AWS CLI can read credentials from the `config` file. If there are credentials in both files for a profile sharing the same name, the keys in the credentials file take precedence.

These files are also used by the various language software development kits (SDKs). If you use one of the SDKs in addition to the AWS CLI, confirm if the credentials should be stored in their own file.

Where you find your home directory location varies based on the operating system, but is referred to using the environment variables `%UserProfile%` in Windows and `$HOME` or `~` (tilde) in Unix-based systems. You can specify a non-default location for the files by setting the `AWS_CONFIG_FILE` and `AWS_SHARED_CREDENTIALS_FILE` environment variables to another local path. See Environment variables to configure the AWS CLI (p. 43) for details.

For example, the files generated by the AWS CLI for a default profile configured with `aws configure` looks similar to the following.

```
~/.aws/credentials
[default]
aws_access_key_id=AKIAIOSFODNN7EXAMPLE
aws_secret_access_key=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
```

```
~/.aws/config
[default]
region=us-west-2
output=json
```

For file examples with multiple named profiles, see Named profiles for the AWS CLI (p. 41).

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

Set and view configuration settings

There are several ways to view and set your configuration settings in the files.
**Credentials and config file**

View and edit your settings by directly editing the `config` and `credentials` files in a text editor. For more information see Where are configuration settings stored? (p. 31)

To remove a setting, delete the corresponding setting in your `config` and `credentials` files.

**aws configure**

Run this command to quickly set and view your credentials, Region, and output format. The following example shows sample values.

```
$ 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
```

For more information see Quick configuration with `aws configure` (p. 28)

**aws configure set**

You can set any credentials or configuration settings using `aws configure set`. Specify the profile that you want to view or modify with the `--profile` setting.

For example, the following command sets the `region` in the profile named `integ`.

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

To remove a setting, use an empty string as the value, or manually delete the setting in your `config` and `credentials` files in a text editor.

```
$ aws configure set cli_pager "" --profile integ
```

**aws configure get**

You can retrieve any credentials or configuration settings you've set using `aws configure get`. Specify the profile that you want to view or modify with the `--profile` setting.

For example, the following command retrieves the `region` setting in the profile named `integ`.

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

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

**aws configure list**

To list all configuration data, use the `aws configure list` command. This command displays the AWS CLI name of all settings you've configured, their values, and where the configuration was retrieved from.

```
$ aws configure list
```

```
<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Type</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>profile</td>
<td>&lt;not set&gt;</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>access_key</td>
<td>*******************ABCD</td>
<td>shared-credentials-file</td>
<td></td>
</tr>
<tr>
<td>secret_key</td>
<td>*******************ABCD</td>
<td>shared-credentials-file</td>
<td></td>
</tr>
</tbody>
</table>
```

32
Supported config file settings

Topics
- Global settings (p. 33)
- S3 Custom command settings (p. 38)

The following settings are supported in the config file. The values listed in the specified (or default) profile are used unless they are overridden by the presence of an environment variable with the same name, or a command line option with the same name. For more information on what order settings take precedence, see Configuration settings and precedence (p. 30)

Global settings

api_versions

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

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

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

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

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

aws_access_key_id (p. 28)

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

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

```
aws_access_key_id = AKIAIOSFODNN7EXAMPLE
```

aws_secret_access_key (p. 28)

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

Can be overridden by the AWS_SECRET_ACCESS_KEY environment variable. You can't specify the secret access key as a command line option.
aws_secret_access_key = wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY

aws_session_token

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

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

aws_session_token = AQoEXAMPLEH4aoAH0gNCAPyJxz48lCFFxNNE1OPTgk5TthT+FwqKnWcOifzRk3c/LTo6UDdySvQ0vVpVcLXrrU7dnniCEXAMPLE/IVw1dYug2RVAJbnL1hb41gRmpRvJsrkuWJ0ga8BIZZaiV2BxIa2R4olgk

cab_bundle

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

Can be overridden by the AWS_CA_BUNDLE (p. 44) environment variable or the --ca-bundle (p. 48) command line option.

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

cli_follow_urlparam

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

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

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

cli_follow_urlparam = false

cli_history

Disabled by default. This setting enables command history for the AWS CLI. After enabling this setting, the AWS CLI records the history of aws commands.

cli_history = enabled

You can list your history using the aws history list command, and use the resulting command_ids in the aws history show command for details. For more information see aws history in the AWS CLI reference guide.

cli_timestamp_format

Specifies the format of timestamp values included in the output. You can specify either of the following values:
• **iso8601** – The default value for the AWS CLI version 2. If specified, the AWS CLI reformats all timestamps according to ISO 8601.

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

```
```

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

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

• **wire** – The default value for the AWS CLI version 1. If specified, the AWS CLI displays all timestamp values exactly as received in the HTTP query response.

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

```
cli_timestamp_format = iso8601
```

**credential_process (p. 57)**

Specifies an external command that the AWS CLI runs to generate or retrieve authentication credentials to use for this command. The command must return the credentials in a specific format. For more information about how to use this setting, see Sourcing credentials with an external process (p. 57).

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

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

**credential_source (p. 61)**

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

This parameter can have one of three values:

• **Environment** – Specifies that the AWS CLI is to retrieve source credentials from environment variables.

• **Ec2InstanceMetadata** – Specifies that the AWS CLI is to use the IAM role attached to the EC2 instance profile to get source credentials.

• **EcsContainer** – Specifies that the AWS CLI is to use the IAM role attached to the ECS container as source credentials.

```
credential_source = Ec2InstanceMetadata
```

duration_seconds

Specifies the maximum duration of the role session, in seconds. The value can range from 900 seconds (15 minutes) up to the maximum session duration setting for the role (which can be a maximum of 43200). This is an optional parameter and by default, the value is set to 3600 seconds.
**external_id (p. 65)**

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

**max_attempts (p. 54)**

Specifies a value of maximum retry attempts the AWS CLI retry handler uses, where the initial call counts toward the max_attempts value that you provide.

You can override this value by using the AWS_MAX_ATTEMPTS environment variable.

```bash
max_attempts = 3
```

**mfa_serial (p. 64)**

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

**output (p. 29)**

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

- **json (p. 89)** – The output is formatted as a JSON string.
- **text (p. 90)** – The output is formatted as multiple lines of tab-separated string values. This can be useful to pass the output to a text processor, like grep, sed, or awk.
- **table (p. 92)** – The output is formatted as a table using the characters +|- to form the cell borders. It typically presents the information in a "human-friendly" format that is much easier to read than the others, but not as programmatically useful.

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

```bash
output = table
```

**parameter_validation**

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

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

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

```bash
parameter_validation = false
```

**region (p. 29)**

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

• aws_global enables you to specify the global endpoint for services that support a global endpoint in addition to Regional endpoints, such as AWS Security Token Service (AWS STS) and Amazon Simple Storage Service (Amazon S3).

You can override this value by using the AWS_DEFAULT_REGION environment variable, or the --region command line option.

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

**retry_mode (p. 54)**

Specifies which retry mode AWS CLI uses. There are three retry modes available: legacy (default), standard, and adaptive. For more information on retries, see AWS CLI retries (p. 54).

You can override this value by using the AWS_RETRY_MODE environment variable.

```bash
retry_mode = standard
```

**role_arn (p. 61)**

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

• source_profile

• credential_source

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

The environment variable AWS_ROLE_ARN (p. 45) overrides this setting.

For more information on using web identities, see the section called "Assume role with web identity" (p. 66).

**role_session_name (p. 65)**

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

```bash
role_session_name = maria_garcia_role
```

The environment variable AWS_ROLE_SESSION_NAME (p. 46) overrides this setting.

For more information on using web identities, see the section called "Assume role with web identity" (p. 66).

**source_profile (p. 61)**

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

Specifies how the AWS CLI determines the AWS service endpoint that the AWS CLI client uses to talk to the AWS Security Token Service (AWS STS). The default value for AWS CLI version 1 is `legacy`.

You can specify one of two values:

- **legacy** – Uses the global STS endpoint, `sts.amazonaws.com`, for the following AWS Regions: `ap-northeast-1`, `ap-south-1`, `ap-southeast-1`, `ap-southeast-2`, `aws-global`, `ca-central-1`, `eu-central-1`, `eu-north-1`, `eu-west-1`, `eu-west-2`, `eu-west-3`, `sa-east-1`, `us-east-1`, `us-east-2`, `us-west-1`, and `us-west-2`. All other Regions automatically use their respective Regional endpoint.

- **regional** – The AWS CLI always uses the AWS STS endpoint for the currently configured Region. For example, if the client is configured to use `us-west-2`, all calls to AWS STS are made to the Regional endpoint `sts.us-west-2.amazonaws.com` instead of the global `sts.amazonaws.com` endpoint. To send a request to the global endpoint while this setting is enabled, you can set the Region to `aws-global`.

This setting can be overwritten by using the `AWS_STS_REGIONAL_ENDPOINTS` environment variable. You can't set this value as a command line parameter.

**web_identity_token_file (p. 66)**

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

The environment variable `AWS_WEB_IDENTITY_TOKEN_FILE` (p. 47) overrides this setting.

For more information on using web identities, see the section called “Assume role with web identity” (p. 66).

**tcp_keepalive**

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

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

```
tcp_keepalive = false
```

### S3 Custom command settings

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

All of these options can be configured by specifying the `s3` nested setting in your `config` file. Each setting is then indented on its own line.
Note
These settings are entirely optional. You should be able to successfully use the `aws s3` transfer commands without configuring any of these settings. These settings are provided to enable you to tune for performance or to account for the specific environment where you are running these `aws s3` commands.

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

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

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

addressing_style

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

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

payload_signing_enabled

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

If set to true, S3 requests receive additional content validation in the form of a SHA256 checksum which is calculated for you and included in the request signature. If set to false, the checksum isn't calculated. Disabling this can be useful to reduce the performance overhead created by the checksum calculation.

use_dualstack_endpoint

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

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

use_accelerate_endpoint

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

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

max_bandwidth

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

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

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

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

max_concurrent_requests

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

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

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

- Decreasing this value – On some environments, the default of 10 concurrent requests can overwhelm a system. This can cause connection timeouts or slow the responsiveness of the system. Lowering this value makes the S3 transfer commands less resource intensive. The tradeoff is that S3 transfers can take longer to complete. Lowering this value might be necessary if you use a tool to limit bandwidth.
- Increasing this value – In some scenarios, you might want the Amazon S3 transfers to complete as quickly as possible, using as much network bandwidth as necessary. In this scenario, the default number of concurrent requests might not be sufficient to use all of the available network bandwidth. Increasing this value can improve the time it takes to complete an Amazon S3 transfer.

max_queue_size

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

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

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

`multipart_chunksize`

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

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

`multipart_threshold`

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

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

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

**Note**

S3 can impose constraints on valid values that can be used for multipart operations. For more information, see the S3 Multipart Upload documentation in the Amazon Simple Storage Service User Guide.

## Named profiles for the AWS CLI

A *named profile* is a collection of settings and credentials that you can apply to a AWS CLI command. When you specify a profile to run a command, the settings and credentials are used to run that command. Multiple *named profiles* can be stored in the `config` and `credentials` files.

You can specify one default profile that is used when no profile is explicitly referenced. Other profiles have names that you can specify as a parameter on the command line for individual commands. Alternatively, you can specify a profile in the `AWS_PROFILE (p. 43)` environment variable which overrides the default profile for commands that run in that session.

### Topics

- Creating named profiles (p. 41)
- Using named profiles (p. 42)

## Creating named profiles

You can configure additional profiles by using `aws configure (p. 31)` with the `--profile` option, or by manually adding entries to the `config` and `credentials` files. For more information on the `config` and `credentials` files, see the section called “Configuration and credential file settings” (p. 30).
Using named profiles

Credentials profile

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

The credentials file uses a different naming format than the AWS CLI config file for named profiles. Do not use the word profile when creating an entry in the credentials file.

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

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

[user1]
aws_access_key_id=AKIAI44QH8DHBEEXAMPLE
aws_secret_access_key=je7MtGbClwBF/2Zp9UtK/h3yCo8nvbEXAMPLEKEY
```

Config profile

Each profile can specify different credentials—perhaps from different IAM users—and can also specify different AWS Regions and output formats. When naming the profile in a config file, include the prefix word "profile".

The following example specifies Region and output information for the default and user1 profiles.

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

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

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

Using named profiles

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

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

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

Linux or macOS

```
$ export AWS_PROFILE=user1
```

Windows

```
C:\> setx AWS_PROFILE user1
```
Using `set` to set an environment variable changes the value used until the end of the current command prompt session, or until you set the variable to a different value.

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

Setting the environment variable changes the default profile until the end of your shell session, or until you set the variable to a different value. You can make environment variables persistent across future sessions by putting them in your shell’s startup script. For more information, see Environment variables to configure the AWS CLI (p. 43).

**Note**
If you specify a profile with `--profile` on an individual command, that overrides the setting specified in the environment variable for only that command.

### Environment variables to configure the AWS CLI

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

**Precedence of options**

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

For more information about precedence and how the AWS CLI determines which credentials to use, see Configuration settings and precedence (p. 30).

**Topics**

- How to set environment variables (p. 43)
- AWS CLI supported environment variables (p. 44)

### How to set environment variables

The following examples show how you can configure environment variables for the default user.

**Linux or macOS**

```bash
# export AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
# export AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
# export AWS_DEFAULT_REGION=us-west-2
```

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

**Windows Command Prompt**

**To set for all sessions**

```cmd
C:\> setx AWS_ACCESS_KEY_ID AKIAIOSFODNN7EXAMPLE
C:\> setx AWS_SECRET_ACCESS_KEY wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
```
Using `setx` to set an environment variable changes the value used in both the current command prompt session and all command prompt sessions that you create after running the command. It does not affect other command shells that are already running at the time you run the command. You may need to restart your terminal for settings to load.

**To set for current session only**

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

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

**PowerShell**

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

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

**AWS CLI supported environment variables**

The AWS CLI supports the following environment variables.

**AWS_ACCESS_KEY_ID**

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

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

**AWS_CA_BUNDLE**

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

If defined, this environment variable overrides the value for the profile setting `ca_bundle` (p. 34). You can override this environment variable by using the `--ca-bundle` (p. 48) command line parameter.

**AWS_CONFIG_FILE**

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

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

**AWS_DATA_PATH**

A list of additional directories to check outside of the built-in search path of `~/.aws/models` when loading AWS CLI data. Setting this environment variable indicates additional directories to check
first before falling back to the built-in search path. Multiple entries should be separated with the os.pathsep character, which is : on Linux or macOS and ; on Windows.

AWS_DEFAULT_OUTPUT (p. 29)

Specifies the output format (p. 88) to use. If defined, this environment variable overrides the value for the profile setting output. You can override this environment variable by using the --output command line parameter.

AWS_DEFAULT_REGION (p. 29)

Specifies the AWS Region to send the request to. If defined, this environment variable overrides the value for the profile setting region and . You can override this environment variable by using the --region command line parameter.

AWS_EC2_METADATA_DISABLED

Disables the use of the Amazon EC2 instance metadata service (IMDS). If set to true, user credentials or configuration (like the Region) are not requested from IMDS.

AWS_MAX_ATTEMPTS (p. 36)

Specifies a value of maximum retry attempts the AWS CLI retry handler uses, where the initial call counts toward the value that you provide. For more information on retries, see AWS CLI retries (p. 54).

If defined, this environment variable overrides the value for the profiles setting max_attempts.

AWS_METADATA_SERVICE_NUM_ATTEMPTS

When attempting to retrieve credentials on an Amazon EC2 instance that has been configured with an IAM role, the AWS CLI attempts to retrieve credentials once from the instance metadata service before stopping. If you know your commands will run on an Amazon EC2 instance, you can increase this value to make AWS CLI retry multiple times before giving up.

AWS_METADATA_SERVICE_TIMEOUT

The number of seconds before a connection to the instance metadata service should time out. When attempting to retrieve credentials on an Amazon EC2 instance that is configured with an IAM role, a connection to the instance metadata service times out after 1 second by default. If you know you're running on an Amazon EC2 instance with an IAM role configured, you can increase this value if needed.

AWS_PROFILE (p. 41)

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

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

AWS_RETRY_MODE (p. 37)

Specifies which retry mode AWS CLI uses. There are three retry modes available: legacy (default), standard, and adaptive. For more information on retries, see AWS CLI retries (p. 54).

If defined, this environment variable overrides the value for the profiles setting retry_mode.

AWS_ROLE_ARN

Specifies the Amazon Resource Name (ARN) of an IAM role with a web identity provider that you want to use to run the AWS CLI commands.
Used with the AWS_WEB_IDENTITY_TOKEN_FILE and AWS_ROLE_SESSION_NAME environment variables.

If defined, this environment variable overrides the value for the profile setting role_arn (p. 37). You can't specify a role session name as a command line parameter.

**Note**
This environment variable only applies to an assumed role with web identity provider it does not apply to the general assume role provider configuration.

For more information on using web identities, see the section called “Assume role with web identity” (p. 66).

AWS_ROLE_SESSION_NAME

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

If defined, this environment variable overrides the value for the profile setting role_session_name (p. 37).

Used with the AWS_ROLE_ARN and AWS_WEB_IDENTITY_TOKEN_FILE environment variables.

For more information on using web identities, see the section called “Assume role with web identity” (p. 66).

**Note**
This environment variable only applies to an assumed role with web identity provider it does not apply to the general assume role provider configuration.

AWS_SECRET_ACCESS_KEY

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

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

AWS_SESSION_TOKEN

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

If defined, this environment variable overrides the value for the profile setting aws_session_token.

AWS_SHARED_CREDENTIALS_FILE

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

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

AWS_STS_REGIONAL_ENDPOINTS (p. 38)

 Specifies how the AWS CLI determines the AWS service endpoint that the AWS CLI client uses to talk to the AWS Security Token Service (AWS STS). The default value for AWS CLI version 1 is legacy.

You can specify one of two values:
• **legacy** – Uses the global STS endpoint, sts.amazonaws.com, for the following AWS Regions: ap-northeast-1, ap-south-1, ap-southeast-1, ap-southeast-2, aws-global, ca-central-1, eu-central-1, eu-north-1, eu-west-1, eu-west-2, eu-west-3, sa-east-1, us-east-1, us-east-2, us-west-1, and us-west-2. All other Regions automatically use their respective Regional endpoint.

• **regional** – The AWS CLI always uses the AWS STS endpoint for the currently configured Region. For example, if the client is configured to use us-west-2, all calls to AWS STS are made to the Regional endpoint sts.us-west-2.amazonaws.com instead of the global sts.amazonaws.com endpoint. To send a request to the global endpoint while this setting is enabled, you can set the Region to aws-global.

**AWS_WEB_IDENTITY_TOKEN_FILE (p. 43)**

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

Used with the AWS_ROLE_ARN and AWS_ROLE_SESSION_NAME environment variables.

If defined, this environment variable overrides the value for the profile setting web_identity_token_file.

For more information on using web identities, see the section called “Assume role with web identity” (p. 66).

**Note**

This environment variable only applies to an assumed role with web identity provider it does not apply to the general assume role provider configuration.

---

## Command line options

In the AWS CLI, command line options are global parameters you can use to override the default configuration settings, any corresponding profile setting, or environment variable setting for that single command. You can't use command line options to directly specify credentials, although you can specify which profile to use.

**Topics**

- How to use command line options (p. 47)
- AWS CLI supported global command line options (p. 48)
- Common uses of command line options (p. 49)

### How to use command line options

Most command line options are simple strings, such as the profile name `profile1` in the following example:

```bash
$ aws s3 ls --profile profile1
example-bucket-1
example-bucket-2
...
```

Each option that takes an argument requires a space or equals sign (=) separating the argument from the option name. If the argument value is a string that contains a space, you must use quotation marks around the argument. For details on argument types and formatting for parameters, see Specifying parameter values for the AWS CLI (p. 74).
AWS CLI supported global command line options

In the AWS CLI you can use the following command line options to override the default configuration settings, any corresponding profile setting, or environment variable setting for that single command.

--ca-bundle <string>

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

If defined, this option overrides the value for the profile setting ca_bundle (p. 34) and the AWS_CA_BUNDLE (p. 44) environment variable.

--cli-connect-timeout <integer>

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

--cli-read-timeout <integer>

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

--color <string>

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

--debug

A Boolean switch that enables debug logging. The AWS CLI by default provides cleaned up information regarding any successes or failures regarding command outcomes in the command output. The --debug option provides the full Python logs. This includes additional stderr diagnostic information about the operation of the command that can be useful when troubleshooting why a command provides unexpected results. To easily view debug logs, we suggest sending the logs to a file to more easily search the information. You can do this by using one of the following.

To send only the stderr diagnostic information, append 2> debug.txt where debug.txt is the name you want to use for your debug file:

```
$ aws servicename commandname options --debug 2> debug.txt
```

To send both the output and stderr diagnostic information, append &> debug.txt where debug.txt is the name you want to use for your debug file:

```
$ aws servicename commandname options --debug &> debug.txt
```

--endpoint-url <string>

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

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

--no-paginate

A Boolean switch that disables the multiple calls the automatically AWS CLI makes to receive all command results that creates pagination of the output. This means only the first page of your output is displayed.
Common uses of command line options

Common uses for command line options include checking your resources in multiple AWS Regions, and changing the output format for legibility or ease of use when scripting. In the following examples, we run the `describe-instances` command against each Region until we find which Region our instance is in.

```
$ aws ec2 describe-instances --output table --region us-west-1
DescribeInstances
+-----------------+
| DescribeInstances |
+-----------------+

$ aws ec2 describe-instances --output table --region us-west-2
DescribeInstances
+-----------------+
<table>
<thead>
<tr>
<th>DescribeInstances</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reservations</td>
</tr>
<tr>
<td>OwnerId</td>
</tr>
<tr>
<td>ReservationId</td>
</tr>
<tr>
<td>Instances</td>
</tr>
<tr>
<td>AmiLaunchIndex</td>
</tr>
<tr>
<td>Architecture</td>
</tr>
</tbody>
</table>
```
Command completion

The AWS Command Line Interface (AWS CLI) includes a bash-compatible command-completion feature that enables you to use the Tab key to complete a partially entered command. On most systems you need to configure this manually.

Topics

• How it works (p. 50)
• Configuring command completion on Linux or macOS (p. 50)
• Configuring command completion on Windows (p. 53)

How it works

When you partially enter a command, parameter, or option, the command-completion feature either automatically completes your command or displays a suggested list of commands. To prompt command completion, you partially enter in a command and press the completion key, which is typically Tab in most shells.

The following examples show different ways that you can use command completion:

• Partially enter a command and press Tab to display a suggested list of commands.

```
$ aws dynamodb dTAB
delete-backup delete-item delete-table describe-backup describe-continuous-backups describe-contributor-insights describe-endpoints
```

describe-global-table describe-global-table-settings describe-limits describe-table describe-table-replica-auto-scaling describe-time-to-live

• Partially enter a parameter and press Tab to display a suggested list of parameters.

```
$ aws dynamodb delete-table --TAB
--ca-bundle --cli-connect-timeout --cli-input-json --cli-read-timeout --color --debug
--cli-connect-timeout --cli-input-json --cli-read-timeout --color --debug
```

--endpoint-url --generate-cli-skeleton --query --no-paginate --region
--no-sign-request --table-name
--no-verify-ssl --version

• Enter a parameter and press Tab to display a suggested list of resource values. This feature is available only in the AWS CLI version 2.

```
$ aws dynamodb db delete-table --table-name TAB
Table 1 Table 2 Table 3
```

Configuring command completion on Linux or macOS

To configure command completion on Linux or macOS, you must know the name of the shell you're using and the location of the aws_completer script.
Note
Command completion is automatically configured and enabled by default on Amazon EC2 instances that run Amazon Linux.

Topics
- Confirm the completer's folder is in your path (p. 51)
- Enable command completion (p. 52)
- Verify command completion (p. 53)

Confirm the completer's folder is in your path

For the AWS completer to work successfully, the `aws_completer` needs to be in your shell's path. The `which` command can check if the completer is in your path.

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

If the `which` command can't find the completer, then use the following steps to add the completer's folder to your path.

Step 1: Locate the AWS completer

The location of the AWS completer can vary depending on the installation method used.

- **Package Manager** - Programs such as `pip`, `yum`, `brew`, and `apt-get` typically install the AWS completer (or a symlink to it) to a standard path location.
  - If you used `pip` without the `--user` parameter, the default path is `/usr/local/bin/aws_completer`.
  - If you used `pip` with the `--user` parameter the default path is `/home/username/.local/bin/aws_completer`.
- **Bundled Installer** - If you used the bundled installer, the default path is `/usr/local/bin/aws_completer`.

If all else fails, you can use the `find` command to search your file system for the AWS completer.

```
$ find / -name aws_completer
/usr/local/bin/aws_completer
```

Step 2: Identify your shell

To identify which shell you're using, you can use one of the following commands.

- **`echo $SHELL`** – Displays the shell's program file name. This usually matches the name of the in-use shell, unless you launched a different shell after logging in.

```
$ echo $SHELL
/bin/bash
```

- **`ps`** – Displays the processes running for the current user. One of them is the shell.

```
$ ps
  PID TTY     TIME CMD
```
Step 3: Add the completer to your path

1. Find your shell's profile script in your user folder.

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

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

2. Add an export command at the end of your profile script that's similar to the following example. Replace `/usr/local/bin/` with the folder that you discovered in the previous section.

   ```bash
   export PATH=/usr/local/bin/:$PATH
   ```

3. Reload the profile into the current session to put those changes into effect. Replace `.bash_profile` with the name of the shell script you discovered in the first section.

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

Enable command completion

After confirming the completer is in your path, enable command completion by running the appropriate command for the shell that you're using. You can add the command to your shell's profile to run it each time you open a new shell. In each command, replace the `/usr/local/bin/` path with the one found on your system in Confirm the completer's folder is in your path (p. 51).

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

  ```bash
  $ complete -C '/usr/local/bin/aws_completer' aws
  ```

  Add the previous command to `~/.bashrc` to run it each time you open a new shell. Your `~/.bash_profile` should source `~/.bashrc` to ensure that the command is also run in login shells.

- **zsh** – To run command completion, you need to run `bashcompinit` by adding the following autoload line at the end of your `~/.zshrc` profile script.

  ```bash
  $ autoload bashcompinit && bashcompinit
  $ autoload -Uz compinit && compinit
  ```

  To enable command completion, use the built-in command `complete`.

  ```bash
  $ complete -C '/usr/local/bin/aws_completer' aws
  ```

  Add the previous commands to `~/.zshrc` to run it each time you open a new shell.

- **tcsh** – Complete for `tcsh` takes a word type and pattern to define the completion behavior.

  ```bash
  > complete aws 'p/*/`aws_completer`/'
  ```
Add the previous command to ~/.tschrc to run it each time you open a new shell.

After you've enabled command completion, Verify command completion (p. 53) is working.

### Verify command completion

After enabling command completion, reload your shell, enter a partial command, and press Tab to see the available commands.

