



User Guide for Version 2

AWS Command Line Interface



AWS Command Line Interface: User Guide for Version 2

Copyright © 2024 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

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

Table of Contents

.....	ix
About the AWS CLI	1
About AWS CLI version 2	1
Maintenance and support for SDK major versions	2
About Amazon Web Services	2
About the examples	2
Additional documentation and resources	4
AWS CLI documentation and resources	4
Other AWS SDKs	4
Get started	5
Prerequisites	6
Create an IAM or IAM Identity Center administrative account	6
Next steps	7
Install/Update	7
AWS CLI install and update instructions	8
Troubleshooting AWS CLI install and uninstall errors	21
Next steps	22
Past releases	22
Troubleshooting AWS CLI install and uninstall errors	40
Next steps	40
Build and install from source	40
Why build from source?	41
Quicksteps	41
Step 1: Setup all requirements	44
Step 2: Configuring the AWS CLI source installation	48
Step 3: Building the AWS CLI	54
Step 4: Installing the AWS CLI	55
Step 5: Verifying the AWS CLI installation	57
Workflow examples	57
Troubleshooting AWS CLI install and uninstall errors	60
Next steps	60
Amazon ECR Public/Docker	61
Prerequisites	61
Deciding between Amazon ECR Public and Docker Hub	61

Run the official images	62
Notes on interfaces and backwards compatibility of the official images	63
Use specific versions and tags	63
Update to the latest official image	64
Share host files, credentials, environment variables, and configuration	65
Shorten the docker run command	71
Setup	74
Gather your credential information for programmatic access	74
Setting up new configuration and credentials	75
Using existing configuration and credentials files	83
Configure the AWS CLI	84
Configuration and credentials precedence	84
Additional topics in this section	85
Configuration and credential file settings	86
Format of the configuration and credential files	86
Where are configuration settings stored?	94
Using named profiles	95
Set and view configuration settings using commands	96
Setting new configuration and credentials command examples	99
Supported config file settings	102
Environment Variables	120
How to set environment variables	121
AWS CLI supported environment variables	122
Command line options	131
How to use command line options	132
AWS CLI supported global command line options	132
Common uses of command line options	137
Command completion	138
How it works	138
Configuring command completion on Linux or macOS	139
Configuring command completion on Windows	143
Retries	144
Available retry modes	144
Configuring a retry mode	147
Viewing logs of retry attempts	148
Use an HTTP proxy	149

Using the examples	149
Authenticating to a proxy	150
Using a proxy on Amazon EC2 instances	151
Troubleshooting	152
Endpoints	152
Set endpoint for a single command	152
Set global endpoint for all AWS services	152
Set to use FIPs endpoints for all AWS services	154
Set to use dual-stack endpoints for all AWS services	155
Set service-specific endpoints	156
Endpoint configuration and settings precedence	191
Authentication and access credentials	193
Configuration and credential precedence	194
Additional topics in this section	195
IAM Identity Center authentication	195
Configure automatic token refresh	196
Configure legacy non-refreshable	203
Using an IAM Identity Center profile	209
Short-term credentials	212
IAM roles	213
Prerequisites	213
Overview of using IAM roles	213
Configuring and using a role	215
Using MFA	217
Cross-account roles and external ID	218
Specifying a role session name for easier auditing	219
Assume role with web identity	219
Clearing cached credentials	221
IAM users	221
Step 1: Create your IAM user	222
Step 2: Get your access keys	222
Configure the AWS CLI	223
Use credentials for Amazon EC2 instance metadata	225
Prerequisites	225
Configuring a profile for Amazon EC2 metadata	226
External credentials	227

Use the AWS CLI	230
Get Help	230
The built-in AWS CLI help command	231
AWS CLI reference guide	236
API documentation	236
Troubleshooting errors	237
Additional help	237
Command Structure	237
Command structure	237
Wait commands	238
Specify Parameter Values	240
Common Parameter Types	241
Quotes with Strings	246
Parameters from Files	250
Generate a CLI Skeleton Template	253
Shorthand Syntax	265
Auto-prompt	267
How it works	268
Auto-prompt features	268
Auto-prompt modes	271
Configure auto-prompt	272
Control Command Output	272
Output Format	273
Pagination	282
Filter output	288
Return Codes	312
Wizards	313
How it works	314
Aliases	315
Prerequisites	315
Step 1: Creating the alias file	315
Step 2: Creating an alias	317
Step 3: Calling an alias	320
Alias repository examples	322
Resources	323
Code examples	324

Guided command examples	324
DynamoDB	325
Amazon EC2	328
S3 Glacier	347
IAM	354
Amazon S3	359
Amazon SNS	377
Bash script examples	380
Actions and scenarios	380
Security	545
Data Protection	545
Data encryption	546
Identity and Access Management	547
Audience	547
Authenticating with identities	548
Managing access using policies	551
How AWS services work with IAM	554
Troubleshooting AWS identity and access	554
Compliance Validation	556
Resilience	557
Infrastructure Security	557
Enforcing a minimum TLS version	558
Troubleshoot errors	559
General troubleshooting to try first	559
Check your AWS CLI command formatting	560
Confirm that you're running a recent version of the AWS CLI	560
Use the --debug option	561
Enable and review the AWS CLI command history logs	566
Confirm that your AWS CLI is configured	567
Command not found errors	567
The "aws --version" command returns a different version than you installed	570
The "aws --version" command returns a version after uninstalling the AWS CLI	571
The AWS CLI processed a command with an incomplete parameter name	572
Access denied errors	574
Invalid credentials and key errors	575
Signature does not match errors	576

SSL certificate errors	578
Invalid JSON errors	578
Additional resources	580
Migration guide	581
New features and changes	581
AWS CLI version 2 new features	581
Breaking changes between AWS CLI version 1 and AWS CLI version 2	583
Migration instructions	590
Replacing version 1 with version 2	591
Side-by-side install	591
Uninstall	593
Troubleshooting AWS CLI install and uninstall errors	596
Document History	597
AWS Glossary	602

What is the AWS Command Line Interface?

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 2

The AWS CLI version 2 is the most recent major version of the AWS CLI and supports all of the latest features. Some features introduced in version 2 are not 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 [Migrate from AWS CLI version 1 to version 2](#).

The AWS CLI version 2 is available to install only as a bundled installer. While you might find it in package managers, these are unsupported and unofficial packages that are not produced

or managed by AWS. We recommend that you install the AWS CLI from only the official AWS distribution points, as documented in this guide.

To install the AWS CLI version 2, see [the section called “Install/Update”](#).

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

```
$ aws --version  
aws-cli/2.15.19 Python/3.11.6 Linux/5.10.205-195.807.amzn2.x86_64 botocore/1.18.6
```

For version history, see the [AWS CLI version 2 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**.

About 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.

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

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.

```
$ 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 2 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 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](#)

Get started with the AWS CLI

This chapter provides steps to get started with version 2 of the AWS Command Line Interface (AWS CLI) and provides links to the relevant instructions.

1. [**Complete all prerequisites**](#) - To access AWS services with the AWS CLI, you need at minimum an AWS account and IAM credentials. To increase the security of your AWS account, we recommend that you do not use your root account credentials. You should create a user with least privilege to provide access credentials to the tasks you'll be running in AWS.
2. Install or gain access to the AWS CLI using one of the following methods:
 - [**\(Recommended\)** the section called "Install/Update".](#)
 - [**the section called "Past releases"**](#). Installing a specific version is primarily used if your team aligns their tools to a specific version.
 - [**the section called "Build and install from source"**](#). Building the AWS CLI from GitHub source is a more in-depth method that is primarily used by customers who work on platforms that we do not directly support with our pre-built installers.
 - [**the section called "Amazon ECR Public/Docker"**](#).
 - Access the AWS CLI version 2 in the AWS console from your browser using AWS CloudShell.
For more information, see the [AWS CloudShell User Guide](#).
3. [**After you have access to the AWS CLI, configure your AWS CLI with your IAM credentials for first time use.**](#)

Troubleshooting installer or configure errors

If you have issues after installing, uninstalling, or configuring the AWS CLI, see [Troubleshoot errors](#) for troubleshooting steps.

Topics

- [Prerequisites to use the AWS CLI version 2](#)
- [Install or update to the latest version of the AWS CLI](#)
- [Install past releases of the AWS CLI version 2](#)
- [Build and install the AWS CLI from source](#)
- [Run the AWS CLI from the official Amazon ECR Public or Docker images](#)

- [Set up the AWS CLI](#)

Prerequisites to use the AWS CLI version 2

To access AWS services with the AWS CLI, you need an AWS account and IAM credentials. When running AWS CLI commands, the AWS CLI needs to have access to those AWS credentials. To increase the security of your AWS account, we recommend that you do not use your root account credentials. You should create a user with least privilege to provide access credentials to the tasks you'll be running in AWS.

Topics

- [Create an IAM or IAM Identity Center administrative account](#)
- [Next steps](#)

Create an IAM or IAM Identity Center administrative account

Before you can configure

To create an administrator user, choose one of the following options.

Choose one way to manage your administrator	To	By	You can also
In IAM Identity Center (Recommended)	Use short-term credentials to access AWS. This aligns with the security best practices. For information about best practices, see Security best	Following the instructions in Getting started in the AWS IAM Identity Center User Guide .	Configure programmatic access by Configuring the AWS CLI to use AWS IAM Identity Center in the AWS Command Line Interface User Guide .

Choose one way to manage your administrator	To	By	You can also
	practices in IAM in the <i>IAM User Guide</i> .		
In IAM (Not recommended)	Use long-term credentials to access AWS.	Following the instructions in Creating your first IAM admin user and user group in the <i>IAM User Guide</i> .	Configure programmatic access by Managing access keys for IAM users in the <i>IAM User Guide</i> .

Next steps

After creating an AWS account and IAM credentials, to use the AWS CLI you can do one of the following:

- [Install the latest release](#) of the AWS CLI version 2 on your computer.
- [Install a past release](#) of the AWS CLI version 2 on your computer.
- Access the AWS CLI version 2 from your computer [using a Docker image](#).
- Access the AWS CLI version 2 in the AWS console from your browser using AWS CloudShell. For more information see the [AWS CloudShell User Guide](#).

Install or update to the latest version of the AWS CLI

This topic describes how to install or update the latest release of the AWS Command Line Interface (AWS CLI) on supported operating systems. For information on the latest releases of AWS CLI, see the [AWS CLI version 2 Changelog](#) on GitHub.

To install a past release of the AWS CLI, see [the section called “Past releases”](#). For uninstall instructions, see [Uninstall](#).

⚠️ Important

AWS CLI versions 1 and 2 use the same aws command name. If you previously installed AWS CLI version 1, see [Migrate from AWS CLI version 1 to version 2](#).

Topics

- [AWS CLI install and update instructions](#)
- [Troubleshooting AWS CLI install and uninstall errors](#)
- [Next steps](#)

AWS CLI install and update instructions

For installation instructions, expand the section for your operating system.

Linux

Install and update requirements

- You must be able to extract or "unzip" the downloaded package. If your operating system doesn't have the built-in unzip command, use an equivalent.
- The AWS CLI uses glibc, groff, and less. These are included by default in most major distributions of Linux.
- We support the AWS CLI on 64-bit versions of recent distributions of CentOS, Fedora, Ubuntu, Amazon Linux 1, Amazon Linux 2, Amazon Linux 2023, and Linux ARM.
- Because AWS doesn't maintain third-party repositories, we can't guarantee that they contain the latest version of the AWS CLI.

Install or update the AWS CLI

⚠️ Warning

If this is your first time updating on Amazon Linux, to install the latest version of the AWS CLI, you must uninstall the pre-installed yum version using the following command:

```
$ sudo yum remove awscli
```

After the yum installation of the AWS CLI is removed, follow the below Linux install instructions.

To update your current installation of AWS CLI, download a new installer each time you update to overwrite previous versions. Follow these steps from the command line to install the AWS CLI on Linux.

We provide the steps in one easy to copy and paste group based on whether you use 64-bit Linux or Linux ARM. See the descriptions of each line in the steps that follow.

Linux x86 (64-bit)

 **Note**

(Optional) The following command block downloads and installs the AWS CLI without first verifying the integrity of your download. To verify the integrity of your download, use the below step by step instructions.

To install the AWS CLI, run the following commands.

```
$ curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"  
unzip awscliv2.zip  
sudo ./aws/install
```

To update your current installation of the AWS CLI, add your existing symlink and installer information to construct the install command using the --bin-dir, --install-dir, and --update parameters. The following command block uses an example symlink of */usr/local/bin* and example installer location of */usr/local/aws-cli*.

```
$ curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"  
unzip awscliv2.zip  
sudo ./aws/install --bin-dir /usr/local/bin --install-dir /usr/local/aws-cli --  
update
```

Linux ARM

Note

(Optional) The following command block downloads and installs the AWS CLI without first verifying the integrity of your download. To verify the integrity of your download, use the below step by step instructions.

To install the AWS CLI, run the following commands.

```
$ curl "https://awscli.amazonaws.com/awscli-exe-linux-aarch64.zip" -o "awscliv2.zip"  
unzip awscliv2.zip  
sudo ./aws/install
```

To update your current installation of the AWS CLI, add your existing symlink and installer information to construct the `install` command using the `--bin-dir`, `--install-dir`, and `--update` parameters. The following command block uses an example symlink of `/usr/local/bin` and example installer location of `/usr/local/aws-cli`.

```
$ curl "https://awscli.amazonaws.com/awscli-exe-linux-aarch64.zip" -o "awscliv2.zip"  
unzip awscliv2.zip  
sudo ./aws/install --bin-dir /usr/local/bin --install-dir /usr/local/aws-cli --  
update
```

1. Download the installation file in one of the following ways:

Linux x86 (64-bit)

- **Use the `curl` command** – The `-o` option specifies the file name that the downloaded package is written to. The options on the following example command write the downloaded file to the current directory with the local name `awscliv2.zip`.

```
$ curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o  
"awscliv2.zip"
```

- **Downloading from the URL** – To download the installer with your browser, use the following URL: https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip

Linux ARM

- **Use the curl command** – The -o option specifies the file name that the downloaded package is written to. The options on the following example command write the downloaded file to the current directory with the local name awscliv2.zip.

```
$ curl "https://awscli.amazonaws.com/awscli-exe-linux-aarch64.zip" -o  
"awscliv2.zip"
```

- **Downloading from the URL** – To download the installer with your browser, use the following URL: <https://awscli.amazonaws.com/awscli-exe-linux-aarch64.zip>

2. (Optional) Verifying the integrity of your downloaded zip file

If you chose to manually download the AWS CLI installer package .zip in the above steps, you can use the following steps to verify the signatures by using the GnuPG tool.

The AWS CLI installer package .zip files are cryptographically signed using PGP signatures. If there is any damage or alteration of the files, this verification fails and you should not proceed with installation.

- a. Download and install the gpg command using your package manager. For more information about GnuPG, see the [GnuPG website](#).
- b. To create the public key file, create a text file and paste in the following text.

```
-----BEGIN PGP PUBLIC KEY BLOCK-----  
  
mQINBF2Cr7UBEADJZHcgus0J17ENSy়umXh85z0TRV0xJorM2B/JL0kH0yigQluUG  
ZMLhENaG0bYatdrKP+3H911vK050pXwn0/R7fB/FSTouki4ciIx50uLlnJZIxSzx  
PqG10mkxImLNbGWoi6Lto0LYxqHN2iQtzlwTVmq9733zd3XfcXrZ3+Lb1HAgEt5G  
TfNxEKJ8soPLyWmwDH6HWCnjZ/aIQRBTIQ05uVeEoYxSh6wOai7ss/KveoSNBbYz  
gbdzoqI2Y8cgH2nbfgp3DSasaLZEEdCSsIsK1u05CinE7k2qZ7KgKAUICt/cR/grk  
C6VwsnDU00UCideXcQ8WeHutqvgZH1JgKDbznoIzeQHJD238GEu+eKhRHcz8/jeG  
94zkcgJ0z3KbZGYMiTh277Fvj9zzvZsbMCedV1BTg3TqgvdX4bdkhf5cH+7NtW0  
1rFj6UwAsGukBTA0xC01/dnSmZhJZ1KmEWilro/g0rjt0xqRqutlIqG22TaqoPG  
FYVN+en3Zwbt97kcgZDwqbuykNt64oZWc4XKCa3mprEGC3IbJTBFqglXmZ719ywG  
EEUJY01b2XrSuPwm139beWdKM8kzr10jn10m6+1pTRCBfo0wa9F8YZRhHPAkwKkX  
XDe0GpWRj4oh0x0d2GWkyV5xyN14p2tQ0Cd00Dmz80yUTgRpPVQu0EhXQARAQAB  
tCFBV1MgQ0xJIFR1YW0gPGF3cy1jbG1AYW1hem9uLmNvbT6JA1QEEwETAD4CGwMF  
CwkIBwIGFQoJCAsCBBYCAwECHgECF4AWIQT7Xbd/1cEYuAURraimMQrMRnJHXAUC
```

```
ZMKcEgUJCSEf3QAKCRCmMQrMRnJHXCi1D/4vior9J5tB+icri5WbDudS3ak/ve4q
XS6ZLm5S81+CBxy5aLQU1yFhuaaEHDC11fG780duxatzeHENASYVo3mmKNwrCBza
NJaewKLGQT0MKwBSP5aa3dva8P/4oUP9GsQn0uWoXwNDWfrMbNI8gn+jC/3MiGw
vD3fu6zC0WWLITNv2SJ0lwlmb/uGfha68o4iTBovcftVRua06DyqF+CrHX/0j0
k1EDQFMY9M4tsYT7X8NWF18Vmc89nzpvL9fwda44WwpKIw1FBZP8S0sgDx2xDs xv
L8kM2Gt0iH0cHqF0+V7xtTKZyloLiDbJKhu80Kc+YC/TmozD8oeGU2rEFXfLegwS
zT9N+jB38+dqaP9pRDsi45iGqyA8yavVBabpL0IQ9jU6eIV+kmcjIjcun/Uo8Sj J
0xQasm41rxPaKV6vJUn10wVNuhSkKk8mzN01Sz wu7Hua6rdcc aGeB8uJ44AP3QzW
BNn rjtoN6A1N0D2wFmfE/YL/rHPxU1XwPntubYB/t3rXFL7ENQ00QH0KVXgRC1ey
sHMglg46c+nQLRzVTshjDjmtzvh9rcV9RKRoPetEggzCoD89veDA9jPR2Kw6RYkS
XzYm2fEv16/HRNYt7hJzneFqRIjHW5qAgSs/bcaRWpAU/QQzzJPVKCQNr4y0weyg
B8HCtGjfod0p1A==
=gdMc
-----END PGP PUBLIC KEY BLOCK-----
```

For reference, the following are the details of the public key.

Key ID:	A6310ACC4672
Type:	RSA
Size:	4096/4096
Created:	2019-09-18
Expires:	2024-07-26
User ID:	AWS CLI Team <aws-cli@amazon.com>
Key fingerprint:	FB5D B77F D5C1 18B8 0511 ADA8 A631 0ACC 4672 475C

- c. Import the AWS CLI public key with the following command, substituting *public-key-file-name* with the file name of the public key you created.

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

- d. Download the AWS CLI signature file for the package you downloaded. It has the same path and name as the .zip file it corresponds to, but has the extension .sig. In the following examples, we save it to the current directory as a file named awscliv2.sig.

Linux x86 (64-bit)

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

```
$ curl -o awscliv2.sig https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip.sig
```

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 **2.0.30** would be `awscli-exe-linux-x86_64-2.0.30.zip.sig` resulting in the following command:

```
$ curl -o awscliv2.sig https://awscli.amazonaws.com/awscli-exe-linux-x86_64-2.0.30.zip.sig
```

For a list of versions, see the [AWS CLI version 2 Changelog](#) on *GitHub*.

Linux ARM

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

```
$ curl -o awscliv2.sig https://awscli.amazonaws.com/awscli-exe-linux-aarch64.zip.sig
```

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 **2.0.30** would be `awscli-exe-linux-aarch64-2.0.30.zip.sig` resulting in the following command:

```
$ curl -o awscliv2.sig https://awscli.amazonaws.com/awscli-exe-linux-aarch64-2.0.30.zip.sig
```

For a list of versions, see the [AWS CLI version 2 Changelog](#) on *GitHub*.

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

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

The output should look similar to the following.

```
gpg: Signature made Mon Nov  4 19:00:01 2019 PST
gpg:                               using RSA key FB5D B77F D5C1 18B8 0511 ADA8 A631 0ACC 4672
475C
gpg: Good signature from "AWS CLI Team <aws-cli@amazon.com>" [unknown]
```

```
gpg: WARNING: This key is not certified with a trusted signature!
gpg:                               There is no indication that the signature belongs to the owner.
Primary key fingerprint: FB5D B77F D5C1 18B8 0511 ADA8 A631 0ACC 4672 475C
```

Important

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

3. Unzip the installer. If your Linux distribution doesn't have a built-in unzip command, use an equivalent to unzip it. The following example command unzips the package and creates a directory named aws under the current directory.

```
$ unzip awscliv2.zip
```

Note

When updating from a previous version, the unzip command prompts to overwrite existing files. To skip these prompts, such as with script automation, use the -u update flag for unzip. This flag automatically updates existing files and creates new ones as needed.

```
$ unzip -u awscliv2.zip
```

4. Run the install program. The installation command uses a file named `install` in the newly unzipped `aws` directory. By default, the files are all installed to `/usr/local/aws-cli`, and a symbolic link is created in `/usr/local/bin`. The command includes `sudo` to grant write permissions to those directories.

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

You can install without `sudo` if you specify directories that you already have write permissions to. Use the following instructions for the `install` command to specify the installation location:

- Ensure that the paths you provide to the `-i` and `-b` parameters contain no volume name or directory names that contain any space characters or other white space characters. If there is a space, the installation fails.
- `--install-dir` or `-i` – This option specifies the directory to copy all of the files to.

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

- `--bin-dir` or `-b` – This option specifies that the main aws program in the install directory is symbolically linked to the file aws in the specified path. You must have write permissions to the specified directory. Creating a symlink to a directory that is already in your path eliminates the need to add the install directory to the user's \$PATH variable.

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

```
$ ./aws/install -i /usr/local/aws-cli -b /usr/local/bin
```

Note

To update your current installation of the AWS CLI, add your existing symlink and installer information to construct the `install` command with the `--update` parameter.

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

To locate the existing symlink and installation directory, use the following steps:

1. Use the `which` command to find your symlink. This gives you the path to use with the `--bin-dir` parameter.

```
$ which aws  
/usr/local/bin/aws
```

2. Use the `ls` command to find the directory that your symlink points to. This gives you the path to use with the `--install-dir` parameter.

```
$ ls -l /usr/local/bin/aws
```

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

5. Confirm the installation with the following command.

```
$ aws --version  
aws-cli/2.15.19 Python/3.11.6 Linux/5.10.205-195.807.amzn2.x86_64 botocore/2.4.5
```

If the aws command cannot be found, you might need to restart your terminal or follow the troubleshooting in [Troubleshoot errors](#).

macOS

Install and update requirements

- We support the AWS CLI on macOS versions 10.9 and later. For more information, see [macOS support policy updates for the AWS CLI v2](#) on the *AWS Developer Tools Blog*.
- Because AWS doesn't maintain third-party repositories, we can't guarantee that they contain the latest version of the AWS CLI.

Install or update the AWS CLI

If you are updating to the latest version, use the same installation method that you used in your current version. You can install the AWS CLI on macOS in the following ways.

GUI installer

The following steps show how to install the latest version of the AWS CLI by using the standard macOS user interface and your browser.

1. In your browser, download the macOS pkg file: <https://awscli.amazonaws.com/AWSCLIV2.pkg>
2. Run your downloaded file and follow the on-screen instructions. You can choose to install the AWS CLI in the following ways:
 - **For all users on the computer (requires sudo)**
 - You can install to any folder, or choose the recommended default folder of /usr/local/aws-cli.

- The installer automatically creates a symlink at `/usr/local/bin/aws` that links to the main program in the installation folder you chose.
- **For only the current user (doesn't require sudo)**
 - You can install to any folder to which you have write permission.
 - Due to standard user permissions, after the installer finishes, you must manually create a symlink file in your `$PATH` that points to the `aws` and `aws_completer` programs by using the following commands at the command prompt. If your `$PATH` includes a folder you can write to, you can run the following command without sudo if you specify that folder as the target's path. If you don't have a writable folder in your `$PATH`, you must use sudo in the commands to get permissions to write to the specified target folder. The default location for a symlink is `/usr/local/bin/`.

```
$ sudo ln -s /folder/installed/aws-cli/aws /usr/local/bin/aws  
$ sudo ln -s /folder/installed/aws-cli/aws_completer /usr/local/bin/  
aws_completer
```

Note

You can view debug logs for the installation by pressing **Cmd+L** anywhere in the installer. This opens a log pane that enables you to filter and save the log. The log file is also automatically saved to `/var/log/install.log`.

3. To verify that the shell can find and run the `aws` command in your `$PATH`, use the following commands.

```
$ which aws  
/usr/local/bin/aws  
$ aws --version  
aws-cli/2.15.19 Python/3.11.6 Darwin/23.3.0 botocore/2.4.5
```

If the `aws` command cannot be found, you might need to restart your terminal or follow the troubleshooting in [Troubleshoot errors](#).

Command line installer - All users

If you have sudo permissions, you can install the AWS CLI for all users on the computer. We provide the steps in one easy to copy and paste group. See the descriptions of each line in the following steps.

```
$ curl "https://awscli.amazonaws.com/AWSCLIV2.pkg" -o "AWSCLIV2.pkg"  
$ sudo installer -pkg AWSCLIV2.pkg -target /
```

1. Download the file using the `curl` command. The `-o` option specifies the file name that the downloaded package is written to. In this example, the file is written to `AWSCLIV2.pkg` in the current folder.

```
$ curl "https://awscli.amazonaws.com/AWSCLIV2.pkg" -o "AWSCLIV2.pkg"
```

2. Run the standard macOS `installer` program, specifying the downloaded `.pkg` file as the source. Use the `-pkg` parameter to specify the name of the package to install, and the `-target /` parameter for which drive to install the package to. The files are installed to `/usr/local/aws-cli`, and a symlink is automatically created in `/usr/local/bin`. You must include `sudo` on the command to grant write permissions to those folders.

```
$ sudo installer -pkg ./AWSCLIV2.pkg -target /
```

After installation is complete, debug logs are written to `/var/log/install.log`.

3. To verify that the shell can find and run the `aws` command in your `$PATH`, use the following commands.

```
$ which aws  
/usr/local/bin/aws  
$ aws --version  
aws-cli/2.15.19 Python/3.11.6 Darwin/23.3.0 botocore/2.4.5
```

If the `aws` command cannot be found, you might need to restart your terminal or follow the troubleshooting in [Troubleshoot errors](#).

Command line - Current user

1. To specify which folder the AWS CLI is installed to, you must create an XML file with any file name. This file is an XML-formatted file that looks like the following example. Leave all values as shown, except you must replace the path `/Users/myusername` in line 9 with the path to the folder you want the AWS CLI installed to. *The folder must already exist, or the command fails.* The following XML example, named `choices.xml`, specifies the installer to install the AWS CLI in the folder `/Users/myusername`, where it creates a folder named `aws-cli`.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/
PropertyList-1.0.dtd">
<plist version="1.0">
<array>
<dict>
<key>choiceAttribute</key>
<string>customLocation</string>
<key>attributeSetting</key>
<string>/Users/myusername</string>
<key>choiceIdentifier</key>
<string>default</string>
</dict>
</array>
</plist>
```

2. Download the pkg installer using the curl command. The -o option specifies the file name that the downloaded package is written to. In this example, the file is written to `AWSCLIV2.pkg` in the current folder.

```
$ curl "https://awscli.amazonaws.com/AWSCLIV2.pkg" -o "AWSCLIV2.pkg"
```

3. Run the standard macOS installer program with the following options:
 - Specify the name of the package to install by using the `-pkg` parameter.
 - Specify installing to a *current user only* by setting the `-target` parameter to `CurrentUserHomeDirectory`.
 - Specify the path (relative to the current folder) and name of the XML file that you created in the `-applyChoiceChangesXML` parameter.

The following example installs the AWS CLI in the folder `/Users/myusername/aws-cli`.

```
$ installer -pkg AWSCLIV2.pkg \
    -target CurrentUserHomeDirectory \
    -applyChoiceChangesXML choices.xml
```

4. Because standard user permissions typically don't allow writing to folders in your \$PATH, the installer in this mode doesn't try to add the symlinks to the aws and aws_completer programs. For the AWS CLI to run correctly, you must manually create the symlinks after the installer finishes. If your \$PATH includes a folder you can write to and you specify the folder as the target's path, you can run the following command without sudo. If you don't have a writable folder in your \$PATH, you must use sudo for permissions to write to the specified target folder. The default location for a symlink is `/usr/local/bin/`. Replace `folder/installed` with the path to your AWS CLI installation.

```
$ sudo ln -s /folder/installed/aws-cli/aws /usr/local/bin/aws
$ sudo ln -s /folder/installed/aws-cli/aws_completer /usr/local/bin/
aws_completer
```

After installation is complete, debug logs are written to `/var/log/install.log`.

5. To verify that the shell can find and run the aws command in your \$PATH, use the following commands.

```
$ which aws
/usr/local/bin/aws
$ aws --version
aws-cli/2.15.19 Python/3.11.6 Darwin/23.3.0 botocore/2.4.5
```

If the aws command cannot be found, you might need to restart your terminal or follow the troubleshooting in [Troubleshoot errors](#).

Windows

Install and update requirements

- We support the AWS CLI on Microsoft-supported versions of 64-bit Windows.
- Admin rights to install software

Install or update the AWS CLI

To update your current installation of AWS CLI on Windows, download a new installer each time you update to overwrite previous versions. AWS CLI is updated regularly. To see when the latest version was released, see the [AWS CLI version 2 Changelog](#) on GitHub.

1. Download and run the AWS CLI MSI installer for Windows (64-bit):

<https://awscli.amazonaws.com/AWSCLIV2.msi>

Alternatively, you can run the `msiexec` command to run the MSI installer.

```
C:\> msiexec.exe /i https://awscli.amazonaws.com/AWSCLIV2.msi
```

For various parameters that can be used with `msiexec`, see [msiexec](#) on the *Microsoft Docs* website. For example, you can use the `/qn` flag for a silent installation.

```
C:\> msiexec.exe /i https://awscli.amazonaws.com/AWSCLIV2.msi /qn
```

2. To confirm the installation, open the **Start** menu, search for cmd to open a command prompt window, and at the command prompt use the `aws --version` command.

```
C:\> aws --version
aws-cli/2.15.19 Python/3.11.6 Windows/10 exe/AMD64 prompt/off
```

If Windows is unable to find the program, you might need to close and reopen the command prompt window to refresh the path, or follow the troubleshooting in [Troubleshoot errors](#).

Troubleshooting AWS CLI install and uninstall errors

If you come across issues after installing or uninstalling the AWS CLI, see [Troubleshoot errors](#) for troubleshooting steps. For the most relevant troubleshooting steps, see [the section called "Command not found errors"](#), [the section called "The "aws --version" command returns a different version than you installed"](#), and [the section called "The "aws --version" command returns a version after uninstalling the AWS CLI"](#).

Next steps

After you successfully install the AWS CLI, you can safely delete your downloaded installer files.

After completing the steps in [the section called “Prerequisites”](#) and installing the AWS CLI, you should perform a [the section called “Setup”](#).

Install past releases of the AWS CLI version 2

This topic describes how to install the past releases of the AWS Command Line Interface version 2 (AWS CLI) on supported operating systems. For information on the AWS CLI version 2 releases, see the [AWS CLI version 2 Changelog](#) on GitHub.

AWS CLI version 2 installation instructions:

Linux

Installation requirements

- You know which release of the AWS CLI version 2 you'd like to install. For a list of versions, see the [AWS CLI version 2 Changelog](#) on GitHub.
- You must be able to extract or "unzip" the downloaded package. If your operating system doesn't have the built-in unzip command, use an equivalent.
- The AWS CLI version 2 uses glibc, groff, and less. These are included by default in most major distributions of Linux.
- We support the AWS CLI version 2 on 64-bit versions of recent distributions of CentOS, Fedora, Ubuntu, Amazon Linux 1, Amazon Linux 2 and Linux ARM.
- Because AWS doesn't maintain third-party repositories, we can't guarantee that they contain the latest version of the AWS CLI.

Installation instructions

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

We provide the steps in one easy to copy and paste group based on whether you use 64-bit Linux or Linux ARM. See the descriptions of each line in the steps that follow.

Linux x86 (64-bit)

Note

(Optional) The following command block downloads and installs the AWS CLI without first verifying the integrity of your download. To verify the integrity of your download, use the below step by step instructions.

For a list of versions, see the [AWS CLI version 2 Changelog](#) on *GitHub*.

To install the AWS CLI, run the following commands.

To specify a version, append a hyphen and the version number to the filename. For this example the filename for version **2.0.30** would be `awscli-exe-linux-x86_64-2.0.30.zip` resulting in the following command:

```
$ curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64-2.0.30.zip" -o
"awscliv2.zip"
unzip awscliv2.zip
sudo ./aws/install
```

To update your current installation of the AWS CLI, add your existing symlink and installer information to construct the `install` command using the `--bin-dir`, `--install-dir`, and `--update` parameters. The following command block uses an example symlink of `/usr/local/bin` and example installer location of `/usr/local/aws-cli`.

```
$ curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64-2.0.30.zip" -o
"awscliv2.zip"
unzip awscliv2.zip
sudo ./aws/install --bin-dir /usr/local/bin --install-dir /usr/local/aws-cli --
update
```

Linux ARM

Note

(Optional) The following command block downloads and installs the AWS CLI without first verifying the integrity of your download. To verify the integrity of your download, use the below step by step instructions.

For a list of versions, see the [AWS CLI version 2 Changelog](#) on GitHub.

To install the AWS CLI, run the following commands.

To specify a version, append a hyphen and the version number to the filename. For this example the filename for version *2.0.30* would be awscli-exe-linux-aarch64-2.0.30.zip resulting in the following command:

```
$ curl "https://awscli.amazonaws.com/awscli-exe-linux-aarch64-2.0.30.zip" -o
"awscliv2.zip"
unzip awscliv2.zip
sudo ./aws/install
```

To update your current installation of the AWS CLI, add your existing symlink and installer information to construct the `install` command using the `--bin-dir`, `--install-dir`, and `--update` parameters. The following command block uses an example symlink of */usr/local/bin* and example installer location of */usr/local/aws-cli*.

```
$ curl "https://awscli.amazonaws.com/awscli-exe-linux-aarch64-2.0.30.zip" -o
"awscliv2.zip"
unzip awscliv2.zip
sudo ./aws/install --bin-dir /usr/local/bin --install-dir /usr/local/aws-cli --
update
```

1. Download the installation file in one of the following ways:

Linux x86 (64-bit)

- **Use the `curl` command** – The `-o` option specifies the file name that the downloaded package is written to. The options on the following example command write the downloaded file to the current directory with the local name `awscliv2.zip`.

To specify a version, append a hyphen and the version number to the filename. For this example the filename for version *2.0.30* would be `awscli-exe-linux-x86_64-2.0.30.zip` resulting in the following command:

```
$ curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64-2.0.30.zip" -o
"awscliv2.zip"
```

For a list of versions, see the [AWS CLI version 2 Changelog](#) on GitHub.

- **Downloading from the URL –**

In your browser, download your specific version of the AWS CLI by appending a hyphen and the version number to the filename.

```
https://awscli.amazonaws.com/awscli-exe-linux-x86_64-<version.number>.zip
```

For this example the filename for version **2.0.30** would be awscli-exe-linux-x86_64-2.0.30.zip resulting in the following link: https://awscli.amazonaws.com/awscli-exe-linux-x86_64-2.0.30.zip

Linux ARM

- **Use the curl command** – The -o option specifies the file name that the downloaded package is written to. The options on the following example command write the downloaded file to the current directory with the local name awscliv2.zip.

To specify a version, append a hyphen and the version number to the filename.

For this example the filename for version **2.0.30** would be awscli-exe-linux-aarch64-2.0.30.zip resulting in the following command:

```
$ curl "https://awscli.amazonaws.com/<b>awscli-exe-linux-aarch64-2.0.30.zip</b>" -o  
"awscliv2.zip"  
unzip awscliv2.zip  
sudo ./aws/install
```

- **Downloading from the URL –**

In your browser, download your specific version of the AWS CLI by appending a hyphen and the version number to the filename.

```
https://awscli.amazonaws.com/awscli-exe-linux-aarch64-<version.number>.zip
```

For this example the filename for version **2.0.30** would be awscli-exe-linux-aarch64-2.0.30.zip resulting in the following link: <https://awscli.amazonaws.com/awscli-exe-linux-aarch64-2.0.30.zip>

2. (Optional) Verifying the integrity of your downloaded zip file

If you chose to manually download the AWS CLI installer package .zip in the above steps, you can use the following steps to verify the signatures by using the GnuPG tool.

The AWS CLI installer package .zip files are cryptographically signed using PGP signatures. If there is any damage or alteration of the files, this verification fails and you should not proceed with installation.

- a. Download and install the gpg command using your package manager. For more information about GnuPG, see the [GnuPG website](#).
- b. To create the public key file, create a text file and paste in the following text.

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

```
mQINBF2Cr7UBEADJZHcgus0J17ENSy়umXh85z0TRV0xJorM2B/JL0kH0yigQluUG
ZMLhENaG0bYatdrKP+3H911vK050pXwn0/R7fB/FSTouki4ciIx50uLlnJZIxSzx
PqG10mkxImLNbGwoi6Lto0LYxqHN2iQtzlwTVmq9733zd3XfcXrZ3+Lb1HAgEt5G
TfNxEKJ8soPLyWmwDH6HWcnjZ/aIQRBTIQ05uVeEoYxSh6w0ai7ss/KveoSNBbYz
gbdzoqI2Y8cgH2nbfgp3DSasaLZEEdCSsIsK1u05CinE7k2qZ7KgKAUIcT/cR/grk
C6VwsnDU00UCideXcQ8WeHutqvgZH1JgKDbznoIzeQHJD238GEu+eKhRHcz8/jeG
94zkcgJ0z3KbZGYMiTh277Fvj9zzvZsbMB CedV1BTg3TqgvdX4bdkhf5cH+7NtW0
1rFj6UwAsGukBTA0xC01/dnSmZhJ7Z1KmEWilro/g0rjt0xqRQut1IqG22TaqoPG
fYVN+en3Zwbt97kcgZDwqbuykNt64oZWc4XKCa3mprEGC3IbJTBFqglXmZ719ywG
EEUJY01b2XrSuPwm139beWdKM8kzr10jn10m6+1pTRCBfo0wa9F8YZRhHPAkwKkX
XDe0GpWRj4oh0x0d2GwkyV5xyN14p2tQ0Cd00Dmz80yUTgRpPVQu0EhXQARAQAB
tCFBV1MgQ0xJIFR1YW0gPGF3cy1jbG1AYW1hem9uLmNvbT6JA1QEEwEIAD4CGwMF
CwkIBwIGFQoJCAcCBBYCAwECHgECF4AWIQT7Xbd/1cEYuAURraimMQRMRnJHXAUC
ZMKcEgUJCSEf3QAKCRCmMQxMRnJHXCi1D/4vior9J5tB+icri5WbDudS3ak/ve4q
XS6ZLm5S81+CBxy5aLQU1yFhuaaEHDC11fG780duxatzeHENASYVo3mmKNwrCBza
NJaeaWKLGQT0MKwBSP5aa3dva8P/4oUP9GsQn0uWoXwNDWfrMbNI8gn+jC/3MigW
vD3fu6zCOWWLITNv2SJ0qlwILmb/uGfha68o4iTBovcftVRua06DyqF+CrHX/0j0
k1EDQFMY9M4tsYT7X8NWF18VmC89nzpvL9fwda44WwpKIw1FBZP8S0sgDx2xDs xv
L8kM2Gt0iH0cHqF0+V7xtTKZylo1iDbJKhu80Kc+YC/TmozD8oeGU2rEFXfLegwS
zT9N+jB38+dqaP9pRDsi45iGqyA8yavVBabpL0IQ9jU6eIV+kmcjIjcun/Uo8sjJ
0xQasm41rxPaKV6vJUn10wVNuhSkKk8mzN01SZwu7Hua6rdccGeB8uJ44AP3QzW
BNnrjtoN6A1N0D2wFmfE/YL/rHPxU1XwPntubYB/t3rXFL7ENQOOQH0KVXgRC1ey
sHMglg46c+nQLRzVTshjDjmtzvh9rcV9RKRoPetEggzCoD89veDA9jPR2Kw6RYkS
XzYm2fEv16/HRNYt7hJzneFqRIjHW5qAgSs/bcaRWpAU/QQzzJPVKCQNr4y0weyg
B8HCtGjfod0p1A==
=gdMc
-----END PGP PUBLIC KEY BLOCK-----
```

For reference, the following are the details of the public key.

Key ID:	A6310ACC4672
Type:	RSA
Size:	4096/4096
Created:	2019-09-18
Expires:	2024-07-26
User ID:	AWS CLI Team <aws-cli@amazon.com>
Key fingerprint:	FB5D B77F D5C1 18B8 0511 ADA8 A631 0ACC 4672 475C

- c. Import the AWS CLI public key with the following command, substituting *public-key-file-name* with the file name of the public key you created.

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

- d. Download the AWS CLI signature file for the package you downloaded. It has the same path and name as the .zip file it corresponds to, but has the extension .sig. In the following examples, we save it to the current directory as a file named awscliv2.sig.

Linux x86 (64-bit)

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

```
$ curl -o awscliv2.sig https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip.sig
```

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 *2.0.30* would be awscli-exe-linux-x86_64-2.0.30.zip.sig resulting in the following command:

```
$ curl -o awscliv2.sig https://awscli.amazonaws.com/awscli-exe-linux-x86_64-2.0.30.zip.sig
```

For a list of versions, see the [AWS CLI version 2 Changelog on GitHub](#).

Linux ARM

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

```
$ curl -o awscliv2.sig https://awscli.amazonaws.com/awscli-exe-linux-aarch64.zip.sig
```

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 **2.0.30** would be awscli-exe-linux-aarch64-2.0.30.zip.sig resulting in the following command:

```
$ curl -o awscliv2.sig https://awscli.amazonaws.com/awscli-exe-linux-aarch64-2.0.30.zip.sig
```

For a list of versions, see the [AWS CLI version 2 Changelog](#) on *GitHub*.

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

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

The output should look similar to the following.

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

Important

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

3. Unzip the installer. If your Linux distribution doesn't have a built-in `unzip` command, use an equivalent to unzip it. The following example command unzips the package and creates a directory named `aws` under the current directory.

```
$ unzip awscliv2.zip
```

4. Run the install program. The installation command uses a file named `install` in the newly unzipped `aws` directory. By default, the files are all installed to `/usr/local/aws-cli`, and a symbolic link is created in `/usr/local/bin`. The command includes `sudo` to grant write permissions to those directories.

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

You can install without `sudo` if you specify directories that you already have write permissions to. Use the following instructions for the `install` command to specify the installation location:

- Ensure that the paths you provide to the `-i` and `-b` parameters contain no volume name or directory names that contain any space characters or other white space characters. If there is a space, the installation fails.
- `--install-dir` or `-i` – This option specifies the directory to copy all of the files to.

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

- `--bin-dir` or `-b` – This option specifies that the main `aws` program in the `install` directory is symbolically linked to the file `aws` in the specified path. You must have write permissions to the specified directory. Creating a symlink to a directory that is already in your path eliminates the need to add the `install` directory to the user's `$PATH` variable.

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

```
$ ./aws/install -i /usr/local/aws-cli -b /usr/local/bin
```

Note

To update your current installation of the AWS CLI version 2 to a newer version, add your existing symlink and installer information to construct the `install` command with the `--update` parameter.

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

To locate the existing symlink and installation directory, use the following steps:

1. Use the `which` command to find your symlink. This gives you the path to use with the `--bin-dir` parameter.

```
$ which aws  
/usr/local/bin/aws
```

2. Use the `ls` command to find the directory that your symlink points to. This gives you the path to use with the `--install-dir` parameter.

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

5. Confirm the installation with the following command.

```
$ aws --version  
aws-cli/2.15.19 Python/3.11.6 Linux/5.10.205-195.807.amzn2.x86_64 botocore/2.4.5
```

If the `aws` command cannot be found, you might need to restart your terminal or follow the troubleshooting in [Troubleshoot errors](#).

(Optional) Verifying the integrity of your downloaded zip file

If you chose to manually download the AWS CLI version 2 installer package `.zip` in the above steps, you can use the following steps to verify the signatures by using the GnuPG tool.

The AWS CLI version 2 installer package `.zip` files are cryptographically signed using PGP signatures. If there is any damage or alteration of the files, this verification fails and you should not proceed with installation.

1. Download and install the `gpg` command using your package manager. For more information about GnuPG, see the [GnuPG website](#).
2. To create the public key file, create a text file and paste in the following text.

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

```
mQINBF2Cr7UBEADJZHcgus0J17ENSy whole key block removed for brevity
```

-----END PGP PUBLIC KEY BLOCK-----

For reference, the following are the details of the public key.

Key ID:	A6310ACC4672
Type:	RSA
Size:	4096/4096
Created:	2019-09-18
Expires:	2024-07-26
User ID:	AWS CLI Team <aws-cli@amazon.com>
Key fingerprint:	FB5D B77F D5C1 18B8 0511 ADA8 A631 0ACC 4672 475C

- Import the AWS CLI public key with the following command, substituting *public-key-file-name* with the file name of the public key you created.

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

4. Download the AWS CLI signature file for the package you downloaded. It has the same path and name as the .zip file it corresponds to, but has the extension .sig. In the following examples, we save it to the current directory as a file named awscliv2.sig.

Linux x86 (64-bit)

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

```
$ curl -o awscliv2.sig https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip.sig
```

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 **2.0.30** would be awscli-exe-linux-x86_64-2.0.30.zip.sig resulting in the following command:

```
$ curl -o awscliv2.sig https://awscli.amazonaws.com/awscli-exe-linux-x86_64-2.0.30.zip.sig
```

For a list of versions, see the [AWS CLI version 2 Changelog](#) on GitHub.

Linux ARM

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

```
$ curl -o awscliv2.sig https://awscli.amazonaws.com/awscli-exe-linux-aarch64.zip.sig
```

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 **2.0.30** would be awscli-exe-linux-aarch64-2.0.30.zip.sig resulting in the following command:

```
$ curl -o awscliv2.sig https://awscli.amazonaws.com/awscli-exe-linux-aarch64-2.0.30.zip.sig
```

For a list of versions, see the [AWS CLI version 2 Changelog](#) on GitHub.

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

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

The output should look similar to the following.

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

Important

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

macOS

Installation requirements

- You know which release of the AWS CLI version 2 you'd like to install. For a list of versions, see the [AWS CLI version 2 Changelog](#) on GitHub.
- We support the AWS CLI version 2 on Apple-supported versions of 64-bit macOS.
- Because AWS doesn't maintain third-party repositories, we can't guarantee that they contain the latest version of the AWS CLI.

Installation instructions

You can install the AWS CLI version 2 on macOS in the following ways.

GUI installer

The following steps show how to install or update to the latest version of the AWS CLI version 2 by using the standard macOS user interface and your browser. If you are updating to the latest version, use the same installation method that you used for your current version.

1. In your browser, download your specific version of the AWS CLI by appending a hyphen and the version number to the filename.

```
https://awscli.amazonaws.com/AWSCLIV2-version.number.pkg
```

For this example, the filename for version **2.0.30** would be AWSCLIV2-2.0.30.pkg resulting in the following link: <https://awscli.amazonaws.com/AWSCLIV2-2.0.30.pkg>.

2. Run your downloaded file and follow the on-screen instructions. You can choose to install the AWS CLI version 2 in the following ways:

- **For all users on the computer (requires sudo)**

- You can install to any folder, or choose the recommended default folder of /usr/local/aws-cli.
- The installer automatically creates a symlink at /usr/local/bin/aws that links to the main program in the installation folder you chose.

- **For only the current user (doesn't require sudo)**

- You can install to any folder to which you have write permission.
- Due to standard user permissions, after the installer finishes, you must manually create a symlink file in your \$PATH that points to the aws and aws_completer programs by using the following commands at the command prompt. If your \$PATH includes a folder you can write to, you can run the following command without sudo if you specify that folder as the target's path. If you don't have a writable folder in your \$PATH, you must use sudo in the commands to get permissions to write to the specified target folder. The default location for a symlink is /usr/local/bin/.

```
$ sudo ln -s /folder/installed/aws-cli/aws /usr/local/bin/aws  
$ sudo ln -s /folder/installed/aws-cli/aws_completer /usr/local/bin/  
aws_completer
```

Note

You can view debug logs for the installation by pressing **Cmd+L** anywhere in the installer. This opens a log pane that enables you to filter and save the log. The log file is also automatically saved to `/var/log/install.log`.

3. To verify that the shell can find and run the aws command in your \$PATH, use the following commands.

```
$ which aws  
/usr/local/bin/aws  
$ aws --version  
aws-cli/2.15.19 Python/3.11.6 Darwin/23.3.0 botocore/2.4.5
```

If the aws command cannot be found, you might need to restart your terminal or follow the troubleshooting in [Troubleshoot errors](#).

Command line installer - All users

If you have sudo permissions, you can install the AWS CLI version 2 for all users on the computer. We provide the steps in one easy to copy and paste group. See the descriptions of each line in the following steps.

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 **2.0.30** would be `AWSCLIV2-2.0.30.pkg` resulting in the following command:

```
$ curl "https://awscli.amazonaws.com/AWSCLIV2-2.0.30.pkg" -o "AWSCLIV2.pkg"  
$ sudo installer -pkg AWSCLIV2.pkg -target /
```

1. Download the file using the curl command. The -o option specifies the file name that the downloaded package is written to. In this example, the file is written to `AWSCLIV2.pkg` in the current folder.

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 **2.0.30** would be `AWSCLIV2-2.0.30.pkg` resulting in the following command:

```
$ curl "https://awscli.amazonaws.com/AWSCLIV2-2.0.30.pkg" -o "AWSCLIV2.pkg"
```

For a list of versions, see the [AWS CLI version 2 Changelog](#) on GitHub.

2. Run the standard macOS installer program, specifying the downloaded .pkg file as the source. Use the -pkg parameter to specify the name of the package to install, and the -target / parameter for which drive to install the package to. The files are installed to /usr/local/aws-cli, and a symlink is automatically created in /usr/local/bin. You must include sudo on the command to grant write permissions to those folders.

```
$ sudo installer -pkg ./AWSCLIV2.pkg -target /
```

After installation is complete, debug logs are written to /var/log/install.log.

3. To verify that the shell can find and run the aws command in your \$PATH, use the following commands.

```
$ which aws
/usr/local/bin/aws
$ aws --version
aws-cli/2.15.19 Python/3.11.6 Darwin/23.3.0 botocore/2.4.5
```

If the aws command cannot be found, you might need to restart your terminal or follow the troubleshooting in [Troubleshoot errors](#).

Command line - Current user

1. To specify which folder the AWS CLI is installed to, you must create an XML file. This file is an XML-formatted file that looks like the following example. Leave all values as shown, except you must replace the path **/Users/myusername** in line 9 with the path to the folder you want the AWS CLI version 2 installed to. *The folder must already exist, or the command fails.* This XML example specifies that the installer installs the AWS CLI in the folder /Users/myusername, where it creates a folder named aws-cli.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/
PropertyList-1.0.dtd">
<plist version="1.0">
```

```
<array>
  <dict>
    <key>choiceAttribute</key>
    <string>customLocation</string>
    <key>attributeSetting</key>
    <string>/Users/myusername</string>
    <key>choiceIdentifier</key>
    <string>default</string>
  </dict>
</array>
</plist>
```

2. Download the pkg installer using the curl command. The -o option specifies the file name that the downloaded package is written to. In this example, the file is written to AWSCLIV2.pkg in the current folder.

For the specific version of the AWS CLI, append a hyphen and the version number to the filename. For this example the filename for version **2.0.30** would be AWSCLIV2-2.0.30.pkg resulting in the following command:

```
$ curl "https://awscli.amazonaws.com/AWSCLIV2-2.0.30.pkg" -o "AWSCLIV2.pkg"
```

For a list of versions, see the [AWS CLI version 2 Changelog](#) on GitHub.

3. Run the standard macOS installer program with the following options:
 - Specify the name of the package to install by using the -pkg parameter.
 - Specify installing to a *current user only* by setting the -target parameter to CurrentUserHomeDirectory.
 - Specify the path (relative to the current folder) and name of the XML file that you created in the -applyChoiceChangesXML parameter.

The following example installs the AWS CLI in the folder /Users/myusername/aws-cli.

```
$ installer -pkg AWSCLIV2.pkg \
            -target CurrentUserHomeDirectory \
            -applyChoiceChangesXML choices.xml
```

4. Because standard user permissions typically don't allow writing to folders in your \$PATH, the installer in this mode doesn't try to add the symlinks to the aws and aws_completer

programs. For the AWS CLI to run correctly, you must manually create the symlinks after the installer finishes. If your \$PATH includes a folder you can write to and you specify the folder as the target's path, you can run the following command without sudo. If you don't have a writable folder in your \$PATH, you must use sudo for permissions to write to the specified target folder. The default location for a symlink is /usr/local/bin/.

```
$ sudo ln -s /folder/installed/aws-cli/aws /usr/local/bin/aws  
$ sudo ln -s /folder/installed/aws-cli/aws_completer /usr/local/bin/  
aws_completer
```

After installation is complete, debug logs are written to /var/log/install.log.

5. To verify that the shell can find and run the aws command in your \$PATH, use the following commands.

```
$ which aws  
/usr/local/bin/aws  
$ aws --version  
aws-cli/2.15.19 Python/3.11.6 Darwin/23.3.0 botocore/2.4.5
```

If the aws command cannot be found, you might need to restart your terminal or follow the troubleshooting in [Troubleshoot errors](#).

Windows

Installation requirements

- You know which release of the AWS CLI version 2 you'd like to install. For a list of versions, see the [AWS CLI version 2 Changelog](#) on GitHub.
- We support the AWS CLI on Microsoft-supported versions of 64-bit Windows.
- Admin rights to install software

Installation instructions

To update your current installation of AWS CLI version 2 on Windows, download a new installer each time you update to overwrite previous versions. AWS CLI is updated regularly. To see when the latest version was released, see the [AWS CLI version 2 Changelog](#) on GitHub.

1. Download and run the AWS CLI MSI installer for Windows (64-bit) in one of the following ways:

- **Downloading and running the MSI installer:** To create your download link for a specific version of the AWS CLI, append a hyphen and the version number to the filename.

```
https://awscli.amazonaws.com/AWSCLIV2-version.number.msi
```

For this example the filename for version **2.0.30** would be AWSCLIV2-2.0.30.msi resulting in the following link: <https://awscli.amazonaws.com/AWSCLIV2-2.0.30.msi>.

- **Using the msieexec command:** Alternatively, you can use the MSI installer by adding the link to the msieexec command. For a specific version of the AWS CLI, append a hyphen and the version number to the filename.

```
C:\> msieexec.exe /i https://awscli.amazonaws.com/AWSCLIV2-version.number.msi
```

For this example the filename for version **2.0.30** would be AWSCLIV2-2.0.30.msi resulting in the following link <https://awscli.amazonaws.com/AWSCLIV2-2.0.30.msi>.

```
C:\> msieexec.exe /i https://awscli.amazonaws.com/AWSCLIV2-2.0.30.msi
```

For various parameters that can be used with msieexec, see [msieexec](#) on the *Microsoft Docs* website.

For a list of versions, see the [AWS CLI version 2 Changelog](#) on *GitHub*.

2. To confirm the installation, open the **Start** menu, search for cmd to open a command prompt window, and at the command prompt use the aws --version command.

```
C:\> aws --version
aws-cli/2.15.19 Python/3.11.6 Windows/10 exe/AMD64 prompt/off
```

If Windows is unable to find the program, you might need to close and reopen the command prompt window to refresh the path, or follow the troubleshooting in [Troubleshoot errors](#).

Troubleshooting AWS CLI install and uninstall errors

If you come across issues after installing or uninstalling the AWS CLI, see [Troubleshoot errors](#) for troubleshooting steps. For the most relevant troubleshooting steps, see [the section called "Command not found errors"](#), [the section called "The "aws --version" command returns a different version than you installed"](#), and [the section called "The "aws --version" command returns a version after uninstalling the AWS CLI"](#).

Next steps

After completing the steps in [the section called "Prerequisites"](#) and installing the AWS CLI, you should perform a [the section called "Setup"](#).

Build and install the AWS CLI from source

This topic describes how to install or update from source to the latest release of the AWS Command Line Interface (AWS CLI) on supported operating systems.

For information on the latest releases of AWS CLI, see the [AWS CLI version 2 Changelog](#) on GitHub.

Important

AWS CLI versions 1 and 2 use the same aws command name. If you previously installed AWS CLI version 1, see [Migrate from AWS CLI version 1 to version 2](#).

Topics

- [Why build from source?](#)
- [Quicksteps](#)
- [Step 1: Setup all requirements](#)
- [Step 2: Configuring the AWS CLI source installation](#)
- [Step 3: Building the AWS CLI](#)
- [Step 4: Installing the AWS CLI](#)
- [Step 5: Verifying the AWS CLI installation](#)
- [Workflow examples](#)

- [Troubleshooting AWS CLI install and uninstall errors](#)
- [Next steps](#)

Why build from source?

The AWS CLI is [available as pre-built installers](#) for most platforms and environments as well as a Docker image.

Generally, these installers provide coverage for most use-cases. The instructions for installing from source are to help with the use-cases our installers do not cover. Some of these use-cases include the following:

- The pre-built installers do not support your environment. For example, ARM 32-bit is not supported by the pre-built installers.
- The pre-built installers have dependencies your environment lacks. For example, Alpine Linux uses [musl](#), but the current installers require glibc causing the pre-built installers to not immediately work.
- The pre-built installers require resources your environment restricts access to. For example, security hardened systems might not give permissions to shared memory. This is needed for the frozen aws installer.
- The pre-built installers are often blockers for maintainers in package managers, as full control over the building process for code and packages is preferred. Building from source enables distribution maintainers a more streamlined process to keep the AWS CLI updated. Enabling maintainers provides customers more up-to-date versions of the AWS CLI when installing from a 3rd party package manager such as brew, yum, and apt.
- Customers that patch AWS CLI functionality require building and installing the AWS CLI from source. This is especially important for community members that want to test changes they've made to the source prior to contributing the change to the AWS CLI GitHub repository.

Quicksteps

 **Note**

All code examples are assumed to run from the root of the source directory.

To build and install the AWS CLI from source, follow the steps in this section. The AWS CLI leverages [GNU Autotools](#) to install from source. In the simplest case, the AWS CLI can be installed from source by running the default example commands from the root of the AWS CLI GitHub repository.

1. [Setup all requirements for your environment](#). This includes being able to run [GNU Autotools](#) generated files and Python 3.8 or later is installed.
2. In your terminal, navigate to the top level of the AWS CLI source folder and run the `./configure` command. This command checks the system for all required dependencies and generates a `Makefile` for building and installing the AWS CLI based on detected and specified configurations.

Linux and macOS

The following `./configure` command example sets the build configuration for the AWS CLI using default settings.

```
$ ./configure
```

Windows PowerShell

Before running any commands calling MSYS2, you must preserve your current working directory:

```
PS C:\> $env:CHERE_INVOKING = 'yes'
```

Then use the following `./configure` command example to set the build configuration for the AWS CLI using your local path to your Python executable, installing to `C:\Program Files\AWSCLI`, and downloading all dependencies.

```
PS C:\> C:\msys64\usr\bin\bash -lc " PYTHON='C:\path\to\python.exe' ./configure --prefix='C:\Program Files\AWSCLI' --with-download-deps "
```

For details, available configuration options, and default setting information, see the [the section called “Step 2: Configuring the AWS CLI source installation”](#) section.

- Run the `make` command. This command builds the AWS CLI according to your configuration settings.

The following `make` command example builds with default options using your existing `./configure` settings.

Linux and macOS

```
$ make
```

Windows PowerShell

```
PS C:\> C:\msys64\usr\bin\bash -lc "make"
```

For details and available build options, see the [the section called “Step 3: Building the AWS CLI” section](#).

- Run the `make install` command. This command installs your built AWS CLI to the configured location on your system.

The following `make install` command example installs your built AWS CLI and creates symlinks in your configured locations using default command settings.

Linux and macOS

```
$ make install
```

Windows PowerShell

```
PS C:\> C:\msys64\usr\bin\bash -lc "make install"
```

After installing, add the path to the AWS CLI using the following:

```
PS C:\> $Env: PATH +=";C:\Program Files\AWSCLI\bin\"
```

For details and available install options, see the [the section called “Step 4: Installing the AWS CLI” section](#).

5. Confirm the AWS CLI successfully installed using the following command:

```
$ aws --version  
aws-cli/2.15.19 Python/3.11.6 Windows/10 exe/AMD64 prompt/off
```

For troubleshooting steps for install errors see the [the section called “Troubleshooting AWS CLI install and uninstall errors”](#) section.

Step 1: Setup all requirements

To build the AWS CLI from source you need the following completed beforehand:

Note

All code examples are assumed to run from the root of the source directory.

1. Download the AWS CLI source by either forking the AWS CLI GitHub repository or downloading the source tarball. The instructions is one of the following:

- Fork and clone the [AWS CLI repository](#) from *Github*. For more information, see [Fork a repo](#) in the *Github Docs*.
- Download the latest source tarball at <https://awscli.amazonaws.com/awscli.tar.gz> extract the contents using the following commands:

```
$ curl -o awscli.tar.gz https://awscli.amazonaws.com/awscli.tar.gz  
$ tar -xzf awscli.tar.gz
```

Note

To download a specific version, use the following link format: <https://awscli.amazonaws.com/awscli-<versionnumber>.tar.gz>

For example, for version 2.10.0 the link is the following: <https://awscli.amazonaws.com/awscli-2.10.0.tar.gz>

Source versions are available starting with version **2.10.0** of the AWS CLI.

(Optional) Verifying the integrity of your downloaded zip file by completing the following steps:

1. You can use the following steps to verify the signatures by using the GnuPG tool.

The AWS CLI installer package .zip files are cryptographically signed using PGP signatures. If there is any damage or alteration of the files, this verification fails and you should not proceed with installation.

2. Download and install the gpg command using your package manager. For more information about GnuPG, see the [GnuPG website](#).
3. To create the public key file, create a text file and paste in the following text.

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

```
mQINBF2Cr7UBEADJZHcgus0J17ENSy whole line of base64 encoded public key
```

-----END PGP PUBLIC KEY BLOCK-----

For reference, the following are the details of the public key.

Key ID:	A6310ACC4672
Type:	RSA
Size:	4096/4096
Created:	2019-09-18
Expires:	2023-09-17
User ID:	AWS CLI Team <aws-cli@amazon.com>
Key fingerprint:	FB5D B77F D5C1 18B8 0511 ADA8 A631 0ACC 4672 475C

4. Import the AWS CLI public key with the following command, substituting *public-key-file-name* with the file name of the public key you created.

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

5. Download the AWS CLI signature file for the package you downloaded at <https://awscli.amazonaws.com/awscli.tar.gz.sig>. It has the same path and name as the tarball file it corresponds to, but has the extension .sig. Save it in the same path as the tarball file. Or use the following command block:

```
$ curl -o awscliv2.sig https://awscli.amazonaws.com/awscli.tar.gz.sig
```

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

```
$ gpg --verify awscliv2.sig awscli.tar.gz
```

The output should look similar to the following.

```
gpg: Signature made Mon Nov  4 19:00:01 2019 PST
gpg:                               using RSA key FB5D B77F D5C1 18B8 0511 ADA8 A631 0ACC 4672
gpg:                               475C
gpg: Good signature from "AWS CLI Team <aws-cli@amazon.com>" [unknown]
gpg: WARNING: This key is not certified with a trusted signature!
```

```
gpg: There is no indication that the signature belongs to the owner.  
Primary key fingerprint: FB5D B77F D5C1 18B8 0511 ADA8 A631 0ACC 4672 475C
```

Important

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

2. You have an environment that can run [GNU Autotools](#) generated files such as `configure` and `Makefile`. These files are widely portable across POSIX platforms.

Linux and macOS

If Autotools is not already installed in your environment or you need to update them, then follow the installation instructions found in [How do I install the Autotools \(as user\)?](#) or [Basic Installation](#) in the *GNU documentation*.

Windows PowerShell

Warning

We suggest if you are in a Windows environment, you use the pre-built installers. For install instructions on the pre-built installers, see [the section called “Install/Update”](#)

Since Windows does not come with a POSIX-compliant shell, you need to install additional software to install the AWS CLI from source. [MSYS2](#) provides a collection of tools and libraries to help build and install Windows software, especially for the POSIX-based scripting that Autotools uses.

1. Install MSYS2. For information on installing and using MSYS2, see the [install and usage instructions](#) in the *MSYS2 Documentation*.
2. Open the MSYS2 terminal and install autotools using the following command.

```
$ pacman -S autotools
```

Note

When using the configure, build, and install code examples in this guide for Windows, the default MSYS2 install path of C:\msys64\usr\bin\bash is assumed. When calling MSYS2 inside of PowerShell you'll be using the following format, with the bash command in quotes:

```
PS C:\> C:\msys64\usr\bin\bash -lc "command example"
```

The following command example calls the ./configure command.

```
PS C:\> C:\msys64\usr\bin\bash -lc "./configure"
```

3. A Python 3.8 or later interpreter is installed. The minimum Python version required follows the same timelines as the official [Python support policy for AWS SDKs and Tools](#). An interpreter is only supported 6 months after its end-of-support date.
4. **(Optional)** Install all build and runtime Python library dependencies of the AWS CLI. The ./configure command informs you if you are missing any dependencies and how to install them.

You can automatically install and use these dependencies through configuration, see [the section called “Downloading dependencies”](#) for more information.

Step 2: Configuring the AWS CLI source installation

Configuration for building and installing the AWS CLI is specified using the configure script. For the documentation of all configuration options, run the configure script with the --help option:

Linux and macOS

```
$ ./configure --help
```

Windows PowerShell

```
PS C:\> C:\msys64\usr\bin\bash -lc "./configure --help"
```

The most important options are the following:

- [Install location](#)
- [Python interpreter](#)
- [Downloading dependencies](#)
- [Install type](#)

Install location

The source installation of the AWS CLI uses two configurable directories to install the AWS CLI:

- libdir - Parent directory where the AWS CLI will be installed. The path to the AWS CLI installation is <libdir-value>/aws-cli. The default libdir value for Linux and macOS is /usr/local/lib making the default installation directory */usr/local/lib/aws-cli*
- bindir - Directory where the AWS CLI executables are installed. The default location is /usr/local/bin.

The following configure options control the directories used:

- --prefix - Sets the directory prefix to use for the installation. The default value for Linux and macOS is /usr/local.
- --libdir - Sets the libdir to use for installing the AWS CLI. The default value is <prefix-value>/lib. If both --libdir and --prefix are not specified, the default for Linux and macOS is /usr/local/lib/.
- --bindir - Sets the bindir to use for installing the AWS CLI aws and aws_completer executables. The default value is <prefix-value>/bin. If both bindir and --prefix are not specified, the default for Linux and macOS is /usr/local/bin/.

Linux and macOS

The following command example uses the --prefix option to do a local user install of the AWS CLI. This command installs the AWS CLI in \$HOME/.local/lib/aws-cli and the executables in \$HOME/.local/bin:

```
$ ./configure --prefix=$HOME/.local
```

The following command example uses the `--libdir` option to install the AWS CLI as an add-on application in the `/opt` directory. This command installs the AWS CLI at `/opt/aws-cli` and the executables at their default location of `/usr/local/bin`.