```bash
$ aws s TAB
s3       ses       sqs       sts       swf
s3api     sns       storagegateway support
```

### Configuring command completion on Windows

**Note**

For information on how PowerShell handles their completion, including their various completion keys, see about_Tab_Expansion in the Microsoft PowerShell Docs.

To enable command completion for PowerShell on Windows, complete the following steps in PowerShell.

1. Open your $PROFILE with the following command.

   ```bash
   PS C:\> Notepad $PROFILE
   ```

   If you do not have a $PROFILE, create a user profile using the following command.

   ```bash
   PS C:\> if (!(Test-Path -Path $PROFILE )) { New-Item -Type File -Path $PROFILE -Force }
   ```

   For more information on PowerShell profiles, see How to Use Profiles in Windows PowerShell ISE on the Microsoft Docs website.

2. To enable command completion, add the following code block to your profile, save, and then close the file.

   ```powershell
   Register-ArgumentCompleter -Native -CommandName aws -ScriptBlock {
     param($commandName, $wordToComplete, $cursorPosition)

     $env:COMP_LINE=$wordToComplete
     if ($env:COMP_LINE.Length -lt $cursorPosition){
       $env:COMP_LINE+ "$ "$
     }

     $env:COMP_POINT=$cursorPosition

     aws_completer.exe | ForEach-Object {
     }

     Remove-Item Env:\COMP_LINE
     Remove-Item Env:\COMP_POINT
   }
   ```

3. After enabling command completion, reload your shell, enter a partial command, and press Tab to cycle through the available commands.

   ```bash
   $ aws s Tab
   ```
AWS CLI retries

This topic describes how the AWS CLI might see calls to AWS services fail due to unexpected issues. These issues can occur on the server side or might fail due to rate limiting from the AWS service you’re attempting to call. These kinds of failures usually don’t require special handling and the call is automatically made again, often after a brief waiting period. The AWS CLI provides many features to assist in retrying client calls to AWS services when these kinds of errors or exceptions are experienced.

Topics
- Available retry modes (p. 54)
- Configuring a retry mode (p. 56)
- Viewing logs of retry attempts (p. 56)

Available retry modes

The AWS CLI has multiple modes to choose from depending on your version:

- Legacy retry mode (p. 54)
- Standard retry mode (p. 55)
- Adaptive retry mode (p. 55)

Legacy retry mode

Legacy mode is the default mode used by the AWS CLI version 1. Legacy mode uses an older retry handler that has limited functionality which includes:

- A default value of 4 for maximum retry attempts, making a total of 5 call attempts. This value can be overwritten through the `max_attempts` configuration parameter.
- DynamoDB has a default value of 9 for maximum retry attempts, making a total of 10 call attempts. This value can be overwritten through the `max_attempts` configuration parameter.
- Retry attempts for the following limited number of errors/exceptions:
  - General socket/connection errors:
    - `ConnectionError`
    - `ConnectionClosedError`
    - `ReadTimeoutError`
    - `EndpointConnectionError`
  - Service-side throttling/limit errors and exceptions:
    - `Throttling`
    - `ThrottlingException`
Available retry modes

- ThrottledException
- RequestThrottledException
- ProvisionedThroughputExceededException

- Retry attempts on several HTTP status codes, including 429, 500, 502, 503, 504, and 509.
- Any retry attempt will include an exponential backoff by a base factor of 2.

Standard retry mode

Standard mode is a standard set of retry rules across the AWS SDKs with more functionality than legacy. Standard mode was created for the AWS CLI version 2 and is backported to AWS CLI version 1. Standard mode’s functionality includes:

- A default value of 2 for maximum retry attempts, making a total of 3 call attempts. This value can be overwritten through the `max_attempts` configuration parameter.
- Retry attempts for the following expanded list of errors/exceptions:
  - Transient errors/exceptions
    - RequestTimeout
    - RequestTimeoutException
    - PriorRequestNotComplete
    - ConnectionError
    - HTTPClientError
  - Service-side throttling/limit errors and exceptions:
    - Throttling
    - ThrottlingException
    - ThrottledException
    - RequestThrottledException
    - TooManyRequestsException
    - ProvisionedThroughputExceededException
    - TransactionInProgressException
    - RequestLimitExceeded
    - BandwidthLimitExceeded
    - LimitExceeded
    - RequestThrottled
    - SlowDown
    - EC2ThrottledException

- Retry attempts on nondescriptive, transient error codes. Specifically, these HTTP status codes: 500, 502, 503, 504.
- Any retry attempt will include an exponential backoff by a base factor of 2 for a maximum backoff time of 20 seconds.

Adaptive retry mode

**Warning**

Adaptive mode is an experimental mode and is subject to change, both in features and behavior.

Adaptive retry mode is an experimental retry mode that includes all the features of standard mode. In addition to the standard mode features, adaptive mode also introduces client-side rate limiting through
the use of a token bucket and rate-limit variables that are dynamically updated with each retry attempt. This mode offers flexibility in client-side retries that adapts to the error/exception state response from an AWS service.

With each new retry attempt, adaptive mode modifies the rate-limit variables based on the error, exception, or HTTP status code presented in the response from the AWS service. These rate-limit variables are then used to calculate a new call rate for the client. Each exception/error or non-success HTTP response (provided in the list above) from an AWS service updates the rate-limit variables as retries occur until success is reached, the token bucket is exhausted, or the configured maximum attempts value is reached.

**Configuring a retry mode**

The AWS CLI includes a variety of both retry configurations as well as configuration methods to consider when creating your client object.

**Available configuration methods**

In the AWS CLI, users can configure retries in the following ways:

- Environment variables
- AWS CLI configuration file

Users can customize the following retry options:

- **Retry mode** - Specifies which retry mode the AWS CLI uses. As described previously, there are three retry modes available: legacy, standard, and adaptive. The default value for the AWS CLI version 1 is legacy.
- **Max attempts** - Specifies the value of maximum retry attempts the AWS CLI retry handler uses, where the initial call counts toward the value that you provide. The default value is 5.

**Defining a retry configuration in your environment variables**

To define your retry configuration for the AWS CLI, update your operating system's environment variables.

The retry environment variables are:

- `AWS_RETRY_MODE`
- `AWS_MAX_ATTEMPTS`

For more information on environment variables, see [Environment variables to configure the AWS CLI](p. 43).

**Viewing logs of retry attempts**

The AWS CLI uses Boto3's retry methodology and logging. You can use the --debug option on any command to receive debug logs. For more information on how to use the --debug option, see [Command line options](p. 47).

If you search for "retry" in your debug logs, you'll find the retry information you need. The client log entries for retry attempts depend on which retry mode you've enabled.

**Legacy mode:**
Retry messages are generated by botocore.retryhandler. You'll see one of three messages:

- No retry needed
- Retry needed, action of: `<action_name>`
- Reached the maximum number of retry attempts: `<attempt_number>`

**Standard or adaptive mode:**

Retry messages are generated by botocore.retries.standard. You'll see one of three messages:

- No retrying request
- Retry needed, retrying request after delay of: `<delay_value>`
- Retry needed but retry quota reached, not retrying request

For the full definition file of botocore retries, see `_retry.json` on the botocore GitHub Repository.

---

**Sourcing credentials with an external process**

**Warning**

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

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

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

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

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

**Syntax**

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

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

**Example for Windows**

```
credential_process = "C:\Path\To\credentials.cmd" parameterWithoutSpaces "parameter with spaces"
```
Example for Linux or macOS

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

Expected output from the Credentials program

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

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

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

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

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

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

Using credentials for Amazon EC2 instance metadata

When you run the AWS CLI from within an Amazon Elastic Compute Cloud (Amazon EC2) instance, you can simplify providing credentials to your commands. Each Amazon EC2 instance contains metadata that the AWS CLI can directly query for temporary credentials. When an IAM role is attached to the instance, the AWS CLI automatically and securely retrieves the credentials from the instance metadata.

To disable this service, use the AWS_EC2_METADATA_DISABLED (p. 45) environment variable.

Topics
- Prerequisites (p. 58)
- Configuring a profile for Amazon EC2 metadata (p. 59)

Prerequisites

To use Amazon EC2 credentials with the AWS CLI, you need to complete the following:

- Install and configure the AWS CLI. For more information, see Installing the AWS CLI (p. 4) and Configuration basics (p. 27).
You understand configuration files and named profiles. For more information, see Configuration and credential file settings (p. 30) and Named profiles for the AWS CLI (p. 41).

You’ve created an AWS Identity and Access Management (IAM) role that has access to the resources needed, and attached that role to the Amazon EC2 instance when you launch it. For more information, see IAM policies for Amazon EC2 in the Amazon EC2 User Guide for Linux Instances and Granting Applications That Run on Amazon EC2 Instances Access to AWS Resources in the IAM User Guide.

Configuring a profile for Amazon EC2 metadata

To specify that you want to use the credentials available in the hosting Amazon EC2 instance profile, use the following syntax in the named profile in your configuration file. See the following steps for more instructions.

```
[profile profilename]
role_arn = arn:aws:iam::123456789012:role/rolename
credential_source = Ec2InstanceMetadata
region = region
```

1. Create a profile in your configuration file.

```
[profile profilename]
```

2. Add your IAM arn role that has access to the resources needed.

```
role_arn = arn:aws:iam::123456789012:role/rolename
```

3. Specify Ec2InstanceMetadata as your credential source.

```
credential_source = Ec2InstanceMetadata
```

4. Set your Region.

```
region = region
```

Example

The following example assumes the marketingadminrole role and uses the us-west-2 Region in an Amazon EC2 instance profile named marketingadmin.

```
[profile marketingadmin]
role_arn = arn:aws:iam::123456789012:role/marketingadminrole
credential_source = Ec2InstanceMetadata
region = us-west-2
```

Using an HTTP proxy

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

Note
The following examples show the environment variable name in all uppercase letters. However, if you specify a variable twice using different cases, the lowercase letters take precedence. We recommend that you define each variable only once to avoid system confusion and unexpected behavior.

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

Linux or macOS

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

Windows Command Prompt

To set for all sessions

```
C:\> setx HTTP_PROXY http://10.15.20.25:1234
C:\> setx HTTP_PROXY http://proxy.example.com:1234
C:\> setx HTTPS_PROXY http://10.15.20.25:5678
C:\> setx HTTPS_PROXY http://proxy.example.com:5678
```

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

To set for current session only

```
C:\> set HTTP_PROXY=http://10.15.20.25:1234
C:\> set HTTP_PROXY=http://proxy.example.com:1234
C:\> set HTTPS_PROXY=http://10.15.20.25:5678
C:\> set HTTPS_PROXY=http://proxy.example.com:5678
```

Authenticating to a proxy

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

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

```
Using a proxy on Amazon EC2 instances

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

**Linux or macOS**

```bash
$ export NO_PROXY=169.254.169.254
```

**Windows Command Prompt**

To set for all sessions

```cmd
C:\> set NO_PROXY=169.254.169.254
```

To set for current session only

```cmd
C:\> set NO_PROXY=169.254.169.254
```

Using an IAM role in the AWS CLI

An [AWS Identity and Access Management (IAM) role](https://docs.aws.amazon.com/cli/latest/userguide/iam-roles-using.html) is an authorization tool that lets an IAM user gain additional (or different) permissions, or get permissions to perform actions in a different AWS account.

**Topics**

- Prerequisites (p. 62)
- Overview of using IAM roles (p. 62)
- Configuring and using a role (p. 63)
- Using multi-factor authentication (p. 64)
- Cross-account roles and external ID (p. 65)
- Specifying a role session name for easier auditing (p. 65)
Prerequisites

To run the `iam` commands, you need to install and configure the AWS CLI. For more information, see *Installing the AWS CLI* (p. 4).

Overview of using IAM roles

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

The following example shows a role profile named `marketingadmin`. If you run commands with `--profile marketingadmin` (or specify it with the `AWS_PROFILE` environment variable (p. 43)), the AWS CLI uses the credentials defined in a separate profile `user1` to assume the role with the Amazon Resource Name (ARN) `arn:aws:iam::123456789012:role/marketingadminrole`. You can run any operations that are allowed by the permissions assigned to that role.

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

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

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

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

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

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

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

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

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

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

The following example shows a trust policy that you could attach to a role. This policy allows the role to be assumed by any IAM user in the account 123456789012, if the administrator of that account explicitly grants the `sts:assume-role` permission to the user.

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

The trust policy doesn't actually grant permissions. The administrator of the account must delegate the permission to assume the role to individual users by attaching a policy with the appropriate permissions. The following example shows a policy that you can attach to an IAM user that allows the user to assume only the `marketingadminrole` role. For more information about granting a user access to assume a role, see Granting a User Permission to Switch Roles in the IAM User Guide.

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

The IAM user doesn't need to have additional permissions to run the AWS CLI commands using the role profile. Instead, the permissions to run the command come from those attached to the role. You attach permission policies to the role to specify which actions can be performed against which AWS resources. For more information about attaching permissions to a role (which works identically to an IAM user), see Changing Permissions for an IAM User in the IAM User Guide.

Now that you have the role profile, role permissions, role trust relationship, and user permissions correctly configured, you can use the role at the command line by invoking the `--profile` option.
For example, the following calls the Amazon S3 ls command using the permissions attached to the marketingadmin role as defined by the example at the beginning of this topic.

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

To use the role for several calls, you can set the AWS_PROFILE environment variable for the current session from the command line. While that environment variable is defined, you don’t have to specify the --profile option on each command.

**Linux or macOS**

```bash
# export AWS_PROFILE=marketingadmin
```

**Windows**

```bash
C:\> setx AWS_PROFILE marketingadmin
```

For more information about configuring IAM users and roles, see Users and Groups and Roles in the IAM User Guide.

**Using multi-factor authentication**

For additional security, you can require that users provide a one-time key generated from a multi-factor authentication (MFA) device, a U2F device, or mobile app when they attempt to make a call using the role profile.

First, you can choose to modify the trust relationship on the IAM role to require MFA. This prevents anyone from using the role without first authenticating by using MFA. For an example, see the Condition line in the following example. This policy allows the IAM user named anika to assume the role the policy is attached to, but only if they authenticate by using MFA.

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

Next, add a line to the role profile that specifies the ARN of the user's MFA device. The following sample config file entries show two role profiles that both use the access keys for the IAM user anika to request temporary credentials for the role cli-role. The user anika has permissions to assume the role, granted by the role's trust policy.

```
[profile role-without-mfa]
region = us-west-2
role_arn= arn:aws:iam::128716708097:role/cli-role
source_profile=cli-user

[profile role-with-mfa]
region = us-west-2
role_arn= arn:aws:iam::128716708097:role/cli-role
```
source_profile = cli-user
mfa_serial = arn:aws:iam::128716708097:mfa/cli-user

[profile anika]
region = us-west-2
output = json

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

The first profile, `role-without-mfa`, doesn't require MFA. However, because the previous example trust policy attached to the role requires MFA, any attempt to run a command with this profile fails.

```shell
$ aws iam list-users --profile role-without-mfa
An error occurred (AccessDenied) when calling the AssumeRole operation: Access denied
```

The second profile entry, `role-with-mfa`, identifies an MFA device to use. When the user attempts to run a AWS CLI command with this profile, the AWS CLI prompts the user to enter the one-time password (OTP) that the MFA device provides. If the MFA authentication succeeds, the command performs the requested operation. The OTP is not displayed on the screen.

```shell
$ aws iam list-users --profile role-with-mfa
Enter MFA code for arn:aws:iam::123456789012:mfa/cli-user:
{
  "Users": [
    ...
```

Cross-account roles and external ID

You can enable IAM users to use roles that belong to different accounts by configuring the role as a cross-account role. During role creation, set the role type to Another AWS account, as described in Creating a Role to Delegate Permissions to an IAM user. Optionally, select Require MFA. Require MFA configures the appropriate condition in the trust relationship, as described in Using multi-factor authentication (p. 64).

If you use an external ID to provide additional control over who can use a role across accounts, you must also add the `external_id` parameter to the role profile. You typically use this only when the other account is controlled by someone outside your company or organization.

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

Specifying a role session name for easier auditing

When many individuals share a role, auditing becomes more of a challenge. You want to associate each operation invoked with the individual who invoked the action. However, when the individual uses a role, the assumption of the role by the individual is a separate action from the invoking of an operation, and you must manually correlate the two.

You can simplify this by specifying unique role session names when users assume a role. You do this by adding a `role_session_name` parameter to each named profile in the `config` file that specifies a role. The `role_session_name` value is passed to the AssumeRole operation and becomes part of the ARN for the role session. It is also included in the AWS CloudTrail logs for all logged operations.
For example, you could create a role-based profile as follows.

```
[profile namedsessionrole]
role_arn = arn:aws:iam::234567890123:role/SomeRole
source_profile = default
role_session_name = Session_Maria_Garcia
```

This results in the role session having the following ARN.

```
arn:aws:iam::234567890123:assumed-role/SomeRole/Session_Maria_Garcia
```

Also, all AWS CloudTrail logs include the role session name in the information captured for each operation.

### Assume role with web identity

You can configure a profile to indicate that the AWS CLI should assume a role using web identity federation and Open ID Connect (OIDC). When you specify this in a profile, the AWS CLI automatically makes the corresponding AWS STS `AssumeRoleWithWebIdentity` call for you.

**Note**

When you specify a profile that uses an IAM role, the AWS CLI makes the appropriate calls to retrieve temporary credentials. These credentials are stored in `~/.aws/cli/cache`. Subsequent AWS CLI commands that specify the same profile use the cached temporary credentials until they expire. At that point, the AWS CLI automatically refreshes the credentials.

To retrieve and use temporary credentials using web identity federation, you can specify the following configuration values in a shared profile.

- **role_arn (p. 61)**
  
  Specifies the ARN of the role to assume.

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

- **role_session_name (p. 65)**
  
  Specifies an optional name applied to this assume-role session.

The following is an example of the minimal amount of configuration needed to configure an assume role with web identity profile.

```
# In ~/.aws/config
[profile web-identity]
role_arn=arn:aws:iam::123456789012:role/RoleNameToAssume
web_identity_token_file=/path/to/a/token
```

You can also provide this configuration by using environment variables (p. 43).

**AWS_ROLE_ARN**

  The ARN of the role to assume.
Clearing cached credentials

When you use a role, the AWS CLI caches the temporary credentials locally until they expire. The next time you try to use them, the AWS CLI attempts to renew them on your behalf.

If your role’s temporary credentials are revoked, they are not renewed automatically, and attempts to use them fail. However, you can delete the cache to force the AWS CLI to retrieve new credentials.

**Linux or macOS**

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

**Windows**

```cmd
C:\> del /s /q %UserProfile%\.aws\cli\cache
```
Using the AWS CLI

This section introduces you to many of the common features and options available in the AWS Command Line Interface (AWS CLI). For a list of commands, see the AWS CLI reference guide.

Note
By default, the AWS CLI sends requests to AWS services by using HTTPS on TCP port 443. To use the AWS CLI successfully, you must be able to make outbound connections on TCP port 443.

Topics in this guide
• Getting help with the AWS CLI (p. 68)
• Command structure in the AWS CLI (p. 72)
• Specifying parameter values for the AWS CLI (p. 74)
• Controlling command output from the AWS CLI (p. 88)
• Understanding return codes from the AWS CLI (p. 111)
• Creating and using AWS CLI aliases (p. 112)

Getting help with the AWS CLI

This topic describes how to access help content for the AWS Command Line Interface (AWS CLI).

Topics
• The built-in AWS CLI help command (p. 68)
• AWS CLI reference guide (p. 71)
• API documentation (p. 72)
• Troubleshooting errors (p. 72)
• Additional help (p. 72)

The built-in AWS CLI help command

You can get help with any command when using the AWS Command Line Interface (AWS CLI). To do so, simply type `help` at the end of a command name.

For example, the following command displays help for the general AWS CLI options and the available top-level commands.

```bash
$ aws help
```

The following command displays the available Amazon Elastic Compute Cloud (Amazon EC2) specific commands.

```bash
$ aws ec2 help
```

The following example displays detailed help for the Amazon EC2 DescribeInstances operation. The help includes descriptions of its input parameters, available filters, and what is included as output. It also includes examples showing how to type common variations of the command.

```bash
$ aws ec2 describe-instances help
```
The help for each command is divided into six sections:

Name

The name of the command.

NAME

describe-instances -

Description

A description of the API operation that the command invokes.

DESCRIPTION

Describes one or more of your instances.

If you specify one or more instance IDs, Amazon EC2 returns information for those instances. If you do not specify instance IDs, Amazon EC2 returns information for all relevant instances. If you specify an instance ID that is not valid, an error is returned. If you specify an instance that you do not own, it is not included in the returned results.

Synopsis

The basic syntax for using the command and its options. If an option is shown in square brackets, it's optional, has a default value, or has an alternative option that you can use.

SYNOPSIS

describe-instances

[--dry-run | --no-dry-run]

[--instance-ids <value>]

[--filters <value>]

[--cli-input-json <value>]

[--starting-token <value>]

[--page-size <value>]

[--max-items <value>]

[--generate-cli-skeleton]

For example, describe-instances has a default behavior that describes all instances in the current account and AWS Region. You can optionally specify a list of instance-ids to describe one or more instances; dry-run is an optional Boolean flag that doesn't take a value. To use a Boolean flag, specify either shown value, in this case --dry-run or --no-dry-run. Likewise, --generate-cli-skeleton doesn't take a value. If there are conditions on an option's use, they are described in the OPTIONS section, or shown in the examples.

Options

A description of each of the options shown in the synopsis.

OPTIONS

--dry-run | --no-dry-run (boolean)

Checks whether you have the required permissions for the action, without actually making the request, and provides an error response. If you have the required permissions, the error response is DryRun-Operation. Otherwise, it is UnauthorizedOperation.

--instance-ids (list)

One or more instance IDs.
Default: Describes all your instances.

Examples

Examples showing the usage of the command and its options. If no example is available for a command or use case that you need, request one using the feedback link on this page, or in the AWS CLI command reference on the help page for the command.

EXAMPLES

To describe an Amazon EC2 instance

Command:

```bash
aws ec2 describe-instances --instance-ids i-5203422c
```

To describe all instances with the instance type m1.small

Command:

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

To describe all instances with an Owner tag

Command:

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

Output

Descriptions of each of the fields and data types included in the response from AWS.

For describe-instances, the output is a list of reservation objects, each of which contains several fields and objects that contain information about the instances associated with it. This information comes from the API documentation for the reservation data type used by Amazon EC2.

```bash
OUTPUT

Reservations -> (list)
  One or more reservations.

(structure)
  Describes a reservation.

  ReservationId -> (string)
  The ID of the reservation.

  OwnerId -> (string)
  The ID of the AWS account that owns the reservation.

  RequesterId -> (string)
  The ID of the requester that launched the instances on your behalf (for example, AWS Management Console or Auto Scaling).

  Groups -> (list)
  One or more security groups.

  (structure)
    Describes a security group.

    GroupName -> (string)
    The name of the security group.
```
GroupId -> (string)
The ID of the security group.

Instances -> (list)
One or more instances.

(structure)
Describes an instance.

InstanceId -> (string)
The ID of the instance.

ImageId -> (string)
The ID of the AMI used to launch the instance.

State -> (structure)
The current state of the instance.

Code -> (integer)
The low byte represents the state. The high byte is an opaque internal value and should be ignored.

When the AWS CLI renders the output into JSON, it becomes an array of reservation objects, similar to the following example.

```json
{
  "Reservations": [
    {
      "OwnerId": "012345678901",
      "ReservationId": "r-4c58f8a0",
      "Groups": [],
      "RequesterId": "012345678901",
      "Instances": [
        {
          "Monitoring": {
            "State": "disabled"
          },
          "PublicDnsName": "ec2-52-74-16-12.us-west-2.compute.amazonaws.com",
          "State": {
            "Code": 16,
            "Name": "running"
          }
        }
      ]
    }
  ]
}
```

Each reservation object contains fields describing the reservation and an array of instance objects, each with its own fields (for example, PublicDnsName) and objects (for example, State) that describe it.

Windows users
You can pipe (|) the output of the help command to the more command to view the help file one page at a time. Press the space bar or PgDn to view more of the document, and q to quit.

```
C:\> aws ec2 describe-instances help | more
```

AWS CLI reference guide

The help files contain links that cannot be viewed or navigated to from the command line. You can view and interact with these links by using the online AWS CLI version 1 reference guide. The reference also
contains the help content for all AWS CLI commands. The descriptions are presented for easy navigation and viewing on mobile, tablet, or desktop screens.

**API documentation**

All commands in the AWS CLI correspond to requests made to an AWS service's public API. Each service with a public API has an API reference that can be found on the service's home page on the AWS Documentation website. The content for an API reference varies based on how the API is constructed and which protocol is used. Typically, an API reference contains detailed information about the operations supported by the API, the data sent to and from the service, and any error conditions that the service can report.

**API Documentation Sections**

- **Actions** – Detailed information on each operation and its parameters (including constraints on length or content, and default values). It lists the errors that can occur for this operation. Each operation corresponds to a subcommand in the AWS CLI.
- **Data Types** – Detailed information about structures that a command might require as a parameter, or return in response to a request.
- **Common Parameters** – Detailed information about the parameters that are shared by all of action for the service.
- **Common Errors** – Detailed information about errors that can be returned by any of the service's operations.

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

**Service-specific CLIs**

Some services have a separate CLI that dates from before a single AWS CLI was created to work with all services. These service-specific CLIs have separate documentation that is linked from the service's documentation page. Documentation for service-specific CLIs do not apply to the AWS CLI.

**Troubleshooting errors**

For help diagnosing and fixing AWS CLI errors, see *Troubleshooting errors (p. 167).*

**Additional help**

For additional help with your AWS CLI issues, visit the AWS CLI community on GitHub.

**Command structure in the AWS CLI**

This topic covers how AWS Command Line Interface (AWS CLI) command is structured, and how to use wait commands.

**Topics**

- Command structure (p. 72)
- Wait commands (p. 73)

**Command structure**

The AWS CLI uses a multipart structure on the command line that must be specified in this order:
1. The base call to the `aws` program.
2. The top-level `command`, which typically corresponds to an AWS service supported by the AWS CLI.
3. The `subcommand` that specifies which operation to perform.
4. General AWS CLI options or parameters required by the operation. You can specify these in any order as long as they follow the first three parts. If an exclusive parameter is specified multiple times, only the last value applies.

```
$ aws <command> <subcommand> [options and parameters]
```

Parameters can take various types of input values, such as numbers, strings, lists, maps, and JSON structures. What is supported is dependent upon the command and subcommand you specify.

**Examples**

**Amazon S3**

The following example lists all of your Amazon S3 buckets.

```
$ aws s3 ls
2018-12-11 17:08:50 my-bucket
2018-12-14 14:55:44 my-bucket2
```

For more information on the Amazon S3 commands, see `aws s3` in the [AWS CLI Command Reference](https://docs.aws.amazon.com/cli/latest/reference/s3/).

**AWS CloudFormation**

The following `create-change-set` command example changes the cloudformation stack name to `my-change-set`.

```
$ aws cloudformation create-change-set --stack-name my-stack --change-set-name my-change-set
```

For more information on the AWS CloudFormation commands, see `aws cloudformation` in the [AWS CLI Command Reference](https://docs.aws.amazon.com/cli/latest/reference/cloudformation/).

**Wait commands**

Some AWS services have `wait` commands available. Any command that uses `aws wait` usually waits until a command is complete before it moves on to the next step. This is especially useful for multipart commands or scripting, as you can use a wait command to prevent moving to subsequent steps if the wait command fails.

The AWS CLI uses a multipart structure on the command line for the `wait` command that must be specified in this order:

1. The base call to the `aws` program.
2. The top-level `command`, which typically corresponds to an AWS service supported by the AWS CLI.
3. The `wait` command.
4. The `subcommand` that specifies which operation to perform.
5. General CLI options or parameters required by the operation. You can specify these in any order as long as they follow the first three parts. If an exclusive parameter is specified multiple times, only the last value applies.
Parameters can take various types of input values, such as numbers, strings, lists, maps, and JSON structures. What is supported is dependent upon the command and subcommand you specify.

**Note**
Not every AWS service supports `wait` commands. See the AWS CLI reference guide to see if your service supports `wait` commands.

**Examples**

**AWS CloudFormation**

The following `wait change-set-create-complete` command examples pauses and resumes only after it can confirm that the `my-change-set` change set in the `my-stack` stack is ready to run.

```bash
$ aws cloudformation wait change-set-create-complete --stack-name my-stack --change-set-name my-change-set
```

For more information on the AWS CloudFormation `wait` commands, see `wait` in the AWS CLI Command Reference.

**AWS CodeDeploy**

The following `wait deployment-successful` command examples pauses until the `d-A1B2C3111` deployment completes successfully.

```bash
$ aws deploy wait deployment-successful --deployment-id d-A1B2C3111
```

For more information on the AWS CodeDeploy `wait` commands, see `wait` in the AWS CLI Command Reference.

**Specifying parameter values for the AWS CLI**

Many parameters used in the AWS Command Line Interface (AWS CLI) are simple string or numeric values, such as the key-pair name `my-key-pair` in the following example.

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

You can surround strings that do not contain any space characters with quotation marks or not. However, you must use quotation marks around strings that include one or more space characters. For more information about using quotation marks around complex parameters, see Using quotation marks with strings in the AWS CLI (p. 78).

**Parameter topics**

- Common AWS CLI parameter types (p. 75)
- Using quotation marks with strings in the AWS CLI (p. 78)
- Loading AWS CLI parameters from a file (p. 80)
- AWS CLI skeletons and input files (p. 83)
- Using shorthand syntax with the AWS CLI (p. 87)
Common AWS CLI parameter types

This section describes some of the common parameter types and the typical required format.

If you are having trouble formatting a parameter for a specific command, check the help by entering `help` after the command name. The help for each subcommand includes an option's name and description. The option's parameter type is listed in parentheses. For more information on viewing help, see the section called "Getting Help" (p. 68).

Parameter types include:
- String (p. 75)
- Timestamp (p. 75)
- List (p. 75)
- Boolean (p. 76)
- Integer (p. 76)
- Binary / blob (binary large object) and streaming blob (p. 76)
- Map (p. 76)
- Document (p. 77)

String

String parameters can contain alphanumeric characters, symbols, and white spaces from the ASCII character set. Strings that contain white spaces must be surrounded by quotation marks. We recommend that you don't use symbols or white spaces other than the standard space character and to observe your terminal's quoting rules (p. 78) to prevent unexpected results.

Some string parameters can accept binary data from a file. See Binary files (p. 82) for an example.

Timestamp

Timestamps are formatted according to the ISO 8601 standard. These are often referred to as "DateTime" or "Date" parameters.

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

Acceptable formats include:
- YYYY-MM-DDThh:mm:ss.sssTZD (UTC), for example, 2014-10-01T20:30:00.000Z
- YYYY-MM-DDThh:mm:ss.sssTZD (with offset), for example, 2014-10-01T12:30:00.000-08:00
- YYYY-MM-DD, for example, 2014-10-01
- Unix time in seconds, for example, 1412195400. This is sometimes referred to as Unix Epoch time and represents the number of seconds since midnight, January 1, 1970 UTC.

You can set the timestamp format by using the `cli_timestamp_format` (p. 34) file setting.

List

One or more strings separated by spaces. If any of the string items contain a space, you must put quotation marks around that item. Observe your terminal's quoting rules (p. 78) to prevent unexpected results.

$ aws ec2 describe-spot-price-history --instance-types m1.xlarge m1.medium
Boolean

Binary flag that turns an option on or off. For example, `ec2 describe-spot-price-history` has a Boolean `--dry-run` parameter that, when specified, validates the query with the service without actually running the query.

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

The output indicates whether the command was well formed. This command also includes a `--no-dry-run` version of the parameter that you can use to explicitly indicate that the command should be run normally. Including it isn't necessary because this is the default behavior.

Integer

An unsigned, whole number.

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

Binary / blob (binary large object) and streaming blob

In the AWS CLI version 1, to pass a value to a parameter with type blob, you must specify a path to a local file that contains the binary data. The path should not contain any protocol identifier, such as `http://` or `file://`. The specified path is interpreted as being relative to the current working directory. For example, the `--body` parameter for `aws s3api put-object` is a blob.

```bash
$ aws s3api put-object --bucket my-bucket --key testimage.png --body /tmp/image.png
```

**Note**

Streaming blobs such as `aws cloudsearchdomain upload-documents` are not prefixed by `file://` or `fileb://`. Instead, streaming blob parameters are formatted using the direct file path. The following example uses the direct file path `document-batch.json` for the `aws cloudsearchdomain upload-documents` command:

```bash
$ aws cloudsearchdomain upload-documents \   --endpoint-url https://doc-my-domain.us-west-1.cloudsearch.amazonaws.com \   --content-type application/json \   --documents document-batch.json
```

Map

A set of key-value pairs specified in JSON or by using the CLI's shorthand syntax (p. 87). The following JSON example reads an item from an Amazon DynamoDB table named `my-table` with a map parameter, `--key`. The parameter specifies the primary key named `id` with a number value of 1 in a nested JSON structure.

For more advanced JSON usage in a command line, consider using a command line JSON processor, like `jq`, to create JSON strings. For more information on `jq`, see the `jq` repository on GitHub.

```bash
$ aws dynamodb get-item --table-name my-table --key '{"id": {"N":"1"}}'
{   "Item": {   "name": {   "S": "John"   },
```
Note

Shorthand syntax (p. 87) is not compatible with document types.

Document types are used to send data without needing to embed JSON inside strings. The document type enables services to provide arbitrary schemas for you to use more flexible data types.

This allows for sending JSON data without needing to escape values. For example, instead of using the following escaped JSON input:

```
{"document": "{\"key\":true}\"}
```

You can use the following document type:

```
{"document": {"key": true}}
```

Valid values for document types

Due to the flexible nature of document types, there are multiple valid value types. Valid values include the following:

**String**

```
--option '"value"'
```

**Number**

```
--option 123
--option 123.456
```

**Boolean**

```
--option true
```

**Null**

```
--option null
```

**Array**

```
--option ['value1', 'value2', 'value3']
--option ['value', 1, true, null, ['key1', 2.34], {"key2": "value2"}]
```

**Object**

```
--option '{"key": "value"}'
--option '{"key1": "value1", "key2": 123, "key3": true, "key4": null, "key5": ["value3", "value4"], "key6": {"value5": "value6"}}'
```
Using quotation marks with strings in the AWS CLI

There are primarily two ways single and double quotes are used in the AWS CLI.

- Using quotation marks around strings that contain white spaces (p. 78)
- Using quotation marks inside strings (p. 78)

Using quotation marks around strings that contain white spaces

Parameter names and their values are separated by spaces on the command line. If a string value contains an embedded space, then you must surround the entire string with quotation marks to prevent the AWS CLI from misinterpreting the space as a divider between the value and the next parameter name. Which type of quotation mark you use depends on the operating system you are running the AWS CLI on.

Linux and macOS

Use single quotation marks ' '

```bash
# aws ec2 create-key-pair --key-name 'my key pair'
```

For more information on using quotes, see the user documentation for your preferred shell.

PowerShell

**Single quotations (recommended)**

Single quotation marks ' ' are called verbatim strings. The string is passed to the command exactly as you type it, which means PowerShell variables will not pass through.

```powershell
PS C:\> aws ec2 create-key-pair --key-name 'my key pair'
```

**Double quotations**

Double quotation marks " " are called expandable string. Variables can be passed in expandable strings.

```powershell
PS C:\> aws ec2 create-key-pair --key-name "my key pair"
```

For more information on using quotes, see About Quoting Rules in the *Microsoft PowerShell Docs.*

Windows command prompt

Use double quotation marks " ".

```cmd
C:\> aws ec2 create-key-pair --key-name "my key pair"
```

Optionally, you can separate the parameter name from the value with an equals sign = instead of a space. This is typically necessary only if the value of the parameter starts with a hyphen.

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

Using quotation marks inside strings

Strings might contain quotation marks, and your shell might require escaping quotations for them to work properly. One of the common parameter value types is a JSON string. This is complex since it
includes spaces and double quotation marks " " around each element name and value in the JSON structure. The way you enter JSON-formatted parameters on the command line differs depending on your operating system.

For more advanced JSON usage in the command line, consider using a command line JSON processor, like jq, to create JSON strings. For more information on j q, see the j q repository on GitHub.

**Linux and macOS**

For Linux and macOS to interpret strings literally use single quotation marks ' ' to enclose the JSON data structure, as in the following example. You do not need to escape double quotation marks embedded in the JSON string, as they are being treated literally. Since the JSON is enclosed in single quotation marks, any single quotation marks in the string will need to be escaped, this is usually accomplished using a backslash before the single quote \'.

```bash
$ aws ec2 run-instances \
   --image-id ami-12345678 \
   --block-device-mappings '[["DeviceName":"/dev/sdb","Ebs":
   {"VolumeSize":20,"DeleteOnTermination":false,"VolumeType":"standard"}]]'
```

For more information on using quotes, see the user documentation for your preferred shell.

**PowerShell**

Use single quotation marks ’ ’ or double quotation marks " ".

**Single quotations (recommended)**

Single quotation marks ’ ’ are called verbatim strings. The string is passed to the command exactly as you type it, which means PowerShell variables will not pass through.

Since JSON data structures include double quotes, we suggest single quotation marks ’ ’ to enclose it. If you use single quotation marks, you do not need to escape double quotation marks embedded in the JSON string. However, you need to escape each single quotation mark with a backtick ` within the JSON structure.

```powershell
PS C:\> aws ec2 run-instances \
   --image-id ami-12345678 \
   --block-device-mappings '[["DeviceName":"/dev/sdb","Ebs":
   {"VolumeSize":20,"DeleteOnTermination":false,"VolumeType":"standard"}]]'
```

**Double quotations**

Double quotation marks " " are called expandable strings. Variables can be passed in expandable strings.

If you use double quotation marks, you do not need to escape single quotation marks embedded in the JSON string. However, you need to escape each double quotation mark with a backtick ` within the JSON structure, as with the following example.

```powershell
PS C:\> aws ec2 run-instances \
   --image-id ami-12345678 \
   --block-device-mappings "[["DeviceName":"/dev/sdb","Ebs":
   {"VolumeSize":20,"DeleteOnTermination":false,"VolumeType":"standard"}]]"
```

For more information on using quotes, see About Quoting Rules in the Microsoft PowerShell Docs.

**Warning**

Before PowerShell sends a command to the AWS CLI, it determines if your command is interpreted using typical PowerShell or CommandLineToArgvW quoting rules. When
PowerShell processes using `CommandLineToArgvW`, you must escape characters with a backslash `\`.

For more information on `CommandLineToArgvW` in PowerShell, see What's up with the strange treatment of quotation marks and backslashes by `CommandLineToArgvW` in the Microsoft DevBlogs, Everyone quotes command line arguments the wrong way in the Microsoft Docs Blog, and `CommandLineToArgvW` function in the Microsoft Docs.

**Single quotations**

Single quotation marks `' ' are called verbatim strings. The string is passed to the command exactly as you type it, which means PowerShell variables will not pass through. Escape characters with a backslash `\`.

```
PS C:\> aws ec2 run-instances `
  --image-id ami-12345678 `
  --block-device-mappings '[[{"DeviceName": "/dev/sdb"}, {"Ebs": {
  "VolumeSize": 20, "DeleteOnTermination": false, "VolumeType": "standard"}}]'
```

**Double quotations**

Double quotation marks `" "` are called expandable strings. Variables can be passed in expandable strings. For double quoted strings you have to escape twice using `\` for each quote instead of only using a backtick. The backtick escapes the backslash, and then the backslash is used as an escape character for the `CommandLineToArgvW` process.

```
PS C:\> aws ec2 run-instances `
  --image-id ami-12345678 `
  --block-device-mappings "[[{"DeviceName": "/dev/sdb"}, {"Ebs": {
  "VolumeSize": 20, "DeleteOnTermination": false, "VolumeType": "standard"}}]"
```

**Blobs (recommended)**

To bypass PowerShell quoting rules for JSON data input, use Blobs to pass your JSON data directly to the AWS CLI. For more information on Blobs, see Binary / blob (binary large object) and streaming blob (p. 76).

Windows command prompt

The Windows command prompt requires double quotation marks `" "` to enclose the JSON data structure. Also, to prevent the command processor from misinterpreting the double quotation marks embedded in the JSON, you must also escape (precede with a backslash `\`) each double quotation mark `"` within the JSON data structure itself, as in the following example.

```
C:\> aws ec2 run-instances ^
  --image-id ami-12345678 ^
  --block-device-mappings "[[{"DeviceName": "/dev/sdb"}, {"Ebs": {
  "VolumeSize": 20, "DeleteOnTermination": false, "VolumeType": "standard"}}]"
```

Only the outermost double quotation marks are not escaped.

**Loading AWS CLI parameters from a file**

Some parameters expect file names as arguments, from which the AWS CLI loads the data. Other parameters enable you to specify the parameter value as either text typed on the command line or read from a file. Whether a file is required or optional, you must encode the file correctly so that the AWS CLI can understand it. The file's encoding must match the reading system's default locale. You can determine this by using the Python `locale.getpreferredencoding()` method.

**Note**

By default, Windows PowerShell outputs text as UTF-16, which conflicts with the UTF-8 encoding used by JSON files and many Linux systems. We recommend that you use `-Encoding`
ascii with your PowerShell Out-File commands to ensure the AWS CLI can read the resulting file.

Topics
- How to load parameters from a file (p. 81)
- Binary files (p. 82)
- Remote files (p. 82)

How to load parameters from a file

Sometimes it's convenient to load a parameter value from a file instead of trying to type it all as a command line parameter value, such as when the parameter is a complex JSON string. To specify a file that contains the value, specify a file URL in the following format.

```
file:///complete/path/to/file
```

- The first two slash `/' characters are part of the specification. If the required path begins with a `/', the result is three slash characters: `file:///folder/file'.
- The URL provides the path to the file that contains the actual parameter content.
- When using files with spaces or special characters, following the quoting and escaping rules (p. 78) for your terminal.

**Note**
This behavior is disabled automatically for parameters that already expect a URL, such as parameter that identifies a AWS CloudFormation template URL. You can also disable this behavior by disabling the `cli_follow_urlparam` (p. 34) setting in your AWS CLI configuration file.

The file paths in the following examples are interpreted to be relative to the current working directory.

Linux or macOS

```
// Read from a file in the current directory
$ aws ec2 describe-instances --filters file://filter.json

// Read from a file in /tmp
$ aws ec2 describe-instances --filters file:///tmp/filter.json

// Read from a file with a filename with whitespaces
$ aws ec2 describe-instances --filters "file://filter content.json"
```

Windows command prompt

```
// Read from a file in C:\temp
C:\> aws ec2 describe-instances --filters file:///C:\temp/filter.json

// Read from a file with a filename with whitespaces
C:\> aws ec2 describe-instances --filters "file:///C:\temp/filter content.json"
```

The `file://` prefix option supports Unix-style expansions, including "~/", "./", and ".../". On Windows, the "-/" expression expands to your user directory, stored in the `%USERPROFILE%` environment variable. For example, on Windows 10 you would typically have a user directory under `C:\Users\UserName`.

You must still escape JSON documents that are embedded as the value of another JSON document.
$ aws sqs create-queue --queue-name my-queue --attributes file://attributes.json

attributes.json

}

Binary files

For commands that take binary data as a parameter, specify that the data is binary content by using the fileb:// prefix. Commands that accept binary data include:

- **aws ec2 run-instances**: --user-data parameter.
- **aws s3api put-object**: --sse-customer-key parameter.
- **aws kms decrypt**: --ciphertext-blob parameter.

The following example generates a binary 256-bit AES key using a Linux command line tool, and then provides it to Amazon S3 to encrypt an uploaded file server-side.

$ dd if=/dev/urandom bs=1 count=32 > sse.key
32+0 records in
32+0 records out
32 bytes (32 B) copied, 0.000164441 s, 195 kB/s

$ aws s3api put-object \
   --bucket my-bucket \
   --key test.txt \
   --body test.txt \
   --sse-customer-key fileb://sse.key \
   --sse-customer-algorithm AES256
{
   "SSECustomerKeyMD5": "iVg8oWa8sy714+FjtesrJg==",
   "SSECustomerAlgorithm": "AES256",
   "ETag": "a6118e84b76cf98bf04bbee14b6045c6c"
}

Remote files

The AWS CLI also supports loading parameters from a file hosted on the internet with an http:// or https:// URL. The following example references a file stored in an Amazon S3 bucket. This allows you to access parameter files from any computer, but it does require that the container is publicly accessible.

$ aws ec2 run-instances \
   --image-id ami-12345678 \
   --block-device-mappings http://my-bucket.s3.amazonaws.com/filename.json

The preceding example assumes that the file filename.json contains the following JSON data.

[  
   {  
      "DeviceName": "/dev/sdb",
      "Ebs": {  
         "VolumeSize": 20,
         "DeleteOnTermination": false,
   
82
"VolumeType": "standard"
}
}
]

For another example referencing a file containing JSON-formatted parameters, see Attaching an IAM managed policy to an IAM user (p. 140).

AWS CLI skeletons and input files

Most of the AWS Command Line Interface (AWS CLI) commands accept all parameter inputs from a file. These templates can be generated using the `generate-cli-skeleton` option.

Topics

- About AWS CLI skeletons and input files (p. 83)
- Generating a command skeleton (p. 85)

About AWS CLI skeletons and input files

Most of the AWS Command Line Interface (AWS CLI) commands support the ability to accept all parameter inputs from a file using the `--cli-input-json` parameters.

Those same commands helpfully provide the `--generate-cli-skeleton` parameter to generate a file in JSON format with all of the parameters that you can edit and fill in. Then you can run the command with the relevant `--cli-input-json` parameter and point to the filled-in file.

Important

Several AWS CLI commands don't map directly to individual AWS API operations, such as the `aws s3` commands. Such commands don't support either the `--generate-cli-skeleton` or `--cli-input-json` parameters described in this topic. If you don't know whether a specific command supports these parameters, run the following command, replacing the `service` and `command` names with the ones you're interested in.

```bash
$ aws service command help
```

The output includes a Synopsis section that shows the parameters that the specified command supports.

```bash
$ aws iam list-users help
...
SYNOPSIS
  list-users
    ...
    [--cli-input-json]
    ...
    [--generate-cli-skeleton <value>]
    ...
```

The `--generate-cli-skeleton` parameter causes the command not to run, but instead to generate and display a parameter template that you can customize and use as input on a later command. The generated template includes all of the parameters that the command supports.

The `--generate-cli-skeleton` parameter accepts one of the following values:

- `input` – The generated template includes all input parameters formatted as JSON. This is the default value.
output – The generated template includes all output parameters formatted as JSON.

Because the AWS CLI is essentially a "wrapper" around the service's API, the skeleton file expects you to reference all parameters by their underlying API parameter names. This is likely different from the AWS CLI parameter name. For example, an AWS CLI parameter named `user-name` might map to the AWS service's API parameter named `UserName` (notice the altered capitalization and missing dash). We recommend that you use the `--generate-cli-skeleton` option to generate the template with the "correct" parameter names to avoid errors. You can also reference the API Reference Guide for the service to see the expected parameter names. You can delete any parameters from the template that are not required and for which you don't want to supply a value.

For example, if you run the following command, it generates the parameter template for the Amazon Elastic Compute Cloud (Amazon EC2) command `run-instances`.

```
$ aws ec2 run-instances --generate-cli-skeleton
```

**JSON**

The following example shows how to generate a template formatted in JSON by using the default value (input) for the `--generate-cli-skeleton` parameter.

```
$ 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": "",
```
Generating a command skeleton

To generate and use a parameter skeleton file

1. Run the command with the `--generate-cli-skeleton` parameter to produce JSON and direct the output to a file to save it.

   JSON

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

2. Open the parameter skeleton file in your text editor and remove any of the parameters that you don't need. For example, you might strip the template down to the following. Be sure that the file is still valid JSON after you remove the elements you don't need.

   JSON

   ```json
   {   "DryRun": true,
   "ImageId": "",
   "KeyName": "",
   "SecurityGroups": [   "",
   ],
   "InstanceType": "",
   "Monitoring": {   "Enabled": true
   }`
In this example, we leave the `DryRun` parameter set to `true` to use the Amazon EC2 dry run feature. This feature lets you safely test the command without actually creating or modifying any resources.

3. Fill in the remaining values with values appropriate for your scenario. In this example, we provide the instance type, key name, security group, and identifier of the Amazon Machine Image (AMI) to use. This example assumes the default AWS Region. The AMI `ami-dfc39aef` is a 64-bit Amazon Linux image hosted in the `us-west-2` Region. If you use a different Region, you must find the correct AMI ID to use.

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

4. Run the command with the completed parameters by passing the completed template file to the `--cli-input-json` parameter by using the `file://` prefix. The AWS CLI interprets the path to be relative to your current working directory, so in the following example that displays only the file name with no path, it looks for the file directly in the current working directory.

```bash
$ aws ec2 run-instances --cli-input-json file://ec2runinst.json
```

The dry run error indicates that the JSON is formed correctly and that the parameter values are valid. If other issues are reported in the output, fix them and repeat the previous step until the "Request would have succeeded" message is displayed.

5. Now you can set the `DryRun` parameter to `false` to disable dry run.

```json
{
  "DryRun": false,
  "ImageId": "ami-dfc39aef",
  "KeyName": "mykey",
  "SecurityGroups": [
    "my-sg"
  ],
  "InstanceType": "t2.micro",
  "Monitoring": {
    "Enabled": true
  }
}
```
6. Run the command, and `run-instances` actually launches an Amazon EC2 instance and displays the details generated by the successful launch. The format of the output is controlled by the `--output` parameter, separately from the format of your input parameter template.

**JSON**

```
$ aws ec2 run-instances --cli-input-json file://ec2runinst.json --output json
```

```
{
  "OwnerId": "123456789012",
  "ReservationId": "r-d94a2b1",
  "Groups": [],
  "Instances": [
  ...
```

## Using shorthand syntax with the AWS CLI

The AWS Command Line Interface (AWS CLI) can accept many of its option parameters in JSON format. However, it can be tedious to enter large JSON lists or structures on the command line. To make this easier, the AWS CLI also supports a shorthand syntax that enables a simpler representation of your option parameters than using the full JSON format.

**Topics**
- Structure parameters (p. 87)
- Using shorthand syntax with the AWS Command Line Interface (p. 88)

### Structure parameters

The shorthand syntax in the AWS CLI makes it easier for users to input parameters that are flat (non-nested structures). The format is a comma-separated list of key-value pairs. Be sure to use the quoting (p. 78) and escaping rules appropriate for your terminal as shorthand syntax are strings.

**Linux or macOS**

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

**PowerShell**

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

These are both equivalent to the following example, formatted in JSON.

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

There must be no white space between each comma-separated key-value pair. Here is an example of the Amazon DynamoDB `update-table` command with the `--provisioned-throughput` option specified in shorthand.

```
$ aws dynamodb update-table \
   --provisioned-throughput ReadCapacityUnits=15,WriteCapacityUnits=10 \
   --table-name MyDDBTable
```
This is equivalent to the following example formatted in JSON.

```
$ aws dynamodb update-table \
   --provisioned-throughput '{"ReadCapacityUnits":15,"WriteCapacityUnits":10}' \
   --table-name MyDDBTable
```

**Using shorthand syntax with the AWS Command Line Interface**

You can specify input parameters in a list form in two ways: JSON or shorthand. The AWS CLI shorthand syntax is designed to make it easier to pass in lists with number, string, or non-nested structures.

The basic format is shown here, where values in the list are separated by a single space.

```
--option value1 value2 value3
```

This is equivalent to the following example, formatted in JSON.

```
--option '[value1,value2,value3]' 
```

As previously mentioned, you can specify a list of numbers, a list of strings, or a list of non-nested structures in shorthand. The following is an example of the `stop-instances` command for Amazon Elastic Compute Cloud (Amazon EC2), where the input parameter (list of strings) for the `--instance-ids` option is specified in shorthand.

```
$ aws ec2 stop-instances \
   --instance-ids i-1486157a i-1286157c i-ec3a7e87
```

This is equivalent to the following example formatted in JSON.

```
$ aws ec2 stop-instances \
   --instance-ids '["i-1486157a","i-1286157c","i-ec3a7e87"]'
```

The following example shows the Amazon EC2 `create-tags` command, which takes a list of non-nested structures for the `--tags` option. The `--resources` option specifies the ID of the instance to tag.

```
$ aws ec2 create-tags \
   --resources i-1286157c \
   --tags Key=My1stTag,Value=Value1 Key=My2ndTag,Value=Value2 Key=My3rdTag,Value=Value3
```

This is equivalent to the following example, formatted in JSON. The JSON parameter is written over multiple lines for readability.

```
$ aws ec2 create-tags \
   --resources i-1286157c \
   --tags \
   [\n     {"Key": "My1stTag", "Value": "Value1"},\n     {"Key": "My2ndTag", "Value": "Value2"},\n     {"Key": "My3rdTag", "Value": "Value3"}\n   ]
```

**Controlling command output from the AWS CLI**

This section describes the different ways to control the output from the AWS Command Line Interface (AWS CLI).
Setting the AWS CLI output format

This topic describes the different output formats for the AWS Command Line Interface (AWS CLI). The AWS CLI supports the following output formats:

- **json** (p. 89) – The output is formatted as a JSON string.
- **text** (p. 90) – The output is formatted as multiple lines of tab-separated string values. This can be useful to pass the output to a text processor, like `grep`, `sed`, or `awk`.
- **table** (p. 92) – The output is formatted as a table using the characters `|`- to form the cell borders. It typically presents the information in a "human-friendly" format that is much easier to read than the others, but not as programmatically useful.

How to select the output format

As explained in the configuration (p. 27) topic, you can specify the output format in three ways:

- **Using the `output` option in a named profile in the `config` file** – The following example sets the default output format to `text`.

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

- **Using the `AWS_DEFAULT_OUTPUT` environment variable** – The following output sets the format to `table` for the commands in this command line session until the variable is changed or the session ends. Using this environment variable overrides any value set in the `config` file.

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

- **Using the `--output` option on the command line** – The following example sets the output of only this one command to `json`. Using this option on the command overrides any currently set environment variable or the value in the `config` file.

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

JSON output format

**JSON** is the default output format of the AWS CLI. Most programming languages can easily decode JSON strings using built-in functions or with publicly available libraries. You can combine JSON output with the `--query` option (p. 95) in powerful ways to filter and format the AWS CLI JSON-formatted output.

For more advanced filtering that you might not be able to do with `--query`, you can consider `jq`, a command line JSON processor. You can download it and find the official tutorial at [http://stedolan.github.io/jq/](http://stedolan.github.io/jq/).

The following is an example of JSON output.

```
$ aws iam list-users --output json
```
Text output format

The text format organizes the AWS CLI output into tab-delimited lines. It works well with traditional Unix text tools such as grep, sed, and awk, and the text processing performed by PowerShell.

The text output format follows the basic structure shown below. The columns are sorted alphabetically by the corresponding key names of the underlying JSON object.

<table>
<thead>
<tr>
<th>IDENTIFIER</th>
<th>sorted-column1</th>
<th>sorted-column2</th>
</tr>
</thead>
<tbody>
<tr>
<td>IDENTIFIER2</td>
<td>sorted-column1</td>
<td>sorted-column2</td>
</tr>
</tbody>
</table>

The following is an example of text output. Each field is tab separated from the others, with an extra tab where there is an empty field.

```
$ aws iam list-users --output text

USERS   arn:aws:iam::123456789012:user/Admin               2014-10-16T16:03:09+00:00
        2016-06-03T18:37:29+00:00   /          AIDA1111111111EXAMPLE   Admin
USERS   arn:aws:iam::123456789012:user/backup/backup-user  2019-09-17T19:30:40+00:00
        /backup/                  AIDA2222222222EXAMPLE   backup-user
USERS   arn:aws:iam::123456789012:user/cli-user             2019-09-17T19:11:39+00:00
        /          AIDA3333333333EXAMPLE   cli-user
```

The fourth column is the PasswordLastUsed field, and is empty for the last two entries because those users never sign in to the AWS Management Console.

**Important**

*We strongly recommend that if you specify text output, you also always use the --query (p. 95) option to ensure consistent behavior.*

This is because the text format alphabetically orders output columns by the key name of the underlying JSON object returned by the AWS service, and similar resources might not have the same key names. For example, the JSON representation of a Linux-based Amazon EC2 instance
Output Format

might have elements that are not present in the JSON representation of a Windows-based instance, or vice versa. Also, resources might have key-value elements added or removed in future updates, altering the column ordering. This is where `--query` augments the functionality of the `text` output to provide you with complete control over the output format.

In the following example, the command specifies which elements to display and defines the ordering of the columns with the list notation `[key1, key2, ...]`. This gives you full confidence that the correct key values are always displayed in the expected column. Finally, notice how the AWS CLI outputs `None` as the value for keys that don’t exist.

```bash
$ aws iam list-users --output text --query 'Users[*].[UserName,Arn,CreateDate,PasswordLastUsed,UserId]'
```

<table>
<thead>
<tr>
<th>Username</th>
<th>ARN</th>
<th>Create Date</th>
<th>Password Last Used</th>
<th>User ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Admin</td>
<td>arn:aws:iam::123456789012:user/Admin</td>
<td>2014-10-16T16:03:09+00:00</td>
<td>2016-06-03T18:37:29+00:00</td>
<td>AIDA11111111EXAMPLE</td>
</tr>
<tr>
<td>backup-user</td>
<td>arn:aws:iam::123456789012:user/backup-user</td>
<td>2019-09-17T19:30:40+00:00</td>
<td>None</td>
<td>AIDA2222222222EXAMPLE</td>
</tr>
<tr>
<td>cli-user</td>
<td>arn:aws:iam::123456789012:user/cli-backup</td>
<td>2019-09-17T19:11:39+00:00</td>
<td>None</td>
<td>AIDA3333333333EXAMPLE</td>
</tr>
</tbody>
</table>

The following example shows how you can use `grep` and `awk` with the `text` output from the `aws ec2 describe-instances` command. The first command displays the Availability Zone, current state, and the instance ID of each instance in `text` output. The second command processes that output to display only the instance IDs of all running instances in the `us-west-2a` Availability Zone.

```bash
$ 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
```

```bash
$ aws ec2 describe-instances --query 'Reservations[*].Instances[*].[Placement.AvailabilityZone, State.Name, InstanceId]' --output text | grep us-west-2a | grep running | awk '{print $3}'
```

```
i-4b41a37c
i-3045b007
i-6fc67758
```

The following example goes a step further and shows not only how to filter the output, but how to use that output to automate changing instance types for each stopped instance.

```bash
$ aws ec2 describe-instances --query 'Reservations[*].Instances[*].[State.Name, InstanceId]' --output text | grep stopped | awk '{print $2}' | while read line; do aws ec2 modify-instance-attribute --instance-id $line --instance-type '{"Value": "m1.medium"}'; done
```

The `text` output can also be useful in PowerShell. Because the columns in `text` output are tab delimited, you can easily split the output into an array by using PowerShell's `\t` delimiter. The following
command displays the value of the third column (InstanceId) if the first column (AvailabilityZone) matches the string us-west-2a.

```powershell
PS C:\> aws ec2 describe-instances --query 'Reservations[*].Instances[*].[
Placement.AvailabilityZone, State.Name, InstanceId]' --output text |
%{if ($_.split("\t")[0] -match "us-west-2a") { $_.split("\t")[2]; } }
```

-4b41a37c
i-a071c394
i-3045b007
i-6fc67758

Notice that although the previous example does show how to use the --query parameter to parse the underlying JSON objects and pull out the desired column, PowerShell has its own ability to handle JSON, if cross-platform compatibility isn’t a concern. Instead of handling the output as text, as most command shells require, PowerShell lets you use the ConvertFrom-JSON cmdlet to produce a hierarchically structured object. You can then directly access the member you want from that object.

```powershell
/aws ec2 describe-instances --output json | ConvertFrom-Json).Reservations.Instances.InstanceId
```

**Tip**

If you output text, and filter the output to a single field using the --query parameter, the output is a single line of tab-separated values. To get each value onto a separate line, you can put the output field in brackets, as shown in the following examples.

Tab separated, single-line output:

```powershell
$ aws iam list-groups-for-user --user-name susan  --output text --query "Groups[].GroupName"
```

<table>
<thead>
<tr>
<th>HRDepartment</th>
<th>Developers</th>
<th>SpreadsheetUsers</th>
<th>LocalAdmins</th>
</tr>
</thead>
<tbody>
<tr>
<td>HRDepartment</td>
<td>Developers</td>
<td>SpreadsheetUsers</td>
<td>LocalAdmins</td>
</tr>
</tbody>
</table>

Each value on its own line by putting [GroupName] in brackets:

```powershell
$ aws iam list-groups-for-user --user-name susan  --output text --query "Groups[j].[GroupName]"
```

<table>
<thead>
<tr>
<th>HRDepartment</th>
<th>Developers</th>
<th>SpreadsheetUsers</th>
<th>LocalAdmins</th>
</tr>
</thead>
<tbody>
<tr>
<td>HRDepartment</td>
<td>Developers</td>
<td>SpreadsheetUsers</td>
<td>LocalAdmins</td>
</tr>
</tbody>
</table>

**Table output format**

The table format produces human-readable representations of complex AWS CLI output in a tabular form.