```
$ ./configure --libdir=/opt
```

Windows PowerShell

The following command example uses the `--prefix` option to do a local user install of the AWS CLI. This command installs the AWS CLI in `$HOME/.local/lib/aws-cli` and the executables in `$HOME/.local/bin`:

```
$ C:\msys64\usr\bin\bash -lc "./configure --prefix='C:\Program Files\AWSCLI'"
```

The following command example uses the `--libdir` option to install the AWS CLI as an add-on application in the `/opt` directory. This command installs the AWS CLI at `C:\Program Files\AWSCLI\opt\aws-cli`.

Python interpreter

Note

It is highly recommended to specify the Python interpreter when installing for Windows.

The `./configure` script automatically selects an installed Python 3.8 or later interpreter to use in building and running the AWS CLI using the [AM_PATH_PYTHON](#) Autoconf macro.

The Python interpreter to use can be explicitly set using the `PYTHON` environment variable when running the `configure` script:

Linux and macOS

```
$ PYTHON=/path/to/python ./configure
```

Windows PowerShell

```
PS C:\> C:\msys64\usr\bin\bash -lc "PYTHON='C:\path\to\python' ./configure"
```

Downloading dependencies

By default, it is required that all build and runtime dependencies of the AWS CLI are already installed on the system. This includes any Python library dependencies. All dependencies are checked when the `configure` script is run, and if the system is missing any Python dependencies, the `configure` script errors out.

The following code example errors out when your system is missing dependencies:

Linux and macOS

```
$ ./configure
checking for a Python interpreter with version >= 3.8... python
checking for python... /Users/username/.envs/env3.11/bin/python
checking for python version... 3.11
checking for python platform... darwin
checking for GNU default python prefix... ${prefix}
checking for GNU default python exec_prefix... ${exec_prefix}
checking for python script directory (pythondir)... ${PYTHON_PREFIX}/lib/python3.11/
site-packages
checking for python extension module directory (pyexecdir)... ${PYTHON_EXEC_PREFIX}/
lib/python3.11/site-packages
checking for --with-install-type... system-sandbox
checking for --with-download-deps... Traceback (most recent call last):
  File "<frozen runpy>", line 198, in _run_module_as_main
  File "<frozen runpy>", line 88, in _run_code
  File "/Users/username/aws-code/aws-cli/.backends/build_system/__main__.py", line
125, in <module>
    main()
  File "/Users/username/aws-code/aws-cli/.backends/build_system/__main__.py", line
121, in main
    parsed_args.func(parsed_args)
  File "/Users/username/aws-code/aws-cli/.backends/build_system/__main__.py", line
49, in validate
    validate_env(parsed_args.artifact)
  File "/Users/username/aws-code/aws-cli/.backends/build_system/validate_env.py",
line 68, in validate_env
    raise UnmetDependenciesException(unmet_deps, in_venv)
UnmetDependenciesException: Environment requires following Python
dependencies:

awscrt (required: ('>=0.12.4', '<0.17.0')) (version installed: None)
```

We recommend using `--with-download-deps` flag to automatically create a virtualenv and download the dependencies.

If you want to manage the dependencies yourself instead, run the following pip command:

```
/Users/username/.envs/env3.11/bin/python -m pip install --prefer-binary  
'awscrt>=0.12.4,<0.17.0'
```

```
configure: error: "Python dependencies not met."
```

Windows PowerShell

```
PS C:\> C:\msys64\usr\bin\bash -lc "./configure"  
checking for a Python interpreter with version >= 3.8... python  
checking for python... /Users/username/.envs/env3.11/bin/python  
checking for python version... 3.11  
checking for python platform... darwin  
checking for GNU default python prefix... ${prefix}  
checking for GNU default python exec_prefix... ${exec_prefix}  
checking for python script directory (pythondir)... ${PYTHON_PREFIX}/lib/python3.11/  
site-packages  
checking for python extension module directory (pyexecdir)... ${PYTHON_EXEC_PREFIX}/  
lib/python3.11/site-packages  
checking for --with-install-type... system-sandbox  
checking for --with-download-deps... Traceback (most recent call last):  
  File "<frozen runpy>", line 198, in _run_module_as_main  
  File "<frozen runpy>", line 88, in _run_code  
  File "/Users/username/aws-code/aws-cli./backends/build_system/__main__.py", line  
125, in <module>  
    main()  
  File "/Users/username/aws-code/aws-cli./backends/build_system/__main__.py", line  
121, in main  
    parsed_args.func(parsed_args)  
  File "/Users/username/aws-code/aws-cli./backends/build_system/__main__.py", line  
49, in validate  
    validate_env(parsed_args.artifact)  
  File "/Users/username/aws-code/aws-cli./backends/build_system/validate_env.py",  
line 68, in validate_env  
    raise UnmetDependenciesException(unmet_deps, in_venv)  
validate_env.UnmetDependenciesException: Environment requires following Python  
dependencies:  
  
awscrt (required: ('>=0.12.4', '<0.17.0')) (version installed: None)
```

We recommend using `--with-download-deps` flag to automatically create a virtualenv and download the dependencies.

If you want to manage the dependencies yourself instead, run the following pip command:

```
/Users/username/.envs/env3.11/bin/python -m pip install --prefer-binary  
'awscrt>=0.12.4,<0.17.0'
```

```
configure: error: "Python dependencies not met."
```

To automatically install the required Python dependencies, use the `--with-download-deps` option. When using this flag, the build process does the following:

- Skips the Python library dependencies check.
- Configures the settings to download all required Python dependencies and use **only** the downloaded dependencies to build the AWS CLI during the make build.

The following configure command example uses the `--with-download-deps` option to download and use the Python dependencies:

Linux and macOS

```
$ ./configure --with-download-deps
```

Windows PowerShell

```
PS C:\> C:\msys64\usr\bin\bash -lc "./configure --with-download-deps"
```

Install type

The source install process supports the following installation types:

- **system-sandbox - (Default)** Creates an isolated Python virtual environment, installs the AWS CLI into the virtual environment, and symlinks to the `aws` and `aws_completer` executable in the virtual environment. This install of the AWS CLI depends directly on the selected Python interpreter for its runtime.

This is a lightweight install mechanism to get the AWS CLI installed on a system and follows best Python practices by sandboxing the installation in a virtual environment. This installation is intended for customers that want to install the AWS CLI from source in the most frictionless way possible with the installation coupled to your installation of Python.

- **portable-exe** - Freezes the AWS CLI into a standalone executable that can be distributed to environments of similar architectures. This is the same process used to generate the official pre-built executables of the AWS CLI. The portable-exe freezes in a copy of the Python interpreter chosen in the `configure` step to use for the runtime of the AWS CLI. This allows it to be moved to other machines that may not have a Python interpreter.

This type of builds is useful because you can ensure your AWS CLI installation isn't coupled to the environment's installed Python version and you can distribute a build to other system that may not already have Python installed. This enables you to control the dependencies and security on the AWS CLI executables you use.

To configure the installation type, use the `--with-install-type` option and specify a value of `portable-exe` or `system-sandbox`.

The following `./configure` command example specifies a value of `portable-exe`:

Linux and macOS

```
$ ./configure --with-install-type=portable-exe
```

Windows PowerShell

```
PS C:\> C:\msys64\usr\bin\bash -lc "./configure --with-install-type=portable-exe"
```

Step 3: Building the AWS CLI

Use the `make` command to build the AWS CLI using your configuration settings:

Linux and macOS

```
$ make
```

Windows PowerShell

```
PS C:\> C:\msys64\usr\bin\bash -lc "make"
```

Note

When using the make command, the following steps are completed behind the scenes:

1. A virtual environment is created in the build directory using the Python [venv](#) module. The virtual environment is bootstrapped with a [version of pip that is vendored in the Python standard library](#).
2. Copies Python library dependencies. Depending on if the `--with-download-deps` flag is specified in the `configure` command, this step does one of the following:
 - The `--with-download-deps` is specified. Python dependencies are pip installed. This includes wheel, setuptools, and all AWS CLI runtime dependencies. If you are building the portable-exe, pyinstaller is installed. These requirements are all specified in lock files generated from [pip-compile](#).
 - The `--with-download-deps` is not specified. Python libraries from the Python interpreter's site package plus any scripts (e.g. pyinstaller) are copied into the virtual environment being used for the build.
3. Runs `pip install` directly on the AWS CLI codebase to do an offline, in-tree build and install of the AWS CLI into the build virtual environment. This install uses the pip flags [--no-build-isolation](#), [--use-feature=in-tree-build](#), [--no-cache-dir](#), and [--no-index](#).
4. **(Optional)** If the `--install-type` is set to `portable-exe` in the `configure` command, builds a standalone executable using [pyinstaller](#).

Step 4: Installing the AWS CLI

The `make install` command installs your built AWS CLI to the configured location on the system.

Linux and macOS

The following command example installs the AWS CLI using your configuration and build settings:

```
$ make install
```

Windows PowerShell

The following command example installs the AWS CLI using your configuration and build settings, then adds an environment variable with the path for the AWS CLI:

```
PS C:\> C:\msys64\usr\bin\bash -lc " make install "
PS C:\> $Env: PATH +=";C:\Program Files\AWSCLI\bin\"
```

The `make install` rule supports the [DESTDIR](#) variable. When specified, this variable prefixes the specified path to the already configured installation path when installing the AWS CLI. By default, no value is set for this variable.

Linux and macOS

The following code example uses a `--prefix=/usr/local` flag for configuring an install location, and then alters that destination using `DESTDIR=/tmp/stage` for the `make install` command. These commands result in the AWS CLI being installed at `/tmp/stage/usr/local/lib/aws-cli` and its executables located in `/tmp/stage/usr/local/bin`.

```
$ ./configure --prefix=/usr/local
$ make
$ make DESTDIR=/tmp/stage install
```

Windows PowerShell

The following code example uses a `--prefix=\awscli` flag for configuring an install location, and then alters that destination using `DESTDIR=C:\Program Files` for the `make install` command. These commands result in the AWS CLI being installed at `C:\Program Files\awscli`.

```
$ ./configure --prefix=\awscli
$ make
```

```
$ make DESTDIR='C:\Program Files' install
```

Note

When running `make install`, the following steps are completed behind the scenes

1. Moves one of the following to the configured install directory:
 - If the install type is system-sandbox, moves your built virtual environment.
 - If the install type is a portable-exe, moves your built standalone executable.
2. Creates symlinks for both the `aws` and `aws_completer` executables in your configured bin directory.

Step 5: Verifying the AWS CLI installation

Confirm the AWS CLI successfully installed by using the following command:

```
$ aws --version
aws-cli/2.15.19 Python/3.11.6 Windows/10 exe/AMD64 prompt/off
```

If the `aws` command is not recognized, you may need to restart your terminal for new symlinks to update. If you come across additional issues after installing or uninstalling the AWS CLI, see [Troubleshoot errors](#) for common troubleshooting steps

Workflow examples

This section provides some basic workflow examples for installing from source.

Basic Linux and macOS install

The following example is a basic installation workflow where the AWS CLI is installed in the default location of `/usr/local/lib/aws-cli`.

```
$ cd path/to/cli/repository
$ ./configure
$ make
$ make install
```

Automated Windows install

Note

You must run PowerShell as an Administrator to use this workflow.

MSYS2 can be used in an automated fashion in a CI setting, see [Using MSYS2 in CI](#) in the *MSYS2 Documentation*.

Downloaded Tarball

Download the `awscli.tar.gz` file, extract, and install the AWS CLI. When using the following commands, replace the following paths:

- `C:\msys64\usr\bin\bash` with the location of your MSYS2 path.
- `.\awscli-2.x.x\` with the extracted `awscli.tar.gz` folder name.
- `PYTHON='C:\path\to\python.exe'` with your local Python path.

The following code example automates building and installing the AWS CLI from PowerShell using MSYS2, and specifies which local install of Python to use:

```
PS C:\> curl -o awscli.tar.gz https://awscli.amazonaws.com/awscli.tar.gz #  
Download the awscli.tar.gz file in the current working directory  
PS C:\> tar -xvzf ./awscli.tar.gz # Extract awscli.tar.gz file  
PS C:\> cd .\awscli-2.x.x\ # Navigate to the root of the extracted files  
PS C:\> $env:CHERE_INVOKING = 'yes' # Preserve the current working directory  
PS C:\> C:\msys64\usr\bin\bash -lc "PYTHON='C:\path\to\python.exe' ./configure --  
prefix='C:\Program Files\AWSCLI' --with-download-deps"  
PS C:\> C:\msys64\usr\bin\bash -lc "make"  
PS C:\> C:\msys64\usr\bin\bash -lc "make install"  
PS C:\> $Env:PATH +=";C:\Program Files\AWSCLI\bin\"  
PS C:\> aws --version  
aws-cli/2.15.19 Python/3.11.6 Windows/10 source-sandbox/AMD64 prompt/off
```

GitHub Repository

Download the `awscli.tar.gz` file, extract, and install the AWS CLI. When using the following commands, replace the following paths:

- C:\msys64\usr\bin\bash with the location of your MSYS2 path.
- C:\path\to\cli\repository\ with the path to your cloned [AWS CLI repository](#) from GitHub. For more information, see [Fork a repo](#) in the *GitHub Docs*
- PYTHON='C:\path\to\python.exe' with your local Python path.

The following code example automates building and installing the AWS CLI from PowerShell using MSYS2, and specifies which local install of Python to use:

```
PS C:\> cd C:\path\to\cli\repository\  
PS C:\> $env:CHERE_INVOKING = 'yes' # Preserve the current working directory  
PS C:\> C:\msys64\usr\bin\bash -lc "PYTHON='C:\path\to\python.exe' ./configure --prefix='C:\Program Files\AWSCLI' --with-download-deps"  
PS C:\> C:\msys64\usr\bin\bash -lc "make"  
PS C:\> C:\msys64\usr\bin\bash -lc "make install"  
PS C:\> $Env:PATH +=";C:\Program Files\AWSCLI\bin\"  
PS C:\> aws --version
```

Alpine Linux container

Below is an example Dockerfile that can be used to get a working installation of the AWS CLI in an Alpine Linux container as an [alternative to pre-built binaries for Alpine](#). When using this example, replace **AWSCLI_VERSION** with you desired AWS CLI version number:

```
FROM python:3.8-alpine AS builder  
  
ENV AWSCLI_VERSION=2.10.1  
  
RUN apk add --no-cache \  
    curl \  
    make \  
    cmake \  
    gcc \  
    g++ \  
    libc-dev \  
    libffi-dev \  
    openssl-dev \  
    && curl https://awscli.amazonaws.com/awscli-$AWSCLI_VERSION.tar.gz | tar -xz \  
    && cd awscli-$AWSCLI_VERSION \  
    && ./configure --prefix=/opt/aws-cli/ --with-download-deps \  
    && make install
```

```
&& make \
&& make install

FROM python:3.8-alpine

RUN apk --no-cache add groff

COPY --from=builder /opt/aws-cli/ /opt/aws-cli/

ENTRYPOINT ["/opt/aws-cli/bin/aws"]
```

This image is built and the AWS CLI invoked from a container similar to the one that is built on Amazon Linux 2:

```
$ docker build --tag awscli-alpine .
$ docker run --rm -it awscli-alpine --version
aws-cli/2.2.1 Python/3.8.11 Linux/5.10.25-linuxkit source-sandbox/x86_64.alpine.3
prompt/off
```

The final size of this image is smaller than the size of the official AWS CLI Docker image. For information on the official Docker image, see [the section called “Amazon ECR Public/Docker”](#).

Troubleshooting AWS CLI install and uninstall errors

For troubleshooting steps for install errors, see [Troubleshoot errors](#) for common troubleshooting steps. For the most relevant troubleshooting steps, see [the section called “Command not found errors”](#), [the section called “The “aws --version” command returns a different version than you installed”](#), and [the section called “The “aws --version” command returns a version after uninstalling the AWS CLI”](#).

For any issues not covered in the troubleshooting guides, search the issues with the source-distribution label in the [AWS CLI Repository](#) on GitHub. If no existing issues cover your errors, [create a new issue](#) to receive help from the AWS CLI maintainers.

Next steps

After installing the AWS CLI, you should perform a [the section called “Setup”](#).

Run the AWS CLI from the official Amazon ECR Public or Docker images

This topic describes how to run, version control, and configure the AWS CLI version 2 on Docker using either the official Amazon Elastic Container Registry Public (Amazon ECR Public) or Docker Hub image. For more information on how to use Docker, see [Docker's documentation](#).

Official images provide isolation, portability, and security that AWS directly supports and maintains. This enables you to use the AWS CLI version 2 in a container-based environment without having to manage the installation yourself.

Topics

- [Prerequisites](#)
- [Deciding between Amazon ECR Public and Docker Hub](#)
- [Run the official AWS CLI version 2 images](#)
- [Notes on interfaces and backwards compatibility of the official images](#)
- [Use specific versions and tags](#)
- [Update to the latest official image](#)
- [Share host files, credentials, environment variables, and configuration](#)
- [Shorten the docker run command](#)

Prerequisites

You must have Docker installed. For installation instructions, see the [Docker website](#).

To verify your installation of Docker, run the following command and confirm there is an output.

```
$ docker --version  
Docker version 19.03.1
```

Deciding between Amazon ECR Public and Docker Hub

We recommend using Amazon ECR Public over Docker Hub for AWS CLI images. Docker Hub has stricter rate limiting for public consumers which can cause throttling issues. In addition, Amazon ECR Public replicates images in more than one region to provide strong availability and handle region outage issues.

For more information on Docker Hub rate limiting see [Understanding Docker Hub Rate Limiting](#) on the [Docker](#) website.

Run the official AWS CLI version 2 images

The first time you use the `docker run` command, the latest image is downloaded to your computer. Each subsequent use of the `docker run` command runs from your local copy.

To run the AWS CLI version 2 Docker images, use the `docker run` command.

Amazon ECR Public

The official AWS CLI version 2 Amazon ECR Public image is hosted on Amazon ECR Public in the [aws-cli/aws-cli repository](#).

```
$ docker run --rm -it public.ecr.aws/aws-cli/aws-cli command
```

Docker Hub

The official AWS CLI version 2 Docker image is hosted on Docker Hub in the `amazon/aws-cli` repository.

```
$ docker run --rm -it amazon/aws-cli command
```

This is how the command functions:

- `docker run --rm -it repository/name` – The equivalent of the `aws` executable. Each time you run this command, Docker spins up a container of your downloaded image, and executes your `aws` command. By default, the image uses the latest version of the AWS CLI version 2.

For example, to call the `aws --version` command in Docker, you run the following.

Amazon ECR Public

```
$ docker run --rm -it public.ecr.aws/aws-cli/aws-cli --version  
aws-cli/2.15.19 Python/3.7.3 Linux/4.9.184-linuxkit botocore/2.4.5dev10
```

Docker Hub

```
$ docker run --rm -it amazon/aws-cli --version
```

```
aws-cli/2.15.19 Python/3.7.3 Linux/4.9.184-linuxkit botocore/2.4.5dev10
```

- `--rm` – Specifies to clean up the container after the command exits.
- `-it` – Specifies to open a pseudo-TTY with `stdin`. This enables you to provide input to the AWS CLI version 2 while it's running in a container, for example, by using the `aws configure` and `aws help` commands. When choosing whether to omit `-it`, consider the following:
 - If you are running scripts, `-it` is not needed.
 - If you are experiencing errors with your scripts, omitting `-it` from your Docker call might fix the issue.
 - If you are trying to pipe output, `-it` might cause errors and omitting `-it` from your Docker call might resolve this issue. If you'd like to keep the `-it` flag, but still would like to pipe output, disabling the [client-side pager](#) the AWS CLI uses by default should resolve the issue.

For more information about the `docker run` command, see the [Docker reference guide](#).

Notes on interfaces and backwards compatibility of the official images

- The only tool supported on the image is the AWS CLI. Only the `aws` executable should ever be directly run. For example, even though `less` and `groff` are explicitly installed on the image, they should not be executed directly outside of an AWS CLI command.
- The `/aws` working directory is user controlled. The image will not write to this directory, unless instructed by the user in running an AWS CLI command.
- There are no backwards compatibility guarantees in relying on the latest tag. To guarantee backwards compatibility, you must pin to a specific `<major.minor.patch>` tag as those tags are immutable; they will only ever be pushed to once.

Use specific versions and tags

The official AWS CLI version 2 image has multiple versions you can use, starting with version `2.0.6`. To run a specific version of the AWS CLI version 2, append the appropriate tag to your `docker run` command. The first time you use the `docker run` command with a tag, the latest image for that tag is downloaded to your computer. Each subsequent use of the `docker run` command with that tag runs from your local copy.

You can use two types of tags:

- latest – Defines the latest version of the AWS CLI version 2 for the image. We recommend you use the latest tag when you want the latest version of the AWS CLI version 2. However, there are no backward-compatibility guarantees when relying on this tag. The latest tag is used by default in the docker run command. To explicitly use the latest tag, append the tag to the container image name.

Amazon ECR Public

```
$ docker run --rm -it public.ecr.aws/aws-cli/aws-cli:latest command
```

Docker Hub

```
$ docker run --rm -it amazon/aws-cli:latest command
```

- <major.minor.patch> – Defines a specific version of the AWS CLI version 2 for the image. If you plan to use an official image in production, we recommend you use a specific version of the AWS CLI version 2 to ensure backward compatibility. For example, to run version 2.0.6, append the version to the container image name.

Amazon ECR Public

```
$ docker run --rm -it public.ecr.aws/aws-cli/aws-cli:2.0.6 command
```

Docker Hub

```
$ docker run --rm -it amazon/aws-cli:2.0.6 command
```

Update to the latest official image

Because the latest image is downloaded to your computer only the first time you use the docker run command, you need to manually pull an updated image. To manually update to the latest version, we recommend you pull the latest tagged image. Pulling the image downloads the latest version to your computer.

Amazon ECR Public

```
$ docker pull public.ecr.aws/aws-cli/aws-cli:latest
```

Docker Hub

```
$ docker pull amazon/aws-cli:latest
```

Share host files, credentials, environment variables, and configuration

Because the AWS CLI version 2 is run in a container, by default the CLI can't access the host file system, which includes configuration and credentials. To share the host file system, credentials, and configuration to the container, mount the host system's `~/.aws` directory to the container at `/root/.aws` with the `-v` flag to the `docker run` command. This allows the AWS CLI version 2 running in the container to locate host file information.

Amazon ECR Public

Linux and macOS

```
$ docker run --rm -it -v ~/.aws:/root/.aws public.ecr.aws/aws-cli/aws-cli command
```

Windows Command Prompt

```
$ docker run --rm -it -v %userprofile%\.aws:/root/.aws public.ecr.aws/aws-cli/aws-cli command
```

Windows PowerShell

```
C:\> docker run --rm -it -v $env:userprofile\.aws:/root/.aws public.ecr.aws/aws-cli/aws-cli command
```

Docker Hub

Linux and macOS

```
$ docker run --rm -it -v ~/.aws:/root/.aws amazon/aws-cli command
```

Windows Command Prompt

```
$ docker run --rm -it -v %userprofile%\.aws:/root/.aws amazon/aws-cli command
```

Windows PowerShell

```
C:\> docker run --rm -it -v $env:UserProfile\.aws:/root/.aws amazon/aws-cli command
```

For more information about the `-v` flag and mounting, see the [Docker reference guide](#).

 **Note**

For information on config and credentials files, see [the section called “Configuration and credential file settings”](#).

Example 1: Providing credentials and configuration

In this example, we're providing host credentials and configuration when running the `s3 ls` command to list your buckets in Amazon Simple Storage Service (Amazon S3). The below examples use the default location for AWS CLI credentials and configuration files, to use a different location, change the file path.

Amazon ECR Public

Linux and macOS

```
$ docker run --rm -it -v ~/.aws:/root/.aws public.ecr.aws/aws-cli/aws-cli s3 ls
2020-03-25 00:30:48 aws-cli-docker-demo
```

Windows Command Prompt

```
$ docker run --rm -it -v %UserProfile%\aws:/root/.aws public.ecr.aws/aws-cli/aws-
cli s3 ls
2020-03-25 00:30:48 aws-cli-docker-demo
```

Windows PowerShell

```
C:\> docker run --rm -it -v $env:UserProfile\.aws:/root/.aws public.ecr.aws/aws-cli/
aws-cli s3 ls
```

Docker Hub

Linux and macOS

```
$ docker run --rm -it -v ~/.aws:/root/.aws amazon/aws-cli s3 ls  
2020-03-25 00:30:48 aws-cli-docker-demo
```

Windows Command Prompt

```
$ docker run --rm -it -v %userprofile%\aws:/root/.aws amazon/aws-cli s3 ls  
2020-03-25 00:30:48 aws-cli-docker-demo
```

Windows PowerShell

```
C:\> docker run --rm -it -v $env:userprofile\aws:/root/.aws amazon/aws-cli s3 ls
```

You can call specific system's environment variables using the `-e` flag. To use an environment variable, call it by name.

Amazon ECR Public

Linux and macOS

```
$ docker run --rm -it -v ~/.aws:/root/.aws -e ENVVAR_NAME public.ecr.aws/aws-cli/  
aws-cli s3 ls  
2020-03-25 00:30:48 aws-cli-docker-demo
```

Windows Command Prompt

```
$ docker run --rm -it -v %userprofile%\aws:/root/.aws -e ENVVAR_NAME  
public.ecr.aws/aws-cli/aws-cli s3 ls  
2020-03-25 00:30:48 aws-cli-docker-demo
```

Windows PowerShell

```
C:\> docker run --rm -it -v $env:userprofile\aws:/root/.aws -e ENVVAR_NAME  
public.ecr.aws/aws-cli/aws-cli s3 ls
```

Docker Hub

Linux and macOS

```
$ docker run --rm -it -v ~/.aws:/root/.aws -e ENVVAR_NAME amazon/aws-cli s3 ls
```

```
2020-03-25 00:30:48 aws-cli-docker-demo
```

Windows Command Prompt

```
$ docker run --rm -it -v %userprofile%\.aws:/root/.aws -e ENVVAR_NAME amazon/aws-cli  
s3 ls
```

```
2020-03-25 00:30:48 aws-cli-docker-demo
```

Windows PowerShell

```
C:\> docker run --rm -it -v $env:userprofile\.aws:/root/.aws -e ENVVAR_NAME amazon/  
aws-cli s3 ls
```

Example 2: Downloading an Amazon S3 file to your host system

For some AWS CLI version 2 commands, you can read files from the host system in the container or write files from the container to the host system.

In this example, we download the S3 object s3://aws-cli-docker-demo/hello to your local file system by mounting the current working directory to the container's /aws directory. By downloading the hello object to the container's /aws directory, the file is saved to the host system's current working directory also.

Amazon ECR Public

Linux and macOS

```
$ docker run --rm -it -v ~/.aws:/root/.aws -v $(pwd):/aws public.ecr.aws/aws-cli/  
aws-cli s3 cp s3://aws-cli-docker-demo/hello .  
download: s3://aws-cli-docker-demo/hello to ./hello
```

Windows Command Prompt

```
$ docker run --rm -it -v %userprofile%\.aws:/root/.aws -v %cd%:/aws public.ecr.aws/  
aws-cli/aws-cli s3 cp s3://aws-cli-docker-demo/hello .  
download: s3://aws-cli-docker-demo/hello to ./hello
```

Windows PowerShell

```
C:\> docker run --rm -it -v $env:UserProfile\.aws:/root/.aws -v $pwd\aws:/aws public.ecr.aws/aws-cli/aws-cli s3 cp s3://aws-cli-docker-demo/hello .
```

Docker Hub

Linux and macOS

```
$ docker run --rm -it -v ~/.aws:/root/.aws -v $(pwd):/aws amazon/aws-cli s3 cp s3://aws-cli-docker-demo/hello .
download: s3://aws-cli-docker-demo/hello to ./hello
```

Windows Command Prompt

```
$ docker run --rm -it -v %UserProfile%\aws:/root/.aws -v %cd%:/aws amazon/aws-cli s3 cp s3://aws-cli-docker-demo/hello .
download: s3://aws-cli-docker-demo/hello to ./hello
```

Windows PowerShell

```
C:\> docker run --rm -it -v $env:UserProfile\.aws:/root/.aws -v $pwd\aws:/aws amazon/aws-cli s3 cp s3://aws-cli-docker-demo/hello .
```

To confirm the downloaded file exists in the local file system, run the following.

Linux and macOS

```
$ cat hello
Hello from Docker!
```

Windows PowerShell

```
$ type hello
Hello from Docker!
```

Example 3: Using your AWS_PROFILE environment variable

You can call specific system's environment variables using the `-e` flag. Call each environment variable you'd like to use. In this example, we're providing host credentials, configuration, and the

AWS_PROFILE environment variable when running the `s3 ls` command to list your buckets in Amazon Simple Storage Service (Amazon S3).

Amazon ECR Public

Linux and macOS

```
$ docker run --rm -it -v ~/.aws:/root/.aws -e AWS_PROFILE public.ecr.aws/aws-cli/aws-cli s3 ls  
2020-03-25 00:30:48 aws-cli-docker-demo
```

Windows Command Prompt

```
$ docker run --rm -it -v %userprofile%\aws:/root/.aws -e AWS_PROFILE public.ecr.aws/aws-cli/aws-cli s3 ls  
2020-03-25 00:30:48 aws-cli-docker-demo
```

Windows PowerShell

```
C:\> docker run --rm -it -v $env:userprofile\aws:/root/.aws -e AWS_PROFILE public.ecr.aws/aws-cli/aws-cli s3 ls
```

Docker Hub

Linux and macOS

```
$ docker run --rm -it -v ~/.aws:/root/.aws -e AWS_PROFILE amazon/aws-cli s3 ls  
2020-03-25 00:30:48 aws-cli-docker-demo
```

Windows Command Prompt

```
$ docker run --rm -it -v %userprofile%\aws:/root/.aws -e AWS_PROFILE amazon/aws-cli s3 ls  
2020-03-25 00:30:48 aws-cli-docker-demo
```

Windows PowerShell

```
C:\> docker run --rm -it -v $env:userprofile\aws:/root/.aws -e AWS_PROFILE amazon/aws-cli s3 ls
```

Shorten the docker run command

To shorten the `docker run` command, we suggest you use your operating system's ability to create a [symbolic link](#) (symlink) or [alias](#) in Linux and macOS, or [doskey](#) in Windows. To set the aws alias, you can run one of the following commands.

- For basic access to aws commands, run the following.

Amazon ECR Public

Linux and macOS

```
$ alias aws='docker run --rm -it public.ecr.aws/aws-cli/aws-cli'
```

Windows Command Prompt

```
C:\> doskey aws=docker run --rm -it public.ecr.aws/aws-cli/aws-cli $*
```

Windows PowerShell

```
C:\> Function AWSCLI {docker run --rm -it public.ecr.aws/aws-cli/aws-cli $args}
Set-Alias -Name aws -Value AWSCLI
```

Docker Hub

Linux and macOS

```
$ alias aws='docker run --rm -it amazon/aws-cli'
```

Windows Command Prompt

```
C:\> doskey aws=docker run --rm -it amazon/aws-cli $*
```

Windows PowerShell

```
C:\> Function AWSCLI {docker run --rm -it amazon/aws-cli $args}
Set-Alias -Name aws -Value AWSCLI
```

- For access to the host file system and configuration settings when using aws commands, run the following.

Amazon ECR Public

Linux and macOS

```
$ alias aws='docker run --rm -it -v ~/.aws:/root/.aws -v $(pwd):/aws
public.ecr.aws/aws-cli/aws-cli'
```

Windows Command Prompt

```
C:\> doskey aws=docker run --rm -it -v %userprofile%\aws:/root/.aws -v %cd%:/aws
public.ecr.aws/aws-cli/aws-cli $*
```

Windows PowerShell

```
C:\> Function AWSCLI {docker run --rm -it -v $env:userprofile\.aws:/root/.aws -v
$pwd\aws:/aws public.ecr.aws/aws-cli/aws-cli $args}
Set-Alias -Name aws -Value AWSCLI
```

Docker Hub

Linux and macOS

```
$ alias aws='docker run --rm -it -v ~/.aws:/root/.aws -v $(pwd):/aws amazon/aws-
cli'
```

Windows Command Prompt

```
C:\> doskey aws=docker run --rm -it -v %userprofile%\aws:/root/.aws -v %cd%:/aws
amazon/aws-cli $*
```

Windows PowerShell

```
C:\> Function AWSCLI {docker run --rm -it -v $env:userprofile\.aws:/root/.aws -v
$pwd\aws:/aws amazon/aws-cli $args}
Set-Alias -Name aws -Value AWSCLI
```

- To assign a specific version to use in your aws alias, append your version tag.

Shorten the docker run command

Amazon ECR Public

Linux and macOS

```
$ alias aws='docker run --rm -it -v ~/.aws:/root/.aws -v $(pwd):/aws public.ecr.aws/aws-cli/aws-cli:2.0.6'
```

Windows Command Prompt

```
C:\> doskey aws=docker run --rm -it -v %userprofile%\aws:/root/.aws -v %cd%:/aws public.ecr.aws/aws-cli/aws-cli:2.0.6 $*
```

Windows PowerShell

```
C:\> Function AWSCLI {docker run --rm -it -v $env:userprofile\aws:/root/.aws -v $pwd\aws:/aws public.ecr.aws/aws-cli/aws-cli:2.0.6 $args}  
Set-Alias -Name aws -Value AWSCLI
```

Docker Hub

Linux and macOS

```
$ alias aws='docker run --rm -it -v ~/.aws:/root/.aws -v $(pwd):/aws amazon/aws-cli:2.0.6'
```

Windows Command Prompt

```
C:\> doskey aws=docker run --rm -it -v %userprofile%\aws:/root/.aws -v %cd%:/aws amazon/aws-cli:2.0.6 $*
```

Windows PowerShell

```
C:\> Function AWSCLI {docker run --rm -it -v $env:userprofile\aws:/root/.aws -v $pwd\aws:/aws amazon/aws-cli:2.0.6 $args}  
Set-Alias -Name aws -Value AWSCLI
```

After setting your alias, you can run the AWS CLI version 2 from within a container as if it's installed on your host system.

```
$ aws --version  
aws-cli/2.15.19 Python/3.7.3 Linux/4.9.184-linuxkit botocore/2.4.5dev10
```

Set up the AWS CLI

This topic explains how to quickly configure basic 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.

Topics

- [Gather your credential information for programmatic access](#)
- [Setting up new configuration and credentials](#)
- [Using existing configuration and credentials files](#)

Gather your credential information for programmatic access

You'll need programmatic access if you want to interact with AWS outside of the AWS Management Console. For authentication and credential instructions, choose one of the following options:

Which user needs programmatic access?	Purpose	Instructions
Workforce identity (AWS IAM Identity Center users)	(Recommended) Use short-term credentials.	the section called "IAM Identity Center authentication"
IAM	Use short-term credentials.	the section called "Short-term credentials"
IAM or Workforce identity (AWS IAM Identity Center users)	Use Amazon EC2 instance metadata for credentials.	the section called "Use credentials for Amazon EC2 instance metadata"
IAM	Pair another credential method and assume a role for permissions.	the section called "IAM roles"

Which user needs programmatic access?	Purpose	Instructions
or Workforce identity (AWS IAM Identity Center users)		
IAM	(Not recommended) Use long-term credentials.	the section called "IAM users"
IAM or Workforce identity (AWS IAM Identity Center users)	(Not recommended) Pair another credential method but use credential values stored in a location outside of the AWS CLI.	the section called "External credentials"

Setting up new configuration and credentials

The AWS CLI stores your configuration and credential information in a *profile* (a collection of settings) in the credentials and config files.

There are primarily two methods to quickly get setup:

- [Configuring using AWS CLI commands](#)
- [Manually editing the credentials and config files](#)

The following examples use sample values for each of the authentication methods. Replace sample values with your own.

Configuring using AWS CLI commands

For general use, the `aws configure` or `aws configure sso` commands in your preferred terminal are the fastest way to set up your AWS CLI installation. Based on the credential method you prefer, the AWS CLI prompts you for the relevant information. 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 credentials and config files, see [Configuration and credential file settings](#).

IAM Identity Center (SSO)

This example is for AWS IAM Identity Center using the `aws configure sso` wizard. For more information, see [the section called "Configure automatic token refresh".](#)

```
$ aws configure sso
SSO session name (Recommended): my-sso
SSO start URL [None]: https://my-sso-portal.awsapps.com/start
SSO region [None]:us-east-1
```

Attempting to automatically open the SSO authorization page in your default browser.

There are 2 AWS accounts available to you.

```
> DeveloperAccount, developer-account-admin@example.com (111122223333)
ProductionAccount, production-account-admin@example.com (444455556666)
```

Using the account ID **111122223333**

There are 2 roles available to you.

```
> ReadOnly
FullAccess
```

Using the role name "ReadOnly"

```
CLI default client Region [None]: us-west-2
CLI default output format [None]: json
CLI profile name [123456789011_ReadOnly]: user1
```

IAM Identity Center (Legacy SSO)

This example is for the legacy method of AWS IAM Identity Center using the `aws configure sso` wizard. To use the legacy SSO, leave the session name blank. For more information, see [the section called "Configure legacy non-refreshable".](#)

```
$ aws configure sso
SSO session name (Recommended):
SSO start URL [None]: https://my-sso-portal.awsapps.com/start
SSO region [None]:us-east-1
```

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

There are 2 AWS accounts available to you.

```
> DeveloperAccount, developer-account-admin@example.com (111122223333)
  ProductionAccount, production-account-admin@example.com (444455556666)
```

Using the account ID **111122223333**

There are 2 roles available to you.

```
> ReadOnly
  FullAccess
```

Using the role name "ReadOnly"

```
CLI default client Region [None]: us-west-2
CLI default output format [None]: json
CLI profile name [123456789011_ReadOnly]: user1
```

Short-term credentials

This example is for the short-term credentials from AWS Identity and Access Management. The aws configure wizard is used to set initial values and then the aws configure set command assigns the last value needed. For more information, see [the section called "Short-term credentials".](#)

```
$ 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
$ aws configure set
aws_session_token fcZib3JpZ2luX2I0oJb3JpZ2luX2I0oJb3JpZ2luX2I0oJb3JpZ2luX2I0oJb3JpZVERYLONG
```

IAM role

This example is for assuming an IAM role. Profiles that use IAM roles pull credentials from another profile, and then apply IAM role permissions. In the following examples, default is the source profile for credentials and user1 borrows the same credentials then assumes a new role. There is no wizard for this process, therefore each value is set using the aws configure set command. For more information, see [the section called "IAM roles".](#)

```
$ aws configure set role_arn arn:aws:iam::123456789012:role/defaultrole
$ aws configure set source_profile default
$ aws configure set role_session_name session_user1
$ aws configure set region us-west-2
```

```
$ aws configure set output json
```

Amazon EC2 instance metadata credentials

This example is for the credentials obtained from the hosting Amazon EC2 instance metadata. There is no wizard for this process, therefore each value is set using the `aws configure` set command. For more information, see [the section called “Use credentials for Amazon EC2 instance metadata”](#).

```
$ aws configure set role_arn arn:aws:iam::123456789012:role/defaultrole  
$ aws configure set credential_source Ec2InstanceMetadata  
$ aws configure set region us-west-2  
$ aws configure set output json
```

Long-term credentials

Warning

To avoid security risks, don't use IAM users for authentication when developing purpose-built software or working with real data. Instead, use federation with an identity provider such as [AWS IAM Identity Center](#).

This example is for the long-term credentials from AWS Identity and Access Management. For more information, see [the section called “IAM users”](#).

```
$ 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 detailed information on authentication and credential methods see [Authentication and access credentials](#).

Manually editing the credentials and config files

When copy and pasting information, we suggest manually editing the config and credentials file. Based on the credential method you prefer, the files are setup in a different way.

The files are stored in your home directory under the . aws folder. Where you find your home directory location varies based on the operating system, but is referred to using the environment variables %UserProfile% in Windows and \$HOME or ~ (tilde) in Unix-based systems. For more information on where these settings are stored, see [the section called “Where are configuration settings stored?”](#).