```powershell
/aws iam list-users --output table
```

```plaintext
<table>
<thead>
<tr>
<th>ListUsers</th>
</tr>
</thead>
</table>
```
### Users

<table>
<thead>
<tr>
<th>Arn</th>
<th>CreateDate</th>
<th>Path</th>
<th>UserId</th>
<th>UserName</th>
</tr>
</thead>
<tbody>
<tr>
<td>arn:aws:iam::123456789012:user/Admin</td>
<td>2014-10-16T16:03:09+00:00</td>
<td>/</td>
<td>AIDA1111111111EXAMPLE</td>
<td>Admin</td>
</tr>
<tr>
<td>arn:aws:iam::123456789012:user/backup/backup-user</td>
<td>2019-09-17T19:30:40+00:00</td>
<td>/backup/</td>
<td>AIDA2222222222EXAMPLE</td>
<td>backup-user</td>
</tr>
<tr>
<td>arn:aws:iam::123456789012:user/cli-user</td>
<td>2019-09-17T19:11:39+00:00</td>
<td>/</td>
<td>AIDA3333333333EXAMPLE</td>
<td>cli-user</td>
</tr>
</tbody>
</table>

You can combine the `--query` option with the `table` format to display a set of elements preselected from the raw output. Notice the output differences between dictionary and list notations: in the first example, column names are ordered alphabetically, and in the second example, unnamed columns are ordered as defined by the user. For more information about the `--query` option, see Filtering AWS CLI output (p. 95).

```bash
$ aws ec2 describe-volumes --query 'Volumes[*].{ID:VolumeId,InstanceId:Attachments[0].InstanceId,AZ:AvailabilityZone,Size:Size}' --output table
```

<table>
<thead>
<tr>
<th>AZ</th>
<th>ID</th>
<th>InstanceId</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>us-west-2a</td>
<td>vol-e11a5288</td>
<td>i-a071c394</td>
<td>30</td>
</tr>
<tr>
<td>us-west-2a</td>
<td>vol-2e410a47</td>
<td>i-4b41a37c</td>
<td>8</td>
</tr>
</tbody>
</table>

```bash
$ aws ec2 describe-volumes --query 'Volumes[*].[VolumeId,Attachments[0].InstanceId,AvailabilityZone,Size]' --output table
```

<table>
<thead>
<tr>
<th>VolumeId</th>
<th>InstanceId</th>
<th>AZ</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>vol-e11a5288</td>
<td>i-a071c394</td>
<td>us-west-2a</td>
<td>30</td>
</tr>
<tr>
<td>vol-2e410a47</td>
<td>i-4b41a37c</td>
<td>us-west-2a</td>
<td>8</td>
</tr>
</tbody>
</table>

### Using AWS CLI pagination options

This topic describes the different ways to paginate output from the AWS Command Line Interface (AWS CLI).

#### Server-side pagination

For commands that can return a large list of items, the AWS Command Line Interface (AWS CLI) has multiple options to control the number of items included in the output when the AWS CLI calls a service's API to populate the list.
AWS Command Line Interface User Guide for Version 1
Pagination

- --no-paginate
- --page-size
- --max-items
- --starting-token

By default, the AWS CLI uses a page size determined by the individual service and retrieves all available items. For example, Amazon S3 has a default page size of 1000. If you run `aws s3api list-objects` on an Amazon S3 bucket that contains 3,500 objects, the AWS CLI automatically makes four calls to Amazon S3, handling the service-specific pagination logic for you in the background and returning all 3,500 objects in the final output.

**How to use the --no-paginate parameter**

The `--no-paginate` option disables following pagination tokens on the client side. When using a command, by default the AWS CLI automatically makes multiple calls to return all possible results to create pagination. One call for each page. Disabling pagination has the AWS CLI only call once for the first page of command results.

For example, if you run `aws s3api list-objects` on an Amazon S3 bucket that contains 3,500 objects, the AWS CLI only makes the first call to Amazon S3, returning only the first 1,000 objects in the final output.

```
$ aws s3api list-objects \
   --bucket my-bucket \
   --no-paginate \
{
   "Contents": [
   ...
```

**How to use the --page-size parameter**

If you see issues when running list commands on a large number of resources, the default page size might be too high. This can cause calls to AWS services to exceed the maximum allowed time and generate a “timed out” error. You can use the `--page-size` option to specify that the AWS CLI request a smaller number of items from each call to the AWS service. The AWS CLI still retrieves the full list, but performs a larger number of service API calls in the background and retrieves a smaller number of items with each call. This gives the individual calls a better chance of succeeding without a timeout. Changing the page size doesn’t affect the output; it affects only the number of API calls that need to be made to generate the output.

```
$ aws s3api list-objects \
   --bucket my-bucket \
   --page-size 100 \
{
   "Contents": [
   ...
```

**How to use the --max-items parameter**

To include fewer items at a time in the AWS CLI output, use the `--max-items` option. The AWS CLI still handles pagination with the service as described previously, but prints out only the number of items at a time that you specify.

```
$ aws s3api list-objects \
   --bucket my-bucket \
```
How to use the \texttt{--starting-token} parameter

If the number of items output (\texttt{--max-items}) is fewer than the total number of items returned by the underlying API calls, the output includes a NextToken that you can pass to a subsequent command to retrieve the next set of items. The following example shows how to use the NextToken value returned by the previous example, and enables you to retrieve the second 100 items.

\textbf{Note}
The parameter \texttt{--starting-token} cannot be null or empty. If the previous command does not return a NextToken value, there are no more items to return and you do not need to call the command again.

```bash
$ aws s3api list-objects \
  --bucket my-bucket \
  --max-items 100 \
  --starting-token eyJNYXJrZXIiOiBudWxsLCAiYm90b190cnVuY2F0ZV9hbW91bnQiOiAxfQ== \
  { 
    "Contents": [ 
      ...  
    ] 
  }
```

The specified AWS service might not return items in the same order each time you call. If you specify different values for \texttt{--page-size} and \texttt{--max-items}, you can get unexpected results with missing or duplicated items. To prevent this, use the same number for \texttt{--page-size} and \texttt{--max-items} to sync the AWS CLI pagination with the pagination of the underlying service. You can also retrieve the whole list and perform any necessary paging operations locally.

Filtering AWS CLI output

The AWS Command Line Interface (AWS CLI) has both server-side and client-side filtering that you can use individually or together to filter your AWS CLI output. Server-side filtering is processed first and returns your output for client-side filtering.

- Server-side filtering is supported by the API, and you usually implement it with a \texttt{--filter} parameter. The service only returns matching results which can speed up HTTP response times for large data sets.
- Client-side filtering is supported by the AWS CLI client using the \texttt{--query} parameter. This parameter has capabilities the server-side filtering might not have.

Topics

- Server-side filtering (p. 95)
- Client-side filtering (p. 96)
- Combining server-side and client-side filtering (p. 110)
- Additional resources (p. 111)

Server-side filtering

Server-side filtering in the AWS CLI is provided by the AWS service API. The AWS service only returns the records in the HTTP response that match your filter, which can speed up HTTP response times for large
data sets. Since server-side filtering is defined by the service API, the parameter names and functions vary between services. Some common parameter names used for filtering are:

- `--filter` such as `ses` and `ce`.
- `--filters` such as `ec2`, `autoscaling`, and `rds`.
- Names starting with the word `filter`, for example `--filter-expression` for the `aws dynamodb scan` command.

For information about whether a specific command has server-side filtering and the filtering rules, see the AWS CLI reference guide.

**Client-side filtering**

The AWS CLI provides built-in JSON-based client-side filtering capabilities with the `--query` parameter. The `--query` parameter is a powerful tool you can use to customize the content and style of your output. The `--query` parameter takes the HTTP response that comes back from the server and filters the results before displaying them. Since the entire HTTP response is sent to the client before filtering, client-side filtering can be slower than server-side filtering for large data-sets.

Querying uses JMESPath syntax to create expressions for filtering your output. To learn JMESPath syntax, see Tutorial on the JMESPath website.

**Important**
The output type you specify changes how the `--query` option operates:

- If you specify `--output text`, the output is paginated before the `--query` filter is applied, and the AWS CLI runs the query once on each page of the output. Due to this, the query includes the first matching element on each page which can result in unexpected extra output. To additionally filter the output, you can use other command line tools such as `head` or `tail`.
- If you specify `--output json`, the output is completely processed as a single, native structure before the `--query` filter is applied. The AWS CLI runs the query only once against the entire structure, producing a filtered result that is then output.

**Client-side filtering topics**
- Before you start (p. 96)
- Identifiers (p. 98)
- Selecting from a list (p. 99)
- Filtering nested data (p. 103)
- Flattening results (p. 104)
- Filtering for specific values (p. 104)
- Piping expressions (p. 105)
- Filtering for multiple identifier values (p. 105)
- Adding labels to identifier values (p. 107)
- Functions (p. 108)
- Advanced `--query` examples (p. 108)

**Before you start**

When using filter expressions used in these examples, be sure to use the correct quoting rules for your terminal shell. For more information, see the section called "Quotes with Strings" (p. 78).
The following JSON output shows an example of what the `--query` parameter can produce. The output describes three Amazon EBS volumes attached to separate Amazon EC2 instances.

Example output

```bash
$ 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
    },
    {
      "AvailabilityZone": "us-west-2a",
      "Attachments": [
        {
          "AttachTime": "2020-11-20T19:54:06.000Z",
          "InstanceId": "i-1jd73kv8",
          "VolumeId": "vol-alb3c7nd",
          "State": "attached",
          "DeleteOnTermination": true,
          "Device": "/dev/sda1"
        }
      ],
      "VolumeType": "standard",
      "VolumeId": "vol-alb3c7nd",
      "State": "in-use",
      "SnapshotId": "snap-234087fb",
      "CreateTime": "2020-11-20T19:54:05.000Z",
      "Size": 15
    }
  ]
}
Identifiers

Identifier are the labels for output values. When creating filters, you use identifiers to narrow down your query results. In the following output example, all identifiers such as Volumes, AvailabilityZone, and AttachTime are highlighted.

```bash
$ 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
        },
        {
            "AvailabilityZone": "us-west-2a",
            "Attachments": [
                {
                    "AttachTime": "2020-11-20T19:54:06.000Z",
                    "InstanceId": "i-1jd73kwv",
                    "VolumeId": "vol-alb3c7nd",
                    "State": "attached",
                    "DeleteOnTermination": true,
                    "Device": "/dev/sda1"
                }
            ],
            "VolumeType": "standard",
            "VolumeId": "vol-alb3c7nd",
            "State": "in-use",
            "SnapshotId": "snap-708e8348",
            "CreateTime": "2020-11-20T19:54:06.000Z",
            "Size": 8
        }
    ]
}
```
Filtering

"State": "in-use",
"SnapshotId": "snap-234087fb",
"CreateTime": "2020-11-20T19:54:05.000Z",
"Size": 15
"
}
]

For more information, see Identifiers on the JMESPath website.

Selecting from a list

A list or array is an identifier that is followed by a square bracket "[ " such as Volumes and Attachments in the section called “Before you start” (p. 96).

Syntax

<listName>[
]

To filter through all output from an array, you can use the wildcard notation. Wildcard expressions are expressions used to return elements using the * notation.

The following example queries all Volumes content.

$ aws ec2 describe-volumes
   --query '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": "2020-11-20T19:54:06.000Z",
      "InstanceId": "i-1jd73kv8",
      "VolumeId": "vol-3b3c7nd",
      "State": "attached",
      "DeleteOnTermination": true,
      "Device": "/dev/sda1"
     }
    ],
    "VolumeType": "standard",
    "VolumeId": "vol-3b3c7nd",
    "State": "in-use",
    "Size": 15
   }
  ]
Filtering

```json
"SnapshotId": "snap-234087fb",
"CreateTime": "2020-11-20T19:54:05.000Z",
"Size": 15
}
```}

To view a specific volume in the array by index, you call the array index. For example, the first item in the `Volumes` array has an index of 0, resulting in the `Volumes[0]` query. For more information about array indexes, see index expressions on the JMESPath website.

```bash
$ 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
}
```

To view a specific range of volumes by index, use slice with the following syntax, where `start` is the starting array index, `stop` is the index where the filter stops processing, and `step` is the skip interval.

**Syntax**

```
<arrayName>[<start>:<stop>:<step>]
```

If any of these are omitted from the slice expression, they use the following default values:

- **Start** – The first index in the list, 0.
- **Stop** – The last index in the list.
- **Step** – No step skipping, where the value is 1.

To return only the first two volumes, you use a start value of 0, a stop value of 2, and a step value of 1 as shown in the following example.

```bash
$ aws ec2 describe-volumes
  --query 'Volumes[0:2:1]'
[
  {
    "AvailabilityZone": "us-west-2a",
    "Attachments": [
      {
        "AttachTime": "2013-09-17T00:55:03.000Z",
        "InstanceId": "i-a071c394",
        "VolumeId": "vol-e11a5288",
```

100
Filtering

Since this example contains default values, you can shorten the slice from `Volumes[0:2:1]` to `Volumes[:2]`.

The following example omits default values and returns every two volumes in the entire array.

```bash
$ aws ec2 describe-volumes
   --query 'Volumes[:2]'
[
  {
   "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-4b44a37c",
     "VolumeId": "vol-2e410a47",
     "State": "attached",
     "DeleteOnTermination": true,
     "Device": "/dev/sda1"
    }
   ],
   "VolumeType": "standard",
   "VolumeId": "vol-2e410a47",
   "State": "in-use",
   "SnapshotId": "snap-708e834b",
   "CreateTime": "2013-09-18T20:26:15.000Z",
   "Size": 8
  }
]
```
Filtering

Steps can also use negative numbers to filter in the reverse order of an array as shown in the following example.

```bash
$ aws ec2 describe-volumes \
--query 'Volumes[::2]' 
[
 {
   "AvailabilityZone": "us-west-2a",
   "Attachments": [
   {
      "AttachTime": "2020-11-20T19:54:06.000Z",
      "InstanceId": "i-1jd73kv8",
      "VolumeId": "vol-a1b3c7nd",
      "State": "attached",
      "DeleteOnTermination": true,
      "Device": "/dev/sda1"
   }
   ],
   "VolumeType": "standard",
   "VolumeId": "vol-a1b3c7nd",
   "State": "in-use",
   "SnapshotId": "snap-234087fb",
   "CreateTime": "2020-11-20T19:54:05.000Z",
   "Size": 15
  },
  {
   "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
  }
]
```

For more information, see Slices on the JMESPath website.
Filtering nested data

To narrow the filtering of the `Volumes[*]` for nested values, you use subexpressions by appending a period and your filter criteria.

Syntax

\(<expression>.<expression>\)

The following example shows all Attachments information for all volumes.

```bash
$ aws ec2 describe-volumes \
  --query 'Volumes[*].Attachments'
[
  [
    {"AttachTime": "2013-09-17T00:55:03.000Z",
     "InstanceId": "i-a071c394",
     "VolumeId": "vol-e11a5288",
     "State": "attached",
     "DeleteOnTermination": true,
     "Device": "/dev/sda1"
  }
  ],
  [
    {"AttachTime": "2013-09-18T20:26:16.000Z",
     "InstanceId": "i-4b41a37c",
     "VolumeId": "vol-2e410a47",
     "State": "attached",
     "DeleteOnTermination": true,
     "Device": "/dev/sda1"
  }
  ],
  [
    {"AttachTime": "2020-11-20T19:54:06.000Z",
     "InstanceId": "i-1jd73kv8",
     "VolumeId": "vol-alb3c7nd",
     "State": "attached",
     "DeleteOnTermination": true,
     "Device": "/dev/sda1"
  }
  ]
]
```

To filter further into the nested values, append the expression for each nested identifier. The following example lists the State for all Volumes.

```bash
$ aws ec2 describe-volumes \
  --query 'Volumes[*].Attachments[*].State'
[
  ["attached"],
  ["attached"],
  ["attached"]
]
```
Flattening results

For more information, see SubExpressions on the JMESPath website.

You can flatten the results for Volumes[*].Attachments[*].State by removing the wildcard notation resulting in the Volumes[*].Attachments[].State query. Flattening often is useful to improve the readability of results.

```
$ aws ec2 describe-volumes \
  --query 'Volumes[*].Attachments[].State'
[  
  "attached",
  "attached",
  "attached"
]
```

For more information, see Flatten on the JMESPath website.

Filtering for specific values

To filter for specific values in a list, you use a filter expression as shown in the following syntax.

**Syntax**

```
? <expression> <comparator> <expression>
```

Expression comparators include ==, !=, <, <=, >, and >=. The following example filters for the VolumeIds for all Volumes in an AttachedState.

```
$ aws ec2 describe-volumes \
  --query 'Volumes[*].Attachments[?State==`attached`].VolumeId'
[  
  [  
    "vol-e11a5288"
  ],
  [  
    "vol-2e410a47"
  ],
  [  
    "vol-a1b3c7nd"
  ]
]
```

This can then be flattened resulting in the following example.

```
$ aws ec2 describe-volumes \
  --query 'Volumes[*].Attachments[?State==`attached`].VolumeId[]'
[    
  "vol-e11a5288",
  "vol-2e410a47",
  "vol-a1b3c7nd"
]
```

The following example filters for the VolumeIds of all Volumes that have a size less than 20.

```
$ aws ec2 describe-volumes \
  --query 'Volumes[?Size < `20`].VolumeId'
[    
  "vol-2e410a47",
```
Filtering

"vol-a1b3c7nd"
]

For more information, see Filter Expressions on the JMESPath website.

Piping expressions

You can pipe results of a filter to a new list, and then filter the result with another expression using the following syntax:

Syntax

<expression> | <expression>

The following example takes the filter results of the Volumes[*].Attachments[].InstanceId expression and outputs the first result in the array.

```
$ aws ec2 describe-volumes \
   --query 'Volumes[*].Attachments[].InstanceId | [0]'
"i-a071c394"
```

This example does this by first creating the array from the following expression.

```
$ aws ec2 describe-volumes \
   --query 'Volumes[*].Attachments[].InstanceId'
"i-a071c394",  
"i-4b41a37c",  
"i-1jd73kv8"
```

And then returns the first element in that array.

"i-a071c394"

For more information, see Pipe Expressions on the JMESPath website.

Filtering for multiple identifier values

To filter for multiple identifiers, you use a multiselect list by using the following syntax:

Syntax

<listName>[].<expression>, <expression>

In the following example, VolumeId and VolumeType are filtered in the Volumes list resulting in the following expression.

```
$ aws ec2 describe-volumes \ 
   --query 'Volumes[].[VolumeId, VolumeType]' 
[ 
  [  
    "vol-e11a5288",  
    "standard"  
  ],  
  [  
    "vol-2e410a47",  
    "standard"  
  ]
] 
```
To add nested data to the list, you add another multiselect list. The following example expands on the previous example by also filtering for `InstanceId` and `State` in the nested `Attachments` list. This results in the following expression.

```bash
$ aws ec2 describe-volumes \
   --query 'Volumes[].[VolumeId, VolumeType, Attachments[].[InstanceId, State]]'
```

To be more readable, flatten out the expression as shown in the following example.

```bash
$ aws ec2 describe-volumes \
   --query 'Volumes[].[VolumeId, VolumeType, Attachments[].[InstanceId, State]][]'
```
Filtering

```json
[  
  "i-1jd73kv8",
  "attached"
]
```

For more information, see Multiselect list on the JMESPath website.

Adding labels to identifier values

To make this output easier to read, use a multiselect hash with the following syntax.

**Syntax**

```
<listName>[].{<label>: <expression>, <label>: <expression>}
```

Your identifier label does not need to be the same as the name of the identifier. The following example uses the label `Type` for the `VolumeType` values.

```
$ aws ec2 describe-volumes \
   --query 'Volumes[].{VolumeType: VolumeType}'
[
  
  
  
  
  
]
```

For simplicity, the following example keeps the identifier names for each label and displays the `VolumeId`, `VolumeType`, `InstanceId`, and `State` for all volumes:

```
$ aws ec2 describe-volumes \
   --query 'Volumes[].{VolumeId: VolumeId, VolumeType: VolumeType, InstanceId: Attachments[0].InstanceId, State: Attachments[0].State}'
[
  
  
  
  
]
```

For more information, see Multiselect hash on the JMESPath website.
Functions

The JMESPath syntax contains many functions that you can use for your queries. For information on JMESPath functions, see Built-in Functions on the JMESPath website.

To demonstrate how you can incorporate a function into your queries, the following example uses the `sort_by` function. The `sort_by` function sorts an array using an expression as the sort key using the following syntax:

**Syntax**

```
sort_by(<listName>, <sort expression>)[].<expression>
```

The following example uses the previous `multiselect hash example (p. 107)` and sorts the output by `VolumeId`.

```
$ aws ec2 describe-volumes \
  --query 'sort_by(Volumes, &VolumeId)[].{VolumeId: VolumeId, VolumeType: VolumeType, 
  InstanceId: Attachments[0].InstanceId, State: Attachments[0].State}'
```

```
[{
  "VolumeId": "vol-2e410a47",
  "VolumeType": "standard",
  "InstanceId": "i-4b41a37c",
  "State": "attached"
},
  {
  "VolumeId": "vol-alb3c7nd",
  "VolumeType": "standard",
  "InstanceId": "i-1jd73kv8",
  "State": "attached"
},
  {
  "VolumeId": "vol-e11a5288",
  "VolumeType": "standard",
  "InstanceId": "i-a071c394",
  "State": "attached"
}
```

For more information, see `sort_by` on the JMESPath website.

**Advanced --query examples**

**To extract information from a specific item**

The following example uses the `--query` parameter to find a specific item in a list and then extracts information from that item. The example lists all of the `AvailabilityZones` associated with the specified service endpoint. It extracts the item from the `ServiceDetails` list that has the specified `ServiceName`, then outputs the `AvailabilityZones` field from that selected item.

```
$ aws --region us-east-1 ec2 describe-vpc-endpoint-services \
  --query 'ServiceDetails[?ServiceName==`com.amazonaws.us-east-1.ecs`].AvailabilityZones' 
```

```
[ "us-east-1a", 
  "us-east-1b", 
  "us-east-1c", 
  "us-east-1d", 
  "us-east-1e", 
```
To show snapshots after the specified creation date

The following example shows how to list all of your snapshots that were created after a specified date, including only a few of the available fields in the output.

```
$ aws ec2 describe-snapshots --owner self \
   --output json \
   --query 'Snapshots[?StartTime>=`2018-02-07`].
   {Id:SnapshotId,VId:VolumeId,Size:VolumeSize}'
[
  {
    "id": "snap-0effb42b7a1b2c3d4",
    "vid": "vol-0be9bb0bf12345678",
    "Size": 8
  }
]
```

To show the most recent AMIs

The following example lists the five most recent Amazon Machine Images (AMIs) that you created, sorted from most recent to oldest.

```
$ aws ec2 describe-images \
   --owners self \
   --query 'reverse(sort_by(Images,&CreationDate))[:5].{id:ImageId,date:CreationDate}'
[
  {
    "id": "ami-0a1b2c3d4e5f60001",
    "date": "2018-11-28T17:16:38.000Z"
  },
  {
    "id": "ami-0a1b2c3d4e5f60002",
    "date": "2018-09-15T13:51:22.000Z"
  },
  {
    "id": "ami-0a1b2c3d4e5f60003",
    "date": "2018-08-19T10:22:45.000Z"
  },
  {
    "id": "ami-0a1b2c3d4e5f60004",
    "date": "2018-05-03T12:04:02.000Z"
  },
  {
    "id": "ami-0a1b2c3d4e5f60005",
    "date": "2017-12-13T17:16:38.000Z"
  }
]
```

To show unhealthy Auto Scaling instances

The following example shows only the InstanceId for any unhealthy instances in the specified Auto Scaling group.

```
$ aws autoscaling describe-auto-scaling-groups \
   --auto-scaling-group-name My-AutoScaling-Group-Name \
   --output text \
   --query 'AutoScalingGroups[?].Instances[?HealthStatus==`Unhealthy`].InstanceId'
```
To exclude volumes with the specified tag

The following example describes all instances without a test tag. Using a simple `?Value != `test` expression does not work for excluding a volume as volumes can have multiple tags. As long as there is another tag beside test attached to the volume, the volume is still returned in the results.

To exclude all volumes with the test tag, start with the below expression to return all tags with the test tag in an array. Any tags that are not the test tag contain a null value.

```
$ aws ec2 describe-volumes \
  --query 'Volumes.Tags[?Value == `test`]'
```

Then filter out all the positive test results using the not_null function.

```
$ aws ec2 describe-volumes \
  --query 'Volumes[?not_null(Tags[?Value == `test`].Value)]'
```

Pipe the results to flatten out the results resulting in the following query.

```
$ aws ec2 describe-volumes \
  --query 'Volumes[?not_null(Tags[?Value == `test`].Value)] | []'
```

Combining server-side and client-side filtering

You can use server-side and client-side filtering together. Server-side filtering is completed first, which sends the data to the client that the --query parameter then filters. If you're using large data sets, using server-side filtering first can lower the amount of data sent to the client for each AWS CLI call, while still keeping the powerful customization that client-side filtering provides.

The following example lists Amazon EC2 volumes using both server-side and client-side filtering. The service filters a list of all attached volumes in the us-west-2a Availability Zone. The --query parameter further limits the output to only those volumes with a Size value that is larger than 50, and shows only the specified fields with user-defined names.

```
$ aws ec2 describe-volumes \
  --filters "Name=availability-zone,Values=us-west-2a" "Name=status,Values=attached" \
  --query 'Volumes[?Size > `50`].{Id:VolumeId,Size:Size,Type:VolumeType}'
```

The following example retrieves a list of images that meet several criteria. It then uses the --query parameter to sort the output by CreationDate, selecting only the most recent. Finally, it displays the ImageId of that one image.

```
$ aws ec2 describe-images \
  --owners amazon \
  --filters "Name=name,Values=amzn*gp2" "Name=virtualization-type,Values=hvm" "Name=root-device-type,Values=ebs" \
  --query "sort_by(Images, &CreationDate)[-1].ImageId" \
  --output text
ami-00ced3122871a4921
The following example displays the number of available volumes that are more than 1000 IOPS by using 
length to count how many are in a list.

```
$ aws ec2 describe-volumes \
   --filters "Name=status,Values=available" \
   --query 'length(Volumes[?Iops > `1000`])'
3
```

**Additional resources**

**JMESPath Terminal**

JMESPath Terminal is an interactive terminal command to experiment with JMESPath expressions 
that are used for client-side filtering. Using the `jpterm` command, the terminal shows immediate 
query results as you’re typing. You can directly pipe AWS CLI output to the terminal, enabling 
advanced querying experimentation.

The following example pipes `aws ec2 describe-volumes` output directly to JMESPath Terminal.

```
$ aws ec2 describe-volumes | jpterm
```

For more information on JMESPath Terminal and installation instructions, see [JMESPath Terminal on GitHub](https://github.com/mvb/jpterm).

**jq utility**

The `jq` utility provides you a way to transform your output on the client-side to an output format 
you desire. For more information on `jq` and installation instructions, see [jq on GitHub](https://github.com/stedolan/jq).

**Understanding return codes from the AWS CLI**

The return code is usually a hidden code sent after running a AWS Command Line Interface (AWS CLI) 
command which describes the status of the command. You can use the `echo` command to display 
the code sent from the last AWS CLI command and use these codes to determine if a command was 
successful or if it failed, and why a command might have an error. In addition to the return codes, you 
can view more details about a failure by running your commands with the `--debug` switch. This switch 
produces a detailed report of the steps the AWS CLI uses to process the command, and what the result of 
each step was.

To determine the return code of an AWS CLI command, run one of the following commands immediately 
after running the CLI command.

**Linux and macOS**

```
$ echo $?
0
```

**Windows PowerShell**

```
PS> echo $lastexitcode
0
```

**Windows Command Prompt**

```
C:\> echo %errorlevel%
```
The following are the return code values that can be returned at the end of running an AWS Command Line Interface (AWS CLI) command.

<table>
<thead>
<tr>
<th>Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The service responded with an HTTP response status code of 200 indicating that there were no errors generated by the AWS CLI and AWS service the request was sent to.</td>
</tr>
<tr>
<td>1</td>
<td>One or more Amazon S3 transfer operations failed. <em>Limited to S3 commands.</em></td>
</tr>
</tbody>
</table>
| 2    | The meaning of this return code depends on the command:  
  - *Applicable to all AWS CLI commands* – the command entered couldn't be parsed. Parsing failures can be caused by, but aren't limited to, missing required subcommands or arguments, or using unknown commands or arguments.  
  - *Limited to S3 commands* – One or more files marked for transfer were skipped during the transfer process. However, all other files marked for transfer were successfully transferred. Files that are skipped during the transfer process include: files that don't exist; files that are character special devices, block special device, FIFO queues, or sockets; and files that the user doesn't have read permissions to. |
| 130  | The command was interrupted by a SIGINT. This is the signal sent by you to cancel a command with Ctrl+C. |
| 255  | The command failed. There were errors generated by the AWS CLI or by the AWS service to which the request was sent. |

### Creating and using AWS CLI aliases

Aliases are shortcuts you can create in the AWS Command Line Interface (AWS CLI) to shorten commands or scripts that you frequently use. You create aliases in the `alias` file located in your configuration folder.

#### Topics
- Prerequisites (p. 112)  
- Step 1: Creating the alias file (p. 113)  
- Step 2: Creating an alias (p. 113)  
- Step 3: Calling an alias (p. 115)  
- Alias repository examples (p. 116)  
- Resources (p. 117)

#### Prerequisites

To use alias commands, you need to complete the following:

- Install and configure the AWS CLI. For more information, see *Installing the AWS CLI* (p. 4) and *Configuration basics* (p. 27).
- Use a minimum AWS CLI version of 1.11.24 or 2.0.0.
- (Optional) To use AWS CLI alias bash scripts, you must use a bash-compatible terminal.
Step 1: Creating the alias file

To create the alias file, you can use your file navigation and a text editor, or use your preferred terminal by using the step-by-step procedure. To quickly create your alias file, use the following command block.

Linux and macOS

```
$ mkdir -p ~/.aws/cli
$ echo ' [toplevel]' > ~/.aws/cli/alias
```

Windows

```
C:\> md %USERPROFILE%\aws\cli
C:\> echo [toplevel] > %USERPROFILE%\aws\cli/alias
```

To create the alias file

1. Create a folder named cli in your AWS CLI configuration folder. By default the configuration folder is `~/.aws/` on Linux or macOS and `%USERPROFILE%\aws\` on Windows. You can create this through your file navigation or by using the following command.

Linux and macOS

```
$ mkdir -p ~/.aws/cli
```

Windows

```
C:\> md %USERPROFILE%\aws\cli
```

The resulting cli folder default path is `~/.aws/cli/` on Linux or macOS and `%USERPROFILE%\aws\cli` on Windows.

2. In the cli folder, create a text file named alias with no extension and add `[toplevel]` to the first line. You can create this file through your preferred text editor or use the following command.

Linux and macOS

```
$ echo ' [toplevel]' > ~/.aws/cli/alias
```

Windows

```
C:\> echo [toplevel] > %USERPROFILE%\aws\cli/alias
```

Step 2: Creating an alias

You can create an alias using basic commands or bash scripting.

Creating a basic command alias

You can create your alias by adding a command using the following syntax in the alias file you created in the previous step.
Step 2: Creating an alias

Syntax

```
aliasname = command [--options]
```

The `aliasname` is what you call your alias. The `command` is the command you want to call, which can include other aliases. You can include options or parameters in your alias, or add them when calling your alias.

The following example creates an alias named `aws whoami` using the `aws sts get-caller-identity` command. Since this alias calls an existing AWS CLI command, you can write the command without the `aws` prefix.