The following examples show a default profile and a profile named user1 and use sample values. Replace sample values with your own. For more information on the credentials and config files, see [Configuration and credential file settings](#).

IAM Identity Center (SSO)

This example is for AWS IAM Identity Center. For more information, see [the section called “Configure automatic token refresh”](#).

Credentials file

The credentials file is not used for this authentication method.

Config file

```
[default]
sso_session = my-sso
sso_account_id = 111122223333
sso_role_name = readOnly
region = us-west-2
output = text

[profile user1]
sso_session = my-sso
sso_account_id = 444455556666
sso_role_name = readOnly
region = us-east-1
output = json

[sso-session my-sso]
sso_region = us-east-1
sso_start_url = https://my-sso-portal.awsapps.com/start
sso_registration_scopes = sso:account:access
```

IAM Identity Center (Legacy SSO)

This example is for the legacy method of AWS IAM Identity Center. For more information, see [the section called “Configure legacy non-refreshable”](#).

Credentials file

The credentials file is not used for this authentication method.

Config file

```
[default]
sso_start_url = https://my-sso-portal.awsapps.com/start
sso_region = us-east-1
sso_account_id = 111122223333
sso_role_name = readOnly
region = us-west-2
output = text

[profile user1]
sso_start_url = https://my-sso-portal.awsapps.com/start
sso_region = us-east-1
sso_account_id = 444455556666
sso_role_name = readOnly
region = us-east-1
output = json
```

Short-term credentials

This example is for the short-term credentials from AWS Identity and Access Management. For more information, see [the section called “Short-term credentials”](#).

Credentials file

```
[default]
aws_access_key_id=ASIAIOSFODNN7EXAMPLE
aws_secret_access_key=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
aws_session_token
= IQoJb3JpZ2luX2IQtJb3JpZ2luX2IQtJb3JpZ2luX2IQtJb3JpZ2luX2IQtJb3JpZVERYLONGSTRINGEXAMPLE

[user1]
aws_access_key_id=ASIAI44QH8DHBEXAMPLE
aws_secret_access_key=je7MtGbClwBF/2Zp9Utk/h3yCo8nvbEXAMPLEKEY
```

```
aws_session_token
= fcZib3JpZ2luX2IQuJb3JpZ2luX2IQuJb3JpZ2luX2IQuJb3JpZ2luX2IQuJb3JpZVERYLONGSTRINGEXAMPLE
```

Config file

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

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

IAM role

This example is for assuming an IAM role. Profiles that use IAM roles pull credentials from another profile, and then apply IAM role permissions. In the following examples, default is the source profile for credentials and user1 borrows the same credentials then assumes a new role. For more information, see [the section called “IAM roles”](#).

Credentials file

The credentials file depends on what authentication your source profile uses. For the following example, the source profile uses short-term credentials.

```
[default]
aws_access_key_id=ASIAIOSFODNN7EXAMPLE
aws_secret_access_key=wJa1rXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
aws_session_token
= IQoJb3JpZ2luX2IQuJb3JpZ2luX2IQuJb3JpZ2luX2IQuJb3JpZ2luX2IQuJb3JpZVERYLONGSTRINGEXAMPLE
```

Config file

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

[profile user1]
role_arn=arn:aws:iam::777788889999:role/user1role
source_profile=default
role_session_name=session_user1
region=us-east-1
```

```
output=text
```

Amazon EC2 instance metadata credentials

This example is for the credentials obtained from the hosting Amazon EC2 instance metadata. For more information, see [the section called “Use credentials for Amazon EC2 instance metadata”](#).

Credentials file

The `credentials` file is not used for this authentication method.

Config file

```
[default]
role_arn=arn:aws:iam::123456789012:role/defaultrole
credential_source=Ec2InstanceMetadata
region=us-west-2
output=json

[profile user1]
role_arn=arn:aws:iam::777788889999:role/user1role
credential_source=Ec2InstanceMetadata
region=us-east-1
output=text
```

Long-term credentials

Warning

To avoid security risks, don't use IAM users for authentication when developing purpose-built software or working with real data. Instead, use federation with an identity provider such as [AWS IAM Identity Center](#).

This example is for the long-term credentials from AWS Identity and Access Management. For more information, see [the section called “IAM users”](#).

Credentials file

```
[default]
```

```
aws_access_key_id=AKIAIOSFODNN7EXAMPLE
aws_secret_access_key=wJa1rXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY

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

Config file

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

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

For more detailed information on authentication and credential methods see [Authentication and access credentials](#).

Using existing configuration and credentials files

If you have existing configuration and credentials files, these can be used for the AWS CLI.

To use the config and credentials files, move them to the folder named .aws in your home directory. Where you find your home directory location varies based on the operating system, but is referred to using the environment variables %UserProfile% in Windows and \$HOME or ~ (tilde) in Unix-based systems.

You can specify a non-default location for the config and credentials 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](#) for details.

For more detailed information on configuration and credentials files, see [the section called "Configuration and credential file settings"](#).

Configure 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 the following:

- **Credentials** identify who is calling the API. Access credentials are used to encrypt the request to the AWS servers to confirm your identity and retrieve associated permissions policies. These permissions determine the actions you can perform. For information on setting up your credentials, see [Authentication and access credentials](#).
- **Other configuration details** to tell the AWS CLI how to process requests, such as 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.

Configuration and credentials precedence

Credentials and configuration settings are 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](#)** – Overrides settings in any other location, such as the `--region`, `--output`, and `--profile` parameters.
2. **[Environment variables](#)** – You can store values in your system's environment variables.
3. **[Assume role](#)** – Assume the permissions of an IAM role through configuration or the [`aws sts assume-role`](#) command.
4. **[Assume role with web identity](#)** – Assume the permissions of an IAM role using web identity through configuration or the [`aws sts assume-role`](#) command.

5. [**AWS IAM Identity Center**](#) – The IAM Identity Center configuration settings are stored in the config file. Credentials are authenticated when you run the `aws configure sso` command. The config file is located at `~/.aws/config` on Linux or macOS, or at `C:\Users\USERNAME\.aws\config` on Windows.
6. [**Credentials file**](#) – 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.
7. [**Custom process**](#) – Get your credentials from an external source.
8. [**Configuration file**](#) – 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.
9. [**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*.
- 10 [**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*.

Additional topics in this section

- [the section called “Configuration and credential file settings”](#)
- [the section called “Environment Variables”](#)
- [the section called “Command line options”](#)
- [the section called “Command completion”](#)
- [the section called “Retries”](#)
- [the section called “Use an HTTP proxy”](#)

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.

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 [Configure the AWS CLI](#).

 **Note**

For information on setting up your credentials, see [Authentication and access credentials](#).

Topics

- [Format of the configuration and credential files](#)
- [Where are configuration settings stored?](#)
- [Using named profiles](#)
- [Set and view configuration settings using commands](#)
- [Setting new configuration and credentials command examples](#)
- [Supported config file settings](#)

Format of the configuration and credential files

The `config` and `credentials` files are organized into sections. Sections include *profiles*, *sso-sessions*, and *services*. A section is a named collection of settings, and continues until another section definition line is encountered. Multiple profiles and sections can be stored in the `config` and `credentials` files.

These files are plaintext files that use the following format:

- Section names are enclosed in brackets [] such as `[default]`, `[profile user1]`, and `[sso-session]`.
- All entries in a section take the general form of `setting_name=value`.

- Lines can be commented out by starting the line with a hash character (#).

The config and credentials files contain the following section types:

- [Section type: profile](#)
- [Section type: sso-session](#)
- [Section type: services](#)

Section type: profile

The AWS CLI stores

Depending on the file, profile section names use the following format:

- **Config file:** [default] [profile *user1*]
- **Credentials file:** [default] [*user1*]

Do **not** use the word **profile** when creating an entry in the **credentials** file.

Each profile can specify different credentials and can also specify different AWS Regions and output formats. When naming the profile in a config file, include the prefix word "profile", but do not include it in the credentials file.

The following examples show a **credentials** and **config** file with two profiles, region, and output specified. The first **[default]** is used when you run a AWS CLI command with no profile specified. The second is used when you run a AWS CLI command with the **--profile user1** parameter.

IAM Identity Center (SSO)

This example is for AWS IAM Identity Center. For more information, see [the section called "Configure automatic token refresh".](#)

Credentials file

The **credentials** file is not used for this authentication method.

Config file

[default]

```
sso_session = my-sso
sso_account_id = 111122223333
sso_role_name = readOnly
region = us-west-2
output = text

[profile user1]
sso_session = my-sso
sso_account_id = 444455556666
sso_role_name = readOnly
region = us-east-1
output = json

[sso-session my-sso]
sso_region = us-east-1
sso_start_url = https://my-sso-portal.awsapps.com/start
sso_registration_scopes = sso:account:access
```

IAM Identity Center (Legacy SSO)

This example is for the legacy method of AWS IAM Identity Center. For more information, see [the section called “Configure legacy non-refreshable”](#).

Credentials file

The credentials file is not used for this authentication method.

Config file

```
[default]
sso_start_url = https://my-sso-portal.awsapps.com/start
sso_region = us-east-1
sso_account_id = 111122223333
sso_role_name = readOnly
region = us-west-2
output = text

[profile user1]
sso_start_url = https://my-sso-portal.awsapps.com/start
sso_region = us-east-1
sso_account_id = 444455556666
sso_role_name = readOnly
region = us-east-1
```

```
output = json
```

Short-term credentials

This example is for the short-term credentials from AWS Identity and Access Management. For more information, see [the section called “Short-term credentials”](#).

Credentials file

```
[default]
aws_access_key_id=ASIAIOSFODNN7EXAMPLE
aws_secret_access_key=wJa1rXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
aws_session_token
  = IQoJb3JpZ2luX2IQtJb3JpZ2luX2IQtJb3JpZ2luX2IQtJb3JpZ2luX2IQtJb3JpZVERYLONGSTRINGEXAMPLE

[user1]
aws_access_key_id=ASIAI44QH8DHBEXAMPLE
aws_secret_access_key=je7MtGbClwBF/2Zp9Utk/h3yCo8nvbEXAMPLEKEY
aws_session_token
  = fcZib3JpZ2luX2IQtJb3JpZ2luX2IQtJb3JpZ2luX2IQtJb3JpZ2luX2IQtJb3JpZVERYLONGSTRINGEXAMPLE
```

Config file

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

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

IAM role

This example is for assuming an IAM role. Profiles that use IAM roles pull credentials from another profile, and then apply IAM role permissions. In the following examples, default is the source profile for credentials and user1 borrows the same credentials then assumes a new role. For more information, see [the section called “IAM roles”](#).

Credentials file

The credentials file depends on what authentication your source profile uses. For the following example, the source profile uses short-term credentials.

Config file

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

[profile user1]
role_arn=arn:aws:iam::777788889999:role/user1role
source_profile=default
role_session_name=session_user1
region=us-east-1
output=text
```

Amazon EC2 instance metadata credentials

This example is for the credentials obtained from the hosting Amazon EC2 instance metadata. For more information, see [the section called “Use credentials for Amazon EC2 instance metadata”](#).

Credentials file

The credentials file is not used for this authentication method.

Config file

```
[default]
role_arn=arn:aws:iam::123456789012:role/defaultrole
credential_source=Ec2InstanceMetadata
region=us-west-2
output=json

[profile user1]
role_arn=arn:aws:iam::777788889999:role/user1role
credential_source=Ec2InstanceMetadata
region=us-east-1
output=text
```

Long-term credentials

⚠ Warning

To avoid security risks, don't use IAM users for authentication when developing purpose-built software or working with real data. Instead, use federation with an identity provider such as [AWS IAM Identity Center](#).

This example is for the long-term credentials from AWS Identity and Access Management. For more information, see [the section called “IAM users”](#).

Credentials file

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

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

Config file

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

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

For more information and additional authorization and credential methods see, see [the section called “IAM users”](#).

Section type: sso-session

The sso-session section of the config file is used to group configuration variables for acquiring SSO access tokens, which can then be used to acquire AWS credentials. The following settings are used:

- **(Required)** sso_start_url
- **(Required)** sso_region
- sso_account_id
- sso_role_name
- sso_registration_scopes

You define an sso-session section and associate it to a profile. `sso_region` and `sso_start_url` must be set within the sso-session section. Typically, `sso_account_id` and `sso_role_name` must be set in the profile section so that the SDK can request SSO credentials.

The following example configures the SDK to request SSO credentials and supports automated token refresh:

```
[profile dev]
sso_session = my-sso
sso_account_id = 111122223333
sso_role_name = SampleRole

[sso-session my-sso]
sso_region = us-east-1
sso_start_url = https://my-sso-portal.awsapps.com/start
```

This also allows sso-session configurations to be reused across multiple profiles:

```
[profile dev]
sso_session = my-sso
sso_account_id = 111122223333
sso_role_name = SampleRole

[profile prod]
sso_session = my-sso
sso_account_id = 111122223333
sso_role_name = SampleRole2

[sso-session my-sso]
sso_region = us-east-1
sso_start_url = https://my-sso-portal.awsapps.com/start
```

However, `sso_account_id` and `sso_role_name` aren't required for all scenarios of SSO token configuration. If your application only uses AWS services that support bearer authentication, then traditional AWS credentials are not needed. Bearer authentication is an HTTP authentication scheme that uses security tokens called bearer tokens. In this scenario, `sso_account_id` and `sso_role_name` aren't required. See the individual guide for your AWS service to determine if it supports bearer token authorization.

Additionally, registration scopes can be configured as part of a `sso-session`. Scope is a mechanism in OAuth 2.0 to limit an application's access to a user's account. An application can request one or more scopes, and the access token issued to the application will be limited to the scopes granted. These scopes define the permissions requested to be authorized for the registered OIDC client and access tokens retrieved by the client. The following example sets `sso_registration_scopes` to provide access for listing accounts/roles:

```
[sso-session my-sso]
sso_region = us-east-1
sso_start_url = https://my-sso-portal.awsapps.com/start
sso_registration_scopes = sso:account:access
```

The authentication token is cached to disk under the `~/.aws/sso/cache` directory with a filename based on the session name.

For more information on this configuration type, see [the section called “Configure automatic token refresh”](#).

Section type: services

The `services` section is a group of settings that configures custom endpoints for AWS service requests. A profile then is linked to a `services` section.

```
[profile dev]
services = my-services
```

The `services` section is separated into subsections by `<SERVICE> =` lines, where `<SERVICE>` is the AWS service identifier key. The AWS service identifier is based on the API model's `serviceId` by replacing all spaces with underscores and lowercasing all letters. For a list of all service identifier keys to use in the `services` section, see [Use endpoints in the AWS CLI](#). The service identifier key is followed by nested settings with each on its own line and indented by two spaces.

The following example configures the endpoint to use for requests made to the Amazon DynamoDB service in the *my-services* section that is used in the *dev* profile. Any immediately following lines that are indented are included in that subsection and apply to that service.

```
[profile dev]
services = my-services

[services my-services]
dynamodb =
    endpoint_url = http://localhost:8000
```

For more information on service-specific endpoints, see [Use endpoints in the AWS CLI](#).

If your profile has role-based credentials configured through a `source_profile` parameter for IAM assume role functionality, the SDK only uses service configurations for the specified profile. It does not use profiles that are role chained to it. For example, using the following shared config file:

```
[profile A]
credential_source = Ec2InstanceMetadata
endpoint_url = https://profile-a-endpoint.aws/

[profile B]
source_profile = A
role_arn = arn:aws:iam::123456789012:role/roleB
services = profileB

[services profileB]
ec2 =
    endpoint_url = https://profile-b-ec2-endpoint.aws
```

If you use profile B and make a call in your code to Amazon EC2, the endpoint resolves as `https://profile-b-ec2-endpoint.aws`. If your code makes a request to any other service, the endpoint resolution will not follow any custom logic. The endpoint does not resolve to the global endpoint defined in profile A. For a global endpoint to take effect for profile B, you would need to set `endpoint_url` directly within profile B.

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. We suggest keeping credentials in the `credentials` files. 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](#) for details.

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.

Using named profiles

If no profile is explicitly defined, the default profile is used.

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.

```
$ 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 as the default profile. You can override this setting by using the `--profile` parameter.

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](#).

Set and view configuration settings using commands

There are several ways to view and set your configuration settings using commands.

[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/bPxRfCYEXAMPLEKEY
Default region name [None]: us-west-2
Default output format [None]: json
```

[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 import](#)

Import CSV credentials generated from the IAM web console. This is not for credentials generated from IAM Identity Center; customers who use IAM Identity Center should use aws configure sso. A CSV file is imported with the profile name matching the username. The CSV file must contain the following headers.

- User Name
- Access key ID
- Secret access key

Note

During initial key pair creation, once you close the **Download .csv file** dialog box, you cannot access your secret access key after you close the dialog box. If you need a .csv file, you'll need to create one yourself with the required headers and your stored key pair information. If you do not have access to your key pair information, you need to create a new key pair.

```
$ aws configure import --csv file://credentials.csv
```

[aws configure list](#)

To list configuration data, use the `aws configure list` command. This command lists the profile, access key, secret key, and region configuration information used for the specified profile. For each configuration item, it shows the value, where the configuration value was retrieved, and the configuration variable name.

For example, if you provide the AWS Region in an environment variable, this command shows you the name of the region you've configured, that this value came from an environment variable, and the name of the environment variable.

For temporary credential methods such as roles and IAM Identity Center, this command displays the temporarily cached access key and secret access key is displayed.

```
$ aws configure list
  Name          Value        Type    Location
  ----          ----        ----    -----
  profile       <not set>   None    None
  access_key    ****ABCD**** shared-credentials-file
  secret_key    ****ABCD**** shared-credentials-file
  region        us-west-2    env     AWS_DEFAULT_REGION
```

[aws configure list-profiles](#)

To list all your profile names, use the `aws configure list-profiles` command.

```
$ aws configure list-profiles
default
test
```

[aws configure sso](#)

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

```
$ aws configure sso
SSO session name (Recommended): my-sso
SSO start URL [None]: https://my-sso-portal.awsapps.com/start
SSO region [None]: us-east-1
```

```
SSO registration scopes [None]: sso:account:access
```

[**aws configure sso-session**](#)

Run this command to quickly set and view your AWS IAM Identity Center credentials, Region, and output format in the sso-session section of the credentials and config files. The following example shows sample values.

```
$ aws configure sso-session
SSO session name: my-sso
SSO start URL [None]: https://my-sso-portal.awsapps.com/start
SSO region [None]: us-east-1
SSO registration scopes [None]: sso:account:access
```

Setting new configuration and credentials command examples

The following examples show configuring a default profile with credentials, region, and output specified for different authentication methods.

IAM Identity Center (SSO)

This example is for AWS IAM Identity Center using the `aws configure sso` wizard. For more information, see [the section called “Configure automatic token refresh”](#).

```
$ aws configure sso
SSO session name (Recommended): my-sso
SSO start URL [None]: https://my-sso-portal.awsapps.com/start
SSO region [None]:us-east-1
```

Attempting to automatically open the SSO authorization page in your default browser.

There are 2 AWS accounts available to you.

```
> DeveloperAccount, developer-account-admin@example.com (111122223333)
  ProductionAccount, production-account-admin@example.com (444455556666)
```

Using the account ID **111122223333**

There are 2 roles available to you.

```
> ReadOnly
  FullAccess
```

Using the role name "ReadOnly"

CLI default client Region [None]: **us-west-2**
CLI default output format [None]: **json**
CLI profile name [123456789011_ReadOnly]: **user1**

IAM Identity Center (Legacy SSO)

This example is for the legacy method of AWS IAM Identity Center using the `aws configure sso` wizard. To use the legacy SSO, leave the session name blank. For more information, see [the section called "Configure legacy non-refreshable"](#).

```
$ aws configure sso
SSO session name (Recommended):
SSO start URL [None]: https://my-sso-portal.awsapps.com/start
SSO region [None]:us-east-1
```

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

There are 2 AWS accounts available to you.

> DeveloperAccount, developer-account-admin@example.com (**111122223333**)
ProductionAccount, production-account-admin@example.com (**44445556666**)

Using the account ID **111122223333**

There are 2 roles available to you.

> ReadOnly
FullAccess

Using the role name "ReadOnly"

CLI default client Region [None]: **us-west-2**
CLI default output format [None]: **json**
CLI profile name [123456789011_ReadOnly]: **user1**

Short-term credentials

This example is for the short-term credentials from AWS Identity and Access Management. The `aws configure` wizard is used to set initial values and then the `aws configure set` command assigns the last value needed. For more information, see [the section called "Short-term credentials"](#).

```
$ aws configure
AWS Access Key ID [None]: AKIAIOSFODNN7EXAMPLE
AWS Secret Access Key [None]: wJalrXUtnFEMI/K7MDENG/bPxRfCYEXAMPLEKEY
Default region name [None]: us-west-2
Default output format [None]: json
$ aws configure set
aws_session_token fcZib3JpZ2luX2I0oJb3JpZ2luX2I0oJb3JpZ2luX2I0oJb3JpZ2luX2I0oJb3JpZVERYLONG
```

IAM role

This example is for assuming an IAM role. Profiles that use IAM roles pull credentials from another profile, and then apply IAM role permissions. In the following examples, default is the source profile for credentials and user1 borrows the same credentials then assumes a new role. There is no wizard for this process, therefore each value is set using the `aws configure set` command. For more information, see [the section called “IAM roles”](#).

```
$ aws configure set role_arn arn:aws:iam::123456789012:role/defaultrole  
$ aws configure set source_profile default  
$ aws configure set role_session_name session_user1  
$ aws configure set region us-west-2  
$ aws configure set output json
```

Amazon EC2 instance metadata credentials

This example is for the credentials obtained from the hosting Amazon EC2 instance metadata. There is no wizard for this process, therefore each value is set using the `aws configure` set command. For more information, see [the section called “Use credentials for Amazon EC2 instance metadata”](#).

```
$ aws configure set role_arn arn:aws:iam::123456789012:role/defaultrole  
$ aws configure set credential_source Ec2InstanceMetadata  
$ aws configure set region us-west-2  
$ aws configure set output json
```

Long-term credentials

Warning

To avoid security risks, don't use IAM users for authentication when developing purpose-built software or working with real data. Instead, use federation with an identity provider such as [AWS IAM Identity Center](#).

This example is for the long-term credentials from AWS Identity and Access Management. For more information, see [the section called "IAM users"](#).

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

Supported config file settings

Topics

- [Global settings](#)
- [S3 Custom command settings](#)

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 [Configure the AWS CLI](#)

Global settings

aws_access_key_id

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

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 = wJa1rXUtnFEMI/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 = AQoEXAMPLEH4aoAH0gNCAPyJxz4B1CFFxWNE10PTgk5TthT  
+FvwqnKwRc0IfRh3c/LTo6UDdyJw00vEVpvLXCrrrUtdnniCEXAMPLE/  
IvU1dYUg2RAJBanLiHb4IgRmpRV3zrkuWJ0gQs8IZZaIv2BXIa2R40lgk
```

ca_bundle

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

Can be overridden by the [AWS_CA_BUNDLE](#) environment variable or the [--ca-bundle](#) command line option.

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

cli_auto_prompt

Enables the auto-prompt for the AWS CLI version 2. There are two settings that can be used:

- **on** uses the full auto-prompt mode each time you attempt to run an aws command. This includes pressing **ENTER** after both a complete command or incomplete command.

```
cli_auto_prompt = on
```

- **on-partial** uses partial auto-prompt mode. If a command is incomplete or cannot be run due to client-side validation errors, auto-prompt is used. This mode is particular useful if you have pre-existing scripts, runbooks, or you only want to be auto-prompted for commands you are unfamiliar with rather than prompted on every command.

```
cli_auto_prompt = on-partial
```

You can override this setting by using the [aws_cli_auto_prompt](#) environment variable or the [--cli-auto-prompt](#) and [--no-cli-auto-prompt](#) command line parameters.

For information on the AWS CLI version 2 auto-prompt feature, see [Have the AWS CLI prompt you for commands](#).

cli_binary_format

Specifies how the AWS CLI version 2 interprets binary input parameters. It can be one of the following values:

- **base64** – This is the default value. An input parameter that is typed as a binary large object (BLOB) accepts a base64-encoded string. To pass true binary content, put the content in a file and provide the file's path and name with the `fileb://` prefix as the parameter's value. To pass base64-encoded text contained in a file, provide the file's path and name with the `file://` prefix as the parameter's value.
- **raw-in-base64-out** – Default for the AWS CLI version 1. If the setting's value is `raw-in-base64-out`, files referenced using the `file://` prefix is read as text and then the AWS CLI attempts to encode it to binary.

This entry does not have an equivalent environment variable. You can specify the value on a single command by using the `--cli-binary-format raw-in-base64-out` parameter.

```
cli_binary_format = raw-in-base64-out
```

If you reference a binary value in a file using the `fileb://` prefix notation, the AWS CLI *always* expects the file to contain raw binary content and does not attempt to convert the value.

If you reference a binary value in a file using the `file://` prefix notation, the AWS CLI handles the file according to the current `cli_binary_format` setting. If that setting's value is `base64` (the default when not explicitly set), the AWS CLI expects the file to contain base64-encoded text. If that setting's value is `raw-in-base64-out`, the AWS CLI expects the file to contain raw binary content.

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_pager

Specifies the pager program used for output. By default, AWS CLI version 2 returns all output through your operating system's default pager program.

Can be overridden by the `AWS_PAGER` environment variable.

```
cli_pager=less
```

To disable all use of an external paging program, set the variable to an empty string as shown in the following example.

```
cli_pager=
```

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.

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

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

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

- **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](#)

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 [Source credentials with an external process](#).

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

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

[credential_source](#)

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.

endpoint_url

Specifies the endpoint that is used for all service requests. If this setting is used in the [services](#) section of the config file, then the endpoint is used only for the specified service.

The following example uses the global endpoint `http://localhost:1234` and a service-specific endpoint of `http://localhost:4567` for Amazon S3.

```
[profile dev]
endpoint_url = http://localhost:1234
services = s3-specific

[services s3-specific]
s3 =
  endpoint_url = http://localhost:4567
```

Endpoint configuration settings are 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. The AWS CLI endpoint configuration settings take precedence in the following order:

1. The [--endpoint-url](#) command line option.

2. If enabled, the [AWS_IGNORE_CONFIGURED_ENDPOINT_URLS](#) global endpoint environment variable or profile setting [ignore_configure_endpoint_urls](#) to ignore custom endpoints.
3. The value provided by a service-specific environment variable [AWS_ENDPOINT_URL_<SERVICE>](#), such as `AWS_ENDPOINT_URL_DYNAMODB`.
4. The values provided by the [AWS_USE_DUALSTACK_ENDPOINT](#), [AWS_USE_FIPS_ENDPOINT](#), and [AWS_ENDPOINT_URL](#) environment variables.
5. The service-specific endpoint value provided by the [endpoint_url](#) setting within a services section of the shared config file.
6. The value provided by the [endpoint_url](#) setting within a profile of the shared config file.
7. [use_dualstack_endpoint](#), [use_fips_endpoint](#), and [endpoint_url](#) settings.
8. Any default endpoint URL for the respective AWS service is used last. For a list of the standard service endpoints available in each Region, see [AWS Regions and Endpoints](#) in the *Amazon Web Services General Reference*.

ignore_configure_endpoint_urls

If enabled, the AWS CLI ignores all custom endpoint configurations specified in the config file. Valid values are **true** and **false**.

```
ignore_configure_endpoint_urls = true
```

Endpoint configuration settings are 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. The AWS CLI endpoint configuration settings take precedence in the following order:

1. The [--endpoint-url](#) command line option.
2. If enabled, the [AWS_IGNORE_CONFIGURED_ENDPOINT_URLS](#) global endpoint environment variable or profile setting [ignore_configure_endpoint_urls](#) to ignore custom endpoints.
3. The value provided by a service-specific environment variable [AWS_ENDPOINT_URL_<SERVICE>](#), such as `AWS_ENDPOINT_URL_DYNAMODB`.
4. The values provided by the [AWS_USE_DUALSTACK_ENDPOINT](#), [AWS_USE_FIPS_ENDPOINT](#), and [AWS_ENDPOINT_URL](#) environment variables.

5. The service-specific endpoint value provided by the [`endpoint_url`](#) setting within a services section of the shared config file.
6. The value provided by the [`endpoint_url`](#) setting within a profile of the shared config file.
7. [`use_dualstack_endpoint`](#), [`use_fips_endpoint`](#), and [`endpoint_url`](#) settings.
8. Any default endpoint URL for the respective AWS service is used last. For a list of the standard service endpoints available in each Region, see [AWS Regions and Endpoints](#) in the *Amazon Web Services General Reference*.

[external_id](#)

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 ID when granting access to your AWS resources to a third party](#) in the *IAM User Guide*.

[max_attempts](#)

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.

```
max_attempts = 3
```

[mfa_serial](#)

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](#)

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

- [`json`](#) – The output is formatted as a [`JSON`](#) string.
- [`yaml`](#) – The output is formatted as a [`YAML`](#) string.

- **yaml-stream** – The output is streamed and formatted as a [YAML](#) string. Streaming allows for faster handling of large data types.
- **text** – 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** – The output is formatted as a table using the characters +|- to form the cell borders. It typically presents the information in a "human-friendly" format that is much easier to read than the others, but not as programmatically useful.

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

```
output = table
```

parameter_validation

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

- **true** – This is the default value. If specified, the 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.

```
parameter_validation = false
```

region

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_REGION environment variable, AWS_DEFAULT_REGION environment variable, or the --region command line option.

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

retry_mode

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](#).

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

```
retry_mode = standard
```

role_arn

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

- source_profile
- credential_source

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

The environment variable [AWS_ROLE_ARN](#) overrides this setting.

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

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.

```
role_session_name = maria_garcia_role
```

The environment variable [AWS_ROLE_SESSION_NAME](#) overrides this setting.

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

services

Specifies the service configuration to use for your profile.

```
[profile dev-s3-specific-and-global]
endpoint_url = http://localhost:1234
services = s3-specific

[services s3-specific]
s3 =
  endpoint_url = http://localhost:4567
```

For more information on the services section, see [the section called “services”](#).

The environment variable [`AWS_ROLE_SESSION_NAME`](#) overrides this setting.

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

source_profile

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

```
source_profile = production-profile
```

sso_account_id

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

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

```
sso_account_id = 123456789012
```

sso_region

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

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

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

[sso_registration_scopes](#)

A comma-delimited list of scopes to be authorized for the sso-session. Scopes authorize access to IAM Identity Center bearer token authorized endpoints. A valid scope is a string, such as sso:account:access. This setting isn't applicable to the legacy non-refreshable configuration.

```
sso_registration_scopes = sso:account:access
```

[sso_role_name](#)

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

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

```
sso_role_name = ReadAccess
```

[sso_start_url](#)

Specifies the URL that points to the organization's AWS access portal. The AWS CLI uses this URL to establish a session with the IAM Identity Center service to authenticate its users. To find your AWS access portal URL, use one of the following:

- Open your invitation email, the AWS access portal URL is listed.
- Open the AWS IAM Identity Center console at <https://console.aws.amazon.com/singlesignon/>. The AWS access portal URL is listed in your settings.

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

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

`use_dualstack_endpoint`

Enables the use of dual-stack endpoints to send AWS requests. To learn more about dual-stack endpoints, which support both IPv4 and IPv6 traffic, see [Using Amazon S3 dual-stack endpoints](#) in the *Amazon Simple Storage Service User Guide*. Dual-stack endpoints are available for some services in some regions. If a dual-stack endpoint does not exist for the service or AWS Region, the request fails. This is disabled by default.

This is mutually exclusive with the `use_accelerate_endpoint` setting.

Endpoint configuration settings are 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. The AWS CLI endpoint configuration settings take precedence in the following order:

1. The [`--endpoint-url`](#) command line option.
2. If enabled, the [`AWS_IGNORE_CONFIGURED_ENDPOINT_URLS`](#) global endpoint environment variable or profile setting [`ignore_configure_endpoint_urls`](#) to ignore custom endpoints.
3. The value provided by a service-specific environment variable [`AWS_ENDPOINT_URL_<SERVICE>`](#), such as `AWS_ENDPOINT_URL_DYNAMODB`.
4. The values provided by the [`AWS_USE_DUALSTACK_ENDPOINT`](#), [`AWS_USE_FIPS_ENDPOINT`](#), and [`AWS_ENDPOINT_URL`](#) environment variables.
5. The service-specific endpoint value provided by the [`endpoint_url`](#) setting within a services section of the shared config file.
6. The value provided by the [`endpoint_url`](#) setting within a profile of the shared config file.
7. [`use_dualstack_endpoint`](#), [`use_fips_endpoint`](#), and [`endpoint_url`](#) settings.
8. Any default endpoint URL for the respective AWS service is used last. For a list of the standard service endpoints available in each Region, see [AWS Regions and Endpoints](#) in the *Amazon Web Services General Reference*.

`use_fips_endpoint`

Some AWS services offer endpoints that support [Federal Information Processing Standard \(FIPS\) 140-2](#) in some AWS Regions. When the AWS service supports FIPS, this setting specifies what FIPS endpoint the AWS CLI should use. Unlike standard AWS endpoints, FIPS endpoints

use a TLS software library that complies with FIPS 140-2. These endpoints might be required by enterprises that interact with the United States government.

If this setting is enabled, but a FIPS endpoint does not exist for the service in your AWS Region, the AWS command may fail. In this case, manually specify the endpoint to use in the command using the [--endpoint-url](#) option or use [service-specific endpoints](#).

For more information on specifying FIPS endpoints by AWS Region, see [FIPS Endpoints by Service](#).

Endpoint configuration settings are 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. The AWS CLI endpoint configuration settings take precedence in the following order:

1. The [--endpoint-url](#) command line option.
2. If enabled, the [AWS_IGNORE_CONFIGURED_ENDPOINT_URLS](#) global endpoint environment variable or profile setting [ignore_configure_endpoint_urls](#) to ignore custom endpoints.
3. The value provided by a service-specific environment variable [AWS_ENDPOINT_URL_<SERVICE>](#), such as [AWS_ENDPOINT_URL_DYNAMODB](#).
4. The values provided by the [AWS_USE_DUALSTACK_ENDPOINT](#), [AWS_USE_FIPS_ENDPOINT](#), and [AWS_ENDPOINT_URL](#) environment variables.
5. The service-specific endpoint value provided by the [endpoint_url](#) setting within a services section of the shared config file.
6. The value provided by the [endpoint_url](#) setting within a profile of the shared config file.
7. [use_dualstack_endpoint](#), [use_fips_endpoint](#), and [endpoint_url](#) settings.
8. Any default endpoint URL for the respective AWS service is used last. For a list of the standard service endpoints available in each Region, see [AWS Regions and Endpoints](#) in the *Amazon Web Services General Reference*.

[web_identity_token_file](#)

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](#) overrides this setting.

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

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 URL; 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_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*.

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 [Configure the AWS CLI](#).

Topics

- [How to set environment variables](#)
- [AWS CLI supported environment variables](#)

How to set environment variables

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

Linux or macOS

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

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

Windows Command Prompt

To set for all sessions

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

Using [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 account.

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

`AWS_CA_BUNDLE`

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

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

`AWS_CLI_AUTO_PROMPT`

Enables the auto-prompt for the AWS CLI version 2. There are two settings that can be used:

- **on** uses the full auto-prompt mode each time you attempt to run an aws command. This includes pressing **ENTER** after both a complete command or incomplete command.
- **on-partial** uses partial auto-prompt mode. If a command is incomplete or cannot be run due to client-side validation errors, auto-prompt is used. This mode is useful if you have pre-

existing scripts, runbooks, or you only want to be auto-prompted for commands you are unfamiliar with rather than prompted on every command.

If defined, this environment variable overrides the value for the [cli_auto_prompt](#) profile setting. You can override this environment variable by using the [--cli-auto-prompt](#) and [--no-cli-auto-prompt](#) command line parameters.

For information on the AWS CLI version 2 auto-prompt feature, see [Have the AWS CLI prompt you for commands](#).

AWS_CLI_FILE_ENCODING

Specifies the encoding used for text files. By default encoding matches your locale. To set encoding different from the locale, use the aws_cli_file_encoding environment variable. For example, if you use Windows with default encoding CP1252, setting aws_cli_file_encoding=UTF-8 sets the CLI to open text files using UTF-8.

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

Specifies the [output format](#) 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

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.

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

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_ENDPOINT_URL

Specifies the endpoint that is used for all service requests.

Endpoint configuration settings are 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. The AWS CLI endpoint configuration settings take precedence in the following order:

1. The [`--endpoint-url`](#) command line option.
2. If enabled, the [`AWS_IGNORE_CONFIGURED_ENDPOINT_URLS`](#) global endpoint environment variable or profile setting [`ignore_configure_endpoint_urls`](#) to ignore custom endpoints.
3. The value provided by a service-specific environment variable [`AWS_ENDPOINT_URL_<SERVICE>`](#), such as `AWS_ENDPOINT_URL_DYNAMODB`.
4. The values provided by the [`AWS_USE_DUALSTACK_ENDPOINT`](#), [`AWS_USE_FIPS_ENDPOINT`](#), and [`AWS_ENDPOINT_URL`](#) environment variables.
5. The service-specific endpoint value provided by the [`endpoint_url`](#) setting within a services section of the shared config file.

6. The value provided by the [endpoint_url](#) setting within a profile of the shared config file.
7. [use_dualstack_endpoint](#), [use_fips_endpoint](#), and [endpoint_url](#) settings.
8. Any default endpoint URL for the respective AWS service is used last. For a list of the standard service endpoints available in each Region, see [AWS Regions and Endpoints](#) in the *Amazon Web Services General Reference*.

AWS_ENDPOINT_URL_<SERVICE>

Specifies a custom endpoint that is used for a specific service, where <SERVICE> is replaced with the AWS service identifier. For example, Amazon DynamoDB has a serviceId of [DynamoDB](#). For this service, the endpoint URL environment variable is AWS_ENDPOINT_URL_DYNAMODB.

For a list of all service-specific environment variables, see [List of service-specific identifiers](#).

Endpoint configuration settings are 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. The AWS CLI endpoint configuration settings take precedence in the following order:

1. The [--endpoint-url](#) command line option.
2. If enabled, the [AWS_IGNORE_CONFIGURED_ENDPOINT_URLS](#) global endpoint environment variable or profile setting [ignore_configure_endpoint_urls](#) to ignore custom endpoints.
3. The value provided by a service-specific environment variable [AWS_ENDPOINT_URL_<SERVICE>](#), such as AWS_ENDPOINT_URL_DYNAMODB.
4. The values provided by the [AWS_USE_DUALSTACK_ENDPOINT](#), [AWS_USE_FIPS_ENDPOINT](#), and [AWS_ENDPOINT_URL](#) environment variables.
5. The service-specific endpoint value provided by the [endpoint_url](#) setting within a services section of the shared config file.
6. The value provided by the [endpoint_url](#) setting within a profile of the shared config file.
7. [use_dualstack_endpoint](#), [use_fips_endpoint](#), and [endpoint_url](#) settings.
8. Any default endpoint URL for the respective AWS service is used last. For a list of the standard service endpoints available in each Region, see [AWS Regions and Endpoints](#) in the *Amazon Web Services General Reference*.

AWS_IGNORE_CONFIGURED_ENDPOINT_URLS

If enabled, the AWS CLI ignores all custom endpoint configurations. Valid values are **true** and **false**.

Endpoint configuration settings are 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. The AWS CLI endpoint configuration settings take precedence in the following order:

1. The [--endpoint-url](#) command line option.
2. If enabled, the [AWS_IGNORE_CONFIGURED_ENDPOINT_URLS](#) global endpoint environment variable or profile setting [ignore_configure_endpoint_urls](#) to ignore custom endpoints.
3. The value provided by a service-specific environment variable [AWS_ENDPOINT_URL_<SERVICE>](#), such as [AWS_ENDPOINT_URL_DYNAMODB](#).
4. The values provided by the [AWS_USE_DUALSTACK_ENDPOINT](#), [AWS_USE_FIPS_ENDPOINT](#), and [AWS_ENDPOINT_URL](#) environment variables.
5. The service-specific endpoint value provided by the [endpoint_url](#) setting within a services section of the shared config file.
6. The value provided by the [endpoint_url](#) setting within a profile of the shared config file.
7. [use_dualstack_endpoint](#), [use_fips_endpoint](#), and [endpoint_url](#) settings.
8. Any default endpoint URL for the respective AWS service is used last. For a list of the standard service endpoints available in each Region, see [AWS Regions and Endpoints](#) in the *Amazon Web Services General Reference*.

AWS_MAX_ATTEMPTS

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](#).

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_PAGER

Specifies the pager program used for output. By default, AWS CLI version 2 returns all output through your operating system's default pager program.

To disable all use of an external paging program, set the variable to an empty string.

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

AWS_PROFILE

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_REGION

The AWS SDK compatible environment variable that specifies the AWS Region to send the request to.

If defined, this environment variable overrides the values in the environment variable `AWS_DEFAULT_REGION` and the profile setting `region`. You can override this environment variable by using the `--region` command line parameter.

AWS_RETRY_MODE

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](#).

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](#). 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"](#).

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](#).

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"](#).

 **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_USE_DUALSTACK_ENDPOINT

Enables the use of dual-stack endpoints to send AWS requests. To learn more about dual-stack endpoints, which support both IPv4 and IPv6 traffic, see [Using Amazon S3 dual-stack endpoints](#) in the *Amazon Simple Storage Service User Guide*. Dual-stack endpoints are available for some services in some regions. If a dual-stack endpoint does not exist for the service or AWS Region, the request fails. This is disabled by default.

Endpoint configuration settings are 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. The AWS CLI endpoint configuration settings take precedence in the following order:

1. The [`--endpoint-url`](#) command line option.
2. If enabled, the [`AWS_IGNORE_CONFIGURED_ENDPOINT_URLS`](#) global endpoint environment variable or profile setting [`ignore_configure_endpoint_urls`](#) to ignore custom endpoints.

3. The value provided by a service-specific environment variable [`AWS_ENDPOINT_URL_<SERVICE>`](#), such as `AWS_ENDPOINT_URL_DYNAMODB`.
4. The values provided by the [`AWS_USE_DUALSTACK_ENDPOINT`](#), [`AWS_USE_FIPS_ENDPOINT`](#), and [`AWS_ENDPOINT_URL`](#) environment variables.
5. The service-specific endpoint value provided by the [`endpoint_url`](#) setting within a services section of the shared config file.
6. The value provided by the [`endpoint_url`](#) setting within a profile of the shared config file.
7. [`use_dualstack_endpoint`](#), [`use_fips_endpoint`](#), and [`endpoint_url`](#) settings.
8. Any default endpoint URL for the respective AWS service is used last. For a list of the standard service endpoints available in each Region, see [AWS Regions and Endpoints](#) in the *Amazon Web Services General Reference*.

AWS_USE_FIPS_ENDPOINT

Some AWS services offer endpoints that support [Federal Information Processing Standard \(FIPS\) 140-2](#) in some AWS Regions. When the AWS service supports FIPS, this setting specifies what FIPS endpoint the AWS CLI should use. Unlike standard AWS endpoints, FIPS endpoints use a TLS software library that complies with FIPS 140-2. These endpoints might be required by enterprises that interact with the United States government.

If this setting is enabled, but a FIPS endpoint does not exist for the service in your AWS Region, the AWS command may fail. In this case, manually specify the endpoint to use in the command using the [`--endpoint-url`](#) option or use [service-specific endpoints](#).

For more information on specifying FIPS endpoints by AWS Region, see [FIPS Endpoints by Service](#).

Endpoint configuration settings are 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. The AWS CLI endpoint configuration settings take precedence in the following order:

1. The [`--endpoint-url`](#) command line option.
2. If enabled, the [`AWS_IGNORE_CONFIGURED_ENDPOINT_URLS`](#) global endpoint environment variable or profile setting [`ignore_configure_endpoint_urls`](#) to ignore custom endpoints.

3. The value provided by a service-specific environment variable [`AWS_ENDPOINT_URL_<SERVICE>`](#), such as `AWS_ENDPOINT_URL_DYNAMODB`.
4. The values provided by the [`AWS_USE_DUALSTACK_ENDPOINT`](#), [`AWS_USE_FIPS_ENDPOINT`](#), and [`AWS_ENDPOINT_URL`](#) environment variables.
5. The service-specific endpoint value provided by the [`endpoint_url`](#) setting within a services section of the shared config file.
6. The value provided by the [`endpoint_url`](#) setting within a profile of the shared config file.
7. [`use_dualstack_endpoint`](#), [`use_fips_endpoint`](#), and [`endpoint_url`](#) settings.
8. Any default endpoint URL for the respective AWS service is used last. For a list of the standard service endpoints available in each Region, see [`AWS Regions and Endpoints`](#) in the *Amazon Web Services General Reference*.

[AWS_WEB_IDENTITY_TOKEN_FILE](#)

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”](#).

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](#)
- [AWS CLI supported global command line options](#)
- [Common uses of command line options](#)

How to use command line options

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

```
$ 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 [Specify parameter values for the AWS CLI](#).

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` and the `AWS_CA_BUNDLE` environment variable.

--cli-auto-prompt

Enables auto-prompt mode for a single command. As the following examples show, you can specify it at any point.

```
$ aws --cli-auto-prompt  
$ aws dynamodb --cli-auto-prompt  
$ aws dynamodb describe-table --cli-auto-prompt
```

This option overrides the [aws_cli_auto_prompt](#) environment variable and the [cli_auto_prompt](#) profile setting.

For information on the AWS CLI version 2 auto-prompt feature, see [Have the AWS CLI prompt you for commands](#).

--cli-binary-format

Specifies how the AWS CLI version 2 interprets binary input parameters. It can be one of the following values:

- **base64** – This is the default value. An input parameter that is typed as a binary large object (BLOB) accepts a base64-encoded string. To pass true binary content, put the content in a file and provide the file's path and name with the `fileb://` prefix as the parameter's value. To pass base64-encoded text contained in a file, provide the file's path and name with the `file://` prefix as the parameter's value.
- **raw-in-base64-out** – Default for the AWS CLI version 1. If the setting's value is `raw-in-base64-out`, files referenced using the `file://` prefix is read as text and then the AWS CLI attempts to encode it to binary.

This overrides the [cli_binary_format](#) file configuration setting.

```
$ aws lambda invoke \  
  --cli-binary-format raw-in-base64-out \  
  --function-name my-function \  
  --invocation-type Event \  
  --payload '{ "name": "Bob" }' \  
  response.json
```

If you reference a binary value in a file using the `fileb://` prefix notation, the AWS CLI *always* expects the file to contain raw binary content and does not attempt to convert the value.

If you reference a binary value in a file using the `file://` prefix notation, the AWS CLI handles the file according to the current `cli_binary_format` setting. If that setting's value is `base64` (the default when not explicitly set), the AWS CLI expects the file to contain base64-encoded text. If that setting's value is `raw-in-base64-out`, the AWS CLI expects the file to contain raw binary content.

--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.

The following command example uses a custom Amazon S3 endpoint URL.

```
$ aws s3 ls --endpoint-url http://localhost:4567
```

Endpoint configuration settings are 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. The AWS CLI endpoint configuration settings take precedence in the following order:

1. The [--endpoint-url](#) command line option.
2. If enabled, the [AWS_IGNORE_CONFIGURED_ENDPOINT_URLS](#) global endpoint environment variable or profile setting [ignore_configure_endpoint_urls](#) to ignore custom endpoints.
3. The value provided by a service-specific environment variable [AWS_ENDPOINT_URL_<SERVICE>](#), such as `AWS_ENDPOINT_URL_DYNAMODB`.
4. The values provided by the [AWS_USE_DUALSTACK_ENDPOINT](#), [AWS_USE_FIPS_ENDPOINT](#), and [AWS_ENDPOINT_URL](#) environment variables.
5. The service-specific endpoint value provided by the [endpoint_url](#) setting within a services section of the shared config file.
6. The value provided by the [endpoint_url](#) setting within a profile of the shared config file.
7. [use_dualstack_endpoint](#), [use_fips_endpoint](#), and [endpoint_url](#) settings.
8. Any default endpoint URL for the respective AWS service is used last. 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-cli-auto-prompt

Disables auto-prompt mode for a single command.

```
$ aws dynamodb describe-table --table-name Table1 --no-cli-auto-prompt
```

This option overrides the [aws_cli_auto_prompt](#) environment variable and the [cli_auto_prompt](#) profile setting.

For information on the AWS CLI version 2 auto-prompt feature, see [Have the AWS CLI prompt you for commands](#).

--no-cli-pager

A Boolean switch that disables using a pager for the output of the command.

--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.

--no-sign-request

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

--no-verify-ssl

By default, the AWS CLI uses SSL when communicating with AWS services. For each SSL connection and call, the AWS CLI verifies the SSL certificates. Using this option overrides the default behavior of verifying SSL certificates.

Warning

This option is **not** best practice. If you use `--no-verify-ssl`, your traffic between your client and AWS services is no longer secured. This means your traffic is a security risk and vulnerable to man-in-the-middle exploits. If you're having issues with certificates, it's best to resolve those issues instead. For certificate troubleshooting steps, see [the section called "SSL certificate errors"](#).

--output <string>

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

- [json](#) – The output is formatted as a [JSON](#) string.
- [yaml](#) – The output is formatted as a [YAML](#) string.
- [yaml-stream](#) – The output is streamed and formatted as a [YAML](#) string. Streaming allows for faster handling of large data types.

- **text** – 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** – 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.

--profile <string>

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

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

--query <string>

Specifies a [JMESPath query](#) to use in filtering the response data. For more information, see [Filter AWS CLI output](#).

--region <string>

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

--version

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

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|
+-----+
$ aws ec2 describe-instances --output table --region us-west-2
```

```
|           DescribeInstances          |
+-----+
||           Reservations          ||
|+-----+-----+-----+
||   OwnerId            | 012345678901    ||
||   ReservationId      | r-abcdefg     ||
|+-----+-----+-----+
|||           Instances          ||| |
||+-----+-----+-----+|||
|||   AmiLaunchIndex    | 0          |||
|||   Architecture      | x86_64      |||
...  
...
```

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.

For information on the AWS CLI version 2 auto-prompt feature instead, see [Have the AWS CLI prompt you for commands](#).

Topics

- [How it works](#)
- [Configuring command completion on Linux or macOS](#)
- [Configuring command completion on Windows](#)

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                      describe-global-table
delete-item                         describe-global-table-settings
delete-table                        describe-limits
describe-backup                     describe-table
describe-continuous-backups        describe-table-replica-auto-scaling
describe-contributor-insights      describe-time-to-live
describe-endpoints
```

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

```
$ aws dynamodb delete-table --TAB
--ca-bundle              --endpoint-url      --profile
--cli-connect-timeout   --generate-cli-skeleton --query
--cli-input-json         --no-paginate       --region
--cli-read-timeout      --no-sign-request   --table-name
--color                  --no-verify-ssl    --version
--debug                 --output
```

- 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](#)

- [Enable command completion](#)
- [Verify command completion](#)

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  
2148 pts/1    00:00:00 bash  
8756 pts/1    00:00:00 ps
```

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.

```
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.

```
$ 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](#).

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

```
$ 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.

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

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

```
$ 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.

```
> complete aws 'p/*/'`aws_completer`/'
```

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

After you've enabled command completion, [Verify command completion](#) is working.

Verify command completion

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

```
$ aws sTAB
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.

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

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

```
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.

```
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_LINE + " "
        }
        $env:COMP_POINT=$cursorPosition
        aws_completer.exe | ForEach-Object {
            [System.Management.Automation.CompletionResult]::new($_, $_,
'ParameterValue', $_)
        }
        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.

```
$ aws sTab
```

```
$ aws s3
```

To see all available commands available to your completion, enter a partial command and press **Ctrl + Space**.

```
$ aws sCtrl + Space
s3          ses          sqs          sts          swf
s3api       sns          storagegateway support
```

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](#)
- [Configuring a retry mode](#)
- [Viewing logs of retry attempts](#)

Available retry modes

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

- [Legacy retry mode](#)
- [Standard retry mode](#)
- [Adaptive retry mode](#)

Legacy retry mode

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`
 - `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. This mode is the default for AWS CLI version 2. 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`
 - `LimitExceededException`
 - `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 2 is standard.
- 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](#).

Defining a retry configuration in your AWS configuration file

To change your retry configuration, update your global AWS configuration file. The default location for your AWS config file is `~/.aws/config`.

The following is an example of an AWS config file:

```
[default]
retry_mode = standard
max_attempts = 6
```

For more information on configuration files, see [Configuration and credential file settings](#).

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](#).

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*.

Use 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.

Topics

- [Using the examples](#)
- [Authenticating to a proxy](#)
- [Using a proxy on Amazon EC2 instances](#)
- [Troubleshooting](#)

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

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 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 username and password in the proxy URL, as follows.

Linux or macOS

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

Windows Command Prompt

To set for all sessions

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

To set for current session only

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

Using a proxy on Amazon EC2 instances

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

Linux or macOS

```
$ export NO_PROXY=169.254.169.254
```

Windows Command Prompt

To set for all sessions

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

To set for current session only

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

Troubleshooting

If you come across issues with the AWS CLI, see [Troubleshoot errors](#) for troubleshooting steps. For the most relevant troubleshooting steps, see [the section called “SSL certificate errors”](#).

Use endpoints in the AWS CLI

To connect programmatically to an AWS service, you use an endpoint. An *endpoint* is the URL of the entry point for an AWS web service. The AWS Command Line Interface (AWS CLI) automatically uses the default endpoint for each service in an AWS Region, but you can specify an alternate endpoint for your API requests.

Endpoint topics

- [Set endpoint for a single command](#)
- [Set global endpoint for all AWS services](#)
- [Set to use FIPs endpoints for all AWS services](#)
- [Set to use dual-stack endpoints for all AWS services](#)
- [Set service-specific endpoints](#)
 - [Service-specific endpoints: Environment variables](#)
 - [Service-specific endpoints: Shared config file](#)
 - [Service-specific endpoints: List of service-specific identifiers](#)
- [Endpoint configuration and settings precedence](#)

Set endpoint for a single command

To override any endpoint settings or environment variables for a single command, use the [--endpoint-url](#) command line option. The following command example uses a custom Amazon S3 endpoint URL.

```
$ aws s3 ls --endpoint-url http://localhost:4567
```

Set global endpoint for all AWS services

To route requests for all services to a custom endpoint URL, use one of the following settings:

- Environment variables:

- [AWS_IGNORE_CONFIGURED_ENDPOINT_URLS](#) - Ignore configured endpoint URLs.

Linux or macOS

```
$ export AWS_IGNORE_CONFIGURED_ENDPOINT_URLS=true
```

Windows Command Prompt

To set for all sessions

```
C:\> setx AWS_IGNORE_CONFIGURED_ENDPOINT_URLS true
```

To set for current session only

```
C:\> set AWS_IGNORE_CONFIGURED_ENDPOINT_URLS=true
```

PowerShell

```
PS C:\> $Env:AWS_IGNORE_CONFIGURED_ENDPOINT_URLS="true"
```

- [AWS_ENDPOINT_URL](#) - Set global endpoint URL.

Linux or macOS

```
$ export AWS_ENDPOINT_URL=http://localhost:4567
```

Windows Command Prompt

To set for all sessions

```
C:\> setx AWS_ENDPOINT_URL http://localhost:4567
```

To set for current session only

```
C:\> set AWS_ENDPOINT_URL=http://localhost:4567
```

PowerShell

```
PS C:\> $Env:AWS_ENDPOINT_URL="http://localhost:4567"
```

- The config file:

- [ignore_configure_endpoint_urls](#) - Ignore configured endpoint URLs.

```
ignore_configure_endpoint_urls = true
```

- [endpoint_url](#) - Set global endpoint URL.

```
endpoint_url = http://localhost:4567
```

Service-specific endpoints and the --endpoint-url command line option override any global endpoints.

Set to use FIPs endpoints for all AWS services

To route requests for all services to use FIPs endpoints, use one of the following:

- [AWS_USE_FIPS_ENDPOINT](#) environment variable.

Linux or macOS

```
$ export AWS_USE_FIPS_ENDPOINT=true
```

Windows Command Prompt

To set for all sessions

```
C:\> setx AWS_USE_FIPS_ENDPOINT true
```

To set for current session only

```
C:\> set AWS_USE_FIPS_ENDPOINT=true
```

PowerShell

```
PS C:\> $Env:AWS_USE_FIPS_ENDPOINT="true"
```

- [use_fips_endpoint](#) file setting.

```
use_fips_endpoint = true
```

Some AWS services offer endpoints that support [Federal Information Processing Standard \(FIPS\) 140-2](#) in some AWS Regions. When the AWS service supports FIPS, this setting specifies what FIPS endpoint the AWS CLI should use. Unlike standard AWS endpoints, FIPS endpoints use a TLS software library that complies with FIPS 140-2. These endpoints might be required by enterprises that interact with the United States government.

If this setting is enabled, but a FIPS endpoint does not exist for the service in your AWS Region, the AWS command may fail. In this case, manually specify the endpoint to use in the command using the [--endpoint-url](#) option or use [service-specific endpoints](#).

For more information on specifying FIPS endpoints by AWS Region, see [FIPS Endpoints by Service](#).

Set to use dual-stack endpoints for all AWS services

To route requests for all services to use dual-stack endpoints when available, use one of the following settings:

- [AWS_USE_DUALSTACK_ENDPOINT](#) environment variable.

Linux or macOS

```
$ export AWS_USE_DUALSTACK_ENDPOINT=true
```

Windows Command Prompt

To set for all sessions

```
C:\> setx AWS_USE_DUALSTACK_ENDPOINT true
```

To set for current session only

```
C:\> set AWS_USE_DUALSTACK_ENDPOINT=true
```

PowerShell

```
PS C:\> $Env:AWS_USE_DUALSTACK_ENDPOINT="true"
```

- [use_dualstack_endpoint](#) file setting.

```
use_dualstack_endpoint = true
```

Enables the use of dual-stack endpoints to send AWS requests. To learn more about dual-stack endpoints, which support both IPv4 and IPv6 traffic, see [Using Amazon S3 dual-stack endpoints](#) in the *Amazon Simple Storage Service User Guide*. Dual-stack endpoints are available for some services in some regions. If a dual-stack endpoint does not exist for the service or AWS Region, the request fails. This is disabled by default.

Set service-specific endpoints

Service-specific endpoint configuration provides the option to use a persistent endpoint of your choosing for AWS CLI requests. These settings provide flexibility to support local endpoints, VPC endpoints, and third-party local AWS development environments. Different endpoints can be used for testing and production environments. You can specify an endpoint URL for individual AWS services.

Service-specific endpoints can be specified in the following ways:

- The command line option [--endpoint-url](#) for a single command.
- Environment variables:
 - [AWS_IGNORE_CONFIGURED_ENDPOINT_URLS](#) - Ignore all configured endpoint URLs, unless specified on the command line.
 - [AWS_ENDPOINT_URL_<SERVICE>](#) - Specifies a custom endpoint that is used for a specific service, where <SERVICE> is replace with the AWS service identifier. For all service-specific variables, see [the section called "List of service-specific identifiers"](#).
- config file:
 - [ignore_configure_endpoint_urls](#) - Ignore all configured endpoint URLs, unless specified using environment variables or on the command line.
 - The [services](#) section of the config file combined with the [endpoint_url](#) file setting.

Service-specific endpoints topics:

- [Service-specific endpoints: Environment variables](#)
- [Service-specific endpoints: Shared config file](#)
- [Service-specific endpoints: List of service-specific identifiers](#)

Service-specific endpoints: Environment variables

Environment variables override settings in your config file, but do not override options specified on the command line. Use environment variables if you want all profiles to use the same endpoints on your device.

The following are service-specific environment variables:

- [AWS_IGNORE_CONFIGURED_ENDPOINT_URLS](#) - Ignore all configured endpoint URLs, unless specified on the command line.

Linux or macOS

```
$ export AWS_IGNORE_CONFIGURED_ENDPOINT_URLS=true
```

Windows Command Prompt

To set for all sessions

```
C:\> setx AWS_IGNORE_CONFIGURED_ENDPOINT_URLS true
```

To set for current session only

```
C:\> set AWS_IGNORE_CONFIGURED_ENDPOINT_URLS=true
```

PowerShell

```
PS C:\> $Env:AWS_IGNORE_CONFIGURED_ENDPOINT_URLS="true"
```

- [AWS_ENDPOINT_URL_<SERVICE>](#) - Specifies a custom endpoint that is used for a specific service, where <SERVICE> is replaced with the AWS service identifier. For all service-specific variables, see [the section called “List of service-specific identifiers”](#).

The following environment variable examples sets an endpoint for AWS Elastic Beanstalk:

Linux or macOS

```
$ export AWS_ENDPOINT_URL_ELASTIC_BEANSTALK=http://localhost:4567
```

Windows Command Prompt

To set for all sessions

```
C:\> setx AWS_ENDPOINT_URL_ELASTIC_BEANSTALK http://localhost:4567
```

To set for current session only

```
C:\> set AWS_ENDPOINT_URL_ELASTIC_BEANSTALK=http://localhost:4567
```

PowerShell

```
PS C:\> $Env:AWS_ENDPOINT_URL_ELASTIC_BEANSTALK="http://localhost:4567"
```

For more information on setting environment variables, see [the section called “Environment Variables”](#).

Service-specific endpoints: Shared config file

In the shared config file, endpoint_url is used in multiple sections. To set a service-specific endpoint, use the endpoint_url setting nested under a service identifier key within a services section. For details on defining a services section in your shared config file, see [the section called “services”](#).

The following example uses a services section to configure a service-specific endpoint URL for Amazon S3 and a custom global endpoint used for all other services:

```
[profile dev1]
endpoint_url = http://localhost:1234
services = s3-specific

[services testing-s3]
s3 =
endpoint_url = http://localhost:4567
```

A single profile can configure endpoints for multiple services. The following example sets the service-specific endpoint URLs for Amazon S3 and AWS Elastic Beanstalk in the same profile.

For a list of all service identifier keys to use in the services section, see [List of service-specific identifiers](#).

```
[profile dev1]
services = testing-s3-and-eb

[services testing-s3-and-eb]
s3 =
  endpoint_url = http://localhost:4567
elastic(beanstalk =
  endpoint_url = http://localhost:8000
```

The service configuration section can be used in multiple profiles. The following example has two profiles use the same services definition:

```
[profile dev1]
output = json
services = testing-s3

[profile dev2]
output = text
services = testing-s3

[services testing-s3]
s3 =
  endpoint_url = https://localhost:4567
```

Service-specific endpoints: List of service-specific identifiers

The AWS service identifier is based on the API model's `serviceId` by replacing all spaces with underscores and lowercasing all letters.

The following service identifier example uses AWS Elastic Beanstalk. AWS Elastic Beanstalk has a `serviceId` of [Elastic Beanstalk](#), therefore the service identifier key is `elastic(beanstalk`.

The following table lists all service-specific identifiers, config file keys, and environment variables.

serviceId	Set AWS_ENDPOINT_URL_<SERVICE> environment variable id r ke fo sh A c fil
AccessAnalyzer	aws AWS_ENDPOINT_URL_ACCESSANALYZER l\
Account	aws AWS_ENDPOINT_URL_ACCOUNT
ACM	aws AWS_ENDPOINT_URL_ACM
ACM PCA	aws AWS_ENDPOINT_URL_ACM_PCA
Alexa For Business	aws AWS_ENDPOINT_URL_ALEXA_FOR_BUSINESS _l
amp	aws AWS_ENDPOINT_URL_AMP
Amplify	aws AWS_ENDPOINT_URL_AMPLIFY
AmplifyBackend	aws AWS_ENDPOINT_URL_AMPLIFYBACKEND cl
AmplifyUIBuilder	aws AWS_ENDPOINT_URL_AMPLIFYUIBUILDER bi
API Gateway	aws AWS_ENDPOINT_URL_API_GATEWAY a\
ApiGatewayManagementApi	aws AWS_ENDPOINT_URL_APIGATEWAYMANAGEMENTAPI y\
	n\

serviceId	Set AWS_ENDPOINT_URL_<SERVICE> environment variable id r ke fo sh A co fil
ApiGatewayV2	at AWS_ENDPOINT_URL_APIGATEWAYV2 y'
AppConfig	at AWS_ENDPOINT_URL_APPCONFIG
AppConfigData	at AWS_ENDPOINT_URL_APPCONFIGDATA da
Appflow	at AWS_ENDPOINT_URL_APPFLOW
AppIntegrations	at AWS_ENDPOINT_URL_APPINTEGRATIONS at
Application Auto Scaling	at AWS_ENDPOINT_URL_APPLICATION_AUTO_SCALING or C:
Application Insights	at AWS_ENDPOINT_URL_APPLICATION_INSIGHTS or ts
ApplicationCostProfiler	at AWS_ENDPOINT_URL_APPLICATIONCOSTPROFILER or f:
App Mesh	at AWS_ENDPOINT_URL_APP_MESH

serviceId	Set AWS_ENDPOINT_URL_<SERVICE> environment variable id r ke fo sh A co fil
AppRunner	api AWS_ENDPOINT_URL_APPRUNNER
AppStream	api AWS_ENDPOINT_URL_APPSTREAM
AppSync	api AWS_ENDPOINT_URL_APPSYNC
Athena	api AWS_ENDPOINT_URL_ATHENA
AuditManager	api AWS_ENDPOINT_URL_AUDITMANAGER ge
Auto Scaling	api AWS_ENDPOINT_URL_AUTO_SCALING in
Auto Scaling Plans	api AWS_ENDPOINT_URL_AUTO_SCALING_PLANS in
Backup	api AWS_ENDPOINT_URL_BACKUP
Backup Gateway	api AWS_ENDPOINT_URL_BACKUP_GATEWAY te
BackupStorage	api AWS_ENDPOINT_URL_BACKUPSTORAGE Ia
Batch	api AWS_ENDPOINT_URL_BATCH

serviceId	Set AWS_ENDPOINT_URL_<SERVICE> environment variable id r ke fo sh A co fil
billingconductor	b: AWS_ENDPOINT_URL_BILLINGCONDUCTOR n:
Braket	b: AWS_ENDPOINT_URL_BRAKET
Budgets	b: AWS_ENDPOINT_URL_BUDGETS
Cost Explorer	c: AWS_ENDPOINT_URL_COST_EXPLORER o:
Chime	cl: AWS_ENDPOINT_URL_CHIME
Chime SDK Identity	cl: AWS_ENDPOINT_URL_CHIME_SDK_IDENTITY -:
Chime SDK Media Pipelines	cl: AWS_ENDPOINT_URL_CHIME_SDK_MEDIA_PIPELINES _r pe
Chime SDK Meetings	cl: AWS_ENDPOINT_URL_CHIME_SDK_MEETINGS _r
Chime SDK Messaging	cl: AWS_ENDPOINT_URL_CHIME_SDK_MESSAGING _r g
Cloud9	c: AWS_ENDPOINT_URL_CLOUD9

serviceId	Set AWS_ENDPOINT_URL_<SERVICE> environment variable id r ke fo sh A co fil
CloudControl	c: AWS_ENDPOINT_URL_CLOUDCONTROL r: c:
CloudDirectory	c: AWS_ENDPOINT_URL_CLOUDDIRECTORY c1
CloudFormation	c: AWS_ENDPOINT_URL_CLOUDFORMATION a1
CloudFront	c: AWS_ENDPOINT_URL_CLOUDFRONT t
CloudHSM	c: AWS_ENDPOINT_URL_CLOUDHSM
CloudHSM V2	c: AWS_ENDPOINT_URL_CLOUDHSM_V2 v:
CloudSearch	c: AWS_ENDPOINT_URL_CLOUDSEARCH cl
CloudSearch Domain	c: AWS_ENDPOINT_URL_CLOUDSEARCH_DOMAIN cl
CloudTrail	c: AWS_ENDPOINT_URL_CLOUDTRAIL l
CloudWatch	c: AWS_ENDPOINT_URL_CLOUDWATCH h

serviceId	Set <code>AWS_ENDPOINT_URL_<SERVICE></code> environment variable id r ke fo sh A co fil
codeartifact	Set <code>AWS_ENDPOINT_URL_CODEARTIFACT</code> ac
CodeBuild	Set <code>AWS_ENDPOINT_URL_CODEBUILD</code>
CodeCommit	Set <code>AWS_ENDPOINT_URL_CODECOMMIT</code> t
CodeDeploy	Set <code>AWS_ENDPOINT_URL_CODEDEPLOY</code> y
CodeGuru Reviewer	Set <code>AWS_ENDPOINT_URL_CODEGURU_REVIEWER</code> re
CodeGuruProfiler	Set <code>AWS_ENDPOINT_URL_CODEGURUPROFILER</code> re
CodePipeline	Set <code>AWS_ENDPOINT_URL_CODEPIPELINE</code> in
CodeStar	Set <code>AWS_ENDPOINT_URL_CODESTAR</code>
CodeStar connections	Set <code>AWS_ENDPOINT_URL_CODESTAR_CONNECTIONS</code> Co ns

serviceId	S et <code>AWS_ENDPOINT_URL_<SERVICE></code> environment variable i d r e k e f or s h A W C o f il
codestar notifications	<code>AWS_ENDPOINT_URL_CODESTAR_NOTIFICATIONS</code> no ti
Cognito Identity	<code>AWS_ENDPOINT_URL_COGNITO_IDENTITY</code> de
Cognito Identity Provider	<code>AWS_ENDPOINT_URL_COGNITO_IDENTITY_PROVIDER</code> de ri
Cognito Sync	<code>AWS_ENDPOINT_URL_COGNITO_SYNC</code> yi
Comprehend	<code>AWS_ENDPOINT_URL_COMPREHEND</code> d
ComprehendMedical	<code>AWS_ENDPOINT_URL_COMPREHENDMEDICAL</code> dr
Compute Optimizer	<code>AWS_ENDPOINT_URL_COMPUTE_OPTIMIZER</code> p1
Config Service	<code>AWS_ENDPOINT_URL_CONFIG_SERVICE</code> I'
Connect	<code>AWS_ENDPOINT_URL_CONNECT</code>

serviceId	Set <code>AWS_ENDPOINT_URL_<SERVICE></code> environment variable id r ke fo sh A co fil
Connect Contact Lens	Set <code>AWS_ENDPOINT_URL_CONNECT_CONTACT_LENS</code> or n:
ConnectCampaigns	Set <code>AWS_ENDPOINT_URL_CONNECTCAMPAIGNS</code> m
ConnectCases	Set <code>AWS_ENDPOINT_URL_CONNECTCASES</code> Se
ConnectParticipant	Set <code>AWS_ENDPOINT_URL_CONNECTPARTICIPANT</code> ri
ControlTower	Set <code>AWS_ENDPOINT_URL_CONTROLTOWER</code> We
Cost and Usage Report Service	Set <code>AWS_ENDPOINT_URL_COST_AND_USAGE_REPO</code> us: RT_SERVICE O: C:
Customer Profiles	Set <code>AWS_ENDPOINT_URL_CUSTOMER_PROFILES</code> p:
DataBrew	Set <code>AWS_ENDPOINT_URL_DATABREW</code>

serviceId	Set <code>AWS_ENDPOINT_URL_<SERVICE></code> environment variable id r ke fo sh A co fil
DataExchange	Set <code>AWS_ENDPOINT_URL_DATAEXCHANGE</code> no
Data Pipeline	Set <code>AWS_ENDPOINT_URL_DATA_PIPELINE</code> 1:
DataSync	Set <code>AWS_ENDPOINT_URL_DATASYNC</code>
DAX	Set <code>AWS_ENDPOINT_URL_DAX</code>
Detective	Set <code>AWS_ENDPOINT_URL_DETECTIVE</code>
Device Farm	Set <code>AWS_ENDPOINT_URL_DEVICE_FARM</code> If
DevOps Guru	Set <code>AWS_ENDPOINT_URL_DEVOPS_GURU</code> If
Direct Connect	Set <code>AWS_ENDPOINT_URL_DIRECT_CONNECT</code> no
Application Discovery Service	Set <code>AWS_ENDPOINT_URL_APPLICATION_DISCOVE</code> oRY_SERVICE e: C6
DLM	Set <code>AWS_ENDPOINT_URL_DLM</code>

serviceId	Set AWS_ENDPOINT_URL_<SERVICE> environment variable id r ke fo sh A co fil
Database Migration Service	d: AWS_ENDPOINT_URL_DATABASE_MIGRATION_ m: SERVICE _-`
DocDB	do AWS_ENDPOINT_URL_DOCDB
drs	d: AWS_ENDPOINT_URL_DRS
Directory Service	d: AWS_ENDPOINT_URL_DIRECTORY_SERVICE _-`
DynamoDB	dy AWS_ENDPOINT_URL_DYNAMODB
DynamoDB Streams	dy AWS_ENDPOINT_URL_DYNAMODB_STREAMS s1
EBS	el AWS_ENDPOINT_URL_EBS
EC2	ec AWS_ENDPOINT_URL_EC2
EC2 Instance Connect	ec AWS_ENDPOINT_URL_EC2_INSTANCE_CONNECT no ct
ECR	ec AWS_ENDPOINT_URL_ECR
ECR PUBLIC	ec AWS_ENDPOINT_URL_ECR_PUBLIC c

serviceId	Set AWS_ENDPOINT_URL_<SERVICE> environment variable id r ke fo sh A co fil
ECS	environment variable e: AWS_ENDPOINT_URL_ECS
EFS	environment variable e: AWS_ENDPOINT_URL_EFS
EKS	environment variable el AWS_ENDPOINT_URL_EKS
Elastic Inference	environment variable e: AWS_ENDPOINT_URL_ELASTIC_INFERENCE
ElastiCache	environment variable e: AWS_ENDPOINT_URL_ELASTICACHE
Elastic Beanstalk	environment variable e: AWS_ENDPOINT_URL_ELASTIC_BEANSTALK
Elastic Transcoder	environment variable e: AWS_ENDPOINT_URL_ELASTIC_TRANSCODER
Elastic Load Balancing	environment variable e: AWS_ENDPOINT_URL_ELASTIC_LOAD_BALANCING
Elastic Load Balancing v2	environment variable e: AWS_ENDPOINT_URL_ELASTIC_LOAD_BALANCING_V2
EMR	environment variable er AWS_ENDPOINT_URL_EMR

serviceId	Set <code>AWS_ENDPOINT_URL_<SERVICE></code> environment variable id r ke fo sh A co fil
EMR containers	Set <code>AWS_ENDPOINT_URL_EMR_CONTAINERS</code> in
EMR Serverless	Set <code>AWS_ENDPOINT_URL_EMR_SERVERLESS</code> in
Elasticsearch Service	Set <code>AWS_ENDPOINT_URL_ELASTICSEARCH_SERVICE</code> a: in
EventBridge	Set <code>AWS_ENDPOINT_URL_EVENTBRIDGE</code> ge
Evidently	Set <code>AWS_ENDPOINT_URL_EVIDENTLY</code>
finspace	Set <code>AWS_ENDPOINT_URL_FINSPACE</code>
finspace data	Set <code>AWS_ENDPOINT_URL_FINSPACE_DATA</code> da
Firehose	Set <code>AWS_ENDPOINT_URL_FIREHOSE</code>
fis	Set <code>AWS_ENDPOINT_URL_FIS</code>
FMS	Set <code>AWS_ENDPOINT_URL_FMS</code>
forecast	Set <code>AWS_ENDPOINT_URL_FORECAST</code>

serviceId	Set <code>AWS_ENDPOINT_URL_<SERVICE></code> environment variable id r ke fo sh A co fil
forecastquery	for AWS_ENDPOINT_URL_FORECASTQUERY ue
FraudDetector	f AWS_ENDPOINT_URL_FRAUDDETECTOR c1
FSx	f: AWS_ENDPOINT_URL_FSX
GameLift	g: AWS_ENDPOINT_URL_GAMELIFT
GameSparks	g: AWS_ENDPOINT_URL_GAMESPARKS s
Glacier	g: AWS_ENDPOINT_URL_GLACIER
Global Accelerator	g: AWS_ENDPOINT_URL_GLOBAL_ACCELERATOR ce
Glue	g: AWS_ENDPOINT_URL_GLUE
grafana	g: AWS_ENDPOINT_URL_GRAFANA
Greengrass	g: AWS_ENDPOINT_URL_GREENGRASS s
GreengrassV2	g: AWS_ENDPOINT_URL_GREENGRASSV2 sv

serviceId	Set AWS_ENDPOINT_URL_<SERVICE> environment variable id r ke fo sh A co fil
GroundStation	g: AWS_ENDPOINT_URL_GROUNDSTATION t:
GuardDuty	gu AWS_ENDPOINT_URL_GUARDDUTY
Health	he AWS_ENDPOINT_URL_HEALTH
HealthLake	he AWS_ENDPOINT_URL_HEALTHLAKE e
Honeycode	ho AWS_ENDPOINT_URL_HONEYCODE
IAM	i: AWS_ENDPOINT_URL_IAM
identitystore	id AWS_ENDPOINT_URL_IDENTITYSTORE t:
imagebuilder	ir AWS_ENDPOINT_URL_IMAGEBUILDER de
ImportExport	ir AWS_ENDPOINT_URL_IMPORTEXPORT o:
Inspector	in AWS_ENDPOINT_URL_INSPECTOR

serviceId	Set AWS_ENDPOINT_URL_<SERVICE> environment variable id r ke fo sh A co fil
Inspector2	in AWS_ENDPOINT_URL_INSPECTOR2 2
IoT	io AWS_ENDPOINT_URL_IOT
IoT Data Plane	io AWS_ENDPOINT_URL_IOT_DATA_PLANE p:
IoT Jobs Data Plane	io AWS_ENDPOINT_URL_IOT_JOBS_DATA_PLANE d: e
IoT 1Click Devices Service	io AWS_ENDPOINT_URL_IOT_1CLICK_DEVICES_ k_ SERVICE _:
IoT 1Click Projects	io AWS_ENDPOINT_URL_IOT_1CLICK_PROJECTS k_ s
IoTAalytics	io AWS_ENDPOINT_URL_IOTANALYTICS i:
IotDeviceAdvisor	io AWS_ENDPOINT_URL_IOTDEVICEADVISOR a:
IoT Events	io AWS_ENDPOINT_URL_IOT_EVENTS s

serviceId	Set <code>AWS_ENDPOINT_URL_<SERVICE></code> environment variable id r ke fo sh A co fil
IoT Events Data	Set <code>AWS_ENDPOINT_URL_IOT_EVENTS_DATA</code>
IoTFleetHub	Set <code>AWS_ENDPOINT_URL_IOTFLEETHUB</code>
IoTFleetWise	Set <code>AWS_ENDPOINT_URL_IOTFLEETWISE</code>
IoTSecureTunneling	Set <code>AWS_ENDPOINT_URL_IOTSECURETUNNELING</code>
IoTSiteWise	Set <code>AWS_ENDPOINT_URL_IOTSITEWISE</code>
IoTThingsGraph	Set <code>AWS_ENDPOINT_URL_IOTTHINGSGRAPH</code>
IoTTwinMaker	Set <code>AWS_ENDPOINT_URL_IOTWINMAKER</code>
IoT Wireless	Set <code>AWS_ENDPOINT_URL_IOT_WIRELESS</code>
ivs	Set <code>AWS_ENDPOINT_URL_IVS</code>
ivschat	Set <code>AWS_ENDPOINT_URL_IVSCHAT</code>

serviceId	Set AWS_ENDPOINT_URL_<SERVICE> environment variable id r ke fo sh A co fil
Kafka	k: AWS_ENDPOINT_URL_KAFKA
KafkaConnect	k: AWS_ENDPOINT_URL_KAFKACONNECT e:
kendra	k: AWS_ENDPOINT_URL_KENDRA
Keyspaces	k: AWS_ENDPOINT_URL_KEYSPACES
Kinesis	k: AWS_ENDPOINT_URL_KINESIS
Kinesis Video Archived Media	k: AWS_ENDPOINT_URL_KINESIS_VIDEO_ARCHIVED_MEDIA i: a
Kinesis Video Media	k: AWS_ENDPOINT_URL_KINESIS_VIDEO_MEDIA i: a
Kinesis Video Signaling	k: AWS_ENDPOINT_URL_KINESIS_VIDEO_SIGNALING i: a:
Kinesis Analytics	k: AWS_ENDPOINT_URL_KINESIS_ANALYTICS n:

serviceId	Set AWS_ENDPOINT_URL_<SERVICE> environment variable
	id r ke fo sh A co fil
Kinesis Analytics V2	k: AWS_ENDPOINT_URL_KINESIS_ANALYTICS_V2 n: v:
Kinesis Video	k: AWS_ENDPOINT_URL_KINESIS_VIDEO i:
KMS	kr: AWS_ENDPOINT_URL_KMS
LakeFormation	l: AWS_ENDPOINT_URL_LAKEFORMATION t:
Lambda	l: AWS_ENDPOINT_URL_LAMBDA
Lex Model Building Service	le: AWS_ENDPOINT_URL_LEX_MODEL_BUILDING_ _S _E
Lex Runtime Service	le: AWS_ENDPOINT_URL_LEX_RUNTIME_SERVICE m: e
Lex Models V2	le: AWS_ENDPOINT_URL_LEX_MODELS_V2 S_
Lex Runtime V2	le: AWS_ENDPOINT_URL_LEX_RUNTIME_V2 m:

serviceId	Set <code>AWS_ENDPOINT_URL_<SERVICE></code> environment variable id r ke fo sh A co fil
License Manager	l: <code>AWS_ENDPOINT_URL_LICENSE_MANAGER</code> ai
License Manager User Subscriptions	l: <code>AWS_ENDPOINT_URL_LICENSE_MANAGER_USE</code> ai R_SUBSCRIPTIONS e: i
Lightsail	l: <code>AWS_ENDPOINT_URL_LIGHTSAIL</code>
Location	l: <code>AWS_ENDPOINT_URL_LOCATION</code>
CloudWatch Logs	c: <code>AWS_ENDPOINT_URL_CLOUDWATCH_LOGS</code> h_
LookoutEquipment	l: <code>AWS_ENDPOINT_URL_LOOKOUTEQUIPMENT</code> u:
LookoutMetrics	l: <code>AWS_ENDPOINT_URL_LOOKOUTMETRICS</code> t:
LookoutVision	l: <code>AWS_ENDPOINT_URL_LOOKOUTVISION</code> s:
m2	m: <code>AWS_ENDPOINT_URL_M2</code>
Machine Learning	m: <code>AWS_ENDPOINT_URL_MACHINE_LEARNING</code> e:

serviceId	Set AWS_ENDPOINT_URL_<SERVICE> environment variable id r ke fo sh A co fil
Macie	macie AWS_ENDPOINT_URL_MACIE
Macie2	macie2 AWS_ENDPOINT_URL_MACIE2
ManagedBlockchain	managedblockchain AWS_ENDPOINT_URL_MANAGEDBLOCKCHAIN OR awsmbc AWS_ENDPOINT_URL_MANAGEDBLOCKCHAIN
Marketplace Catalog	marketplacecatalog AWS_ENDPOINT_URL_MARKETPLACE_CATALOG C atalog
Marketplace Entitlement Service	marketplaceentitlementservice AWS_ENDPOINT_URL_MARKETPLACE_ENTITLEMENT_SERVICE er v:
Marketplace Commerce Analytics	marketplacecommerceanalytics AWS_ENDPOINT_URL_MARKETPLACE_COMMERC_E_ANALYTICS C om mer ce an aly tic s
MediaConnect	mediacconnect AWS_ENDPOINT_URL_MEDIACONNECT ec
MediaConvert	mediacconvert AWS_ENDPOINT_URL_MEDIA CONVERT e:

serviceId	Set AWS_ENDPOINT_URL_<SERVICE> environment variable id r ke fo sh A co fil
MediaLive	me AWS_ENDPOINT_URL_MEDIALIVE
MediaPackage	me AWS_ENDPOINT_URL_MEDIAPACKAGE ac
MediaPackage Vod	me AWS_ENDPOINT_URL_MEDIAPACKAGE_VOD ac
MediaStore	me AWS_ENDPOINT_URL_MEDIASTORE e
MediaStore Data	me AWS_ENDPOINT_URL_MEDIASTORE_DATA e_
MediaTailor	me AWS_ENDPOINT_URL_MEDIATAILOR o:
MemoryDB	me AWS_ENDPOINT_URL_MEMORYDB
Marketplace Metering	me AWS_ENDPOINT_URL_MARKETPLACE_METERING C n:
Migration Hub	m: AWS_ENDPOINT_URL_MIGRATION_HUB _I
mgn	me AWS_ENDPOINT_URL_MGN

serviceId	Set AWS_ENDPOINT_URL_<SERVICE> environment variable id r ke fo sh A co fil
Migration Hub Refactor Spaces	m: AWS_ENDPOINT_URL_MIGRATION_HUB_REFAC_TOR_SPACES c: e:
MigrationHub Config	m: AWS_ENDPOINT_URL_MIGRATIONHUB_CONFIG_HI g
MigrationHubOrches trator	m: AWS_ENDPOINT_URL_MIGRATIONHUBORCHESTRATOR_HI t:
MigrationHubStrategy	m: AWS_ENDPOINT_URL_MIGRATIONHUBSTRATEGY_HI g:
Mobile	mo AWS_ENDPOINT_URL_MOBILE
mq	mq AWS_ENDPOINT_URL_MQ
MTurk	mt AWS_ENDPOINT_URL_MTURK
MWAA	mw AWS_ENDPOINT_URL_MWAA
Neptune	ne AWS_ENDPOINT_URL_NEPTUNE
Network Firewall	ne AWS_ENDPOINT_URL_NETWORK_FIREWALL_i:

serviceId	Set AWS_ENDPOINT_URL_<SERVICE> environment variable id r ke fo sh A co fil
NetworkManager	ne AWS_ENDPOINT_URL_NETWORKMANAGER n
nimble	n: AWS_ENDPOINT_URL_NIMBLE
OpenSearch	o: AWS_ENDPOINT_URL_OPENSEARCH h
OpsWorks	o: AWS_ENDPOINT_URL_OPSWORKS
OpsWorksCM	o: AWS_ENDPOINT_URL_OPSWORKSCM m
Organizations	o: AWS_ENDPOINT_URL_ORGANIZATIONS i
Outposts	o: AWS_ENDPOINT_URL_OUTPOSTS
Panorama	p: AWS_ENDPOINT_URL_PANORAMA
Personalize	pe AWS_ENDPOINT_URL_PERSONALIZE ze
Personalize Events	pe AWS_ENDPOINT_URL_PERSONALIZE_EVENTS ze

serviceId	Set AWS_ENDPOINT_URL_<SERVICE> environment variable
	id r ke fo sh A co fil
Personalize Runtime	p: AWS_ENDPOINT_URL_PERSONALIZE_RUNTIME ze e
PI	p: AWS_ENDPOINT_URL_PI
Pinpoint	p: AWS_ENDPOINT_URL_PINPOINT
Pinpoint Email	p: AWS_ENDPOINT_URL_PINPOINT_EMAIL er
Pinpoint SMS Voice	p: AWS_ENDPOINT_URL_PINPOINT_SMS_VOICE sr
Pinpoint SMS Voice V2	p: AWS_ENDPOINT_URL_PINPOINT_SMS_VOICE_V2 sr —'
Polly	p: AWS_ENDPOINT_URL_POLLY
Pricing	p: AWS_ENDPOINT_URL_PRICING
PrivateNetworks	p: AWS_ENDPOINT_URL_PRIVATENETWORKS t\
Proton	p: AWS_ENDPOINT_URL_PROTON
QLDB	q: AWS_ENDPOINT_URL_QLDB

serviceId	Set AWS_ENDPOINT_URL_<SERVICE> environment variable id r ke fo sh A co fil
QLDB Session	q: AWS_ENDPOINT_URL_QLDB_SESSION id
QuickSight	q: AWS_ENDPOINT_URL_QUICKSIGHT t
RAM	r: AWS_ENDPOINT_URL_RAM
rbin	r: AWS_ENDPOINT_URL_RBIN
RDS	r: AWS_ENDPOINT_URL_RDS
RDS Data	r: AWS_ENDPOINT_URL_RDS_DATA
Redshift	r: AWS_ENDPOINT_URL_REDSHIFT
Redshift Data	r: AWS_ENDPOINT_URL_REDSHIFT_DATA d:
Redshift Serverless	r: AWS_ENDPOINT_URL_REDSHIFT_SERVERLESS S: S
Rekognition	r: AWS_ENDPOINT_URL_REKOGNITION O:
resiliencehub	r: AWS_ENDPOINT_URL_RESILIENCEHUB el

serviceId	Set <code>AWS_ENDPOINT_URL_<SERVICE></code> environment variable id r ke fo sh A co fil
Resource Groups	Set <code>AWS_ENDPOINT_URL_RESOURCE_GROUPS</code> g:
Resource Groups Tagging API	Set <code>AWS_ENDPOINT_URL_RESOURCE_GROUPS_TAG</code> : <code>GING_API</code> g:
RoboMaker	Set <code>AWS_ENDPOINT_URL_ROBOMAKER</code>
RolesAnywhere	Set <code>AWS_ENDPOINT_URL_ROLESANYWHERE</code> he
Route 53	Set <code>AWS_ENDPOINT_URL_ROUTE_53</code>
Route53 Recovery Cluster	Set <code>AWS_ENDPOINT_URL_ROUTE53_RECOVERY_CLUSTER</code> re clu
Route53 Recovery Control Config	Set <code>AWS_ENDPOINT_URL_ROUTE53_RECOVERY_CONTROL_CONFIG</code> re on
Route53 Recovery Readiness	Set <code>AWS_ENDPOINT_URL_ROUTE53_RECOVERY_READINESS</code> re

serviceId	Set <code>AWS_ENDPOINT_URL_<SERVICE></code> environment variable id r ke fo sh A co fil
Route 53 Domains	Set <code>AWS_ENDPOINT_URL_ROUTE_53_DOMAINS</code>
Route53Resolver	Set <code>AWS_ENDPOINT_URL_ROUTE53RESOLVER</code>
RUM	Set <code>AWS_ENDPOINT_URL_RUM</code>
S3	Set <code>AWS_ENDPOINT_URL_S3</code>
S3 Control	Set <code>AWS_ENDPOINT_URL_S3_CONTROL</code>
S3Outposts	Set <code>AWS_ENDPOINT_URL_S3OUTPOSTS</code>
SageMaker	Set <code>AWS_ENDPOINT_URL_SAGEMAKER</code>
SageMaker A2I Runtime	Set <code>AWS_ENDPOINT_URL_SAGEMAKER_A2I_RUNTIME</code>
Sagemaker Edge	Set <code>AWS_ENDPOINT_URL_SAGEMAKER_EDGE</code>

serviceId	Set <code>AWS_ENDPOINT_URL_<SERVICE></code> environment variable id r ke fo sh A co fil
SageMaker FeatureStore Runtime	Set <code>AWS_ENDPOINT_URL_SAGEMAKER_FEATURESTORE_RUNTIME</code> to in
SageMaker Runtime	Set <code>AWS_ENDPOINT_URL_SAGEMAKER_RUNTIME</code> to
savingsplans	Set <code>AWS_ENDPOINT_URL_SAVINGSPLANS</code> to
schemas	Set <code>AWS_ENDPOINT_URL_SCHEMAS</code> to
SimpleDB	Set <code>AWS_ENDPOINT_URL_SIMPLEDB</code> to
Secrets Manager	Set <code>AWS_ENDPOINT_URL_SECRETS_MANAGER</code> to
SecurityHub	Set <code>AWS_ENDPOINT_URL_SECURITYHUB</code> to
ServerlessApplicationRepository	Set <code>AWS_ENDPOINT_URL_SERVERLESSAPPLICATIONREPOSITORY</code> to
Service Quotas	Set <code>AWS_ENDPOINT_URL_SERVICE_QUOTAS</code> to

serviceId	Set <code>AWS_ENDPOINT_URL_<SERVICE></code> environment variable id r ke fo sh A co fil
Service Catalog	Set <code>AWS_ENDPOINT_URL_SERVICE_CATALOG</code> at
Service Catalog AppRegistry	Set <code>AWS_ENDPOINT_URL_SERVICE_CATALOG_APP</code> at <code>REGISTRY</code> p:
ServiceDiscovery	Set <code>AWS_ENDPOINT_URL_SERVICEDISCOVERY</code> S
SES	Set <code>AWS_ENDPOINT_URL_SES</code>
SESV2	Set <code>AWS_ENDPOINT_URL_SESV2</code>
Shield	Set <code>AWS_ENDPOINT_URL_SHIELD</code>
signer	Set <code>AWS_ENDPOINT_URL_SIGNER</code>
SMS	Set <code>AWS_ENDPOINT_URL_SMS</code>
Snow Device Management	Set <code>AWS_ENDPOINT_URL_SNOW_DEVICE_MANAGEMENT</code> C me
Snowball	Set <code>AWS_ENDPOINT_URL_SNOWBALL</code>
SNS	Set <code>AWS_ENDPOINT_URL_SNS</code>

serviceId	Set AWS_ENDPOINT_URL_<SERVICE> environment variable id r ke fo sh A co fil
SQS	s: AWS_ENDPOINT_URL_SQS
SSM	s: AWS_ENDPOINT_URL_SSM
SSM Contacts	s: AWS_ENDPOINT_URL_SSM_CONTACTS ct
SSM Incidents	s: AWS_ENDPOINT_URL_SSM INCIDENTS ei
SSO	s: AWS_ENDPOINT_URL_SSO
SSO Admin	s: AWS_ENDPOINT_URL_SSO_ADMIN
SSO OIDC	s: AWS_ENDPOINT_URL_SSO_OIDC
SFN	s: AWS_ENDPOINT_URL_SFN
Storage Gateway	s: AWS_ENDPOINT_URL_STORAGE_GATEWAY at
STS	s: AWS_ENDPOINT_URL_STS
Support	s: AWS_ENDPOINT_URL_SUPPORT
Support App	s: AWS_ENDPOINT_URL_SUPPORT_APP pi
SWF	s: AWS_ENDPOINT_URL_SWF

serviceId	S et AWS_ENDPOINT_URL_<SERVICE> environment variable i d r e k e f or s h A W C LI f ile
synthetics	s et AWS_ENDPOINT_URL_SYNTHETICS s ervice
Textract	t o AWS_ENDPOINT_URL_TEXTRACT
Timestream Query	t : AWS_ENDPOINT_URL_TIMESTREAM_QUERY m _
Timestream Write	t : AWS_ENDPOINT_URL_TIMESTREAM_WRITE m _
Transcribe	t : AWS_ENDPOINT_URL_TRANSCRIBE e ntry
Transfer	t : AWS_ENDPOINT_URL_TRANSFER
Translate	t : AWS_ENDPOINT_URL_TRANSLATE
Voice ID	v c AWS_ENDPOINT_URL_VOICE_ID
WAF	w : AWS_ENDPOINT_URL_WAF
WAF Regional	w : AWS_ENDPOINT_URL_WAF_REGIONAL n :
WAFV2	w : AWS_ENDPOINT_URL_WAFV2

serviceId	Set AWS_ENDPOINT_URL_<SERVICE> environment variable
	id r ke fo sh A co fil
WellArchitected	w: AWS_ENDPOINT_URL_WELLARCHITECTED
Wisdom	w: AWS_ENDPOINT_URL_WISDOM
WorkDocs	w: AWS_ENDPOINT_URL_WORKDOCS
WorkLink	w: AWS_ENDPOINT_URL_WORKLINK
WorkMail	w: AWS_ENDPOINT_URL_WORKMAIL
WorkMailMessageFlow	w: AWS_ENDPOINT_URL_WORKMAILMESSAGEFLOW
WorkSpaces	w: AWS_ENDPOINT_URL_WORKSPACES
WorkSpaces Web	w: AWS_ENDPOINT_URL_WORKSPACES_WEB
XRay	x: AWS_ENDPOINT_URL_XRAY

Endpoint configuration and settings precedence

Endpoint configuration settings are 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. The AWS CLI endpoint configuration settings take precedence in the following order:

1. The [--endpoint-url](#) command line option.
2. If enabled, the [AWS_IGNORE_CONFIGURED_ENDPOINT_URLS](#) global endpoint environment variable or profile setting [ignore_configure_endpoint_urls](#) to ignore custom endpoints.
3. The value provided by a service-specific environment variable [AWS_ENDPOINT_URL_<SERVICE>](#), such as `AWS_ENDPOINT_URL_DYNAMODB`.
4. The values provided by the [AWS_USE_DUALSTACK_ENDPOINT](#), [AWS_USE_FIPS_ENDPOINT](#), and [AWS_ENDPOINT_URL](#) environment variables.
5. The service-specific endpoint value provided by the [endpoint_url](#) setting within a services section of the shared config file.
6. The value provided by the [endpoint_url](#) setting within a profile of the shared config file.
7. [use_dualstack_endpoint](#), [use_fips_endpoint](#), and [endpoint_url](#) settings.
8. Any default endpoint URL for the respective AWS service is used last. For a list of the standard service endpoints available in each Region, see [AWS Regions and Endpoints](#) in the *Amazon Web Services General Reference*.

Authentication and access credentials

You must establish how the AWS CLI authenticates with AWS when you develop with AWS services. To configure credentials for programmatic access for the AWS CLI, choose one of the following options. The options are in order of recommendation.

Which user needs programmatic access?	Purpose	Instructions
Workforce identity (AWS IAM Identity Center users)	(Recommended) Use short-term credentials.	the section called "IAM Identity Center authentication"
IAM	Use short-term credentials.	the section called "Short-term credentials"
IAM or Workforce identity (AWS IAM Identity Center users)	Use Amazon EC2 instance metadata for credentials.	the section called "Use credentials for Amazon EC2 instance metadata"
IAM or Workforce identity (AWS IAM Identity Center users)	Pair another credential method and assume a role for permissions.	the section called "IAM roles"
IAM	(Not recommended) Use long-term credentials.	the section called "IAM users"
IAM or Workforce identity (AWS IAM Identity Center users)	(Not recommended) Pair another credential method but use credential values stored in a location outside of the AWS CLI.	the section called "External credentials"

Configuration and credential precedence

Credentials and configuration settings are 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 authentication take precedence over others. The AWS CLI authentication settings take precedence in the following order:

1. **[Command line options](#)** – Overrides settings in any other location, such as the `--region`, `--output`, and `--profile` parameters.
2. **[Environment variables](#)** – You can store values in your system's environment variables.
3. **[Assume role](#)** – Assume the permissions of an IAM role through configuration or the [`aws sts assume-role`](#) command.
4. **[Assume role with web identity](#)** – Assume the permissions of an IAM role using web identity through configuration or the [`aws sts assume-role`](#) command.
5. **[AWS IAM Identity Center](#)** – The IAM Identity Center configuration settings are stored in the config file. Credentials are authenticated when you run the `aws configure sso` command. The config file is located at `~/.aws/config` on Linux or macOS, or at `C:\Users\<USERNAME>\.aws\config` on Windows.
6. **[Credentials file](#)** – 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.
7. **[Custom process](#)** – Get your credentials from an external source.
8. **[Configuration file](#)** – 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.
9. **[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*.
10. **[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*.

Additional topics in this section

- [the section called “IAM Identity Center authentication”](#)
- [the section called “Short-term credentials”](#)
- [the section called “IAM roles”](#)
- [the section called “IAM users”](#)
- [the section called “Use credentials for Amazon EC2 instance metadata”](#)
- [the section called “External credentials”](#)

Configure the AWS CLI to use AWS IAM Identity Center

There are primarily two ways to authenticate users with AWS IAM Identity Center (IAM Identity Center) to get credentials to run AWS Command Line Interface (AWS CLI) commands through the config file:

- **(Recommended)** [SSO token provider configuration](#). The SSO token provider configuration, your AWS SDK or tool can automatically retrieve refreshed authentication tokens.
- [Legacy non-refreshable configuration](#). When using the legacy non-refreshable configuration, you need to manually refresh the token as it periodically expires.

When using IAM Identity Center, you can login to Active Directory, a built-in IAM Identity Center directory, or [another IdP connected to IAM Identity Center](#). You can map these credentials to an AWS Identity and Access Management (IAM) role for you to run AWS CLI commands.

Regardless of which IdP you use, IAM Identity Center abstracts those distinctions away. For example, you can connect Microsoft Azure AD as described in the blog article [The Next Evolution in IAM Identity Center](#).

 **Note**

For information on using bearer auth, which uses no account ID and role, see [Setting up to use the AWS CLI with CodeCatalyst](#) in the *Amazon CodeCatalyst User Guide*.

Topics in this section

- [Configure the AWS CLI to use IAM Identity Center token provider credentials with automatic authentication refresh](#)
- [Legacy non-refreshable configuration for AWS IAM Identity Center](#)
- [Use an IAM Identity Center named profile](#)

Configure the AWS CLI to use IAM Identity Center token provider credentials with automatic authentication refresh

This topic describes how to configure the AWS CLI to authenticate users with the AWS IAM Identity Center (IAM Identity Center) token provider configuration. Using this SSO token provider configuration, your AWS SDK or tool can automatically retrieve refreshed authentication tokens.

When using IAM Identity Center, you can login to Active Directory, a built-in IAM Identity Center directory, or [another IdP connected to IAM Identity Center](#). You can map these credentials to an AWS Identity and Access Management (IAM) role for you to run AWS CLI commands.

Regardless of which IdP you use, IAM Identity Center abstracts those distinctions away. For example, you can connect Microsoft Azure AD as described in the blog article [The Next Evolution in IAM Identity Center](#).

Note

For information on using bearer auth, which uses no account ID and role, see [Setting up to use the AWS CLI with CodeCatalyst](#) in the *Amazon CodeCatalyst User Guide*.

You can use the SSO token provider configuration to automatically refresh authentication tokens as needed for your application, and to use extended session duration options. You can configure this in the following ways:

- Automatically, using the `aws configure sso` and `aws configure sso-session` commands. The following commands are wizards that guide you through configuring your profile and `sso-session` information are the following:
 - Use [`aws configure sso`](#) to create or edit both your config profiles and `sso-session` sections.
 - Use [`aws configure sso-session`](#) to create or edit only `sso-session` sections.
- [Manually](#), by editing the config file that stores the named profiles.

Prerequisites

- Install the AWS CLI. For more information, see [the section called “Install/Update”](#).
- You must first have access to SSO authentication within IAM Identity Center. Choose one of the following methods to access your AWS credentials.

I do not have established access through IAM Identity Center

Follow the instructions in [Getting started](#) in the *AWS IAM Identity Center User Guide*. This process activates IAM Identity Center, creates an administrative user, and adds an appropriate least-privilege permission set.

 **Note**

For Step 6 – Create a permission set that applies least-privilege permissions. We recommend using the predefined PowerUserAccess permission set, unless your employer has created a custom permission set for this purpose.

Exit the portal and sign in again to see your AWS accounts and options for Administrator or PowerUserAccess. Select PowerUserAccess when working with the SDK. This also helps you find details about programmatic access.

I already have access to AWS through a federated identity provider managed by my employer (such as Azure AD or Okta)

Sign in to AWS through your identity provider’s portal. If your Cloud Administrator has granted you PowerUserAccess (developer) permissions, you see the AWS accounts that you have access to and your permission set. Next to the name of your permission set, you see options to access the accounts manually or programmatically using that permission set.

Custom implementations might result in different experiences, such as different permission set names. If you’re not sure which permission set to use, contact your IT team for help.

I already have access to AWS through the AWS access portal managed by my employer

Sign in to AWS through the AWS access portal. If your Cloud Administrator has granted you PowerUserAccess (developer) permissions, you see the AWS accounts that you have access to

and your permission set. Next to the name of your permission set, you see options to access the accounts manually or programmatically using that permission set.

I already have access to AWS through a federated custom identity provider managed by my employer

Contact your IT team for help.

Configure your profile with the aws configure sso wizard

To configure both an IAM Identity Center profile and sso-session to your AWS CLI

1. Gather your IAM Identity Center information by performing the following:
 1. In the AWS access portal, select the permission set you use for development, and select the **Command line or programmatic access** link.
 2. In the **Get credentials** dialog box, choose either **MacOS and Linux** or **Windows**, depending on your operating system.
 3. Choose the **IAM Identity Center credentials** method to get the **SSO Start URL** and **SSO Region** values that you need to run `aws configure sso`.
 4. For information on which scopes value to register, see [OAuth 2.0 Access scopes](#) in the *IAM Identity Center User Guide*.
2. In your preferred terminal, run the `aws configure sso` command and provide your IAM Identity Center start URL and the AWS Region that hosts the Identity Center directory.

```
$ aws configure sso
SSO session name (Recommended): my-sso
SSO start URL [None]: https://my-sso-portal.awsapps.com/start
SSO region [None]: us-east-1
SSO registration scopes [None]: sso:account:access
```

3. The AWS CLI attempts to open your default browser and begin the login process for your IAM Identity Center account.

Attempting to automatically open the SSO authorization page in your default browser.

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

If the browser does not open or you wish to use a different device to authorize this request, open the following URL:

<https://device.sso.us-west-2.amazonaws.com/>

Then enter the code:

QCFK-N451

IAM Identity Center uses the code to associate the IAM Identity Center session with your current AWS CLI session. The IAM Identity Center browser page prompts you to log in with your IAM Identity Center credentials. This gives permissions to the AWS CLI to retrieve and display the AWS accounts and roles that you are authorized to use with IAM Identity Center.

 **Note**

The sign in process may prompt you to allow the AWS CLI access to your data. Since the AWS CLI is built on top of the SDK for Python, permission messages may contain variations of the botocore name.

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

There are 2 AWS accounts available to you.

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

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

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

Using the account ID [123456789011](#)

There are 2 roles available to you.

```
> ReadOnly  
FullAccess
```

Use the arrow keys to select the IAM role you want to use and press <ENTER>.

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

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

 **Note**

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

- A final message describes the completed profile configuration.

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

```
aws s3 ls --profile my-dev-profile
```

- This results in creating the `sso-session` section and named profile in `~/.aws/config` that looks like the following:

```
[profile my-dev-profile]  
sso_session = my-sso  
sso_account_id = 123456789011  
sso_role_name = readOnly  
region = us-west-2  
output = json  
  
[sso-session my-sso]  
sso_region = us-east-1
```

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

You can now use this sso-session and profile to request refreshed credentials. Use the aws sso login command to request and retrieve the credentials needed to run commands. For instructions, see [Use an IAM Identity Center named profile](#).

Configure only your sso-session section with aws configure sso-session wizard

The aws configure sso-session command only updates the sso-session sections in the `~/.aws/config` file. This command can be used to create or update your sessions. This is useful if you already have existing configuration settings and would like to create new or edit existing sso-session configuration.

Run the aws configure sso-session command and provide your IAM Identity Center start URL and the AWS Region that hosts the Identity Center directory.

```
$ aws configure sso-session
SSO session name: my-sso
SSO start URL [None]: https://my-sso-portal.awsapps.com/start
SSO region [None]: us-east-1
SSO registration scopes [None]: sso:account:access
```

After entering in your information a message describes the completed profile configuration.