```
whoami = sts get-caller-identity
```

The following example takes the previous `whoami` example and adds the `AccountName` filter and `text` output options.

```
whoami2 = sts get-caller-identity --query AccountName --output text
```

Creating a bash scripting alias

**Warning**

To use AWS CLI alias bash scripts, you must use a bash-compatible terminal.

You can create an alias using bash scripts for more advanced processes using the following syntax.

Syntax

```
aliasname =
  !f() {
    script content
  }; f
```

The `aliasname` is what you call your alias and `script content` is the script you want to run when you call the alias.

The following example uses `opendns` to output your current IP address. Since you can use aliases in other aliases, the following `myip` alias is useful to allow or revoke access for your IP address from within other aliases.

```
myip =
  !f() {
    dig +short myip.opendns.com @resolver1.opendns.com
  }; f
```

The following script example calls the previous `aws myip` alias to authorize your IP address for an Amazon EC2 security group ingress.

```
authorize-my-ip =
  !f() {
    ip=$(aws myip)
    aws ec2 authorize-security-group-ingress --group-id ${1} --cidr $ip/32 --protocol tcp --port 22
  }; f
```

When you call aliases that use bash scripting, the variables are always passed in the order that you entered them. In bash scripting, the variable names are not taken into consideration, only the order they
appear. In the following `textalert` alias example, the variable for the `--message` option is first and `--phone-number` option is second.

```
textalert =
  !f() {
    aws sns publish --message "${1}" --phone-number ${2}
  }; f
```

### Step 3: Calling an alias

To run the alias you created in your `alias` file use the following syntax. You can add additional options when you call your alias.

**Syntax**

```
$ aws aliasname
```

The following example uses the `aws whoami` alias.

```
$ aws whoami

{
  "UserId": "A12BCD34E5FCHI6JKLM",
  "Account": "1234567890987",
  "Arn": "arn:aws:iam::1234567890987:user/userName"
}
```

The following example uses the `aws whoami` alias with additional options to only return the *Account* number in *text* output.

```
$ aws whoami --query Account --output text
1234567890987
```

### Calling an alias using bash scripting variables

When you call aliases that use bash scripting, variables are passed in the order they are entered. In bash scripting, the name of the variables are not taken into consideration, only the order they appear. For example, in the following `textalert` alias, the variable for the option `--message` is first and `--phone-number` is second.

```
textalert =
  !f() {
    aws sns publish --message "${1}" --phone-number ${2}
  }; f
```

When you call the `textalert` alias, you need to pass variables in the same order as they are run in the alias. In the following example we use the variables `$message` and `$phone`. The `$message` variable is passed as `$1` for the `--message` option and the `$phone` variable is passed as `$2` for the `--phone-number` option. This results in successfully calling the `textalert` alias to send a message.

```
$ aws textalert $message $phone

{
  "MessageId": "1ab2cd3e4-fg56-7h89-i01j-2klmn34567"
}
In the following example, the order is switched when calling the alias to `$phone` and `$message`. The `$phone` variable is passed as `${1}` for the `--message` option and the `$message` variable is passed as `${2}` for the `--phone-number` option. Since the variables are out of order, the alias passes the variables incorrectly. This causes an error because the contents of `$message` do not match the phone number formatting requirements for the `--phone-number` option.

```bash
$ aws textalert $phone
$message
```

 usage: aws [options] <command> <subcommand> [<subcommand> ...] [parameters]

To see help text, you can run:

```bash
aws help
aws <command> help
aws <command> <subcommand> help
```

Unknown options: text

## Alias repository examples

The AWS CLI alias repository on GitHub contains AWS CLI alias examples created by the AWS CLI developer team and community. You can use the entire alias file example or take individual aliases for your own use.

**Warning**

Running the commands in this section deletes your existing alias file. To avoid overwriting your existing alias file, change your download location.

**To use aliases from the repository**

1. Install Git. For installation instructions, see Getting Started - Installing Git in the Git Documentation.
2. Install the `jp` command. The `jp` command is used in the `tostring` alias. For installation instructions, see the JMESPath (jp) README.md on GitHub.
3. Install the `jq` command. The `jq` command is used in the `tostring-with-jq` alias. For installation instructions, see the JSON processor (jq) on GitHub.
4. Download the alias file by doing one of the following:

   - Run the following commands that downloads from the repository and copies the alias file to your configuration folder.
     Linux and macOS
     ```bash
     $ git clone https://github.com/awslabs/awscli-aliases.git
     $ mkdir -p ~/.aws/cli
     $ cp awscli-aliases/alias ~/.aws/cli/alias
     ```
   - Download directly from the repository and save to the `cli` folder in your AWS CLI configuration folder. By default the configuration folder is `~/.aws/` on Linux or macOS and `%USERPROFILE%\aws\` on Windows.
   ```bash
     C:\> git clone https://github.com/awslabs/awscli-aliases.git
     C:\> md %USERPROFILE%\aws\cli
     C:\> copy awscli-aliases\alias %USERPROFILE%\aws\cli
     ```
5. To verify the aliases are working, run the following alias.
This displays the same response as the `aws sts get-caller-identity` command:

```
{
  "Account": "012345678901",
  "UserId": "AIUAINBADX2VEG2TC6HD6",
  "Arn": "arn:aws:iam::012345678901:user/myuser"
}
```

## Resources

- The [AWS CLI alias repository on GitHub](https://github.com) contains AWS CLI alias examples created by the AWS CLI developer team and the contribution of the AWS CLI community.
- The alias feature announcement from [AWS re:Invent 2016: The Effective AWS CLI User on YouTube](https://www.youtube.com).
- `aws sts get-caller-identity`
- `aws ec2 describe-instances`
- `aws sns publish`
Using the AWS CLI to work with AWS Services

This section provides examples that show how to use the AWS Command Line Interface (AWS CLI) to access various AWS services.

**Note**
For a complete reference of all the available commands for each service, see the AWS CLI reference guide, or use the built-in command line help. For more information, see Getting help with the AWS CLI (p. 68).

**Services**
- Using Amazon DynamoDB with the AWS CLI (p. 118)
- Using Amazon EC2 with the AWS CLI (p. 121)
- Using Amazon S3 Glacier with the AWS CLI (p. 134)
- Using AWS Identity and Access Management from the AWS CLI (p. 138)
- Using Amazon S3 with the AWS CLI (p. 141)
- Using Amazon SNS with the AWS CLI (p. 154)
- Using Amazon Simple Workflow Service with the AWS CLI (p. 156)

Using Amazon DynamoDB with the AWS CLI

**An introduction to Amazon DynamoDB**

<table>
<thead>
<tr>
<th>What is Amazon DynamoDB?</th>
</tr>
</thead>
<tbody>
<tr>
<td>The AWS Command Line Interface (AWS CLI) provides support for all of the AWS database services, including Amazon DynamoDB. You can use the AWS CLI for impromptu operations, such as creating a table. You can also use it to embed DynamoDB operations within utility scripts. For more information about using the AWS CLI with DynamoDB, see <code>dynamodb</code> in the AWS CLI Command Reference. To list the AWS CLI commands for DynamoDB, use the following command:</td>
</tr>
</tbody>
</table>

```
$ aws dynamodb help
```

**Topics**
- Prerequisites (p. 119)
- Creating and using DynamoDB tables (p. 119)
- Using DynamoDB Local (p. 120)
Prerequisites

To run the `dynamodb` commands, you need to:

- AWS CLI installed, see *Installing the AWS CLI* (p. 4) for more information.
- AWS CLI configured, see *Configuration basics* (p. 27) for more information. The profile that you use must have permissions that allow the AWS operations performed by the examples.

Creating and using DynamoDB tables

The command line format consists of an DynamoDB command name, followed by the parameters for that command. The AWS CLI supports the CLI shorthand syntax (p. 87) for the parameter values, and full JSON.

The following example creates a table named `MusicCollection`.

```
$ aws dynamodb create-table \
   --table-name MusicCollection \
   --attribute-definitions AttributeName=Artist,AttributeType=S \n   AttributeName=SongTitle,AttributeType=S \
   --key-schema AttributeName=Artist,KeyType=HASH AttributeName=SongTitle,KeyType=RANGE \n   --provisioned-throughput ReadCapacityUnits=1,WriteCapacityUnits=1
```

You can add new lines to the table with commands similar to those shown in the following example. These examples use a combination of shorthand syntax and JSON.

```
$ aws dynamodb put-item \
   --table-name MusicCollection \
   --item '{
       "Artist": "{S": "No One You Know"},
       "SongTitle": "{S": "Call Me Today"},
       "AlbumTitle": "{S": "Somewhat Famous"}
    }' \
   --return-consumed-capacity TOTAL
{
  "ConsumedCapacity": {
    "CapacityUnits": 1.0,
    "TableName": "MusicCollection"
  }
}

$ aws dynamodb put-item \
   --table-name MusicCollection \
   --item '{
       "Artist": "{S": "Acme Band"},
       "SongTitle": "{S": "Happy Day"},
       "AlbumTitle": "{S": "Songs About Life"}
    }' \
   --return-consumed-capacity TOTAL
{
  "ConsumedCapacity": {
    "CapacityUnits": 1.0,
    "TableName": "MusicCollection"
  }
}
```
It can be difficult to compose valid JSON in a single-line command. To make this easier, the AWS CLI can read JSON files. For example, consider the following JSON snippet, which is stored in a file named expression-attributes.json.

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

You can use that file to issue a query request using the AWS CLI. In the following example, the content of the expression-attributes.json file is used as the value for the --expression-attribute-values parameter.

```bash
$ aws dynamodb query --table-name MusicCollection \
     --key-condition-expression "Artist = :v1 AND SongTitle = :v2" \
     --expression-attribute-values file://expression-attributes.json
```

```json
{
 "Count": 1,
 "Items": [
 {  
  "AlbumTitle": {  
   "S": "Somewhat Famous"
  },
  "SongTitle": {  
   "S": "Call Me Today"
  },
  "Artist": {  
   "S": "No One You Know"
  }
 },
 {  
  "ScannedCount": 1,
  "ConsumedCapacity": null
 }
}
```

### Using DynamoDB Local

In addition to DynamoDB, you can use the AWS CLI with DynamoDB Local. DynamoDB Local is a small client-side database and server that mimics the DynamoDB service. DynamoDB Local enables you to write applications that use the DynamoDB API, without manipulating any tables or data in the DynamoDB web service. Instead, all of the API actions are rerouted to a local database. This lets you save on provisioned throughput, data storage, and data transfer fees.

For more information about DynamoDB Local and how to use it with the AWS CLI, see the following sections of the Amazon DynamoDB Developer Guide:

- DynamoDB Local
- Using the AWS CLI with DynamoDB Local

### Resources

**AWS CLI reference:**

- `aws dynamodb`
- `aws dynamodb create-table`
- `aws dynamodb put-item`
- `aws dynamodb query`
Using Amazon EC2 with the AWS CLI

An introduction to Amazon Elastic Compute Cloud

Introduction to Amazon EC2 - Elastic Cloud Server and Hosting with AWS

You can access the features of Amazon Elastic Compute Cloud (Amazon EC2) using the AWS Command Line Interface (AWS CLI). To list the AWS CLI commands for Amazon EC2, use the following command.

```
aws ec2 help
```

Before you run any commands, set your default credentials. For more information, see Configuring the AWS CLI (p. 27).

This topic shows short-form examples of AWS CLI commands that perform common tasks for Amazon EC2.

For long-form examples of AWS CLI commands, see AWS CLI code examples repository on GitHub.

Topics
- Creating, displaying, and deleting Amazon EC2 key pairs (p. 121)
- Creating, configuring, and deleting security groups for Amazon EC2 (p. 123)
- Launching, listing, and terminating Amazon EC2 instances (p. 127)
- Change an Amazon EC2 instance type using a bash script (p. 131)

Creating, displaying, and deleting Amazon EC2 key pairs

You can use the AWS Command Line Interface (AWS CLI) to create, display, and delete your key pairs for Amazon Elastic Compute Cloud (Amazon EC2). You use key pairs to connect to an Amazon EC2 instance.

You must provide the key pair to Amazon EC2 when you create the instance, and then use that key pair to authenticate when you connect to the instance.

Note
For additional command examples, see the AWS CLI reference guide.

Topics
- Prerequisites (p. 122)
- Create a key pair (p. 122)
- Display your key pair (p. 123)
- Delete your key pair (p. 123)
- References (p. 123)
Prerequisites

To run the `ec2` commands, you need to:

- **Install and configure the AWS CLI.** For more information, see Installing the AWS CLI (p. 4) and Configuration basics (p. 27).
- **Set your IAM permissions to allow for Amazon EC2 access.** For more information about IAM permissions for Amazon EC2, see IAM policies for Amazon EC2 in the Amazon EC2 User Guide for Linux Instances.

Create a key pair

To create a key pair, use the `aws ec2 create-key-pair` command with the --query option, and the --output text option to pipe your private key directly into a file.

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

For PowerShell, the > file redirection defaults to UTF-8 encoding, which cannot be used with some SSH clients. So, you must convert the output by piping it to the out-file command and explicitly set the encoding to ascii.

```
PS C:\>aws ec2 create-key-pair --key-name MyKeyPair --query 'KeyMaterial' --output text | out-file -encoding ascii -filepath MyKeyPair.pem
```

The resulting `MyKeyPair.pem` file looks similar to the following.

```
-----BEGIN RSA PRIVATE KEY-----
EXAMPLEKEYCAQEAy7WzaDsDrA1w3mlRtqtvhwyORRX8ngxDAfRt/gx42kWxS5rXE/bSCpSgie/vBoU7jLxxk92pNNOFnByFp+Dc21eyz9C6vJrTMuAOJwFwi5/akH71O56rUc7d9qKw2du5qUdEOQWZ/aXNvMngQ5E6XqfwnXV8werrqo+ZWQeqiUwMkudEBLeJFLhMCYURpUMSC1oehm4491x9X1FG50TCFeOzlf8dqPC82zQaPIj1U9X/aizO9V+tpU52E/qtmXZC3/npFQ5xvZDOJH67kms6ufPw
oPe2ev/D8v+x+bth6f3j9179D2qvB5hXWkxgBzd2C0U/wei5D/HyywDgAQBdIQBYG2kAvnuq
/uzer/vlg5n5m715L5hj4lHLAIw6U/ezVvchK05kbCCQXurHmuQ2MqYz3/okn2fjYV/uxGbLi
mbv5n7p4j834d6Qs83iKLWVUyjGc0ui1smJoap/GTLUW5Nfnc36P4BUHyp53Y6G7hXb2
babyWy7NfJ1e4M86g3d2YK3V1CmX/c/BOAsHnJ+6e3JrXXPwVn39z5EmCdJa/R15DZyhm/lJZWSd9
81cGk9TopE7CifatEA7yy7LqgQ6k64i1u9mJkA3o3zXmQexXVJ1LZVEHG7bH1Y9d801osR
eQf/FI2xNAZ2iijCcyY/y1pJ403+kc8yEAym9tyhkkFdpwsMARlA18cNABbjwEy7Z5Mqfg1+1p1
YkriL0DBLx1VrFAH+YHRIt2hHjOjU/IN2h4Axv+cpq09qbuI3+43eEy2487G/Uh+GTFbjeXaOxQx/x
p90tyVwc7hsq5TA5P2b+mvkJ50BEKset9XcKwONBYELGhnep7cCgYEA6Vg0v6YHIlHuik9Huw
ayaovxc5zkj9FnhFHFry2/1r1trv2Vdpn+9x481UrpzWV0EKhv+u+xttmaS1s/P1lkq5DwruNW
WA8gk6nO3E3f3qy2N9B6raAKdJJ53L1HvQveTe10HLYTxpJnkEvf+Un1a2j1vWUt5pBbrBuc
gyTjbo+0ZkOscCpZ99bhsyIzX5yX5v2uQwaIjdApP9Fthane599y+BxN5LlywcWQhiv0bH
oM07yvYAB70zdz5wQemB4AlsSjX5sUaRcaOBsOy1as8w6fao53izladoxNeGsArag6VrlGt8uAEBgkZv9wK4fGgf+GwV1lw71rsgD5rWhMFPFTxJnuZa6y5eGxlaCjgFKahy
Whd4Xz9mHnmB5lAyFyRj/T01bxyArnxMhXIOIwAnXMXGB4KSylmSVAQo+fGe+c3Jod1F1lj
jbb0d/1jYfb1NdxAVH86SSbkdsx2fEnELEYyBKJSr9enRaOGAnMrTwXnZeTvskF/S5Fyu0ioEgLdA
NWWU3t8/vNdcQPkiD5m3n3qEcjuJmmbWlv+E+Y2jV7vU7G38MywtGNTGtdb6snqY2a5snrFqS
VrRkakKXYegjkuupUv9FyjxKxcfr/V+QPF50ndHAKJJxjW7a4ejJLncTzmZ5PyZwApce=
-----END RSA PRIVATE KEY-----
```

Your private key isn't stored in AWS and can be retrieved only when it's created. You can't recover it later. Instead, if you lose the private key, you must create a new key pair.

If you're connecting to your instance from a Linux computer, we recommend that you use the following command to set the permissions of your private key file so that only you can read it.
Display your key pair

A "fingerprint" is generated from your key pair, and you can use it to verify that the private key that you have on your local machine matches the public key that’s stored in AWS.

The fingerprint is an SHA1 hash taken from a DER-encoded copy of the private key. This value is captured when the key pair is created, and is stored in AWS with the public key. You can view the fingerprint in the Amazon EC2 console or by running the AWS CLI command `aws ec2 describe-key-pairs`.

The following example displays the fingerprint for `MyKeyPair`.

```bash
$ aws ec2 describe-key-pairs --key-name MyKeyPair
{
  "KeyPairs": [
    {
      "KeyName": "MyKeyPair",
    }
  ]
}
```

For more information about keys and fingerprints, see Amazon EC2 Key Pairs in the Amazon EC2 User Guide for Linux Instances.

Delete your key pair

To delete a key pair, run the `aws ec2 delete-key-pair` command, substituting `MyKeyPair` with the name of the pair to delete.

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

References

AWS CLI reference:

- `aws ec2`
- `aws ec2 create-key-pair`
- `aws ec2 delete-key-pair`
- `aws ec2 describe-key-pairs`

Other reference:

- Amazon Elastic Compute Cloud Documentation
- To view and contribute to AWS SDK and AWS CLI code examples, see the AWS Code Examples Repository on GitHub.

Creating, configuring, and deleting security groups for Amazon EC2

You can create a security group for your Amazon Elastic Compute Cloud (Amazon EC2) instances that essentially operates as a firewall, with rules that determine what network traffic can enter and leave.
Use the AWS Command Line Interface (AWS CLI) to create a security group, add rules to existing security groups, and delete security groups.

**Note**
For additional command examples, see the AWS CLI reference guide.

**Topics**
- Prerequisites (p. 124)
- Create a security group (p. 124)
- Add rules to your security group (p. 125)
- Delete your security group (p. 126)
- References (p. 126)

**Prerequisites**
To run the `ec2` commands, you need to:

- Install and configure the AWS CLI. For more information, see *Installing the AWS CLI* (p. 4) and *Configuration basics* (p. 27).
- Set your IAM permissions to allow for Amazon EC2 access. For more information about IAM permissions for Amazon EC2, see *IAM policies for Amazon EC2* in the Amazon EC2 User Guide for Linux Instances.

**Create a security group**
You can create security groups associated with virtual private clouds (VPCs).

The following `aws ec2 create-security-group` example shows how to create a security group for a specified VPC.

```bash
$ aws ec2 create-security-group --group-name my-sg --description "My security group" --vpc-id vpc-1a2b3c4d
{
  "GroupId": "sg-903004f8"
}
```

To view the initial information for a security group, run the `aws ec2 describe-security-groups` command. You can reference an EC2-VPC security group only by its `vpc-id`, not its name.

```bash
$ 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"
    }
  ]
}
Add rules to your security group

When you run an Amazon EC2 instance, you must enable rules in the security group to allow incoming network traffic for your means of connecting to the image.

For example, if you're launching a Windows instance, you typically add a rule to allow inbound traffic on TCP port 3389 to support Remote Desktop Protocol (RDP). If you're launching a Linux instance, you typically add a rule to allow inbound traffic on TCP port 22 to support SSH connections.

Use the `aws ec2 authorize-security-group-ingress` command to add a rule to your security group. A required parameter of this command is the public IP address of your computer, or the network (in the form of an address range) that your computer is attached to, in CIDR notation.

**Note**

We provide the following service, https://checkip.amazonaws.com/, to enable you to determine your public IP address. To find other services that can help you identify your IP address, use your browser to search for "what is my IP address". If you connect through an ISP or from behind your firewall using a dynamic IP address (through a NAT gateway from a private network), your address can change periodically. In that case, you must find out the range of IP addresses used by client computers.

The following example shows how to add a rule for RDP (TCP port 3389) to an EC2-VPC security group with the ID `sg-903004f8` using your IP address.

To start, find your IP address.

```bash
$ curl https://checkip.amazonaws.com
x.x.x.x
```

You can then add the IP address to your security group by running the `aws ec2 authorize-security-group-ingress` command.

```bash
$ aws ec2 authorize-security-group-ingress --group-id sg-903004f8 --protocol tcp --port 3389 --cidr x.x.x.x
```

The following command adds another rule to enable SSH to instances in the same security group.

```bash
$ aws ec2 authorize-security-group-ingress --group-id sg-903004f8 --protocol tcp --port 22 --cidr x.x.x.x
```

To view the changes to the security group, run the `aws ec2 describe-security-groups` command.

```bash
$ aws ec2 describe-security-groups --group-ids sg-903004f8
{
    "SecurityGroups": [
        {
            "IpPermissionsEgress": []
        }
    ]
}`
Delete your security group

To delete a security group, run the `aws ec2 delete-security-group` command.

**Note**

You can't delete a security group if it's currently attached to an environment.

The following command example deletes an EC2-VPC security group.

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

References

**AWS CLI reference:**

- `aws ec2`
- `aws ec2 authorize-security-group-ingress`
- `aws ec2 create-security-group`
- `aws ec2 delete-security-group`
- `aws ec2 describe-security-groups`

**Other reference:**

- Amazon Elastic Compute Cloud Documentation
- To view and contribute to AWS SDK and AWS CLI code examples, see the AWS Code Examples Repository on GitHub.
Launching, listing, and terminating Amazon EC2 instances

You can use the AWS Command Line Interface (AWS CLI) to launch, list, and terminate Amazon Elastic Compute Cloud (Amazon EC2) instances. If you launch an instance that isn't within the AWS Free Tier, you are billed after you launch the instance and charged for the time that the instance is running, even if it remains idle.

**Note**
For additional command examples, see the AWS CLI reference guide.

Topics
- Prerequisites (p. 127)
- Launch your instance (p. 127)
- Add a block device to your instance (p. 129)
- Add a tag to your instance (p. 130)
- Connect to your instance (p. 130)
- List your instances (p. 130)
- Terminate your instance (p. 131)
- References (p. 131)

Prerequisites

To run the `ec2` commands in this topic, you need to:

- Install and configure the AWS CLI. For more information, see *Installing the AWS CLI* (p. 4) and *Configuration basics* (p. 27).
- Set your IAM permissions to allow for Amazon EC2 access. For more information about IAM permissions for Amazon EC2, see *IAM policies for Amazon EC2* in the *Amazon EC2 User Guide for Linux Instances*.
- Create a key pair (p. 121) and a security group (p. 123).
- Select an Amazon Machine Image (AMI) and note the AMI ID. For more information, see *Finding a Suitable AMI* in the *Amazon EC2 User Guide for Linux Instances*.

Launch your instance

To launch an Amazon EC2 instance using the AMI you selected, use the `aws ec2 run-instances` command. You can launch the instance into a virtual private cloud (VPC).

Initially, your instance appears in the **pending** state, but changes to the **running** state after a few minutes.

The following example shows how to launch a `t2.micro` instance in the specified subnet of a VPC. Replace the *italicized* parameter values with your own.

```
$ aws ec2 run-instances --image-id ami-xxxxxxxx --count 1 --instance-type t2.micro --key-name MyKeyPair --security-group-ids sg-903004f8 --subnet-id subnet-6e7f829e
{
  "OwnerId": "123456789012",
  "ReservationId": "r-5875ca20",
  "Groups": [
    {
      "GroupName": "my-sg",
      "IPPermissions": [
        {
          "IpProtocol": "tcp",
          "FromPort": 22,
          "ToPort": 22,
          "UserIdGroupPairs": [
            {
              "UserId": "123456789012",
              "GroupId": "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-a7edbc9",
"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,
Add a block device to your instance

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

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

The following example parameter provisions a standard Amazon EBS volume that is 20 GB in size, and maps it to your instance using the identifier `/dev/sdf`.

```
--block-device-mappings "[{"DeviceName":"/dev/sdf","Ebs":{"VolumeSize":20, "DeleteOnTermination":false}}]"
```

The following example adds an Amazon EBS volume, mapped to `/dev/sdf`, based on an existing snapshot. A snapshot represents an image that is loaded onto the volume for you. When you specify a snapshot, you don't have to specify a volume size; it will be large enough to hold your image. However, if you do specify a size, it must be greater than or equal to the size of the snapshot.

```
--block-device-mappings "[{"DeviceName":"/dev/sdf","Ebs":{"SnapshotId":"snap-a1b2c3d4"}}]"
```

The following example adds two volumes to your instance. The number of volumes available to your instance depends on its instance type.
The following example creates the mapping (/dev/sdj), but doesn't provision a volume for the instance.

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

For more information, see Block Device Mapping in the Amazon EC2 User Guide for Linux Instances.

### Add a tag to your instance

A tag is a label that you assign to an AWS resource. It enables you to add metadata to your resources that you can use for a variety of purposes. For more information, see Tagging Your Resources in the Amazon EC2 User Guide for Linux Instances.

The following example shows how to add a tag with the key name "Name" and the value "MyInstance" to the specified instance, by using the `aws ec2 create-tags` command.

```
$ aws ec2 create-tags --resources i-5203422c --tags Key=Name,Value=MyInstance
```

### Connect to your instance

When your instance is running, you can connect to it and use it just as you'd use a computer sitting in front of you. For more information, see Connect to Your Amazon EC2 Instance in the Amazon EC2 User Guide for Linux Instances.

### List your instances

You can use the AWS CLI to list your instances and view information about them. You can list all your instances, or filter the results based on the instances that you're interested in.

The following examples show how to use the `aws ec2 describe-instances` command.

The following command lists all your instances.

```
$ aws ec2 describe-instances
```

The following command filters the list to only your `t2.micro` instances and outputs only the `InstanceId` values for each match.

```
$ aws ec2 describe-instances --filters "Name=instance-type,Values=t2.micro" --query "Reservations[].Instances[].InstanceId"

[ "i-05e998023d9e69f9a"
]
```

The following command lists any of your instances that have the tag `Name=MyInstance`.

```
$ aws ec2 describe-instances --filters "Name=tag:Name,Values=MyInstance"
```

The following command lists your instances that were launched using any of the following AMIs: ami-x0123456, ami-y0123456, and ami-z0123456.
Terminate your instance

Terminating an instance deletes it. You can't reconnect to an instance after you've terminated it.

As soon as the state of the instance changes to shutting-down or terminated, you stop incurring charges for that instance. If you want to reconnect to an instance later, use stop-instances instead of terminate-instances. For more information, see Terminate Your Instance in the Amazon EC2 User Guide for Linux Instances.

To delete an instance, you use the command `aws ec2 terminate-instances` to delete it.

References

AWS CLI reference:

- `aws ec2`
- `aws ec2 create-tags`
- `aws ec2 describe-instances`
- `aws ec2 run-instances`
- `aws ec2 terminate-instances`

Other reference:

- Amazon Elastic Compute Cloud Documentation
- To view and contribute to AWS SDK and AWS CLI code examples, see the AWS Code Examples Repository on GitHub.