```
Completed configuring SSO session: my-sso
Run the following to login and refresh access token for this session:
aws sso login --sso-session my-sso
```

Note

If you are signed in to the sso-session you are updating, refresh your token by running the aws sso login command.

Manual configuration using the config file

The sso-session section of the config file is used to group configuration variables for acquiring SSO access tokens, which can then be used to acquire AWS credentials. The following settings are used:

- **(Required)** `sso_start_url`
- **(Required)** `sso_region`
- `sso_account_id`
- `sso_role_name`
- `sso_registration_scopes`

You define an sso-session section and associate it to a profile. `sso_region` and `sso_start_url` must be set within the sso-session section. Typically, `sso_account_id` and `sso_role_name` must be set in the profile section so that the SDK can request SSO credentials.

The following example configures the SDK to request SSO credentials and supports automated token refresh:

```
[profile dev]
sso_session = my-sso
sso_account_id = 111122223333
sso_role_name = SampleRole

[sso-session my-sso]
sso_region = us-east-1
sso_start_url = https://my-sso-portal.awsapps.com/start
```

This also allows sso-session configurations to be reused across multiple profiles:

```
[profile dev]
sso_session = my-sso
sso_account_id = 111122223333
sso_role_name = SampleRole

[profile prod]
sso_session = my-sso
sso_account_id = 111122223333
sso_role_name = SampleRole2
```

```
[sso-session my-sso]
sso_region = us-east-1
sso_start_url = https://my-sso-portal.awsapps.com/start
```

However, `sso_account_id` and `sso_role_name` aren't required for all scenarios of SSO token configuration. If your application only uses AWS services that support bearer authentication, then traditional AWS credentials are not needed. Bearer authentication is an HTTP authentication scheme that uses security tokens called bearer tokens. In this scenario, `sso_account_id` and `sso_role_name` aren't required. See the individual guide for your AWS service to determine if it supports bearer token authorization.

Additionally, registration scopes can be configured as part of a `sso-session`. Scope is a mechanism in OAuth 2.0 to limit an application's access to a user's account. An application can request one or more scopes, and the access token issued to the application will be limited to the scopes granted. These scopes define the permissions requested to be authorized for the registered OIDC client and access tokens retrieved by the client. The following example sets `sso_registration_scopes` to provide access for listing accounts/roles:

```
[sso-session my-sso]
sso_region = us-east-1
sso_start_url = https://my-sso-portal.awsapps.com/start
sso_registration_scopes = sso:account:access
```

The authentication token is cached to disk under the `~/.aws/sso/cache` directory with a filename based on the session name.

Legacy non-refreshable configuration for AWS IAM Identity Center

This topic describes how to configure the AWS CLI to authenticate users with AWS IAM Identity Center (IAM Identity Center) to get credentials to run AWS CLI commands using the legacy method. When using the legacy non-refreshable configuration, you need to manually refresh the token as it periodically expires.

When using IAM Identity Center, you can login to Active Directory, a built-in IAM Identity Center directory, or [another IdP connected to IAM Identity Center](#). You can map these credentials to an AWS Identity and Access Management (IAM) role where you can run AWS CLI commands.

Regardless of which IdP you use, IAM Identity Center abstracts those distinctions away. For example, you can connect Microsoft Azure AD as described in the blog article [The Next Evolution in IAM Identity Center](#).

Note

For information on using bearer auth, which uses no account ID and role, see [Setting up to use the AWS CLI with CodeCatalyst](#) in the *Amazon CodeCatalyst User Guide*.

You can configure one or more of your AWS CLI [named profiles](#) to use a role from a legacy IAM Identity Center in the following ways:

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

Prerequisites

- Install the AWS CLI. For more information, see [the section called “Install/Update”](#).
- You must first have access to SSO authentication within IAM Identity Center. Choose one of the following methods to access your AWS credentials.

I do not have established access through IAM Identity Center

Follow the instructions in [Getting started](#) in the *AWS IAM Identity Center User Guide*. This process activates IAM Identity Center, creates an administrative user, and adds an appropriate least-privilege permission set.

Note

For Step 6 – Create a permission set that applies least-privilege permissions. We recommend using the predefined PowerUserAccess permission set, unless your employer has created a custom permission set for this purpose.

Exit the portal and sign in again to see your AWS accounts and options for Administrator or PowerUserAccess. Select PowerUserAccess when working with the SDK. This also helps you find details about programmatic access.

I already have access to AWS through a federated identity provider managed by my employer (such as Azure AD or Okta)

Sign in to AWS through your identity provider's portal. If your Cloud Administrator has granted you PowerUserAccess (developer) permissions, you see the AWS accounts that you have access to and your permission set. Next to the name of your permission set, you see options to access the accounts manually or programmatically using that permission set.

Custom implementations might result in different experiences, such as different permission set names. If you're not sure which permission set to use, contact your IT team for help.

I already have access to AWS through the AWS access portal managed by my employer

Sign in to AWS through the AWS access portal. If your Cloud Administrator has granted you PowerUserAccess (developer) permissions, you see the AWS accounts that you have access to and your permission set. Next to the name of your permission set, you see options to access the accounts manually or programmatically using that permission set.

I already have access to AWS through a federated custom identity provider managed by my employer

Contact your IT team for help.

Automatic configuration for legacy configuration

To configure an IAM Identity Center profile to your AWS CLI

1. Run the `aws configure sso` command and provide your IAM Identity Center start URL and the AWS Region that hosts the Identity Center directory.

```
$ aws configure sso
SSO session name (Recommended):
SSO start URL [None]: https://my-sso-portal.awsapps.com/start
SSO region [None]:us-east-1
```

2. The AWS CLI attempts to open your default browser and begin the login process for your IAM Identity Center account.

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

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

Using a browser, open the following URL:

<https://device.sso.us-west-2.amazonaws.com/>

and enter the following code:

QCFK-N451

IAM Identity Center uses the code to associate the IAM Identity Center session with your current AWS CLI session. The IAM Identity Center browser page prompts you to sign in with your IAM Identity Center credentials. This gives permissions to the AWS CLI to retrieve and display the AWS accounts and roles that you are authorized to use with IAM Identity Center.

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

There are 2 AWS accounts available to you.

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

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

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

Using the account ID [123456789011](#)

There are 2 roles available to you.

> ReadOnly
FullAccess

Use the arrow keys to select the IAM role you want to use with this profile and press <ENTER>.

5. The AWS CLI confirms your role selection.

Using the role name "ReadOnly"

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

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

 **Note**

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

- A final message describes the completed profile configuration.

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

```
aws s3 ls --profile my-dev-profile
```

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

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

At this point, you have a profile that you can use to request temporary credentials. You must use the `aws sso login` command to actually request and retrieve the temporary credentials needed to run commands. For instructions, see [Use an IAM Identity Center named profile](#).

Manual configuration for legacy configuration

Automated token refresh isn't supported using the legacy non-refreshable configuration. We recommend using the SSO token configuration.

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

- [sso_start_url](#)
- [sso_region](#)
- [sso_account_id](#)
- [sso_role_name](#)

You can include any other keys and values that are valid in the `.aws/config` file, such as [region](#), [output](#), or [s3](#). To prevent errors, don't include any credential related values, such as [role_arn](#) or [aws_secret_access_key](#).

The following is an example IAM Identity Center profile in `.aws/config`:

```
[profile my-sso-profile]
sso_start_url = https://my-sso-portal.awsapps.com/start
sso_region = us-west-2
sso_account_id = 111122223333
sso_role_name = SSORoleOnlyRole
region = us-west-2
output = json
```

Your profile for temporary credentials is complete.

To run commands, you must first use the `aws sso login` command to request and retrieve your temporary credentials. For instructions, see the next section, [Use an IAM Identity Center named profile](#). The authentication token is cached to disk under the `~/.aws/sso/cache` directory with a filename based on the `sso_start_url`.

Use an IAM Identity Center named profile

This topic describes how to use the AWS CLI to authenticate users with AWS IAM Identity Center (IAM Identity Center) to get credentials to run AWS CLI commands.

Note

Whether your credentials are temporary or automatically refreshing depends on how you previously configured your profile.

Topics

- [Prerequisites](#)
- [Signing in and getting credentials](#)
- [Running a command with your IAM Identity Center profile](#)
- [Signing out of your IAM Identity Center sessions](#)

Prerequisites

You've configured an IAM Identity Center profile. See [the section called "Configure automatic token refresh"](#) and [the section called "Configure legacy non-refreshable"](#) for more information.

Signing in and getting credentials

Note

The sign in process may prompt you to allow the AWS CLI access to your data. Since the AWS CLI is built on top of the SDK for Python, permission messages may contain variations of the botocore name.

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

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

The AWS CLI opens your default browser and verifies your IAM Identity Center log in.

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

If you are not currently signed into IAM Identity Center, you must provide your IAM Identity Center credentials.

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

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

Using a browser, open the following URL:

<https://device.sso.us-west-2.amazonaws.com/>

and enter the following code:

QCFK-N451

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

Your IAM Identity Center session credentials are cached. If these credentials are temporary, it includes an expiration timestamp and when they expire, the AWS CLI requests you to sign in to IAM Identity Center again.

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

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

You can also specify which sso-session profile to use when logging in using the **--sso-session** parameter of the `aws sso login` command.

```
$ aws sso login --sso-session my-dev-session
```

Attempting to automatically open the SSO authorization page in your default browser.
If the browser does not open or you wish to use a different device to authorize this request, open the following URL:

<https://device.sso.us-west-2.amazonaws.com/>

and enter the following code:

QCFK-N451

Successfully logged into Start URL: <https://cli-reinvent.awsapps.com/start>

Running a command with your IAM Identity Center profile

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

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

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

```
$ aws s3 ls --profile my-sso-profile
```

Your short-term credentials have expired. Please sign-in to renew your credentials
SSO authorization page has automatically been opened in your default browser.
Follow the instructions in the browser to complete this authorization request.

Signing out of your IAM Identity Center sessions

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

```
$ aws sso logout
```

Successfully signed out of all SSO profiles.

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

Authenticate with short-term credentials

We recommend configuring your SDK or tool to use [IAM Identity Center authentication](#) with extended session duration options. However, you can copy and use temporary credentials that are available in the AWS access portal. New credentials will need to be copied when these expire. You can use the temporary credentials in a profile or use them as values for system properties and environment variables.

1. [Sign in to the AWS access portal](#).
2. Follow [these instructions](#) to copy IAM role credentials from the AWS access portal.
 1. For step 2 in the linked instructions, choose the AWS account and IAM role name that grants access for your development needs. This role typically has a name like **PowerUserAccess** or **Developer**.
 2. For step 4, select the **Add a profile to your AWS credentials file** option and copy the contents.
3. Create or open the shared credentials file. This file is `~/.aws/credentials` on Linux and macOS systems, and `%USERPROFILE%\.aws\credentials` on Windows. For more information, see [the section called “Configuration and credential file settings”](#).
4. Add the following text to the shared credentials file. Replace the sample values with the credentials you copied.

```
[default]
aws_access_key_id = AKIAIOSFODNN7EXAMPLE
aws_secret_access_key = wJalrXUtnFEMI/K7MDENG/bPxRfCYEXAMPLEKEY
aws_session_token =
IQoJb3JpZ2luX2IQuJb3JpZ2luX2IQuJb3JpZ2luX2IQuJb3JpZ2luX2IQuJb3JpZVERYLONGSTRINGEXAMPLE
```

5. Add your preferred default region and format to the shared config file.

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

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

When the SDK creates a service client, it will access these temporary credentials and use them for each request. The settings for the IAM role chosen in step 2a determine [how long the temporary credentials are valid](#). The maximum duration is twelve hours.

Repeat these steps each time your credentials expire.

Use an IAM role in the AWS CLI

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

Topics

- [Prerequisites](#)
- [Overview of using IAM roles](#)
- [Configuring and using a role](#)
- [Using multi-factor authentication](#)
- [Cross-account roles and external ID](#)
- [Specifying a role session name for easier auditing](#)
- [Assume role with web identity](#)
- [Clearing cached credentials](#)

Prerequisites

To run the `iam` commands, you need to install and configure the AWS CLI. For more information, see [the section called “Install/Update”](#).

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](#)), 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 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 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 user 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 user in the account 123456789012, *if* the administrator of that account explicitly grants the `sts:AssumeRole` permission to the user.

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

The trust policy doesn't actually grant permissions. The administrator of the account must delegate the permission to assume the role to individual users by attaching a policy with the appropriate

permissions. The following example shows a policy that you can attach to a user that allows the user to assume only the `marketingadminrole` role. For more information about granting a user access to assume a role, see [Granting a User Permission to Switch Roles in the IAM User Guide](#).

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

The 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 a user), see [Changing Permissions for an IAM user in the IAM User Guide](#).

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

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

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

Linux or macOS

```
$ export AWS_PROFILE=marketingadmin
```

Windows

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

For more information about configuring 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 user named anika to assume the role the policy is attached to, but only if they authenticate by using MFA.

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

Next, add a line to the role profile that specifies the ARN of the user's MFA device. The following sample config file entries show two role profiles that both use the access keys for the 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 cli-user]
region = us-west-2
output = json
```

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

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

```
$ aws iam list-users --profile role-without-mfa
```

```
An error occurred (AccessDenied) when calling the AssumeRole operation: Access denied
```

The second profile entry, `role-with-mfa`, identifies an MFA device to use. When the user attempts to run a 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.

```
$ 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 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](#).

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

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

Specifying a role session name for easier auditing

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

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

For example, you could create a role-based profile as follows.

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

This results in the role session having the following ARN.

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

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

Assume role with web identity

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

Note

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

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

role_arn

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

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](#).

AWS_ROLE_ARN

The ARN of the role to assume.

AWS_WEB_IDENTITY_TOKEN_FILE

The path to the web identity token file.

AWS_ROLE_SESSION_NAME

The name applied to this assume-role session.

Note

These environment variables currently apply only to the assume role with web identity provider. They don't apply to the general assume role provider configuration.

Clearing cached credentials

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

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

Linux or macOS

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

Windows

```
C:\> del /s /q %UserProfile%\.aws\cli\cache
```

Authenticate with IAM user credentials

Warning

To avoid security risks, don't use IAM users for authentication when developing purpose-built software or working with real data. Instead, use federation with an identity provider such as [AWS IAM Identity Center](#).

This section explains how to configure basic settings with an IAM user. These include your security credentials using the config and credentials files. To instead see configuration instructions for AWS IAM Identity Center, see [the section called "IAM Identity Center authentication"](#).

Topics

- [Step 1: Create your IAM user](#)
- [Step 2: Get your access keys](#)
- [Configure the AWS CLI](#)
 - [Using aws configure](#)
 - [Importing access keys via .CSV file](#)
 - [Directly editing the config and credentials files](#)

Step 1: Create your IAM user

Create your IAM user by following the [Creating IAM users \(console\)](#) procedure in the *IAM User Guide*.

- For **Permission options**, choose **Attach policies directly** for how you want to assign permissions to this user.
- Most "Getting Started" SDK tutorials use the Amazon S3 service as an example. To provide your application with full access to Amazon S3, select the AmazonS3FullAccess policy to attach to this user.

Step 2: Get your access keys

1. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation pane of the IAM console, select **Users** and then select the **User name** of the user that you created previously.
3. On the user's page, select the **Security credentials** page. Then, under **Access keys**, select **Create access key**.
4. For **Create access key Step 1**, choose **Command Line Interface (CLI)**.
5. For **Create access key Step 2**, enter an optional tag and select **Next**.

6. For **Create access key Step 3**, select **Download .csv file** to save a .csv file with your IAM user's access key and secret access key. You need this information for later.
7. Select Done.

Configure the AWS CLI

For general use, the AWS CLI needs the following pieces of information:

- Access key ID
- Secret access key
- AWS Region
- Output format

The AWS CLI stores this information in a *profile* (a collection of settings) named default in the 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 credentials file, see [Configuration and credential file settings](#).

To configure the AWS CLI, use one of the following procedures:

Topics

- [Using aws configure](#)
- [Importing access keys via .CSV file](#)
- [Directly editing the config and credentials files](#)

Using aws configure

For general use, the `aws configure` command is the fastest way to set up your AWS CLI installation. This configure wizard prompts you for each piece of information you need to get started. Unless otherwise specified by using the `--profile` option, the AWS CLI stores this information in the default profile.

The following example configures a default profile using 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/bPxRfCYEXAMPLEKEY
Default region name [None]: us-west-2
Default output format [None]: json
```

The following example configures a profile named `userprod` using sample values. Replace them with your own values as described in the following sections.

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

Importing access keys via .CSV file

Instead of using `aws configure` to enter in access keys, you can import the `.csv` file you downloaded after you created your access keys.

The `.csv` file must contain the following headers.

- User Name - This column must be added to your `.csv`. This is used to create the profile name when you import.
- Access key ID
- Secret access key

Note

During initial access keys creation, once you close the **Download .csv file** dialog box, you cannot access your secret access key after you close the dialog box. If you need a `.csv` file, you'll need to create one yourself with the required headers and your stored access keys information. If you do not have access to your access keys information, you need to create a new access keys.

To import the `.csv` file, use the `aws configure import` command with the `--csv` option as follows:

```
$ aws configure import --csv file://credentials.csv
```

For more information, see [aws_configure_import](#).

Directly editing the config and credentials files

To directly edit the config and credentials files, perform the following.

1. Create or open the shared AWS credentials file. This file is `~/.aws/credentials` on Linux and macOS systems, and `%USERPROFILE%\.aws\credentials` on Windows. For more information, see [the section called “Configuration and credential file settings”](#).
2. Add the following text to the shared `credentials` file. Replace the sample values in the `.csv` file that you downloaded earlier and save the file.

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

Use 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](#) environment variable.

Topics

- [Prerequisites](#)
- [Configuring a profile for Amazon EC2 metadata](#)

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 [the section called “Install/Update”](#) and [Authentication and access credentials](#).
- You understand configuration files and named profiles. For more information, see [Configuration and credential file settings](#).
- 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
```

Source 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

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

Expected output from the Credentials program

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

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

Note

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

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

 **Note**

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

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

Use the AWS CLI

This section provides information about general use, common features, and options available in the AWS Command Line Interface (AWS CLI), beyond what is written in the Configuration [the section called "Endpoints"](#) section. Instructions include how to write a command, basic structure, formatting, filtering, and locating the help content or documentation for a command.

For AWS service specific examples, see [Code examples](#) or the [AWS CLI version 2 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

- [Get help with the AWS CLI](#)
- [Command structure in the AWS CLI](#)
- [Specify parameter values for the AWS CLI](#)
- [Have the AWS CLI prompt you for commands](#)
- [Control command output from the AWS CLI](#)
- [Return codes from the AWS CLI](#)
- [Interactive commands using the AWS CLI wizards](#)
- [Create and use AWS CLI command shortcuts called aliases](#)

Get 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](#)
- [AWS CLI reference guide](#)
- [API documentation](#)

- [Troubleshooting errors](#)
- [Additional help](#)

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.

```
$ aws help
```

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

```
$ aws ec2 help
```

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

```
$ aws ec2 describe-instances help
```

The help for each command is divided into six sections:

Name

The name of the command.

NAME
describe-instances -

Description

A description of the API operation that the command invokes.

DESCRIPTION

Describes one or more of your instances.

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

...

Synopsis

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

SYNOPSIS

```
describe-instances
[--dry-run | --no-dry-run]
[--instance-ids <value>]
[--filters <value>]
[--cli-input-json <value>]
[--starting-token <value>]
[--page-size <value>]
[--max-items <value>]
[--generate-cli-skeleton]
```

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

Options

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

OPTIONS

```
--dry-run | --no-dry-run (boolean)
Checks whether you have the required permissions for the action,
```

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

```
--instance-ids (list)  
One or more instance IDs.
```

Default: Describes all your instances.

...

Examples

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

EXAMPLES

To describe an Amazon EC2 instance

Command:

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

To describe all instances with the instance type m1.small

Command:

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

To describe all instances with an Owner tag

Command:

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

...

Output

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

For `describe-instances`, the output is a list of reservation objects, each of which contains several fields and objects that contain information about the instances associated with it. This

information comes from the [API documentation for the reservation data type](#) used by Amazon EC2.

OUTPUT

```
Reservations -> (list)
  One or more reservations.

  (structure)
    Describes a reservation.

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

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

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

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

      (structure)
        Describes a security group.

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

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

    Instances -> (list)
      One or more instances.

      (structure)
        Describes an instance.

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

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

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

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

...
```

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

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

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

Windows users

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

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

AWS CLI 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 2 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 [Troubleshoot errors](#).

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](#)
- [Wait commands](#)

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*.

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*.

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.

```
$ aws <command> wait <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.

 **Note**

Not every AWS service supports wait commands. See the [AWS CLI version 2 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.

```
$ 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.

```
$ 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*.

Specify parameter values for the AWS CLI

Many parameters used in the AWS Command Line Interface (AWS CLI) are simple string or numeric values, such as the key-pair name `my-key-pair` in the following example.

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

Formatting between terminals can vary. For example, most terminals are case sensitive but Powershell is case insensitive. This means the two following command examples would yield different results for case sensitive terminals as they view `MyFile*.txt` and `myfile*.txt` as **different** parameters.

However, PowerShell would process these requests as the same as it sees `MyFile*.txt` and `myfile*.txt` as the **same** parameters.

```
$ aws s3 cp . s3://my-bucket/path --include "MyFile*.txt"  
$ aws s3 cp . s3://my-bucket/path --include "myfile*.txt"
```

For more information on PowerShell's case insensitivity, see [about_Case-Sensitivity](#) in the *PowerShell documentation*.

Sometimes you need to use quotation marks or literals around strings that include special or space characters. The rules around this formatting can also vary between terminals. For more information about using quotation marks around complex parameters, see [Quotation marks with strings in the AWS CLI](#).

Parameter topics

- [Common AWS CLI parameter types](#)
- [Quotation marks with strings in the AWS CLI](#)
- [Load AWS CLI parameters from a file](#)
- [AWS CLI skeletons and input files](#)
- [Use shorthand syntax with the AWS CLI](#)

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 "Get Help"](#).

Parameter types include:

- [String](#)
- [Timestamp](#)
- [List](#)
- [Boolean](#)
- [Integer](#)
- [Binary / blob \(binary large object\) and streaming blob](#)
- [Map](#)
- [Document](#)

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](#) to prevent unexpected results.

Some string parameters can accept binary data from a file. See [Binary files](#) 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.

By default, the AWS CLI version 2 translates all *response* DateTime values to ISO 8601 format.

You can set the timestamp format by using the [`cli_timestamp_format`](#) 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](#) 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.

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

The output indicates whether the command was well formed. This command also includes a `--no-dry-run` version of the parameter that you can use to explicitly indicate that the command should be run normally. Including it isn't necessary because this is the default behavior.

Integer

An unsigned, whole number.

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

Binary / blob (binary large object) and streaming blob

In the AWS CLI, you can pass a binary value as a string directly on the command line. There are two types of blobs:

- [Blob](#)
- [Streaming blob](#)

Blob

To pass a value to a parameter with type blob, you must specify a path to a local file that contains the binary data using the `fileb://` prefix. Files referenced using the `fileb://` prefix are always treated as raw unencoded binary. The specified path is interpreted as being relative to the current working directory. For example, the `--plaintext` parameter for `aws kms encrypt` is a blob.

```
$ aws kms encrypt \  
  --key-id 1234abcd-12ab-34cd-56ef-1234567890ab \  
  --plaintext fileb://ExamplePlaintextFile \  
  --output text \  
  --query CiphertextBlob | base64 \  
  --decode > ExampleEncryptedFile
```

Note

For backwards compatibility, you can use the `file://` prefix. There are two formats used based on the file setting [`cli_binary_format`](#) or [`--cli-binary-format`](#) command line option:

- Default for the AWS CLI version 2. If the setting's value is `base64`, files referenced using the `file://` prefix are treated as base64-encoded text.
- Default for the AWS CLI version 1. If the setting's value is `raw-in-base64-out`, files referenced using the `file://` prefix is read as text and then the AWS CLI attempts to encode it to binary.

For more information, see the file setting [`cli_binary_format`](#) or [`--cli-binary-format`](#) command line option.

Streaming blob

Streaming blobs such as `aws cloudsearchdomain upload-documents` do not use prefixes. 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:

```
$ 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](#). 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.

```
$ aws dynamodb get-item --table-name my-table --key '{"id": {"N": "1"}' \
  {
    "Item": {
      "name": {
        "S": "John"
      },
      "id": {
        "N": "1"
      }
    }
  }
```

Document

Note

[Shorthand syntax](#) 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": {"key7": "value5", "key8": "value6"}'}
```

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](#)
- [Using quotation marks inside strings](#)

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 ' '

```
$ 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.

```
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.

```
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 " ".

```
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.

```
$ 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 jq, see the [jq 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 \ '.

```
$ 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.

```
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.

```
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 [Blob](#).

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 \ character) each double quotation mark " within the JSON data structure itself, as in the following example.

```
C:\> aws ec2 run-instances ^
--image-id ami-12345678 ^
--block-device-mappings "[{\\"DeviceName\": \"/dev/sdb\", \\"Ebs\": {
\\\"VolumeSize\":20, \\"DeleteOnTermination\":false, \\"VolumeType\": \"standard\"}}]"
```

Only the outermost double quotation marks are not escaped.

Load 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](#)

- [Binary files](#)

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](#) for your terminal.

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

```
{  
    "RedrivePolicy": "{\"deadLetterTargetArn\":\"arn:aws:sqs:us-west-2:0123456789012:deadletter\", \"maxReceiveCount\":5}"  
}
```

Binary files

For commands that take binary data as a parameter, specify that the data is binary content by using the `fileb://` prefix. Commands that accept binary data include:

- **aws ec2 run-instances**: `--user-data` parameter.
- **aws s3api put-object**: `--sse-customer-key` parameter.
- **aws kms decrypt**: `--ciphertext-blob` parameter.

The following example generates a binary 256-bit AES key using a Linux command line tool, and then provides it to Amazon S3 to encrypt an uploaded file server-side.

```
$ dd if=/dev/urandom bs=1 count=32 > sse.key  
32+0 records in  
32+0 records out  
32 bytes (32 B) copied, 0.000164441 s, 195 kB/s  
$ aws s3api put-object \  
  --bucket my-bucket \  
  --key test.txt \  
  --body test.txt \  
  --sse-customer-key fileb://sse.key \  
  --sse-customer-algorithm AES256
```

```
{  
    "SSECustomerKeyMD5": "iVg8oWa8sy714+FjtesrJg==",  
    "SSECustomerAlgorithm": "AES256",  
    "ETag": "\"a6118e84b76cf98bf04bbe14b6045c6c\""  
}
```

For another example referencing a file containing JSON-formatted parameters, see [Attach an IAM managed policy to a user](#).

AWS CLI skeletons and input files

Most of the 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](#)
- [Generating a command skeleton](#)

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` and `--cli-input-yaml` parameters.

Those same commands helpfully provide the `--generate-cli-skeleton` parameter to generate a file in either JSON or YAML 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` or `--cli-input-yaml` 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` and `--cli-input-yaml` 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.

```
$ aws service command help
```

The output includes a **Synopsis** section that shows the parameters that the specified command supports.

```
$ aws iam list-users help
...
SYNOPSIS
    list-users
    ...
    [--cli-input-json | --cli-input-yaml]
    ...
    [--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.
- `yaml-input` – The generated template includes all input parameters formatted as YAML.
- `output` – The generated template includes all output parameters formatted as JSON. You can't currently request the output parameters as YAML.

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**.

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": false  
        }  
    ]  
}
```

```
        "NoDevice": "",  
    },  
],  
"Monitoring": {  
    "Enabled": true  
},  
"SubnetId": "",  
"DisableApiTermination": true,  
"InstanceInitiatedShutdownBehavior": "",  
"PrivateIpAddress": "",  
"ClientToken": "",  
"AdditionalInfo": "",  
"NetworkInterfaces": [  
    {  
        "NetworkInterfaceId": "",  
        "DeviceIndex": 0,  
        "SubnetId": "",  
        "Description": "",  
        "PrivateIpAddress": "",  
        "Groups": [  
            ""  
        ],  
        "DeleteOnTermination": true,  
        "PrivateIpAddresses": [  
            {  
                "PrivateIpAddress": "",  
                "Primary": true  
            }  
        ],  
        "SecondaryPrivateIpAddressCount": 0,  
        "AssociatePublicIpAddress": true  
    }  
],  
"IamInstanceProfile": {  
    "Arn": "",  
    "Name": ""  
},  
"EbsOptimized": true  
}
```

YAML

The following example shows how to generate a template formatted in YAML by using the value `yaml-input` for the `--generate-cli-skeleton` parameter.

```
$ aws ec2 run-instances --generate-cli-skeleton yaml-input
```

```
BlockDeviceMappings: # The block device mapping entries.  
- DeviceName: '' # The device name (for example, /dev/sdh or xvdh).  
  VirtualName: '' # The virtual device name (ephemeralN).  
  Ebs: # Parameters used to automatically set up Amazon EBS volumes when the  
    instance is launched.  
    DeleteOnTermination: true # Indicates whether the EBS volume is deleted on  
    instance termination.  
    Iops: 0 # The number of I/O operations per second (IOPS) that the volume  
    supports.  
    SnapshotId: '' # The ID of the snapshot.  
    VolumeSize: 0 # The size of the volume, in GiB.  
    VolumeType: st1 # The volume type. Valid values are: standard, io1, gp2, sc1,  
    st1.  
    Encrypted: true # Indicates whether the encryption state of an EBS volume is  
    changed while being restored from a backing snapshot.  
    KmsKeyId: '' # Identifier (key ID, key alias, ID ARN, or alias ARN) for a  
    customer managed KMS key under which the EBS volume is encrypted.  
  NoDevice: '' # Suppresses the specified device included in the block device  
  mapping of the AMI.  
ImageId: '' # The ID of the AMI.  
InstanceType: c4.4xlarge # The instance type. Valid values are: t1.micro, t2.nano,  
t2.micro, t2.small, t2.medium, t2.large, t2.xlarge, t2.2xlarge, t3.nano, t3.micro,  
t3.small, t3.medium, t3.large, t3.xlarge, t3.2xlarge, t3a.nano, t3a.micro,  
t3a.small, t3a.medium, t3a.large, t3a.xlarge, t3a.2xlarge, m1.small, m1.medium,  
m1.large, m1.xlarge, m3.medium, m3.large, m3.xlarge, m3.2xlarge, m4.large,  
m4.xlarge, m4.2xlarge, m4.4xlarge, m4.10xlarge, m4.16xlarge, m2.xlarge, m2.2xlarge,  
m2.4xlarge, cr1.8xlarge, r3.large, r3.xlarge, r3.2xlarge, r3.4xlarge, r3.8xlarge,  
r4.large, r4.xlarge, r4.2xlarge, r4.4xlarge, r4.8xlarge, r4.16xlarge, r5.large,  
r5.xlarge, r5.2xlarge, r5.4xlarge, r5.8xlarge, r5.12xlarge, r5.16xlarge,  
r5.24xlarge, r5.metal, r5a.large, r5a.xlarge, r5a.2xlarge, r5a.4xlarge,  
r5a.8xlarge, r5a.12xlarge, r5a.16xlarge, r5a.24xlarge, r5d.large, r5d.xlarge,  
r5d.2xlarge, r5d.4xlarge, r5d.8xlarge, r5d.12xlarge, r5d.16xlarge, r5d.24xlarge,  
r5d.metal, r5ad.large, r5ad.xlarge, r5ad.2xlarge, r5ad.4xlarge, r5ad.8xlarge,  
r5ad.12xlarge, r5ad.16xlarge, r5ad.24xlarge, x1.16xlarge, x1.32xlarge, x1e.xlarge,  
x1e.2xlarge, x1e.4xlarge, x1e.8xlarge, x1e.16xlarge, x1e.32xlarge, i2.xlarge,  
i2.2xlarge, i2.4xlarge, i2.8xlarge, i3.large, i3.xlarge, i3.2xlarge, i3.4xlarge,  
i3.8xlarge, i3.16xlarge, i3.metal, i3en.large, i3en.xlarge, i3en.2xlarge,  
i3en.3xlarge, i3en.6xlarge, i3en.12xlarge, i3en.24xlarge, i3en.metal, hi1.4xlarge,  
hs1.8xlarge, c1.medium, c1.xlarge, c3.large, c3.xlarge, c3.2xlarge, c3.4xlarge,  
c3.8xlarge, c4.large, c4.xlarge, c4.2xlarge, c4.4xlarge, c4.8xlarge, c5.large,  
c5.xlarge, c5.2xlarge, c5.4xlarge, c5.9xlarge, c5.12xlarge, c5.18xlarge,
```

```
c5.24xlarge, c5.metal, c5d.large, c5d.xlarge, c5d.2xlarge, c5d.4xlarge,
c5d.9xlarge, c5d.18xlarge, c5n.large, c5n.xlarge, c5n.2xlarge, c5n.4xlarge,
c5n.9xlarge, c5n.18xlarge, cc1.4xlarge, cc2.8xlarge, g2.2xlarge, g2.8xlarge,
g3.4xlarge, g3.8xlarge, g3.16xlarge, g3s.xlarge, g4dn.xlarge, g4dn.2xlarge,
g4dn.4xlarge, g4dn.8xlarge, g4dn.12xlarge, g4dn.16xlarge, cg1.4xlarge, p2.xlarge,
p2.8xlarge, p2.16xlarge, p3.2xlarge, p3.8xlarge, p3.16xlarge, p3dn.24xlarge,
d2.xlarge, d2.2xlarge, d2.4xlarge, d2.8xlarge, f1.2xlarge, f1.4xlarge, f1.16xlarge,
m5.large, m5.xlarge, m5.2xlarge, m5.4xlarge, m5.8xlarge, m5.12xlarge, m5.16xlarge,
m5.24xlarge, m5.metal, m5a.large, m5a.xlarge, m5a.2xlarge, m5a.4xlarge,
m5a.8xlarge, m5a.12xlarge, m5a.16xlarge, m5a.24xlarge, m5d.large, m5d.xlarge,
m5d.2xlarge, m5d.4xlarge, m5d.8xlarge, m5d.12xlarge, m5d.16xlarge, m5d.24xlarge,
m5d.metal, m5ad.large, m5ad.xlarge, m5ad.2xlarge, m5ad.4xlarge, m5ad.8xlarge,
m5ad.12xlarge, m5ad.16xlarge, m5ad.24xlarge, h1.2xlarge, h1.4xlarge, h1.8xlarge,
h1.16xlarge, z1d.large, z1d.xlarge, z1d.2xlarge, z1d.3xlarge, z1d.6xlarge,
z1d.12xlarge, z1d.metal, u-6tb1.metal, u-9tb1.metal, u-12tb1.metal, u-18tb1.metal,
u-24tb1.metal, a1.medium, a1.large, a1.xlarge, a1.2xlarge, a1.4xlarge, a1.metal,
m5dn.large, m5dn.xlarge, m5dn.2xlarge, m5dn.4xlarge, m5dn.8xlarge, m5dn.12xlarge,
m5dn.16xlarge, m5dn.24xlarge, m5n.large, m5n.xlarge, m5n.2xlarge, m5n.4xlarge,
m5n.8xlarge, m5n.12xlarge, m5n.16xlarge, m5n.24xlarge, r5dn.large, r5dn.xlarge,
r5dn.2xlarge, r5dn.4xlarge, r5dn.8xlarge, r5dn.12xlarge, r5dn.16xlarge,
r5dn.24xlarge, r5n.large, r5n.xlarge, r5n.2xlarge, r5n.4xlarge, r5n.8xlarge,
r5n.12xlarge, r5n.16xlarge, r5n.24xlarge.

Ipv6AddressCount: 0 # [EC2-VPC] The number of IPv6 addresses to associate with the primary network interface.

Ipv6Addresses: # [EC2-VPC] The IPv6 addresses from the range of the subnet to associate with the primary network interface.

- Ipv6Address: '' # The IPv6 address.

KernelId: '' # The ID of the kernel.

KeyName: '' # The name of the key pair.

MaxCount: 0 # [REQUIRED] The maximum number of instances to launch.

MinCount: 0 # [REQUIRED] The minimum number of instances to launch.

Monitoring: # Specifies whether detailed monitoring is enabled for the instance.

    Enabled: true # [REQUIRED] Indicates whether detailed monitoring is enabled.

Placement: # The placement for the instance.

    AvailabilityZone: '' # The Availability Zone of the instance.

    Affinity: '' # The affinity setting for the instance on the Dedicated Host.

    GroupName: '' # The name of the placement group the instance is in.

    PartitionNumber: 0 # The number of the partition the instance is in.

    HostId: '' # The ID of the Dedicated Host on which the instance resides.

    Tenancy: dedicated # The tenancy of the instance (if the instance is running in a VPC). Valid values are: default, dedicated, host.

    SpreadDomain: '' # Reserved for future use.

RamdiskId: '' # The ID of the RAM disk to select.

SecurityGroupIds: # The IDs of the security groups.
```

```
- ''  
SecurityGroups: # [default VPC] The names of the security groups.  
- ''  
SubnetId: '' # [EC2-VPC] The ID of the subnet to launch the instance into.  
UserData: '' # The user data to make available to the instance.  
AdditionalInfo: '' # Reserved.  
ClientToken: '' # Unique, case-sensitive identifier you provide to ensure the  
    idempotency of the request.  
DisableApiTermination: true # If you set this parameter to true, you can't terminate  
    the instance using the Amazon EC2 console, CLI, or API; otherwise, you can.  
DryRun: true # Checks whether you have the required permissions for the action,  
    without actually making the request, and provides an error response.  
EbsOptimized: true # Indicates whether the instance is optimized for Amazon EBS I/O.  
IamInstanceProfile: # The IAM instance profile.  
    Arn: '' # The Amazon Resource Name (ARN) of the instance profile.  
    Name: '' # The name of the instance profile.  
InstanceInitiatedShutdownBehavior: stop # Indicates whether an instance stops or  
    terminates when you initiate shutdown from the instance (using the operating system  
    command for system shutdown). Valid values are: stop, terminate.  
NetworkInterfaces: # The network interfaces to associate with the instance.  
- AssociatePublicIpAddress: true # Indicates whether to assign a public IPv4  
    address to an instance you launch in a VPC.  
    DeleteOnTermination: true # If set to true, the interface is deleted when the  
    instance is terminated.  
    Description: '' # The description of the network interface.  
    DeviceIndex: 0 # The position of the network interface in the attachment order.  
    Groups: # The IDs of the security groups for the network interface.  
- ''  
    Ipv6AddressCount: 0 # A number of IPv6 addresses to assign to the network  
    interface.  
    Ipv6Addresses: # One or more IPv6 addresses to assign to the network interface.  
    - Ipv6Address: '' # The IPv6 address.  
    NetworkInterfaceId: '' # The ID of the network interface.  
    PrivateIpAddress: '' # The private IPv4 address of the network interface.  
    PrivateIpAddresses: # One or more private IPv4 addresses to assign to the network  
    interface.  
    - Primary: true # Indicates whether the private IPv4 address is the primary  
    private IPv4 address.  
        PrivateIpAddress: '' # The private IPv4 addresses.  
    SecondaryPrivateIpAddressCount: 0 # The number of secondary private IPv4  
    addresses.  
    SubnetId: '' # The ID of the subnet associated with the network interface.  
    InterfaceType: '' # The type of network interface.  
    PrivateIpAddress: '' # [EC2-VPC] The primary IPv4 address.
```

```
ElasticGpuSpecification: # An elastic GPU to associate with the instance.  
- Type: '' # [REQUIRED] The type of Elastic Graphics accelerator.  
ElasticInferenceAccelerators: # An elastic inference accelerator to associate with  
the instance.  
- Type: '' # [REQUIRED] The type of elastic inference accelerator.  
TagSpecifications: # The tags to apply to the resources during launch.  
- ResourceType: network-interface # The type of resource to tag. Valid values  
are: client-vpn-endpoint, customer-gateway, dedicated-host, dhcp-options, elastic-  
ip, fleet, fpga-image, host-reservation, image, instance, internet-gateway,  
launch-template, natgateway, network-acl, network-interface, reserved-instances,  
route-table, security-group, snapshot, spot-instances-request, subnet, traffic-  
mirror-filter, traffic-mirror-session, traffic-mirror-target, transit-gateway,  
transit-gateway-attachment, transit-gateway-route-table, volume, vpc, vpc-peering-  
connection, vpn-connection, vpn-gateway.  
Tags: # The tags to apply to the resource.  
- Key: '' # The key of the tag.  
Value: '' # The value of the tag.  
LaunchTemplate: # The launch template to use to launch the instances.  
LaunchTemplateId: '' # The ID of the launch template.  
LaunchTemplateName: '' # The name of the launch template.  
Version: '' # The version number of the launch template.  
InstanceMarketOptions: # The market (purchasing) option for the instances.  
MarketType: spot # The market type. Valid values are: spot.  
SpotOptions: # The options for Spot Instances.  
MaxPrice: '' # The maximum hourly price you're willing to pay for the Spot  
Instances.  
SpotInstanceType: one-time # The Spot Instance request type. Valid values are:  
one-time, persistent.  
BlockDurationMinutes: 0 # The required duration for the Spot Instances (also  
known as Spot blocks), in minutes.  
ValidUntil: 1970-01-01 00:00:00 # The end date of the request.  
InstanceInterruptionBehavior: terminate # The behavior when a Spot Instance is  
interrupted. Valid values are: hibernate, stop, terminate.  
CreditSpecification: # The credit option for CPU usage of the T2 or T3 instance.  
CpuCredits: '' # [REQUIRED] The credit option for CPU usage of a T2 or T3  
instance.  
CpuOptions: # The CPU options for the instance.  
CoreCount: 0 # The number of CPU cores for the instance.  
ThreadsPerCore: 0 # The number of threads per CPU core.  
CapacityReservationSpecification: # Information about the Capacity Reservation  
targeting option.  
CapacityReservationPreference: none # Indicates the instance's Capacity  
Reservation preferences. Valid values are: open, none.  
CapacityReservationTarget: # Information about the target Capacity Reservation.
```

```
CapacityReservationId: '' # The ID of the Capacity Reservation.  
HibernationOptions: # Indicates whether an instance is enabled for hibernation.  
    Configured: true # If you set this parameter to true, your instance is enabled  
    for hibernation.  
LicenseSpecifications: # The license configurations.  
- LicenseConfigurationArn: '' # The Amazon Resource Name (ARN) of the license  
configuration.
```