Change an Amazon EC2 instance type using a bash script

This bash scripting example for Amazon EC2 changes the instance type for an Amazon EC2 instance using the AWS Command Line Interface (AWS CLI). It stops the instance if it's running, changes the
instance type, and then, if requested, restarts the instance. Shell scripts are programs designed to run in a command line interface.

**Note**
For additional command examples, see the AWS CLI reference guide.

**Topics**
- Before you start (p. 132)
- About this example (p. 132)
- Parameters (p. 132)
- Files (p. 133)
- References (p. 133)

**Before you start**

Before you can run any of the below examples, the following things need to be completed.

- AWS CLI installed, see *Installing the AWS CLI* (p. 4) for more information.
- AWS CLI configured, see *Configuration basics* (p. 27) for more information. The profile that you use must have permissions that allow the AWS operations performed by the examples.
- A running Amazon EC2 instance in the account for which you have permission to stop and modify. If you run the test script, it launches an instance for you, tests changing the type, and then terminates the instance.
- As an AWS best practice, grant this code least privilege, or only the permissions required to perform a task. For more information, see *Grant Least Privilege* in the *AWS Identity and Access Management (IAM) User Guide*.
- This code has not been tested in all AWS Regions. Some AWS services are available only in specific Regions. For more information, see *Service Endpoints and Quotas* in the *AWS General Reference Guide*.
- Running this code can result in charges to your AWS account. It is your responsibility to ensure that any resources created by this script are removed when you are done with them.

**About this example**

This example is written as a function in the shell script file `change_ec2_instance_type.sh` that you can source from another script or from the command line. Each script file contains comments describing each of the functions. Once the function is in memory, you can invoke it from the command line. For example, the following commands change the type of the specified instance to `t2.nano`:

```
$ source ./change_ec2_instance_type.sh
$ ./change_ec2_instance_type -i *instance-id* -t new-type
```

For the full example and downloadable script files, see *Change Amazon EC2 Instance Type* in the *AWS Code Examples Repository* on *GitHub*.

**Parameters**

- `i` *(string)* Specifies the instance ID to modify.
- `t` *(string)* Specifies the Amazon EC2 instance type to switch to.
- `r` *(switch)* By default, this is unset. If `-r` is set, restarts the instance after the type switch.
-f - (switch) By default, the script prompts the user to confirm shutting down the instance before making the switch. If -f is set, the function doesn’t prompt the user before shutting down the instance to make the type switch.

-v - (switch) By default, the script operates silently and displays output only in the event of an error. If -v is set, the function displays status throughout its operation.

Files

change_ec2_instance_type.sh

The main script file contains the change_ec2_instance_type() function that performs the following tasks:

• Verifies that the specified Amazon EC2 instance exists.
• Unless -f is selected, warns the user before stopping the instance.
• Changes the instance type.
• If you set -r, restarts the instance and confirms that the instance is running.

View the code for change_ec2_instance_type.sh on GitHub.

test_change_ec2_instance_type.sh

The file change_ec2_instance_type_test.sh script tests the various code paths for the change_ec2_instance_type function. If all steps in the test script work correctly, the test script removes all resources that it created.

You can run the test script with the following parameters:

• -v - (switch) The each test shows a pass/failure status as they run. By default, the tests runs silently and the output includes only the final overall pass/failure status.
• -i - (switch) The script pauses after each test to enable you to browse the intermediate results of each step. Enables you to examine the current status of the instance using the Amazon EC2 console. The script proceeds to the next step after you press ENTER at the prompt.

View the code for test_change_ec2_instance_type.sh on GitHub.

awsdocs_general.sh

The script file awsdocs_general.sh holds general purpose functions used across advanced examples for the AWS CLI.

View the code for awsdocs_general.sh on GitHub.

References

AWS CLI reference:

• `aws ec2`
• `aws ec2 describe-instances`
• `aws ec2 modify-instance-attribute`
• `aws ec2 start-instances`
• `aws ec2 stop-instances`
• `aws ec2 wait instance-running`
• `aws ec2 wait instance-stopped`
Using Amazon S3 Glacier with the AWS CLI

An introduction to Amazon S3 Glacier

Introduction to Amazon S3 Glacier

This topic shows examples of AWS CLI commands that perform common tasks for S3 Glacier. The examples demonstrate how to use the AWS CLI to upload a large file to S3 Glacier by splitting it into smaller parts and uploading them from the command line.

You can access Amazon S3 Glacier features using the AWS Command Line Interface (AWS CLI). To list the AWS CLI commands for S3 Glacier, use the following command.

```
aws glacier help
```

**Note**
For command reference and additional examples, see `aws glacier` in the *AWS CLI Command Reference.*

**Topics**
- Prerequisites (p. 134)
- Create an Amazon S3 Glacier vault (p. 134)
- Prepare a file for uploading (p. 135)
- Initiate a multipart upload and upload files (p. 135)
- Complete the upload (p. 136)
- Resources (p. 138)

**Prerequisites**
To run the `glacier` commands, you need to:

- AWS CLI installed, see *Installing the AWS CLI* (p. 4) for more information.
- AWS CLI configured, see Configuration basics (p. 27) for more information. The profile that you use must have permissions that allow the AWS operations performed by the examples.
- This tutorial uses several command line tools that typically come preinstalled on Unix-like operating systems, including Linux and macOS. Windows users can use the same tools by installing Cygwin and running the commands from the Cygwin terminal. We note Windows native commands and utilities that perform the same functions where available.

**Create an Amazon S3 Glacier vault**
Create a vault with the `create-vault` command.
$ aws glacier create-vault --account-id - --vault-name myvault
{
   "location": "/123456789012/vaults/myvault"
}

Note
All S3 Glacier commands require an account ID parameter. Use the hyphen character (--account-id -) to use the current account.

Prepare a file for uploading

Create a file for the test upload. The following commands create a file named `largefile` that contains exactly 3 MiB of random data.

Linux or macOS

$ dd if=/dev/urandom of=largefile bs=3145728 count=1
1+0 records in
1+0 records out
3145728 bytes (3.1 MB) copied, 0.205813 s, 15.3 MB/s

$ is a utility that copies a number of bytes from an input file to an output file. The previous example uses the system device file `/dev/urandom` as a source of random data. `fsutil` performs a similar function in Windows.

Windows

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

Next, split the file into 1 MiB (1,048,576 byte) chunks.

$ split -b 1048576 --verbose largefile chunk
creating file `chunkaa'
creating file `chunkab'
creating file `chunkac'

Note
`HJ-Split` is a free file splitter for Windows and many other platforms.

Initiate a multipart upload and upload files

Create a multipart upload in Amazon S3 Glacier by using the `initiate-multipart-upload` command.

$ aws glacier initiate-multipart-upload --account-id - --archive-description "multipart upload test" --part-size 1048576 --vault-name myvault
{
   "uploadId": "19gaRezEXAMPLES6ry5YYdqthHOC_kGRCT03L9yetr220UmPtBYKk-OssZtLgyPu7sY1_lR7vgPuJV6NtcV5psasJ",
   "location": "/123456789012/vaults/myvault/multipart-uploads/19gaRezEXAMPLES6ry5YYdqthHOC_kGRCT03L9yetr220UmPtBYKk-OssZtLgyPu7sY1_lR7vgPuJV6NtcV5psasJ"
}
S3 Glacier requires the size of each part in bytes (1 MiB in this example), your vault name, and an account ID to configure the multipart upload. The AWS CLI outputs an upload ID when the operation is complete. Save the upload ID to a shell variable for later use.

**Linux or macOS**

```bash
$ UPLOADID="19gaRezEXAMPLES6ry5YYdqthHOC_kGRCT03L9yetr220UmPtyRk-
OssEtLgyFv7sY1_LR7vgFvJv6NtcV5psaJ"
```

**Windows**

```bash
C:\> set UPLOADID="19gaRezEXAMPLES6ry5YYdqthHOC_kGRCT03L9yetr220UmPtyRk-
OssEtLgyFv7sY1_LR7vgFvJv6NtcV5psaJ"
```

Next, use the `upload-multipart-part` command to upload each of the three parts.

```bash
$ aws glacier upload-multipart-part --upload-id $UPLOADID --body chunkaa --range 'bytes 0-1048575/*' --account-id - --vault-name myvault
{
  "checksum": "e1f2a7cd6e047fa606fe2f0280350f69b9f8cfa602097a9a026360a7edc1f553"
}

$ aws glacier upload-multipart-part --upload-id $UPLOADID --body chunkab --range 'bytes 1048576-2097151/*' --account-id - --vault-name myvault
{
  "checksum": "e1f2a7cd6e047fa606fe2f0280350f69b9f8cfa602097a9a026360a7edc1f553"
}

$ aws glacier upload-multipart-part --upload-id $UPLOADID --body chunkac --range 'bytes 2097152-3145727/*' --account-id - --vault-name myvault
{
  "checksum": "e1f2a7cd6e047fa606fe2f0280350f69b9f8cfa602097a9a026360a7edc1f553"
}
```

**Note**
The previous example uses the dollar sign ($) to reference the contents of the UPLOADID shell variable on Linux. On the Windows command line, use a percent sign (%) on either side of the variable name (for example, %UPLOADID%).

You must specify the byte range of each part when you upload it so that S3 Glacier can reassemble it in the correct order. Each piece is 1,048,576 bytes, so the first piece occupies bytes 0-1048575, the second 1048576-2097151, and the third 2097152-3145727.

### Complete the upload

Amazon S3 Glacier requires a tree hash of the original file to confirm that all of the uploaded pieces reached AWS intact.

To calculate a tree hash, you must split the file into 1 MiB parts and calculate a binary SHA-256 hash of each piece. Then you split the list of hashes into pairs, combine the two binary hashes in each pair, and take hashes of the results. Repeat this process until there is only one hash left. If there is an odd number of hashes at any level, promote it to the next level without modifying it.

The key to calculating a tree hash correctly when using command line utilities is to store each hash in binary format and convert to hexadecimal only at the last step. Combining or hashing the hexadecimal version of any hash in the tree will cause an incorrect result.

**Note**

Windows users can use the `type` command in place of `cat`. OpenSSL is available for Windows at [OpenSSL.org](https://www.OpenSSL.org).
To calculate a tree hash

1. If you haven't already, split the original file into 1 MiB parts.

   ```bash
   $ 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.

   ```bash
   $ 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.

   ```bash
   $ 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.

   ```bash
   $ cat hash12hash hash3 > hash123
   $ openssl dgst -sha256 hash123
   SHA256(hash123)= 9628195fcdbcbbe76cdde932d4646fa7de5f219fb39823836d81f0cc0e18aa67
   $ TREEHASH=9628195fcdbcbbe76cdde932d4646fa7de5f219fb39823836d81f0cc0e18aa67
   ```

Finally, complete the upload with the `complete-multipart-upload` command. This command takes the original file's size in bytes, the final tree hash value in hexadecimal, and your account ID and vault name.

```bash
$ aws glacier complete-multipart-upload --checksum $TREEHASH --archive-size 3145728 --upload-id $UPLOADID --account-id - --vault-name myvault
```

You can also check the status of the vault using the `describe-vault` command.

```bash
$ aws glacier describe-vault --account-id - --vault-name myvault
```

**Note**

Vault status is updated about once per day. See Working with Vaults for more information.

Now it's safe to remove the chunk and hash files that you created.
For more information on multipart uploads, see *Uploading Large Archives in Parts* and *Computing Checksums* in the *Amazon S3 Glacier Developer Guide*.

**Resources**

**AWS CLI reference:**

- `aws glacier`
- `aws glacier complete-multipart-upload`
- `aws glacier create-vault`
- `aws glacier describe-vault`
- `aws glacier initiate-multipart-upload`

**Service reference:**

- *Amazon S3 Glacier Developer Guide*
- *Uploading Large Archives in Parts* in the *Amazon S3 Glacier Developer Guide*
- *Computing Checksums* in the *Amazon S3 Glacier Developer Guide*
- *Working with Vaults* in the *Amazon S3 Glacier Developer Guide*

---

Using AWS Identity and Access Management from the AWS CLI

**An introduction to AWS Identity and Access Management**

<table>
<thead>
<tr>
<th>Introduction to AWS Identity and Access Management</th>
</tr>
</thead>
</table>

You can access the features of AWS Identity and Access Management (IAM) using the AWS Command Line Interface (AWS CLI). To list the AWS CLI commands for IAM, use the following command.

`aws iam help`

This topic shows examples of AWS CLI commands that perform common tasks for IAM.

Before you run any commands, set your default credentials. For more information, see *Configuring the AWS CLI* (p. 27).

For more information on the IAM service, see the *AWS Identity and Access Management User Guide*.

**Topics**

- Creating IAM users and groups (p. 139)
- Attaching an IAM managed policy to an IAM user (p. 140)
- Setting an initial password for an IAM user (p. 140)
- Create an access key for an IAM user (p. 141)
Creating IAM users and groups

This topic describes how to use AWS Command Line Interface (AWS CLI) commands to create an AWS Identity and Access Management (IAM) group and a new IAM user, and then add the user to the group. For more information on the IAM service, see the AWS Identity and Access Management User Guide.

Before you run any commands, set your default credentials. For more information, see Configuring the AWS CLI (p. 27).

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

1. Use the `create-group` command to create the group.

   ```bash
   $ aws iam create-group --group-name MyIamGroup
   {
     "Group": {
       "GroupName": "MyIamGroup",
       "CreateDate": "2018-12-14T03:03:52.834Z",
       "GroupId": "AGPAJNUJ2W4IJVEXAMPLE",
       "Arn": "arn:aws:iam::123456789012:group/MyIamGroup",
       "Path": "/"
     }
   }
   ```

2. Use the `create-user` command to create the user.

   ```bash
   $ aws iam create-user --user-name MyUser
   {
     "User": {
       "UserName": "MyUser",
       "Path": "/",
       "CreateDate": "2018-12-14T03:13:02.581Z",
       "UserId": "AIDAJY2PE5XUZ4EXAMPLE",
       "Arn": "arn:aws:iam::123456789012:user/MyUser"
     }
   }
   ```

3. Use the `add-user-to-group` command to add the user to the group.

   ```bash
   $ 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.

   ```bash
   $ aws iam get-group --group-name MyIamGroup
   {
     "Group": {
       "GroupName": "MyIamGroup",
       "CreateDate": "2018-12-14T03:03:52Z",
       "GroupId": "AGPAJNUJ2W4IJVEXAMPLE",
       "Arn": "arn:aws:iam::123456789012:group/MyIamGroup",
       "Path": "/"
     },
     "Users": [
       {
         "UserName": "MyUser",
         "Path": "/",
         "CreateDate": "2018-12-14T03:13:02Z",
         "UserId": "AIDAJY2PE5XUZ4EXAMPLE",
         "Arn": "arn:aws:iam::123456789012:user/MyUser"
       }
     ]
   }
   ```
Attaching an IAM managed policy to an IAM user

This topic describes how to use AWS Command Line Interface (AWS CLI) commands to attach an AWS Identity and Access Management (IAM) policy to an IAM user. The policy in this example provides the user with "Power User Access". For more information on the IAM service, see the AWS Identity and Access Management User Guide.

Before you run any commands, set your default credentials. For more information, see Configuring the AWS CLI (p. 27).

To attach an IAM managed policy to an IAM user

1. Determine the Amazon Resource Name (ARN) of the policy to attach. The following command uses list-policies to find the ARN of the policy with the name PowerUserAccess. It then stores that ARN in an environment variable.

   ```bash
   $ export POLICYARN=$(aws iam list-policies --query 'Policies[? PolicyName=='PowerUserAccess'].{ARN:Arn}' --output text) ~
   $ echo $POLICYARN
   arn:aws:iam::aws:policy/PowerUserAccess
   ```

2. To attach the policy, use the `attach-user-policy` command, and reference the environment variable that holds the policy ARN.

   ```bash
   $ aws iam attach-user-policy --user-name MyUser --policy-arn $POLICYARN
   ```

3. Verify that the policy is attached to the user by running the `list-attached-user-policies` command.

   ```bash
   $ aws iam list-attached-user-policies --user-name MyUser
   {  
     "AttachedPolicies": [  
       {  
         "PolicyName": "PowerUserAccess",  
         "PolicyArn": "arn:aws:iam::aws:policy/PowerUserAccess"
       }
     ]
   }
   ```

For more information, see Access Management Resources. This topic provides links to an overview of permissions and policies, and links to examples of policies for accessing Amazon S3, Amazon EC2, and other services.

Setting an initial password for an IAM user

This topic describes how to use AWS Command Line Interface (AWS CLI) commands to set an initial password for an AWS Identity and Access Management (IAM) user. For more information on the IAM service, see the AWS Identity and Access Management User Guide.

Before you run any commands, set your default credentials. For more information, see Configuring the AWS CLI (p. 27).
Create an access key for an IAM user

This topic describes how to use AWS Command Line Interface (AWS CLI) commands to create a set of access keys for an AWS Identity and Access Management (IAM) user. For more information on the IAM service, see the AWS Identity and Access Management User Guide.

Before you run any commands, set your default credentials. For more information, see Configuring the AWS CLI (p. 27).

You can use the create-access-key command to create an access key for an IAM user. An access key is a set of security credentials that consists of an access key ID and a secret key.

An IAM user can create only two access keys at one time. If you try to create a third set, the command returns a LimitExceeded error.

```
$ aws iam create-access-key --user-name MyUser
{
  "AccessKey": {
    "UserName": "MyUser",
    "AccessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "Status": "Active",
    "SecretAccessKey": "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY",
    "CreateDate": "2018-12-14T17:34:16Z"
  }
}
```

Use the delete-access-key command to delete an access key for an IAM user. Specify which access key to delete by using the access key ID.

```
$ aws iam delete-access-key --user-name MyUser --access-key-id AKIAIOSFODNN7EXAMPLE
```

Using Amazon S3 with the AWS CLI

An introduction to Amazon Simple Storage Service (Amazon S3)

Introduction to Amazon Simple Storage Service (Amazon S3 - Cloud Storage on AWS)
You can access the features of Amazon Simple Storage Service (Amazon S3) using the AWS Command Line Interface (AWS CLI). The AWS CLI provides two tiers of commands for accessing Amazon S3:

- **s3** – High-level commands that simplify performing common tasks, such as creating, manipulating, and deleting objects and buckets.
- **s3api** – Exposes direct access to all Amazon S3 API operations which enables you to carry out advanced operations.

**Topics in this guide:**
- Using high-level (s3) commands with the AWS CLI (p. 142)
- Using API-Level (s3api) commands with the AWS CLI (p. 150)
- Amazon S3 bucket lifecycle operations scripting example (p. 152)

**Using high-level (s3) commands with the AWS CLI**

This topic describes some of the commands you can use to manage Amazon S3 buckets and objects using the `aws s3` commands in the AWS CLI. For commands not covered in this topic and additional command examples, see the `aws s3` commands in the AWS CLI Reference.

The high-level `aws s3` commands simplify managing Amazon S3 objects. These commands enable you to manage the contents of Amazon S3 within itself and with local directories.

**Topics**
- Prerequisites (p. 142)
- Before you start (p. 143)
- Create a bucket (p. 143)
- List buckets and objects (p. 143)
- Delete buckets (p. 144)
- Delete objects (p. 144)
- Move objects (p. 145)
- Copy objects (p. 145)
- Sync objects (p. 146)
- Frequently used options for s3 commands (p. 148)
- Resources (p. 149)

**Prerequisites**

To run the `s3` commands, you need to:

- AWS CLI installed, see *Installing the AWS CLI* (p. 4) for more information.
- AWS CLI configured, see *Configuration basics* (p. 27) for more information. The profile that you use must have permissions that allow the AWS operations performed by the examples.
- Understand these Amazon S3 terms:
  - **Bucket** – A top-level Amazon S3 folder.
  - **Prefix** – An Amazon S3 folder in a bucket.
  - **Object** – Any item that's hosted in an Amazon S3 bucket.
Before you start

This section describes a few things to note before you use `aws s3` commands.

Large object uploads

When you use `aws s3` commands to upload large objects to an Amazon S3 bucket, the AWS CLI automatically performs a multipart upload. You can't resume a failed upload when using these `aws s3` commands.

If the multipart upload fails due to a timeout, or if you manually canceled in the AWS CLI, the AWS CLI stops the upload and cleans up any files that were created. This process can take several minutes.

If the multipart upload or cleanup process is canceled by a kill command or system failure, the created files remain in the Amazon S3 bucket. To clean up the multipart upload, use the `s3api abort-multipart-upload` command.

File properties and tags in multipart copies

When you use the AWS CLI version 1 version of commands in the `aws s3` namespace to copy a file from one Amazon S3 bucket location to another Amazon S3 bucket location, and that operation uses multipart copy, no file properties from the source object are copied to the destination object.

Create a bucket

Use the `s3 mb` command to make a bucket. Bucket names must be *globally* unique (unique across all of Amazon S3) and should be DNS compliant.

Bucket names can contain lowercase letters, numbers, hyphens, and periods. Bucket names can start and end only with a letter or number, and cannot contain a period next to a hyphen or another period.

Syntax

```
$ aws s3 mb <target> [--options]
```

`s3 mb examples`

The following example creates the `s3://bucket-name` bucket.

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

List buckets and objects

To list your buckets, folders, or objects, use the `s3 ls` command. Using the command without a target or options lists all buckets.

Syntax

```
$ aws s3 ls <target> [--options]
```

For a few common options to use with this command, and examples, see Frequently used options for `s3` commands (p. 148). For a complete list of available options, see `s3 ls` in the AWS CLI Command Reference.

`s3 ls examples`

The following example lists all of your Amazon S3 buckets.
The following command lists all objects and prefixes in a bucket. In this example output, the prefix `example/` has one file named `MyFile1.txt`.

```bash
$ aws s3 ls s3://bucket-name
PRE example/ 2018-12-04 19:05:48          3 MyFile1.txt
```

You can filter the output to a specific prefix by including it in the command. The following command lists the objects in `bucket-name/example/` (that is, objects in `bucket-name` filtered by the prefix `example/`).

```bash
$ aws s3 ls s3://bucket-name/example/ 2018-12-06 18:59:32          3 MyFile1.txt
```

### Delete buckets

To delete a bucket, use the `s3 rb` command.

**Syntax**

```bash
$ aws s3 rb <target> [--options]
```

**s3 rb examples**

The following example removes the `s3://bucket-name` bucket.

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

By default, the bucket must be empty for the operation to succeed. To remove a bucket that's not empty, you need to include the `--force` option. If you're using a versioned bucket that contains previously deleted—but retained—objects, this command does not allow you to remove the bucket. You must first remove all of the content.

The following example deletes all objects and prefixes in the bucket, and then deletes the bucket.

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

### Delete objects

To delete objects in a bucket or your local directory, use the `s3 rm` command.

**Syntax**

```bash
$ aws s3 rm <target> [--options]
```

For a few common options to use with this command, and examples, see [Frequently used options for s3 commands](p. 148). For a complete list of options, see `s3 rm` in the *AWS CLI Command Reference*.

**s3 rm examples**

The following example deletes `filename.txt` from `s3://bucket-name/example`. 
$ aws s3 rm s3://bucket-name/example/filename.txt --recursive

The following example deletes all objects from s3://bucket-name/example using the --recursive option.

$ aws s3 rm s3://bucket-name/example --recursive

### Move objects

Use the `s3 mv` command to move objects from a bucket or a local directory.

**Syntax**

```
$ aws s3 mv <source> <target> [--options]
```

For a few common options to use with this command, and examples, see Frequently used options for `s3` commands (p. 148). For a complete list of available options, see `s3 mv` in the AWS CLI Command Reference.

**s3 mv examples**

The following example moves all objects from s3://bucket-name/example to s3://my-bucket/.

```
$ aws s3 mv s3://bucket-name/example s3://my-bucket/
```

The following example moves a local file from your current working directory to the Amazon S3 bucket with the `s3 mv` command.

```
$ aws s3 mv filename.txt s3://bucket-name
```

The following example moves a file from your Amazon S3 bucket to your current working directory, where ./ specifies your current working directory.

```
$ aws s3 mv s3://bucket-name/filename.txt ./
```

### Copy objects

Use the `s3 cp` command to copy objects from a bucket or a local directory.

**Syntax**

```
$ aws s3 cp <source> <target> [--options]
```

You can use the dash parameter for file streaming to standard input (`stdin`) or standard output (`stdout`).

**Warning**

If you're using PowerShell, the shell might alter the encoding of a CRLF or add a CRLF to piped input or output, or redirected output.

The `s3 cp` command uses the following syntax to upload a file stream from `stdin` to a specified bucket.

**Syntax**

```
$ aws s3 cp - <target> [--options]
```
The `s3 cp` command uses the following syntax to download an Amazon S3 file stream for `stdout`.

**Syntax**

```
$ aws s3 cp <target> [--options] -
```

For a few common options to use with this command, and examples, see Frequently used options for s3 commands (p. 148). For the complete list of options, see `s3 cp` in the AWS CLI Command Reference.

**s3 cp examples**

The following example copies all objects from s3://bucket-name/example to s3://my-bucket/.

```
$ aws s3 cp s3://bucket-name/example s3://my-bucket/
```

The following example copies a local file from your current working directory to the Amazon S3 bucket with the `s3 cp` command.

```
$ aws s3 cp filename.txt s3://bucket-name
```

The following example copies a file from your Amazon S3 bucket to your current working directory, where `./` specifies your current working directory.

```
$ aws s3 cp s3://bucket-name/filename.txt ./
```

The following example uses echo to stream the text "hello world" to the s3://bucket-name/filename.txt file.

```
$ echo "hello world" | aws s3 cp - s3://bucket-name/filename.txt
```

The following example streams the contents of s3://bucket-name/filename.txt file to `stdout` and prints the contents to the console.

```
$ aws s3 cp s3://bucket-name/filename.txt -
hello world
```

The following example streams the contents of s3://bucket-name/pre to `stdout`, uses the `bzip2` command to compress the files, and uploads the new compressed file named `key.bz2` to s3://bucket-name.

```
$ aws s3 cp s3://bucket-name/pre - | bzip2 --best | aws s3 cp - s3://bucket-name/key.bz2
```

**Sync objects**

The `s3 sync` command synchronizes the contents of a bucket and a directory, or the contents of two buckets. Typically, `s3 sync` copies missing or outdated files or objects between the source and target. However, you can also supply the `--delete` option to remove files or objects from the target that are not present in the source.

**Syntax**

```
$ aws s3 sync <source> <target> [--options]
```

For a few common options to use with this command, and examples, see Frequently used options for s3 commands (p. 148). For a complete list of options, see `s3 sync` in the AWS CLI Command Reference.
s3 sync examples

The following example synchronizes the contents of an Amazon S3 prefix named path in the bucket named my-bucket with the current working directory.

s3 sync updates any files that have a size or modified time that are different from 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
```

The following example, which extends the previous one, shows how to use the --delete option.

```
// 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
```

When using the --delete option, the --exclude and --include options can filter files or objects to delete during an s3 sync operation. In this case, the parameter string must specify files to exclude from, or include for, deletion in the context of the target directory or bucket. The following shows an example.

```
Assume local directory and s3://my-bucket/path currently in sync and each contains 3 files: MyFile1.txt MyFile2.rtf MyFile88.txt

// 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 "path/MyFile?.txt"
delete: s3://my-bucket/path/MyFile88.txt

// 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
```

147
Frequently used options for s3 commands

The following options are frequently used for the commands described in this topic. For a complete list of options you can use on a command, see the specific command in the AWS CLI reference guide.

**acl**

`s3 sync` and `s3 cp` can use the `--acl` option. This enables you to set the access permissions for files copied to Amazon S3. The `--acl` option accepts `private`, `public-read`, and `public-read-write` values. For more information, see Canned ACL in the Amazon Simple Storage Service User Guide.

```bash
$ aws s3 sync . s3://my-bucket/path --acl public-read
```

**exclude**

When you use the `s3 cp`, `s3 mv`, `s3 sync`, or `s3 rm` command, you can filter the results by using the `--exclude` or `--include` option. The `--exclude` option sets rules to only exclude objects from the command, and the options apply in the order specified. This is shown in the following example.

Local directory contains 3 files:
MyFile1.txt
MyFile2.rtf
MyFile88.txt

// Exclude all .txt files, resulting in only MyFile2.rtf being copied
$ aws s3 cp . s3://my-bucket/path --exclude "*.txt"

// Exclude all .txt files but include all files with the "MyFile*.txt" format, resulting in, MyFile1.txt, MyFile2.rtf, MyFile88.txt being copied
$ aws s3 cp . s3://my-bucket/path --exclude "*.txt" --include "MyFile*.txt"

// Exclude all .txt files, but include all files with the "MyFile*.txt" format, but exclude all files with the "MyFile?.txt" format resulting in, MyFile2.rtf and MyFile88.txt being copied
$ aws s3 cp . s3://my-bucket/path --exclude "*.txt" --include "MyFile*.txt" --exclude "MyFile?.txt"

**include**

When you use the `s3 cp`, `s3 mv`, `s3 sync`, or `s3 rm` command, you can filter the results using the `--exclude` or `--include` option. The `--include` option sets rules to only include objects specified for the command, and the options apply in the order specified. This is shown in the following example.

Local directory contains 3 files:
MyFile1.txt
MyFile2.rtf
MyFile88.txt

// Include all .txt files, resulting in MyFile1.txt and MyFile88.txt being copied
$ aws s3 cp . s3://my-bucket/path --include "*.txt"

// Include all .txt files but exclude all files with the "MyFile*.txt" format, resulting in no files being copied
$ aws s3 cp . s3://my-bucket/path --include "*.txt" --exclude "MyFile*.txt"
// Include all .txt files, but exclude all files with the "MyFile*.txt" format, but 
// include all files with the "MyFile?.txt" format resulting in MyFile1.txt being copied

$ aws s3 cp . s3://my-bucket/path --include "*.txt" --exclude "MyFile*.txt" --include 
// "MyFile?.txt"

grant

The s3 cp, s3 mv, and s3 sync commands include a --grants option that you can use to 
grant permissions on the object to specified users or groups. Set the --grants option to a list of 
permissions using the following syntax. Replace Permission, Grantee_Type, and Grantee_ID 
with your own values.

Syntax

```
--grants Permission=Grantee_Type=Grantee_ID
 [Permission=Grantee_Type=Grantee_ID ...]
```

Each value contains the following elements:

- **Permission** – Specifies the granted permissions. Can be set to read, readacl, writeacl, or 
  full.
- **Grantee_Type** – Specifies how to identify the grantee. 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 about Amazon S3 access control, see Access control.

The following example copies an object into a bucket. It grants read permissions on the object to 
everyone, and full permissions (read, readacl, and writeacl) to the account associated with 
user@example.com.

```
$ aws s3 cp file.txt s3://my-bucket/ --grants read=uri=http://acs.amazonaws.com/groups/
global/AllUsers full=emailaddress=user@example.com
```

You can also specify a nondefault storage class (REDUCED_REDUNDANCY or STANDARD_IA) for 
objects that you upload to Amazon S3. To do this, use the --storage-class option.

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

recursive

When you use this option, the command is performed on all files or objects under the specified 
directory or prefix. The following example deletes s3://my-bucket/path and all of its contents.

```
$ aws s3 rm s3://my-bucket/path --recursive
```

Resources

AWS CLI reference:

- aws s3
- aws s3 cp
- aws s3 mb
Using API-Level (s3api) commands with the AWS CLI

The API-level commands (contained in the s3api command set) provide direct access to the Amazon Simple Storage Service (Amazon S3) APIs, and enable some operations that are not exposed in the high-level s3 commands. These commands are the equivalent of the other AWS services that provide API-level access to the services' functionality. For more information on the s3 commands, see Using high-level (s3) commands with the AWS CLI (p. 142).

This topic provides examples that demonstrate how to use the lower-level commands that map to the Amazon S3 APIs. In addition, you can find examples for each S3 API command in the s3api section of the AWS CLI reference guide.

Prerequisites

To run the s3api commands, you need to:

- AWS CLI installed, see Installing the AWS CLI (p. 4) for more information.
- AWS CLI configured, see Configuration basics (p. 27) for more information. The profile that you use must have permissions that allow the AWS operations performed by the examples.
- Understand these Amazon S3 terms:
  - Bucket – A top-level Amazon S3 folder.
  - Prefix – An Amazon S3 folder in a bucket.
  - Object – Any item that's hosted in an Amazon S3 bucket.

Apply a custom ACL

With high-level commands, you can use the --acl option to apply predefined access control lists (ACLs) to Amazon S3 objects. But you can't use that command to set bucket-wide ACLs. However, you can do this by using the put-bucket-acl API-level command.
The following example shows how to grant full control to two AWS users (user1@example.com and user2@example.com) and read permission to everyone. The identifier for "everyone" comes from a special URI that you pass as a parameter.

```
$ aws s3api put-bucket-acl --bucket MyBucket --grant-full-control 'emailaddress="user1@example.com",emailaddress="user2@example.com"' --grant-read 'uri="http://acs.amazonaws.com/groups/global/AllUsers"'
```

For details about how to construct the ACLs, see PUT Bucket acl in the Amazon Simple Storage Service API Reference. The s3api ACL commands in the CLI, such as put-bucket-acl, use the same shorthand argument notation.

Configure a logging policy

The API command put-bucket-logging configures a bucket logging policy.

In the following example, the AWS user user@example.com is granted full control over the log files, and all users have read access to them. Notice that the put-bucket-acl command is also required to grant the Amazon S3 log delivery system (specified by a URI) the permissions needed to read and write the logs to the bucket.

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

The logging.json file in the previous command has the following content.

```
{
  "LoggingEnabled": {
    "TargetBucket": "MyBucket",
    "TargetPrefix": "MyBucketLogs/",
    "TargetGrants": [
      {
        "Grantee": {
          "Type": "AmazonCustomerByEmail",
          "EmailAddress": "user@example.com"
        },
        "Permission": "FULL_CONTROL"
      },
      {
        "Grantee": {
          "Type": "Group",
          "URI": "http://acs.amazonaws.com/groups/global/AllUsers"
        },
        "Permission": "READ"
      }
    ]
  }
}
```

Resources

AWS CLI reference:

- `aws s3api`
- `aws s3api put-bucket-acl`
Bucket lifecycle scripting example (s3api)

This topic uses a bash scripting example for Amazon S3 bucket lifecycle operations using the AWS Command Line Interface (AWS CLI). This scripting example uses the `aws s3api` set of commands. Shell scripts are programs designed to run in a command line interface.

**Topics**
- Before you start (p. 152)
- About this example (p. 152)
- Files (p. 153)
- References (p. 153)

**Before you start**

Before you can run any of the below examples, the following things need to be completed.

- AWS CLI installed, see *Installing the AWS CLI* (p. 4) for more information.
- AWS CLI configured, see *Configuration basics* (p. 27) for more information. The profile that you use must have permissions that allow the AWS operations performed by the examples.

- As an AWS best practice, grant this code least privilege, or only the permissions required to perform a task. For more information, see *Grant Least Privilege* in the *IAM User Guide*.
- This code has not been tested in all AWS Regions. Some AWS services are available only in specific Regions. For more information, see *Service Endpoints and Quotas* in the *AWS General Reference Guide*.
- Running this code can result in charges to your AWS account. It is your responsibility to ensure that any resources created by this script are removed when you are done with them.

The Amazon S3 service uses the following terms:

- Bucket — A top level Amazon S3 folder.
- Prefix — An Amazon S3 folder in a bucket.
- Object — Any item hosted in an Amazon S3 bucket.

**About this example**

This example demonstrates how to interact with some of the basic Amazon S3 operations using a set of functions in shell script files. The functions are located in the shell script file named `bucket-
You can call these functions in another file. Each script file contains comments describing each of the functions.

To see the intermediate results of each step, run the script with a `-i` parameter. You can view the current status of the bucket or its contents using the Amazon S3 console. The script only proceeds to the next step when you press `enter` at the prompt.

For the full example and downloadable script files, see Amazon S3 Bucket Lifecycle Operations in the AWS Code Examples Repository on GitHub.

**Files**

The example contains the following files:

*bucket-operations.sh*

This main script file can be sourced from another file. It includes functions that perform the following tasks:
- Creating a bucket and verifying that it exists
- Copying a file from the local computer to a bucket
- Copying a file from one bucket location to a different bucket location
- Listing the contents of a bucket
- Deleting a file from a bucket
- Deleting a bucket

View the code for *bucket-operations.sh* on GitHub.

*test-bucket-operations.sh*

The shell script file *test-bucket-operations.sh* demonstrates how to call the functions by sourcing the *bucket-operations.sh* file and calling each of the functions. After calling functions, the test script removes all resources that it created.

View the code for *test-bucket-operations.sh* on GitHub.

*awsdocs-general.sh*

The script file *awsdocs-general.sh* holds general purpose functions used across advanced code examples for the AWS CLI.

View the code for *awsdocs-general.sh* on GitHub.

**References**

**AWS CLI reference:**

- `aws s3api`
- `aws s3api create-bucket`
- `aws s3api copy-object`
- `aws s3api delete-bucket`
- `aws s3api delete-object`
- `aws s3api head-bucket`
- `aws s3api list-objects`
Using Amazon SNS with the AWS CLI

You can access the features of Amazon Simple Notification Service (Amazon SNS) using the AWS Command Line Interface (AWS CLI). To list the AWS CLI commands for Amazon SNS, use the following command.

```
aws sns help
```

Before you run any commands, set your default credentials. For more information, see Configuring the AWS CLI (p. 27).

This topic shows examples of AWS CLI commands that perform common tasks for Amazon SNS.

Topics
- Create a topic (p. 154)
- Subscribe to a topic (p. 154)
- Publish to a topic (p. 155)
- Unsubscribe from a topic (p. 155)
- Delete a topic (p. 155)

Create a topic

To create a topic, use the `sns create-topic` command and specify the name to assign to the topic.

```
$ aws sns create-topic --name my-topic
{
}
```

Make a note of the response's TopicArn, which you use later to publish a message.

Subscribe to a topic

To subscribe to a topic, use the `sns subscribe` command.

The following example specifies the email protocol and an email address for the notification-endpoint.

```
$ aws sns subscribe --topic-arn arn:aws:sns:us-west-2:123456789012:my-topic --protocol email --notification-endpoint saanvi@example.com
{

```

Other reference:
- Working with Amazon S3 buckets in the Amazon Simple Storage Service User Guide
- Working with Amazon S3 objects in the Amazon Simple Storage Service User Guide
- To view and contribute to AWS SDK and AWS CLI code examples, see the AWS Code Examples Repository on GitHub.
AWS immediately sends a confirmation message by email to the address you specified in the subscribe command. The email message has the following text.

You have chosen to subscribe to the topic:
To confirm this subscription, click or visit the following link (If this was in error no action is necessary):
Confirm subscription

After the recipient clicks the Confirm subscription link, the recipient's browser displays a notification message with information similar to the following.

Subscription confirmed!
You have subscribed saanvi@example.com to the topic:my-topic.
Your subscription's id is:
If it was not your intention to subscribe, click here to unsubscribe.

Publish to a topic

To send a message to all subscribers of a topic, use the sns publish command.

The following example sends the message "Hello World!" to all subscribers of the specified topic.

```bash
```

In this example, AWS sends an email message with the text "Hello World!" to saanvi@example.com.

Unsubscribe from a topic

To unsubscribe from a topic and stop receiving messages published to that topic, use the sns unsubscribe command and specify the ARN of the topic you want to unsubscribe from.

```bash
```

To verify that you successfully unsubscribed, use the sns list-subscriptions command to confirm that the ARN no longer appears in the list.

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

Delete a topic

To delete a topic, run the sns delete-topic command.
To verify that AWS successfully deleted the topic, use the `sns list-topics` command to confirm that the topic no longer appears in the list.

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

### Using Amazon Simple Workflow Service with the AWS CLI

An introduction to Amazon Simple Workflow Service

Amazon Simple Workflow Service

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

To list the AWS CLI commands for Amazon SWF, use the following command.

```
aws swf help
```

Before you run any commands, set your default credentials. For more information, see Configuring the AWS CLI (p. 27).

The following topics show examples of AWS CLI commands that perform common tasks for Amazon SWF.

**Topics**

- List of Amazon SWF commands by category (p. 156)
- Working with Amazon SWF domains using the AWS CLI (p. 159)

### List of Amazon SWF commands by category

You can use the AWS Command Line Interface (AWS CLI) to create, display, and manage workflows in Amazon Simple Workflow Service (Amazon SWF).

This section lists the reference topics for Amazon SWF commands in the AWS CLI, grouped by functional category.

For an alphabetic list of commands, see the Amazon SWF section of the AWS CLI Command Reference, or use the following command.

```
$ aws swf help
```

You can also get help for an individual command, by placing the `help` directive after the command name. The following shows an example.

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

- Commands related to activities (p. 157)
- Commands related to deciders (p. 157)
- Commands related to workflow executions (p. 157)
- Commands related to administration (p. 157)
- Visibility commands (p. 158)

Commands related to activities

Activity workers use `poll-for-activity-task` to get new activity tasks. After a worker receives an activity task from Amazon SWF, it performs the task and responds using `respond-activity-task-completed` if successful or `respond-activity-task-failed` if unsuccessful.

The following are commands that are performed by activity workers:

- `poll-for-activity-task`
- `respond-activity-task-completed`
- `respond-activity-task-failed`
- `respond-activity-task-canceled`
- `record-activity-task-heartbeat`

Commands related to deciders

Deciders use `poll-for-decision-task` to get decision tasks. After a decider receives a decision task from Amazon SWF, it examines its workflow execution history and decides what to do next. It calls `respond-decision-task-completed` to complete the decision task and provides zero or more next decisions.

The following are commands that are performed by deciders:

- `poll-for-decision-task`
- `respond-decision-task-completed`

Commands related to workflow executions

The following commands operate on a workflow execution:

- `request-cancel-workflow-execution`
- `start-workflow-execution`
- `signal-workflow-execution`
- `terminate-workflow-execution`

Commands related to administration

Although you can perform administrative tasks from the Amazon SWF console, you can use the commands in this section to automate functions or build your own administrative tools.

Activity management

- `register-activity-type`
• deprecate-activity-type

Workflow management
• register-workflow-type
• deprecate-workflow-type

Domain management
• register-domain
• deprecate-domain

For more information and examples of these domain management commands, see Working with Amazon SWF domains using the AWS CLI (p. 159).

Workflow execution management
• request-cancel-workflow-execution
• terminate-workflow-execution

Visibility commands
Although you can perform visibility actions from the Amazon SWF console, you can use the commands in this section to build your own console or administrative tools.

Activity visibility
• list-activity-types
• describe-activity-type

Workflow visibility
• list-workflow-types
• describe-workflow-type

Workflow execution visibility
• describe-workflow-execution
• list-open-workflow-executions
• list-closed-workflow-executions
• count-open-workflow-executions
• count-closed-workflow-executions
• get-workflow-execution-history

Domain visibility
• list-domains
• describe-domain
For more information and examples of these domain visibility commands, see Working with Amazon SWF domains using the AWS CLI (p. 159).

Task list visibility

- count-pending-activity-tasks
- count-pending-decision-tasks

Working with Amazon SWF domains using the AWS CLI

You can use the AWS Command Line Interface (AWS CLI) to manage your Amazon Simple Workflow Service (Amazon SWF) domains.

Topics

- List your domains (p. 159)
- Get information about a domain (p. 159)
- Register a domain (p. 160)
- Deprecate a domain (p. 160)

List your domains

To list the Amazon SWF domains that you have registered for your AWS account, you can use `swf list-domains`. You must include `--registration-status` and specify either `REGISTERED` or `DEPRECATED`.

Here's a minimal example.

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

Note
For an example of using `DEPRECATED`, see Deprecate a domain (p. 160).

For more information, see `swf list-domains` in the AWS CLI Command Reference.

Get information about a domain

To get detailed information about a particular domain, use `swf describe-domain`. There is one required parameter, `--name`, which takes the name of the domain you want information about, as shown in the following example.

```
$ aws swf describe-domain --name ExampleDomain
```
Working with Amazon SWF Domains

```
{
  "domainInfo": {
    "status": "REGISTERED",
    "name": "ExampleDomain"
  },
  "configuration": {
    "workflowExecutionRetentionPeriodInDays": "1"
  }
}
```

For more information, see `swf describe-domain` in the *AWS CLI Command Reference*.

### Register a domain

To register new domains, use `swf register-domain`.

There are two required parameters: `--name` and `--workflow-execution-retention-period-in-days`. The `--name` parameter takes the domain name to register. The `--workflow-execution-retention-period-in-days` parameter takes an integer to specify the number of days to retain workflow execution data on this domain, up to a maximum period of 90 days (for more information, see the Amazon SWF FAQ).

If you specify zero (0) for this value, the retention period is automatically set at the maximum duration. Otherwise, workflow execution data isn't retained after the specified number of days have passed. The following example shows how to register a new domain.

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

The command doesn't return any output, but you can use `swf list-domains` or `swf describe-domain` to see the new domain, as shown in the following example.

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

For more information, see `swf register-domain` in the *AWS CLI Command Reference*.

### Deprecate a domain

To deprecate a domain (you can still see it, but cannot create new workflow executions or register types on it), use `swf deprecate-domain`. It has a sole required parameter, `--name`, which takes the name of the domain to deprecate.

```
$ aws swf deprecate-domain --name MyNeatNewDomain
```

As with `register-domain`, no output is returned. If you use `list-domains` to view the registered domains, however, you will see that the domain no longer appears among them. You can also use `--registration-status DEPRECATED`.

```
$ aws swf list-domains --registration-status DEPRECATED
```

160
For more information, see deprecate-domain in the AWS CLI Command Reference.
Security in the AWS Command Line Interface

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from a data center and network architecture that is built to meet the requirements of the most security-sensitive organizations.

Security is a shared responsibility between AWS and you. The shared responsibility model describes this as security of the cloud and security in the cloud:

- **Security of the cloud** – AWS is responsible for protecting the infrastructure that runs AWS services in the AWS Cloud. AWS also provides you with services that you can use securely. Third-party auditors regularly test and verify the effectiveness of our security as part of the AWS Compliance Programs. To learn about the compliance programs that apply to AWS Command Line Interface, see AWS Services in Scope by Compliance Program.

- **Security in the cloud** – Your responsibility is determined by the AWS service that you use. You are also responsible for other factors including the sensitivity of your data, your company’s requirements, and applicable laws and regulations.

This documentation helps you understand how to apply the shared responsibility model when using the AWS Command Line Interface (AWS CLI). The following topics show you how to configure the AWS CLI to meet your security and compliance objectives. You also learn how to use the AWS CLI to help you to monitor and secure your AWS resources.

Topics
- Data protection in the AWS CLI (p. 162)
- Identity and Access Management for the AWS CLI (p. 163)
- Compliance validation for the AWS CLI (p. 164)
- Enforcing a minimum version of TLS (p. 164)

Data protection in the AWS CLI

The AWS shared responsibility model applies to data protection in AWS Command Line Interface. As described in this model, AWS is responsible for protecting the global infrastructure that runs all of the AWS Cloud. You are responsible for maintaining control over your content that is hosted on this infrastructure. This content includes the security configuration and management tasks for the AWS services that you use. For more information about data privacy, see the Data Privacy FAQ. For information about data protection in Europe, see the AWS Shared Responsibility Model and GDPR blog post on the AWS Security Blog.

For data protection purposes, we recommend that you protect AWS account credentials and set up individual user accounts with AWS Identity and Access Management (IAM). That way each user is given only the permissions necessary to fulfill their job duties. We also recommend that you secure your data in the following ways:

- Use multi-factor authentication (MFA) with each account.
- Use SSL/TLS to communicate with AWS resources. We recommend TLS 1.2 or later.
- Set up API and user activity logging with AWS CloudTrail.
• Use AWS encryption solutions, along with all default security controls within AWS services.
• Use advanced managed security services such as Amazon Macie, which assists in discovering and securing personal data that is stored in Amazon S3.
• If you require FIPS 140-2 validated cryptographic modules when accessing AWS through a command line interface or an API, use a FIPS endpoint. For more information about the available FIPS endpoints, see Federal Information Processing Standard (FIPS) 140-2.

We strongly recommend that you never put confidential or sensitive information, such as your customers' email addresses, into tags or free-form fields such as a Name field. This includes when you work with AWS CLI or other AWS services using the console, API, AWS CLI, or AWS SDKs. Any data that you enter into tags or free-form fields used for names may be used for billing or diagnostic logs. If you provide a URL to an external server, we strongly recommend that you do not include credentials information in the URL to validate your request to that server.

Data encryption

A key feature of any secure service is that information is encrypted when it is not being actively used.

Encryption at rest

The AWS CLI does not itself store any customer data other than the credentials it needs to interact with the AWS services on the user's behalf.

If you use the AWS CLI to invoke an AWS service that transmits customer data to your local computer for storage, then refer to the Security & Compliance chapter in that service's User Guide for information on how that data is stored, protected, and encrypted.

Encryption in transit

By default, all data transmitted from the client computer running the AWS CLI and AWS service endpoints is encrypted by sending everything through a HTTPS/TLS connection.

You don't need to do anything to enable the use of HTTPS/TLS. It is always enabled unless you explicitly disable it for an individual command by using the --no-verify-ssl command line option.

Identity and Access Management for the AWS CLI

The AWS Command Line Interface (AWS CLI) uses the same users and roles to access your AWS resources and their services. The policies that grant permissions are the same because the AWS CLI calls the same API operations that are used by the service console. For more information, see the "Identity and Access Management" section in the "Security" chapter of the AWS service that you want to use.

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

The credentials used by the AWS CLI are stored in plaintext files and are not encrypted.

• The $HOME/.aws/credentials file stores long-term credentials required to access your AWS resources. These include your access key ID and secret access key.
• Short-term credentials, such as those for roles that you assume, or that are for AWS IAM Identity Center (successor to AWS Single Sign-On) services, are also stored in the $HOME/.aws/cli/cache and $HOME/aws/sso/cache folders, respectively.
Mitigation of Risk

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

Compliance validation for the AWS CLI

Third-party auditors assess the security and compliance of AWS services as part of multiple AWS compliance programs. Using the AWS Command Line Interface (AWS CLI) to access a service does not alter that service's compliance.

For a list of AWS services in scope of specific compliance programs, see AWS Services in Scope by Compliance Program. For general information, see AWS Compliance Programs.

You can download third-party audit reports using the AWS Artifact. For more information, see Downloading Reports in AWS Artifact.

Your compliance responsibility when using AWS CLI is determined by the sensitivity of your data, your company’s compliance objectives, and applicable laws and regulations. AWS provides the following resources to help with compliance:

- Security and Compliance Quick Start Guides – These deployment guides discuss architectural considerations and provide steps for deploying security- and compliance-focused baseline environments on AWS.
- Architecting for HIPAA Security and Compliance Whitepaper – This whitepaper describes how companies can use AWS to create HIPAA-compliant applications.
- AWS Compliance Resources – This collection of workbooks and guides might apply to your industry and location.
- Evaluating Resources with Rules in the AWS Config Developer Guide – The AWS Config service assesses how well your resource configurations comply with internal practices, industry guidelines, and regulations.
- AWS Security Hub – This AWS service provides a comprehensive view of your security state within AWS that helps you check your compliance with security industry standards and best practices.

Enforcing a minimum version of TLS

To add increased security when communicating with AWS services, you should use TLS 1.2 or later. When you use the AWS CLI, Python is used to set the TLS version.

To ensure the AWS CLI version 1 uses no TLS version earlier than TLS 1.2, you might need to recompile OpenSSL to enforce this minimum and then recompile Python to use the newly built OpenSSL.

Topics

- Determine your currently supported protocols (p. 164)
- Compile OpenSSL and Python (p. 166)

Determine your currently supported protocols

First, create a self-signed certificate to use for the test server and the Python SDK using OpenSSL.
Enforcing a minimum TLS version

$ openssl req -subj '/CN=localhost' -x509 -newkey rsa:4096 -nodes -keyout key.pem -out cert.pem -days 365

Then spin up a test server using OpenSSL.

$ openssl s_server -key key.pem -cert cert.pem -www

In a new terminal window, create a virtual environment and install the SDK for Python.

$ python3 -m venv test-env
source test-env/bin/activate
pip install botocore

Create a new Python script named check.py that uses the SDK's underlying HTTP library.

```
$ import urllib3
URL = 'https://localhost:4433/'
http = urllib3.PoolManager(
    ca_certs='cert.pem',
    cert_reqs='CERT_REQUIRED',
)
r = http.request('GET', URL)
print(r.data.decode('utf-8'))
```

Run your new script.

$ python check.py

This displays details about the connection made. Search for "Protocol : " in the output. If the output is "TLSv1.2" or later, the SDK defaults to TLS v1.2 or later. If it's an earlier version, you need to recompile OpenSSL and recompile Python.

However, even if your installation of Python defaults to TLS v1.2 or later, it's still possible for Python to renegotiate to a version earlier than TLS v1.2 if the server doesn't support TLS v1.2 or later. To check that Python doesn't automatically renegotiate to earlier versions, restart the test server with the following.

$ openssl s_server -key key.pem -cert cert.pem -no_tls1_3 -no_tls1_2 -www

If you're using an earlier version of OpenSSL, you might not have the -no_tls_3 flag available. If this is the case, remove the flag because the version of OpenSSL you're using doesn't support TLS v1.3. Then rerun the Python script.

$ python check.py

If your installation of Python correctly doesn't renegotiate for versions earlier than TLS 1.2, you should receive an SSL error.

```

If you're able to make a connection, you need to recompile OpenSSL and Python to disable negotiation of protocols earlier than TLS v1.2.
Enforcing a minimum TLS version

Compile OpenSSL and Python

To ensure the SDK or AWS CLI doesn't negotiate for anything earlier than TLS 1.2, you need to recompile OpenSSL and Python. To do this, copy the following content to create a script and run it.

```bash
#!/usr/bin/env bash
set -e

OPENSSL_VERSION="1.1.1d"
OPENSSL_PREFIX="/opt/openssl-with-min-tls1_2"
PYTHON_VERSION="3.8.1"
PYTHON_PREFIX="/opt/python-with-min-tls1_2"

curl -O "https://www.openssl.org/source/openssl-$OPENSSL_VERSION.tar.gz"
tar -xzf "openssl-$OPENSSL_VERSION.tar.gz"
cd openssl-$OPENSSL_VERSION
./config --prefix=$OPENSSL_PREFIX no-ssl3 no-tls1 no-tls1_1 no-shared
make > /dev/null
sudo make install_sw > /dev/null

cd /tmp
curl -O "https://www.python.org/ftp/python/$PYTHON_VERSION/Python-$PYTHON_VERSION.tgz"
tar -xzf "Python-$PYTHON_VERSION.tgz"
cd Python-$PYTHON_VERSION
./configure --prefix=$PYTHON_PREFIX --with-openssl=$OPENSSL_PREFIX --disable-shared > /dev/null
make > /dev/null
sudo make install > /dev/null
```

This compiles a version of Python that has a statically linked OpenSSL that doesn't automatically negotiate anything earlier than TLS 1.2. This also installs OpenSSL in the `/opt/openssl-with-min-tls1_2` directory and installs Python in the `/opt/python-with-min-tls1_2` directory. After you run this script, confirm installation of the new version of Python.

```bash
$ /opt/python-with-min-tls1_2/bin/python3 --version
```

This should print out the following.

```
Python 3.8.1
```

To confirm this new version of Python doesn't negotiate a version earlier than TLS 1.2, rerun the steps from Determine your currently supported protocols (p. 164) using the newly installed Python version (that is, `/opt/python-with-min-tls1_2/bin/python3`).
Troubleshooting AWS CLI errors

This section covers common errors and troubleshooting steps to follow to resolve your issue. We suggest following the general troubleshooting (p. 167) first.

Contents

• General troubleshooting to try first (p. 167)
  • Check your AWS CLI command formatting (p. 167)
  • Confirm that you’re running a recent version of the AWS CLI (p. 168)
  • Use the --debug option (p. 168)
  • Enable and review the AWS CLI command history logs (p. 172)
  • Confirm that your AWS CLI is configured (p. 172)
• Command not found errors (p. 172)
• The "aws --version" command returns a different version than you installed (p. 174)
• The "aws --version" command returns a version after uninstalling the AWS CLI (p. 175)
• Access denied errors (p. 176)
• Invalid credentials and key errors (p. 176)
• Signature does not match errors (p. 177)
• No Windows console found errors (p. 178)
• SSL certificate errors (p. 179)
• Invalid JSON errors (p. 179)
• Additional resources (p. 181)

General troubleshooting to try first

If you receive an error or encounter an issue with the AWS CLI, we suggest the following general tips to help you troubleshoot.

Check your AWS CLI command formatting

If you receive an error that indicates that a command doesn't exist, or that it doesn't recognize a parameter (Parameter validation failed) that the documentation says is available, then your command might be formatted incorrectly. We suggest that you check the following:

• Check your command for spelling and formatting errors.
• Confirm all quotes and escaping appropriate for your terminal (p. 78) is correct in your command.
• Generate an AWS CLI skeleton (p. 83) to confirm your command structure.
• For JSON, see the additional troubleshooting for JSON values (p. 179). If you're having issues with your terminal processing JSON formatting, we suggest skipping past the terminal's quoting rules by using Blobs to pass JSON data directly to the AWS CLI (p. 76).

For more information on how a specific command should be structured, see the AWS CLI reference guide.
Confirm that you're running a recent version of the AWS CLI

If you receive an error that indicates that a command doesn't exist, or that it doesn't recognize a parameter that the AWS CLI reference guide says is available, first confirm that your command is correctly formatted. If the formatting is correct, then we recommend that you upgrade to the most recent version of the AWS CLI. Updated versions of the AWS CLI are released almost every business day. New AWS services, features, and parameters are introduced in those new versions of the AWS CLI. The only way to get access to those new services, features, or parameters is to upgrade to a version that was released after that element was first introduced.

How you update your version of the AWS CLI depends on how you originally installed it as described in Installing the AWS CLI (p. 4).

If you used one of the bundled installers, you might need to remove the existing installation before you download and install the latest version for your operating system.

Back to top (p. 167)

Use the --debug option

When the AWS CLI reports an error that you don't immediately understand, or produces results that you don't expect, you can get more detail about the error by running the command again with the --debug option. With this option, the AWS CLI outputs details about every step it takes to process your command. The details in the output can help you to determine when the error occurs and provides clues about where it started.

You can send the output to a text file for later review, or to send to AWS Support when asked for it.

When you include the --debug option, some of the details include:

- Looking for credentials
- Parsing the provided parameters
- Constructing the request sent to AWS servers
- The contents of the request sent to AWS
- The contents of the raw response
- The formatted output

Here's an example of a command run with and without the --debug option.

```bash
$ aws iam list-groups --profile MyTestProfile
{
    "Groups": [
    
        {
            "Path": "/",
            "GroupName": "MyTestGroup",
            "GroupId": "AGPA0123456789EXAMPLE",
            "Arn": "arn:aws:iam::123456789012:group/MyTestGroup",
            "CreateDate": "2019-08-12T19:34:04Z"
        }
    ]
}

$ aws iam list-groups --profile MyTestProfile --debug
```
Use the `--debug` option

```
2019-08-12 12:36:18,305 - MainThread - awscli.clidriver - DEBUG - Arguments entered to CLI: ['iam', 'list-groups', '--debug']
2019-08-12 12:36:18,305 - MainThread - botocore.hooks - DEBUG - Event session-initialized: calling handler <function add_scalar_parsers at 0x7fdf17316e0>
2019-08-12 12:36:18,305 - MainThread - botocore.hooks - DEBUG - Event session-initialized: calling handler <function register_uri_param_handler at 0x7fdf17dec40>
2019-08-12 12:36:18,305 - MainThread - botocore.hooks - DEBUG - Event session-initialized: calling handler <function inject_assume_role_provider_cache at 0x7fdf17da9378>
2019-08-12 12:36:18,305 - MainThread - botocore.credentials - DEBUG - Skipping environment variable credential check because profile name was explicitly set.
2019-08-12 12:36:18,306 - MainThread - botocore.hooks - DEBUG - Event session-initialized: calling handler <function attach_history_handler at 0x7fdf173ed9d8>
2019-08-12 12:36:18,317 - MainThread - botocore.hooks - DEBUG - Event building-argument-table.iam: calling handler <function add_waiters at 0x7fdf1731a840>
2019-08-12 12:36:18,321 - MainThread - awscli.clidriver - DEBUG - OrderedDict([('path-prefix', <awscli.arguments.CLIArgument object at 0x7fdf171ac780>), ('marker', <awscli.arguments.CLIArgument object at 0x7fdf171b09e8>), ('max-items', <awscli.arguments.CLIArgument object at 0x7fdf171b09b0>)]
2019-08-12 12:36:18,322 - MainThread - botocore.hooks - DEBUG - Event building-argument-table.iam.list-groups: calling handler <function add_streaming_output_arg at 0x7fdf17316510>
2019-08-12 12:36:18,322 - MainThread - botocore.hooks - DEBUG - Event building-argument-table.iam.list-groups: calling handler <function add_cli_input_json at 0x7fdf17da9d90>
2019-08-12 12:36:18,322 - MainThread - botocore.hooks - DEBUG - Event building-argument-table.iam.list-groups: calling handler <function unify_paging_params at 0x7fdf17328048>
2019-08-12 12:36:18,326 - MainThread - awscli.customizations.paginate - DEBUG - Modifying paging parameters for operation: ListGroups
2019-08-12 12:36:18,326 - MainThread - botocore.hooks - DEBUG - Event building-argument-table.iam.list-groups: calling handler <function add_generate_skeleton at 0x7fdf17328158>
2019-08-12 12:36:18,327 - MainThread - botocore.hooks - DEBUG - Event before-building-argument-table-parser.iam.list-groups: calling handler <bound method OverrideRequiredArgsArgument.override_required_args of <awscli.customizations.cliinputjson.CliInputJSONArgument object at 0x7fdf171b0a58>>
2019-08-12 12:36:18,327 - MainThread - botocore.hooks - DEBUG - Event before-building-argument-table-parser.iam.list-groups: calling handler <bound method GenerateCliSkeletonArgument.override_required_args of <awscli.customizations.generatecliskeleton.GenerateCliSkeletonArgument object at 0x7fdf1715978>>
2019-08-12 12:36:18,327 - MainThread - botocore.hooks - DEBUG - Event operation-args-parsed.iam.list-groups: calling handler functiontools.partial(<function check_should_enable-pagination at 0x7fdf17328158>, ['marker', 'max-items'], {'max-items': <awscli.arguments.CLIArgument object at 0x7fdf171b09b0>}, OrderedDict([('path-prefix', <awscli.arguments.CLIArgument object at 0x7fdf171ac780>), ('marker', <awscli.arguments.CLIArgument object at 0x7fdf171b09e8>), ('max-items', <awscli.customizations.paginate.PageArgument object at 0x7fdf171c58d0>), ('cli-input-json', <awscli.customizations.clinputjson.CliInputJSONArgument object at 0x7fdf171b0a58>), ('starting-token', <awscli.customizations.paginate.PageArgument object at 0x7fdf171b0a20>), ('page-size', <awscli.customizations.paginate.PageArgument object at 0x7fdf171c5828>), ('generate-cli-skeleton', <awscli.customizations.generatecliskeleton.GenerateCliSkeletonArgument object at 0x7fdf171c5978>)]))
```

169
Use the `--debug` option

```
2019-08-12 12:36:18,328 - MainThread - botocore.hooks - DEBUG - Event load-cli-arg.iam.list-groups.cli-input-json: calling handler <awscli.paramfile.URIArgumentHandler object at 0x7fdf1725c978>
2019-08-12 12:36:18,328 - MainThread - botocore.hooks - DEBUG - Event load-cli-arg.iam.list-groups.starting-token: calling handler <awscli.paramfile.URIArgumentHandler object at 0x7fdf1725c978>
2019-08-12 12:36:18,328 - MainThread - botocore.hooks - DEBUG - Event load-cli-arg.iam.list-groups.page-size: calling handler <awscli.paramfile.URIArgumentHandler object at 0x7fdf1725c978>
2019-08-12 12:36:18,328 - MainThread - botocore.hooks - DEBUG - Event load-cli-arg.iam.list-groups.generate-cli-skeleton: calling handler <awscli.paramfile.URIArgumentHandler object at 0x7fdf1725c978>
2019-08-12 12:36:18,329 - MainThread - botocore.hooks - DEBUG - Event calling-command.iam.list-groups: calling handler <bound method CliInputJSONArgument.add_to_call_parameters of <awscli.customizations.cliinputjson.CliInputJSONArgument object at 0x7fdf171b0a58>>
2019-08-12 12:36:18,329 - MainThread - botocore.hooks - DEBUG - Event calling-command.iam.list-groups: calling handler <bound method GenerateCliSkeletonArgument.generate_json_skeleton of <awscli.customizations.generatecliskeleton.GenerateCliSkeletonArgument object at 0x7fdf171c5978>>
2019-08-12 12:36:18,329 - MainThread - botocore.credentials - DEBUG - Looking for credentials via: assume-role
2019-08-12 12:36:18,334 - MainThread - botocore.hooks - DEBUG - Event choose-service-name: calling handler <function handle_service_name_alias at 0x7fdf1898eb70>
2019-08-12 12:36:18,337 - MainThread - botocore.hooks - DEBUG - Event creating-client-class.iam: calling handler <function add_generate_presigned_url at 0x7fdf18a028c8>
2019-08-12 12:36:18,337 - MainThread - botocore.regions - DEBUG - Using partition endpoint for iam, us-west-2: aws-global
2019-08-12 12:36:18,337 - MainThread - botocore.args - DEBUG - The s3 config key is not a dictionary type, ignoring its value of: None
2019-08-12 12:36:18,340 - MainThread - botocore.endpoint - DEBUG - Setting iam timeout as (60, 60)
2019-08-12 12:36:18,342 - MainThread - botocore.hooks - DEBUG - Event before-call.iam.ListGroups: calling handler <bound method RequestSigner.handler of <botocore.signers.RequestSigner object at 0x7fdf16e9a4a8>>
2019-08-12 12:36:18,342 - MainThread - botocore.hooks - DEBUG - Event request-created.iam.ListGroups: calling handler <bound method RequestSigner.handler of <botocore.signers.RequestSigner object at 0x7fdf16e9a470>>
2019-08-12 12:36:18,342 - MainThread - botocore.hooks - DEBUG - Event request-created.iam.ListGroups: calling handler <bound method RequestSigner.handler of <botocore.signers.RequestSigner object at 0x7fdf16e9a470>>
2019-08-12 12:36:18,342 - MainThread - botocore.hooks - DEBUG - Event request-created.iam.ListGroups: calling handler <bound method RequestSigner.handler of <botocore.signers.RequestSigner object at 0x7fdf16e9a470>>
2019-08-12 12:36:18,342 - MainThread - botocore.hooks - DEBUG - Event request-created.iam.ListGroups: calling handler <bound method RequestSigner.handler of <botocore.signers.RequestSigner object at 0x7fdf16e9a470>>
2019-08-12 12:36:18,342 - MainThread - botocore.hooks - DEBUG - Event request-created.iam.ListGroups: calling handler <bound method RequestSigner.handler of <botocore.signers.RequestSigner object at 0x7fdf16e9a470>>
2019-08-12 12:36:18,342 - MainThread - botocore.hooks - DEBUG - Event request-created.iam.ListGroups: calling handler <bound method RequestSigner.handler of <botocore.signers.RequestSigner object at 0x7fdf16e9a470>>
```
Use the `--debug` option

```python
2019-08-12 12:36:18,343 - MainThread - botocore.auth - DEBUG - CanonicalRequest:
POST

/content-type:application/x-www-form-urlencoded; charset=utf-8
host:iam.amazonaws.com
x-amz-date:20190812T193618Z

2019-08-12 12:36:18,343 - MainThread - botocore.auth - DEBUG - StringToSign:
AWS4-HMAC-SHA256
20190812T193618Z
20190812/us-east-1/iam/aws4_request
ab7e367eEXAMPLE2769f178ea509978c504874b3EXAMPLE9b8cb5eb5d6d4a787a33ae41c8cd6eEXAMPLEca69080e1e1f

2019-08-12 12:36:18,344 - MainThread - botocore.auth - DEBUG - Signature:
d85a0EXAMPLE41c8cd6eadiddff85a1a3ce7

2019-08-12 12:36:18,344 - MainThread - botocore.endpoint - DEBUG - Sending http request:

<AWSPreparedRequest stream_output=False, method=POST, url=https://iam.amazonaws.com/,
headers={'Content-Type': b'application/x-www-form-urlencoded; charset=utf-8',
'User-Agent': b'aws-cli/1.16.215 Python/3.7.3 Linux/4.14.133-113.105.amzn2.x86_64
botocore/1.12.205', 'X-Amz-Date': b'20190812T193618Z', 'Authorization': b'AWS4-HMAC-SHA256
Credential=AKIA01234567890EXAMPLE-east-1/iam/aws4_request, SignedHeaders=content-type;host;x-amz-date,
Signature=d85a07692aceb401EXAMPLEa1b18adiddff85a1a3ce7EXAMPLE'},

2019-08-12 12:36:18,344 - MainThread - urllib3.util.retry - DEBUG - Converted retries
value: False -> Retry(total=False, connect=None, read=None, redirect=0, status=None)

2019-08-12 12:36:18,344 - MainThread - urllib3.connectionpool - DEBUG - Starting new HTTPS
collection (1): iam.amazonaws.com:443

2019-08-12 12:36:18,664 - MainThread - botocore.parsers - DEBUG - Response headers:

{'x-amzn-RequestId': '74c11606-bd38-11e9-9c82-559da0ad349', 'Content-Type': 'text/xml',
'Content-Length': '570', 'Date': 'Mon, 12 Aug 2019 19:36:18 GMT'}

2019-08-12 12:36:18,664 - MainThread - botocore.parsers - DEBUG - Response body:

  <ListGroupsResult>
    <IsTruncated>false</IsTruncated>
    <Groups>
      <member>
        <Path>/</Path>
        <GroupName>MyTestGroup</GroupName>
        <Arn>arn:aws:iam::123456789012:group/MyTestGroup</Arn>
        <GroupId>AGPA1234567890EXAMPLE</GroupId>
        <CreateDate>2019-08-12T19:34:04Z</CreateDate>
      </member>
    </Groups>
  </ListGroupsResult>
</ListGroupsResponse>
'

2019-08-12 12:36:18,665 - MainThread - botocore.retryhandler - DEBUG - No retry needed.

2019-08-12 12:36:18,665 - MainThread - botocore.hooks - DEBUG - Event

```
Enable and review the AWS CLI command history logs

You can enable the AWS CLI command history logs using the `cli_history` (p. 34) file setting. After enabling this setting, the AWS CLI records the history of `aws` commands.

You can list your history using the `aws history list` command, and use the resulting command_ids in the `aws history show` command for details. For more information see `aws history` in the AWS CLI reference guide.

When you include the `--debug` option, some of the details include:

- API calls made to botocore
- Status codes
- HTTP responses
- Headers
- Return codes

You can use this information to confirm parameter data and API calls are behaving in the way you expect, and can then deduce at what step in the process your command is failing.

Confirm that your AWS CLI is configured

Various errors can occur if your `config` and `credentials` files or your IAM user or roles are not configured correctly. For more information on resolving errors with `config` and `credentials` files or your IAM user or roles, see the section called “Access denied errors” (p. 176) and the section called “Invalid credentials and key errors” (p. 176).

Command not found errors

This error means that the operating system can't find the AWS CLI command. The installation might be incomplete or requires updating.

Possible cause: You're trying to use an AWS CLI feature newer than your installed version, or have incorrect formatting

Example error text:

```
$ aws s3 copy
usage: aws [options] <command> <subcommand> [...]
To see help text, you can run:

  aws help
  aws <command> help
  aws <command> <subcommand> help
aws: error: argument subcommand: Invalid choice, valid choices are:
  ls           | website
```
Various errors can occur if your command is formatted incorrectly or you are using an earlier version
from before the feature was released. For more information on resolving errors around these two
issues, see the section called “Check your AWS CLI command formatting” (p. 167) and the section
called “Confirm that you’re running a recent version of the AWS CLI” (p. 168).

**Possible cause: The terminal needs to be restarted after installation**

*Example error text:*

```
$ aws --version
command not found: aws
```

If the `aws` command cannot be found after first installing or updating the AWS CLI, you might need
to restart your terminal for it to recognize any `PATH` updates.

**Possible cause: The AWS CLI did not fully install**

*Example error text:*

```
$ aws --version
command not found: aws
```

If the `aws` command cannot be found after first installing or updating the AWS CLI, it might not have
been fully installed. Try reinstalling by following the steps for your platform in *Installing the AWS
CLI* (p. 4).

**Possible cause: The AWS CLI does not have permissions (Linux)**

If the `aws` command cannot be found after first installing or updating the AWS CLI on Linux, it might
not have `execute` permissions for the folder it installed in. Run the following command with the
`PATH` to your AWS CLI installation, to provide `chmod` permissions to the AWS CLI:

```
$ sudo chmod -R 755 /usr/local/aws-cli/
```

**Possible cause: The operating system `PATH` was not updated during installation**

*Example error text:*

```
$ aws --version
command not found: aws
```

You might need to add the `aws` executable to your operating system's `PATH` environment variable.
To add the AWS CLI to your `PATH`, use the following instructions for your operating system.

**Linux and macOS**

1. Find your shell's profile script in your user directory. If you're not sure which shell you have, run `echo $SHELL`. 
The "aws --version" command returns a different version than you installed

Your terminal might be returning a different PATH for the AWS CLI than you expect.

Possible cause: The terminal needs to be restarted after installing

If the aws command shows the wrong version, you might need to restart your terminal for it to recognize any PATH updates.
Possible cause: You have multiple versions of the AWS CLI

If you updated the AWS CLI and used a different install method than your pre-existing installation, it might cause multiple versions to be installed. For example, if on Linux or macOS you used `pip` for your current install, but tried to update using the `.pkg` install file, this could cause some conflicts especially with your `PATH` pointing to the old version.

To resolve this, uninstall all versions of the AWS CLI (p. 175) and perform a clean install.

After uninstalling all versions, follow instructions appropriate for your operating system to install your desired version of the AWS CLI version 1 or AWS CLI version 2.

Note
If this is happening after you installed the AWS CLI version 2 with a pre-existing install of AWS CLI version 1, follow the migration instructions in Installation instructions when migrating from AWS CLI version 1.

Back to top (p. 167)

The "aws --version" command returns a version after uninstalling the AWS CLI

This often occurs when there is still an AWS CLI installed somewhere on your system.

Possible cause: The terminal needs to be restarted after uninstalling

If the `aws --version` command still works, you might need to restart your terminal for it to recognize any terminal updates.

Back to top (p. 167)

Possible cause: You have multiple versions of the AWS CLI on your system, or didn't use the same uninstall method that you used to originally install the AWS CLI

The AWS CLI might not uninstall correctly if you uninstalled the AWS CLI using a different method than you used to install it, or if you installed multiple versions. For example, if you used `pip` for your current install, you must use `pip` to uninstall it. To resolve this, uninstall AWS CLI using the same method that you used to install it.

1. Follow the instructions appropriate for your operating system and your original installation method to uninstall the AWS CLI version 1 and AWS CLI version 2.
2. Close all terminals you have open.
3. Open your preferred terminal, enter in the following command and confirm that no version is returned.

```
$ aws --version
command not found: aws
```

If you still have a version listed in the output, the AWS CLI was most likely installed using a different method or there are multiple versions. If you don't know which method you installed the AWS CLI, follow the instructions for each uninstall method for the AWS CLI version 1 and AWS CLI version 2 appropriate to your operating system until no version output is received.

Note
If you used a package manager to install the AWS CLI (`pip`, `apt`, `brew`, etc.), you must use the same package manager to uninstall it. Be sure to follow the instructions provided by the package manager on how to uninstall all versions of a package.
Access denied errors

Possible cause: The AWS CLI program file doesn’t have "run" permission

On Linux or macOS, make sure that the aws program has run permissions for the calling user. Typically, the permissions are set to 755.

To add run permission for your user, run the following command, substituting ~/.local/bin/aws with the path to the program on your computer.

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

Possible cause: Your IAM identity doesn’t have permission to perform the operation

Example error text:

```
$ aws s3 ls
An error occurred (AccessDenied) when calling the ListBuckets operation: Access denied.
```

When you run a AWS CLI command, AWS operations are performed on your behalf, using credentials that associate you with an IAM user or role. The policies attached to that IAM user or role must grant you permission to call the API actions that correspond to the commands that you run with the AWS CLI.

Most commands call a single action with a name that matches the command name. However, custom commands like aws s3 sync call multiple APIs. You can see which APIs a command calls by using the --debug option.

If you are sure that the user or role has the proper permissions assigned by policy, make sure that your AWS CLI command is using the credentials you expect. See the next section about credentials (p. 176) to verify that the credentials the AWS CLI is using are the ones that you expect.

For information about assigning permissions to IAM users and roles, see Overview of Access Management: Permissions and Policies in the IAM User Guide.

Invalid credentials and key errors

Example error text:

```
$ aws s3 ls
An error occurred (InvalidAccessKeyId) when calling the ListBuckets operation: The AWS Access Key Id you provided does not exist in our records.
```

```
$ aws s3 ls
An error occurred (InvalidClientTokenId) when calling the ListBuckets operation: The security token
```
Signature does not match errors

Possible cause: The AWS CLI is reading incorrect credentials or from an unexpected location

The AWS CLI might be reading credentials from a different location than you expect, or your key pair information is incorrect. You can run `aws configure list` to confirm which credentials are used.

The following example shows how to check the credentials used for the default profile.

```
$ aws configure list
Name                    Value             Type    Location
----                    -----             ----    --------
profile                 <not set>             None    None
access_key     ****************XYVA shared-credentials-file
secret_key     ****************ZAGY shared-credentials-file
region                us-west-2      config-file    ~/.aws/config
```

The following example shows how to check the credentials of a named profile.

```
$ aws configure list --profile saanvi
Name                    Value             Type    Location
----                    -----             ----    --------
profile                    saanvi           manual    --profile
access_key         **************** shared-credentials-file
secret_key         **************** shared-credentials-file
region                us-west-2      config-file    ~/.aws/config
```

To confirm your key pair details, check your config and credentials files. For more information on config and credentials files, see the section called “Configuration and credential file settings” (p. 30). For more information on key pairs, see the section called “Access key ID and secret access key” (p. 28).

Back to top (p. 167)

Possible cause: Your computer’s clock is out of sync

If you are using valid credentials, your clock might be out of sync. On Linux or macOS, run `date` to check the time.

```
$ date
```

If your system clock is not correct within a few minutes, use `ntpd` to sync it.

```
$ sudo service ntpd stop
$ sudo ntpdate time.nist.gov
$ sudo service ntpd start
$ ntpstat
```

On Windows, use the date and time options in the Control Panel to configure your system clock.

Back to top (p. 167)
When the AWS CLI runs a command, it sends an encrypted request to the AWS servers to perform the appropriate AWS service operations. Your credentials (the access key and secret key) are involved in the encryption and enable AWS to authenticate the person making the request. There are several things that can interfere with the correct operation of this process, as follows.

Possible cause: Your clock is out of sync with the AWS servers

To help protect against replay attacks, the current time can be used during the encryption/decryption process. If the time of the client and server disagree by more than the allowed amount, the process can fail and the request is rejected. This can also happen when you run a command in a virtual machine whose clock is out of sync with the host machine's clock. One possible cause is when the virtual machine hibernates and takes some time after waking up to sync the clock with the host machine.

On Linux or macOS, run `date` to check the time.

```bash
# date
```

If your system clock is not correct within a few minutes, use `ntpd` to sync it.

```bash
# 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.

Back to top (p. 167)

Possible cause: Your operating system is mishandling AWS secret keys that contain certain special characters

If your AWS secret key includes certain special characters, such as -, +, /, or %, some operating system variants process the string improperly and cause the secret key string to be interpreted incorrectly.

If you process your access keys and secret keys using other tools or scripts, such as tools that build the credentials file on a new instance as part of its creation, those tools and scripts might have their own handling of special characters that causes them to be transformed into something that AWS no longer recognizes.

We suggest regenerating the secret key to get one that does not include the special character causing issues.

Back to top (p. 167)

No Windows console found errors

Example error text:

```bash
$ aws s3 ls
```

Back to top (p. 167)
SSL certificate errors

Possible cause: The AWS CLI doesn't trust your proxy's certificate

Example error text:

```bash
$ aws s3 ls
[SSL: CERTIFICATE_VERIFY_FAILED] certificate verify failed
```

When you use an AWS CLI command, you receive an `[SSL: CERTIFICATE_VERIFY_FAILED] certificate verify failed` error message. This is caused by the AWS CLI not trusting your proxy's certificate due to factors such as your proxy's certificate being self-signed, with your company set as the Certification Authority (CA). This prevents the AWS CLI from finding your company's CA root certificate in the local CA registry.

To fix this, instruct the AWS CLI where to find your company's .pem file using the `ca_bundle (p. 34)` configuration file setting, `--ca-bundle (p. 48)` command line option, or the `AWS_CA_BUNDLE (p. 44)` environment variable.

Back to top (p. 167)

Possible cause: Your configuration isn't pointing to the correct CA root certificate location

Example error text:

```bash
$ aws s3 ls
SSL validation failed for regionname [Errno 2] No such file or directory
```

This is caused by your Certification Authority (CA) bundle file location being configured incorrectly in the AWS CLI. To fix this, confirm where your company's .pem file is located and update the AWS CLI configuration by using the `ca_bundle (p. 34)` configuration file setting, `--ca-bundle (p. 48)` command line option, or the `AWS_CA_BUNDLE (p. 44)` environment variable.

Back to top (p. 167)

Invalid JSON errors

Example error text:

```bash
$ aws dynamodb update-table \
   --provisioned-throughput '{"ReadCapacityUnits":15,"WriteCapacityUnits":10}' \
   --table-name MyDDBTable
Error parsing parameter '--provisioned-throughput': Invalid JSON: Expecting property name enclosed in double quotes: line 1 column 25 (char 24)
```
When you use an AWS CLI command, you receive a "Invalid JSON" error message. This is usually an error seen when you enter a command with an expected JSON format and the AWS CLI cannot read your JSON correctly.

**Possible cause: You did not enter valid JSON for the AWS CLI to use**

Confirm you have valid JSON entered for your command. We suggest using a JSON validator for JSON you're having issues formatting.

For more advanced JSON usage in the command line, consider using a command line JSON processor, like jq, to create JSON strings. For more information on jq, see the jq repository on GitHub.

**Possible cause: Your terminal's quoting rules are preventing valid JSON being sent to the AWS CLI**

Before the AWS CLI receives anything from a command, your terminal processes the command using it's own quoting and escaping rules. Due to a terminal's formatting rules, some of your JSON content may be stripped before the command is passed to the AWS CLI. When formulating commands, be sure to use your terminal's quoting rules (p. 78).

To troubleshoot, use the echo command to see how the shell is handling your parameters:

```
$ echo{"ReadCapacityUnits":15,"WriteCapacityUnits":10}  
ReadCapacityUnits:15 WriteCapacityUnits:10
```

```
$ echo '{"ReadCapacityUnits":15,"WriteCapacityUnits":10} '  
{"ReadCapacityUnits":15,"WriteCapacityUnits":10}
```

Modify your command until your until valid JSON is returned.

For more in-depth troubleshooting, use the --debug parameter to view the debug logs as they'll display exactly what got passed to the AWS CLI:

```
# aws dynamodb update-table  
  --provisioned-throughput '{"ReadCapacityUnits":15,\"WriteCapacityUnits":10} '  
  --table-name MyDDBTable \  --debug
```

```
Python/2.7.18 Linux/5.4.196-119.356.amzn2int.x86_64 botocore/1.18.6  
2022-07-19 22:25:07,741 - MainThread - awscli.clidriver - DEBUG - Arguments entered to CLI:  
['dynamodb', 'update-table', '--provisioned-throughput',  
'{"ReadCapacityUnits":15,\"WriteCapacityUnits":10}',  
'--table-name', 'MyDDBTable', '--debug']
```

Use your terminal's quoting rules to fix any issues your JSON input has when being sent to the AWS CLI. For more information on quoting rules, see the section called “Quotes with Strings” (p. 78).

**Note**

If you're having issues with getting valid JSON to the AWS CLI, we recommend to bypass a terminal's quoting rules for JSON data input by using Blobs to pass your JSON data directly to the AWS CLI. For more information on Blobs, see Binary / blob (binary large object) and streaming blob (p. 76).

Back to top (p. 167)
Additional resources

For additional help with your AWS CLI issues, visit the AWS CLI community on GitHub or the AWS re:Post community.

Back to top (p. 167)
AWS CLI user guide document history

The following table describes important additions to the *AWS Command Line Interface User Guide*, beginning in January 2019. For notification about updates to this documentation, you can subscribe to the RSS feed.

<table>
<thead>
<tr>
<th>update-history-change</th>
<th>update-history-description</th>
<th>update-history-date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Content for the AWS CLI V1 and V2 are now separated into their respective guides (p. 182)</td>
<td>For clarity and ease, the AWS CLI version 1 and AWS CLI version 2 content is now separated into their own guides. For AWS CLI version 2, see the latest <em>AWS Command Line Interface User Guide</em></td>
<td>November 2, 2021</td>
</tr>
<tr>
<td>Added AWS CLI alias information</td>
<td>Added AWS CLI alias information. Aliases are shortcuts you can create in the AWS Command Line Interface (AWS CLI) to shorten commands or scripts that you frequently use.</td>
<td>March 11, 2021</td>
</tr>
<tr>
<td>Updated filter output information</td>
<td>Updated information for filters and moved to their own page.</td>
<td>February 1, 2021</td>
</tr>
<tr>
<td>Deprecation announcement for Python 2.7, 3.4, and 3.5 (p. 182)</td>
<td>Python 2.7 was deprecated by the Python Software Foundation on January 1, 2020. Going forward, customers using the AWS CLI version 1 should transition to using Python 3, with a minimum of Python 3.6. Python 2.7 support is deprecated for new versions of the AWS CLI version 1 starting 7/19/2021. Python 3.4 and 3.5 is deprecated starting 2/1/2021.</td>
<td>January 29, 2021</td>
</tr>
<tr>
<td>Added Amazon S3 scripting example</td>
<td>Added an Amazon S3 lifecycle scripting example.</td>
<td>October 15, 2020</td>
</tr>
<tr>
<td>Added Amazon EC2 scripting example</td>
<td>Added an Amazon EC2 instance type scripting example.</td>
<td>October 15, 2020</td>
</tr>
<tr>
<td>Added retries information</td>
<td>Added a retries page for features and behavior of retries in the AWS CLI.</td>
<td>September 17, 2020</td>
</tr>
<tr>
<td>Server-side and client-side pagination page</td>
<td>Updated pagination information and centralized on a single page.</td>
<td>August 17, 2020</td>
</tr>
<tr>
<td>Update to</td>
<td>Description</td>
<td>Date</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>Updated s3 commands page</td>
<td>Updated the high-level s3 commands page with new examples and resources.</td>
<td>July 30, 2020</td>
</tr>
<tr>
<td>Updated installation information</td>
<td>The install, update, and uninstall information for Linux, macOS, and Windows are updated.</td>
<td>May 19, 2020</td>
</tr>
<tr>
<td>Updated to remove support for Python 2.6 and 3.3 from AWS CLI version 1</td>
<td>As of January 10th, 2020, AWS CLI version 1 no longer supports using Python versions 2.6 or 3.3. You must update to a newer version of Python to use AWS CLI version 1.17 or later.</td>
<td>January 10, 2020</td>
</tr>
<tr>
<td>New MFA section</td>
<td>Added a new section describing how to access the CLI using multi-factor authentication and roles.</td>
<td>May 3, 2019</td>
</tr>
<tr>
<td>Update to &quot;Using the CLI&quot; section</td>
<td>Major improvements and additions to the usage instructions and procedures.</td>
<td>March 7, 2019</td>
</tr>
<tr>
<td>Update to &quot;Installing the CLI&quot; section</td>
<td>Major improvements and additions to the AWS CLI installation instructions and procedures.</td>
<td>March 7, 2019</td>
</tr>
<tr>
<td>Update to &quot;Configuring the CLI&quot; section</td>
<td>Major improvements and additions to the AWS CLI configuration instructions and procedures.</td>
<td>March 7, 2019</td>
</tr>
</tbody>
</table>