Generating a command skeleton

To generate and use a parameter skeleton file

1. Run the command with the `--generate-cli-skeleton` parameter to produce either JSON or YAML and direct the output to a file to save it.

JSON

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

YAML

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

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 or YAML after you remove the elements you don't need.

JSON

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

```
}
```

YAML

```
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.

- 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
    }
}
```

YAML

```
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 either the --cli-input-json or --cli-input-yaml 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.

JSON

```
$ aws ec2 run-instances --cli-input-json file://ec2runinst.json
```

A client error (DryRunOperation) occurred when calling the RunInstances operation: Request would have succeeded, but DryRun flag is set.

YAML

```
$ aws ec2 run-instances --cli-input-yaml file://ec2runinst.yaml
```

A client error (DryRunOperation) occurred when calling the RunInstances operation: Request would have succeeded, but DryRun flag is set.

The dry run error indicates that the JSON or YAML 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  
    }  
}
```

YAML

```
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": [  
        ...  
    ]  
}
```

YAML

```
$ aws ec2 run-instances --cli-input-yaml file://ec2runinst.yaml --output yaml
```

```
OwnerId: '123456789012'  
ReservationId: 'r-d94a2b1',  
Groups":  
- ''  
Instances:  
...
```

Use 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](#)
- [Using shorthand syntax with the AWS Command Line Interface](#)

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](#) 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 [  
    {"Key": "My1stTag", "Value": "Value1"},  
    {"Key": "My2ndTag", "Value": "Value2"},  
    {"Key": "My3rdTag", "Value": "Value3"}  
'
```

Have the AWS CLI prompt you for commands

You can have the AWS CLI version 2 prompt you commands, parameters, and resources when you run an `aws` command.

Topics

- [How it works](#)
- [Auto-prompt features](#)
- [Auto-prompt modes](#)
- [Configure auto-prompt](#)

How it works

If enabled, the auto-prompt enables you to use the **ENTER** key to complete a partially entered command. After pressing the **ENTER** key, commands, parameters, and resources are suggested based on what you continue to type. The suggestions list the name of the command, parameter, or resource on the left and a description of it on the right. To select and use a suggestion, use the arrows keys to highlight a row, and then press the **SPACE** key. When you've finished entering in your command, press **ENTER** to use the command. The following example demonstrates what a suggested list from auto-prompt looks like.

```
$ aws
> aws a
    accessanalyzer          Access Analyzer
    acm                      AWS Certificate Manager
    acm-pca                 AWS Certificate Manager Private Certificate
Authority
    alexaforbusiness        Alexa For Business
    amplify                  AWS Amplify
```

Auto-prompt features

The auto-prompt contains the following useful features:

Documentation panel

Provides the help documentation for the current command. To open the documentation, press the **F3** key.

Command completion

Suggests aws commands to use. To see a list, partially enter the command. The following example is searching for a service starting with the letter a.

```
$ aws
> aws a
    accessanalyzer          Access Analyzer
    acm                      AWS Certificate Manager
    acm-pca                 AWS Certificate Manager Private Certificate
Authority
    alexaforbusiness        Alexa For Business
    amplify                  AWS Amplify
```

Parameter completion

After a command is typed, auto-prompt starts to suggest parameters. The descriptions for the parameters include the value type, and a description of what the parameter is. Required parameters are listed first, and are labeled as required. The following example shows the auto-prompt list of parameters for `aws dynamodb describe-table`.

```
$ aws dynamodb describe-table
> aws dynamodb describe-table
    --table-name (required) [string] The name of the
    table to describe.
    --cli-input-json      [string] Reads arguments
    from the JSON string provided. The JSON string follows the format provide...
    --cli-input-yaml      [string] Reads arguments
    from the YAML string provided. The YAML string follows the format provide...
    --generate-cli-skeleton [string] Prints a JSON
    skeleton to standard output without sending an API request. If provided wit...
```

Resource completion

The auto-prompt makes AWS API calls using available AWS resource properties to suggest resource values. This allows for auto-prompt to suggest possible resources you own when entering in parameters. In the following example auto-prompt lists your table names when filling in the `--table-name` parameter for the `aws dynamodb describe-table` command.

```
$ aws dynamodb describe-table
> aws dynamodb describe-table --table-name
    Table1
    Table2
    Table3
```

Shorthand completion

For parameters that use shorthand syntax, auto-prompt suggests values to use. In the following example, auto-prompt lists shorthand syntax values for the `--placement` parameter in the `aws ec2 run-instances` command.

```
$ aws ec2 run-instances
> aws ec2 run-instances --placement
    AvailabilityZone=      [string] The Availability Zone of the instance. If not
    specified, an Availability Zone wil...
```

```
Affinity= [string] The affinity setting for the instance on the Dedicated Host. This parameter is no...
GroupName= [string] The name of the placement group the instance is in.
PartitionNumber= [integer] The number of the partition the instance is in.
Valid only if the placement grou...
```

File completion

When filling out parameters in aws commands, auto-complete suggests local filenames after using the prefix `file://` or `fileb://`. In the following example, auto-prompt suggests local files after entering in `--item file://` for the `aws ec2 run-instances` command.

```
$ aws ec2 run-instances
> aws ec2 run-instances --item file://
    item1.txt
    file1.json
    file2.json
```

Region completion

When using the global parameter `--region`, auto-prompt lists possible Regions to select from. In the following example, auto-prompt suggests Regions in alphabetical order after entering in `--region` for the `aws dynamodb list-tables` command.

```
$ aws dynamodb list-tables
> aws dynamodb list-tables --region
    af-south-1
    ap-east-1
    ap-northeast-1
    ap-northeast-2
```

Profile completion

When using the global parameter `--profile`, auto-prompt lists your profiles. In the following example, auto-prompt suggests your profiles after entering in `--profile` for the `aws dynamodb list-tables` command.

```
$ aws dynamodb list-tables
> aws dynamodb list-tables --profile
    profile1
```

```
profile2  
profile3
```

Fuzzy searching

Complete commands and values that contain a specific set of characters. In the following example, auto-prompt suggests Regions that contain eu after entering in `--region eu` for the `aws dynamodb list-tables` command.

```
$ aws dynamodb list-tables  
> aws dynamodb list-tables --region west  
                                eu-west-1  
                                eu-west-2  
                                eu-west-3  
                                us-west-1
```

History

To view and run previously used commands in auto-prompt mode, press **CTRL + R**. History lists previous commands that you can select by using the arrow keys. In the following example, the auto-prompt mode history is displayed.

```
$ aws  
> aws  
      dynamodb list-tables  
      s3 ls
```

Auto-prompt modes

Auto-prompt for the AWS CLI version 2 has 2 modes that can be configured:

- **Full mode:** Uses auto-prompt each time you attempt to run an aws command, whether you manually call it using the `--cli-auto-prompt` parameter or permanently enabled it. This includes pressing **ENTER** after both a complete command or incomplete command.
- **Partial mode:** Uses auto-prompt if a command is incomplete or cannot be run due to client-side validation errors. This mode is particular useful if you have pre-existing scripts, runbooks, or you only want to be auto-prompted for commands you are unfamiliar with rather than prompted on every command.

Configure auto-prompt

To configure auto-prompt you can use the following methods in order of precedence:

- **Command line options** enable or disable auto-prompt for a single command. Use [--cli-auto-prompt](#) to call auto-prompt and [--no-cli-auto-prompt](#) to disable auto-prompt.
- **Environment variables** use the [aws_cli_auto_prompt](#) variable.
- **Shared config files** use the [cli_auto_prompt](#) setting.

Control command output from the AWS CLI

This section describes the different ways to control the output from the AWS Command Line Interface (AWS CLI). Customizing the AWS CLI output in your terminal can improve readability, streamline scripting automation and provide easier navigation through larger data sets.

The AWS CLI supports multiple [output formats](#), including [json](#), [text](#), [yaml](#), and [table](#). Some services have server-side [pagination](#) for their data and the AWS CLI provides its own client-side features for additional pagination options.

Lastly, the 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 service API. Client-side filtering is supported by the AWS CLI client using the `--query` parameter.

Server-side vs client-side output options

Server-side output options are features directly supported by the AWS service API. Any data that is filtered or paged out is not sent to the client, which can speed up HTTP response times and improve bandwidth for larger data sets.

Client-side output options are features created by the AWS CLI. All data is sent to the client, then the AWS CLI filters or pages the content displayed. Client-side operations do not save on speed or bandwidth for larger datasets.

When server-side and client-side options are used together, server-side operations are completed first and then sent to the client for client-side operations. This uses the potential speed and bandwidth savings of server-side options, while using additional AWS CLI features to get your desired output.

Topics

- [Set the AWS CLI output format](#)
- [Use AWS CLI pagination options](#)
- [Filter AWS CLI output](#)

Set 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** – The output is formatted as a [JSON](#) string.
- **yaml** – The output is formatted as a [YAML](#) string.
- **yaml-stream** – The output is streamed and formatted as a [YAML](#) string. Streaming allows for faster handling of large data types.
- **text** – 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** – 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](#) 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
```

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, --output yaml, or --output yaml-stream 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.

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 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/>.

The following is an example of JSON output.

```
$ aws iam list-users --output json
```

```
{
```

```
"Users": [  
    {  
        "Path": "/",  
        "UserName": "Admin",  
        "UserId": "AIDA111111111EXAMPLE",  
        "Arn": "arn:aws:iam::123456789012:user/Admin",  
        "CreateDate": "2014-10-16T16:03:09+00:00",  
        "PasswordLastUsed": "2016-06-03T18:37:29+00:00"  
    },  
    {  
        "Path": "/backup/",  
        "UserName": "backup-user",  
        "UserId": "AIDA222222222EXAMPLE",  
        "Arn": "arn:aws:iam::123456789012:user/backup/backup-user",  
        "CreateDate": "2019-09-17T19:30:40+00:00"  
    },  
    {  
        "Path": "/",  
        "UserName": "cli-user",  
        "UserId": "AIDA333333333EXAMPLE",  
        "Arn": "arn:aws:iam::123456789012:user/cli-user",  
        "CreateDate": "2019-09-17T19:11:39+00:00"  
    }  
]
```

YAML output format

[YAML](#) is a good choice for handling the output programmatically with services and tools that emit or consume [YAML](#)-formatted strings, such as AWS CloudFormation with its support for [YAML-formatted templates](#).

For more advanced filtering that you might not be able to do with `--query`, you can consider `yq`, a command line YAML processor. You can download `yq` in the [yq repository](#) on GitHub.

The following is an example of YAML output.

```
$ aws iam list-users --output yaml
```

```
Users:  
- Arn: arn:aws:iam::123456789012:user/Admin  
  CreateDate: '2014-10-16T16:03:09+00:00'
```

```
PasswordLastUsed: '2016-06-03T18:37:29+00:00'  
Path: /  
UserId: AIDA1111111111EXAMPLE  
UserName: Admin  
- Arn: arn:aws:iam::123456789012:user/backup/backup-user  
CreateDate: '2019-09-17T19:30:40+00:00'  
Path: /backup/  
UserId: AIDA2222222222EXAMPLE  
UserName: arq-45EFD6D1-CE56-459B-B39F-F9C1F78FBE19  
- Arn: arn:aws:iam::123456789012:user/cli-user  
CreateDate: '2019-09-17T19:30:40+00:00'  
Path: /  
UserId: AIDA3333333333EXAMPLE  
UserName: cli-user
```

YAML stream output format

The `yaml-stream` format takes advantage of the [YAML](#) format while providing more responsive/faster viewing of large data sets by streaming the data to you. You can start viewing and using YAML data before the entire query downloads.

For more advanced filtering that you might not be able to do with `--query`, you can consider `yq`, a command line YAML processor. You can download `yq` in the [yq repository](#) on *GitHub*.

The following is an example of `yaml-stream` output.

```
$ aws iam list-users --output yaml-stream
```

```
- IsTruncated: false  
Users:  
- Arn: arn:aws:iam::123456789012:user/Admin  
CreateDate: '2014-10-16T16:03:09+00:00'  
PasswordLastUsed: '2016-06-03T18:37:29+00:00'  
Path: /  
UserId: AIDA1111111111EXAMPLE  
UserName: Admin  
- Arn: arn:aws:iam::123456789012:user/backup/backup-user  
CreateDate: '2019-09-17T19:30:40+00:00'  
Path: /backup/  
UserId: AIDA2222222222EXAMPLE  
UserName: arq-45EFD6D1-CE56-459B-B39F-F9C1F78FBE19  
- Arn: arn:aws:iam::123456789012:user/cli-user
```

```
CreateDate: '2019-09-17T19:30:40+00:00'  
Path: /  
UserId: AIDA3333333333EXAMPLE  
UserName: cli-user
```

The following is an example of `yaml-stream` output in conjunction with using the `--page-size` parameter to paginate the streamed YAML content.

```
$ aws iam list-users --output yaml-stream --page-size 2
```

```
- IsTruncated: true  
Marker: ab1234cdef5ghi67jk8lmo9p/  
q012rs3t445uv6789w0x1y2z/345a6b78c9d00/1efgh234ij56k1mno78pqrstu90vwxyx  
Users:  
- Arn: arn:aws:iam::123456789012:user/Admin  
  CreateDate: '2014-10-16T16:03:09+00:00'  
  PasswordLastUsed: '2016-06-03T18:37:29+00:00'  
  Path: /  
  UserId: AIDA1111111111EXAMPLE  
  UserName: Admin  
- Arn: arn:aws:iam::123456789012:user/backup/backup-user  
  CreateDate: '2019-09-17T19:30:40+00:00'  
  Path: /backup/  
  UserId: AIDA2222222222EXAMPLE  
  UserName: arq-45EFD6D1-CE56-459B-B39F-F9C1F78FBE19  
- IsTruncated: false  
Users:  
- Arn: arn:aws:iam::123456789012:user/cli-user  
  CreateDate: '2019-09-17T19:30:40+00:00'  
  Path: /  
  UserId: AIDA3333333333EXAMPLE  
  UserName: cli-user
```

Text output format

The text format organizes the AWS CLI output into tab-delimited lines. It works well with traditional Unix text tools such as `grep`, `sed`, and `awk`, and the text processing performed by PowerShell.

The text output format follows the basic structure shown below. The columns are sorted alphabetically by the corresponding key names of the underlying JSON object.

```
IDENTIFIER sorted-column1 sorted-column2
IDENTIFIER2 sorted-column1 sorted-column2
```

The following is an example of 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](#) 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 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.

```
$ aws iam list-users --output text --query 'Users[*].
[UserName,Arn,CreateDate,PasswordLastUsed,UserId]'
```

```
Admin      arn:aws:iam::123456789012:user/Admin
2014-10-16T16:03:09+00:00  2016-06-03T18:37:29+00:00  AIDA111111111EXAMPLE
backup-user  arn:aws:iam::123456789012:user/backup-user
2019-09-17T19:30:40+00:00  None                  AIDA222222222EXAMPLE
cli-user    arn:aws:iam::123456789012:user/cli-backup
2019-09-17T19:11:39+00:00  None                  AIDA333333333EXAMPLE
```

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.

```
$ aws ec2 describe-instances --query 'Reservations[*].Instances[*].
[Placement.AvailabilityZone, State.Name, InstanceId]' --output text
```

```
us-west-2a      running i-4b41a37c
us-west-2a      stopped i-a071c394
us-west-2b      stopped i-97a217a0
us-west-2a      running i-3045b007
us-west-2a      running i-6fc67758
```

```
$ aws ec2 describe-instances --query 'Reservations[*].Instances[*].
[Placement.AvailabilityZone, State.Name, InstanceId]' --output text | grep us-west-2a |
grep running | awk '{print $3}'
```

```
i-4b41a37c
i-3045b007
i-6fc67758
```

The following example goes a step further and shows not only how to filter the output, but how to use that output to automate changing instance types for each stopped instance.

```
$ aws ec2 describe-instances --query 'Reservations[*].Instances[*].[State.Name,
InstanceId]' --output text |
> grep stopped |
> awk '{print $2}' |
```

```
> while read line;
> do aws ec2 modify-instance-attribute --instance-id $line --instance-type '{"Value": "m1.medium"}';
> done
```

The text output can also be useful in PowerShell. Because the columns in text output are tab delimited, you can easily split the output into an array by using PowerShell's `t delimiter. The following command displays the value of the third column (InstanceId) if the first column (AvailabilityZone) matches the string us-west-2a.

```
PS C:\>aws ec2 describe-instances --query 'Reservations[*].Instances[*].[Placement.AvailabilityZone, State.Name, InstanceId]' --output text |
%{if ($.split("`t")[0] -match "us-west-2a") { $.split("`t")[2]; } }
```

```
-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.

```
(aws ec2 describe-instances --output json | ConvertFrom-Json).Reservations.Instances.InstanceId
```

Tip

If you output text, and filter the output to a single field using the --query parameter, the output is a single line of tab-separated values. To get each value onto a separate line, you can put the output field in brackets, as shown in the following examples.

Tab separated, single-line output:

```
$ aws iam list-groups-for-user --user-name susan --output text --query "Groups[].GroupName"
```

```
HRDepartment Developers SpreadsheetUsers LocalAdmins
```

Each value on its own line by putting [GroupName] in brackets:

```
$ aws iam list-groups-for-user --user-name susan --output text --query "Groups[].[GroupName]"
```

```
HRDepartment  
Developers  
SpreadsheetUsers  
LocalAdmins
```

Table output format

The table format produces human-readable representations of complex AWS CLI output in a tabular form.

```
$ aws iam list-users --output table
```

```
|  
+-----+  
+-----+  
||  
|| Users ||  
|+-----+-----+-----+-----+  
+-----+-----+-----+-----+  
|| Arn | CreateDate |  
| PasswordLastUsed | Path | UserId | UserName ||  
|+-----+-----+-----+-----+  
+-----+-----+-----+-----+  
|| arn:aws:iam::123456789012:user/Admin | 2014-10-16T16:03:09+00:00 | | |
| 2016-06-03T18:37:29+00:00 | / | AIDA111111111EXAMPLE | Admin ||  
|| arn:aws:iam::123456789012:user/backup/backup-user | 2019-09-17T19:30:40+00:00 |  
| / | AIDA222222222EXAMPLE | backup-user ||  
|| arn:aws:iam::123456789012:user/cli-user | 2019-09-17T19:11:39+00:00 |  
| / | AIDA333333333EXAMPLE | cli-user ||
```

```
+-----  
+-----
```

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 [Filter AWS CLI output](#).

```
$ aws ec2 describe-volumes --query 'Volumes[*].  
{ID:VolumeId,InstanceId:Attachments[0].InstanceId,AZ:AvailabilityZone,Size:Size}' --  
output table
```

```
+-----  
|           DescribeVolumes           |  
+-----+-----+-----+  
|   AZ     |   ID      | InstanceId | Size  |  
+-----+-----+-----+  
| us-west-2a| vol-e11a5288 | i-a071c394 | 30   |  
| us-west-2a| vol-2e410a47 | i-4b41a37c | 8    |  
+-----+-----+-----+
```

```
$ aws ec2 describe-volumes --query 'Volumes[*].  
[VolumeId,Attachments[0].InstanceId,AvailabilityZone,Size]' --output table
```

```
+-----  
|           DescribeVolumes           |  
+-----+-----+-----+  
| vol-e11a5288| i-a071c394 | us-west-2a | 30 |  
| vol-2e410a47| i-4b41a37c | us-west-2a | 8  |  
+-----+-----+-----+
```

Use AWS CLI pagination options

This topic describes the different ways to paginate output from the AWS CLI.

There are primarily two ways to control pagination from the AWS CLI.

- [Using server-side pagination parameters](#).
- [Using your default output client-side paging program](#).

Server-side pagination parameters process first and any output is sent to client-side pagination.

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.

The options include the following:

- [How to use the --no-paginate parameter](#)
- [How to use the --page-size parameter](#)
- [How to use the --max-items parameter](#)
- [How to use the --starting-token parameter](#)

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 \
  --max-items 100
{
  "NextToken": "eyJNYXJrZXIiOiBudWxsLCAiYm90b190cnVuY2F0ZV9hbW91bnQiOiaxfQ==",
  "Contents": [
    ...
  ]
}
```

How to use the --starting-token parameter

If the number of items output (`--max-items`) is fewer than the total number of items returned by the underlying API calls, the output includes a `NextToken` that you can pass to a subsequent command to retrieve the next set of items. The following example shows how to use the `NextToken` value returned by the previous example, and enables you to retrieve the second 100 items.

Note

The parameter `--starting-token` cannot be null or empty. If the previous command does not return a `NextToken` value, there are no more items to return and you do not need to call the command again.

```
$ aws s3api list-objects \
  --bucket my-bucket \
  --max-items 100 \
  --starting-token eyJNYXJrZXIi0iBudWxsLCAiYm90b190cnVuY2F0ZV9hbW91bnQiOiaxfQ==
{
    "Contents": [
        ...
    ]
}
```

The specified AWS service might not return items in the same order each time you call. If you specify different values for `--page-size` and `--max-items`, you can get unexpected results with missing or duplicated items. To prevent this, use the same number for `--page-size` and `--max-items` to sync the AWS CLI pagination with the pagination of the underlying service. You can also retrieve the whole list and perform any necessary paging operations locally.

Client-side pager

AWS CLI version 2 provides the use of a client-side pager program for output. By default, this feature returns all output through your operating system's default pager program.

In order of precedence, you can specify the output pager in the following ways:

- Using the `cli_pager` setting in the config file in the default or named profile.
- Using the `AWS_PAGER` environment variable.
- Using the `PAGER` environment variable.

In order of precedence, you can disable all use of an external paging program in the following ways:

- Use the `--no-cli-pager` command line option to disable the pager for a single command use.
- Set the `cli_pager` setting or `AWS_PAGER` variable to an empty string.

Client-side pager topics:

- [How to use the cli_pager setting](#)
- [How to use the AWS_PAGER environment variable](#)
- [How to use the --no-cli-pager option](#)
- [How to use pager flags](#)

How to use the cli_pager setting

You can save your frequently used configuration settings and credentials in files that are maintained by the AWS CLI. Settings in a name profile take precedence over settings in the default profile. For more information on configuration settings, see [Configuration and credential file settings](#).

The following example sets the default output pager to the less program.

```
[default]
cli_pager=less
```

The following example sets the default to disable the use of a pager.

```
[default]
cli_pager=
```

How to use the AWS_PAGER environment variable

The following example sets the default output pager to the less program. For more information on environment variables, see [Environment variables to configure the AWS CLI](#).

Linux and macOS

```
$ export AWS_PAGER="less"
```

Windows

```
C:\> setx AWS_PAGER "less"
```

The following example disables the use of a pager.

Linux and macOS

```
$ export AWS_PAGER=""
```

Windows

```
C:\> setx AWS_PAGER ""
```

How to use the --no-cli-pager option

To disable the use of a pager on a single command, use the `--no-cli-pager` option. For more information on command line options, see [Command line options](#).

```
$ aws s3api list-objects \
  --bucket my-bucket \
  --no-cli-pager
{
  "Contents": [
    ...
  ]
}
```

How to use pager flags

You can specify flags to use automatically with your paging program. Flags are dependent on the paging program you use. The below examples are for the typical defaults of `less` and `more`.

Linux and macOS

If you do not specify otherwise, the pager AWS CLI version 2 uses by default is `less`. If you don't have the `LESS` environment variable set, the AWS CLI version 2 uses the `FRX` flags. You can combine flags by specifying them when setting the AWS CLI pager.

The following example uses the `S` flag. This flag then combines with the default `FRX` flags to create a final `FRXS` flag.

```
$ export AWS_PAGER="less -S"
```

If you don't want any of the `FRX` flags, you can negate them. The following example negates the `F` flag to create a final `RX` flag.

```
$ export AWS_PAGER="less -+F"
```

For more information on less flags see [less](#) on *manpages.org*.

Windows

If you do not specify otherwise, the pager AWS CLI version 2 uses by default is more with no additional flags.

The following example uses the /c parameter.

```
C:\> setx AWS_PAGER "more /c"
```

For more information on more flags see [more](#) on *Microsoft Docs*.

Filter 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 --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 --query parameter. This parameter has capabilities the server-side filtering might not have.

Topics

- [Server-side filtering](#)
- [Client-side filtering](#)
- [Combining server-side and client-side filtering](#)
- [Additional resources](#)

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 version 2 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`, `--output yaml`, or `--output yaml-stream` 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](#)

- [Identifiers](#)
- [Selecting from a list](#)
- [Filtering nested data](#)
- [Flattening results](#)
- [Filtering for specific values](#)
- [Piping expressions](#)
- [Filtering for multiple identifier values](#)
- [Adding labels to identifier values](#)
- [Functions](#)
- [Advanced --query examples](#)

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”](#).

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

```
$ 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-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
}
]
```

{

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.

```
$ 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-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
}
]
```

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 [the section called “Before you start”](#).

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-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
}
]
```

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*.

```
$ 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.

```
$ 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",
            "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
}
```

```
 }  
 ]
```

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.

```
$ 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": "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
},
]
```

Steps can also use negative numbers to filter in the reverse order of an array as shown in the following example.

```
$ 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.

```
$ 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-a1b3c7nd",  
  "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.

```
$ 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",
  "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"
  ],
  [
    "vol-a1b3c7nd",
    "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.

```
$ aws ec2 describe-volumes \
  --query 'Volumes[].[VolumeId, VolumeType, Attachments[].[InstanceId, State]]'
[
  [
    "vol-e11a5288",
    "standard",
    [
      [
        [
          {
            "InstanceId": "i-000000000000000000",
            "State": "attached"
          }
        ]
      ]
    ]
  ]
]
```

```
        "i-a071c394",
        "attached"
    ],
],
[
    "vol-2e410a47",
    "standard",
    [
        [
            [
                "i-4b41a37c",
                "attached"
            ]
        ]
    ],
[
    "vol-a1b3c7nd",
    "standard",
    [
        [
            [
                "i-1jd73kv8",
                "attached"
            ]
        ]
    ]
]
]
```

To be more readable, flatten out the expression as shown in the following example.

```
$ aws ec2 describe-volumes \
  --query 'Volumes[].[VolumeId, VolumeType, Attachments[].[InstanceId, State][][][]'
[
    "vol-e11a5288",
    "standard",
    [
        [
            "i-a071c394",
            "attached"
        ],
        "vol-2e410a47",
        "standard",
        [
            [
                "i-4b41a37c",
                "attached"
            ]
        ]
]
```

```
],
"vol-a1b3c7nd",
"standard",
[
  "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 `VolumeType` for the `VolumeType` values.

```
$ aws ec2 describe-volumes \
--query 'Volumes[].[VolumeType: VolumeType]'
[
  {
    "VolumeType": "standard",
  },
  {
    "VolumeType": "standard",
  },
  {
    "VolumeType": "standard",
  }
]
```

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]'
```

```
[  
 {  
   "VolumeId": "vol-e11a5288",  
   "VolumeType": "standard",  
   "InstanceId": "i-a071c394",  
   "State": "attached"  
 },  
 {  
   "VolumeId": "vol-2e410a47",  
   "VolumeType": "standard",  
   "InstanceId": "i-4b41a37c",  
   "State": "attached"  
 },  
 {  
   "VolumeId": "vol-a1b3c7nd",  
   "VolumeType": "standard",  
   "InstanceId": "i-1jd73kv8",  
   "State": "attached"  
 }  
 ]
```

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](#) 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-a1b3c7nd",  
   "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",  
    "us-east-1f"
```

```
]  
]
```

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 include volumes with the specified tag

The following example describes all instances with a test tag. As long as there is another tag beside test attached to the volume, the volume is still returned in the results.

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`]'
```

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}'
[
{
    "Id": "vol-0be9bb0bf12345678",
    "Size": 80,
    "VolumeType": "gp2"
}
]
```

The following example retrieves a list of images that meet several criteria. It then uses the `--query` parameter to sort the output by `CreationDate`, selecting only the most recent. Finally, it displays the `ImageId` of that one image.

```
$ aws ec2 describe-images \
--owners amazon \
```

```
--filters "Name=name,Values=amzn*gp2" "Name=virtualization-type,Values=hvm"  
"Name=root-device-type,Values=ebs" \  
--query "sort_by(Images, &CreationDate)[-1].ImageId" \  
--output text  
ami-00ced3122871a4921
```

The following example 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

AWS CLI autoprompt

When beginning to use filter expressions, you can use the auto-prompt feature in the AWS CLI version 2. The auto-prompt feature provides a preview when you press the **F5** key. For more information, see [the section called “Auto-prompt”](#).

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](#).

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](#).

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%  
0
```

The following are the return code values that can be returned at the end of running an AWS Command Line Interface (AWS CLI) command.

Code	Meaning
0	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.
1	One or more Amazon S3 transfer operations failed. <i>Limited to S3 commands.</i>

Code	Meaning
2	<p>The meaning of this return code depends on the command:</p> <ul style="list-style-type: none">• <i>Applicable to all AWS CLI commands</i> – 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.• <i>Limited to S3 commands</i> – One or more files marked for transfer were skipped during the transfer process. However, all other files marked for transfer were successfully transferred. Files that are skipped during the transfer process include: files that don't exist; files that are character special devices, block special device, FIFO queues, or sockets; and files that the user doesn't have read permissions to.
130	The command was interrupted by a SIGINT. This is the signal sent by you to cancel a command with Ctrl+C.
252	Command syntax was invalid, an unknown parameter was provided, or a parameter value was incorrect and prevented the command from running.
253	The system environment or configuration was invalid. While the command provided might be syntactically valid, missing configuration or credentials prevented the command from running.
254	The command successfully parsed and a request made to the specified service but the service returned an error. This will generally indicate incorrect API usage or other service specific issues.
255	The command failed. There were errors generated by the AWS CLI or by the AWS service to which the request was sent.

Interactive commands using the AWS CLI wizards

The AWS Command Line Interface (AWS CLI) provides the ability to use a wizard for some commands. To contribute or view the full list of available AWS CLI wizards, see the [AWS CLI wizards folder](#) on GitHub.

How it works

Similar to the AWS console, the AWS CLI has a UI wizard that guides you through managing your AWS resources. To use the wizard, you call the `wizard` subcommand and the wizard name after the service name in a command. The command structure is as follows:

Syntax:

```
$ aws <command> wizard <wizardName>
```

The following example is calling the wizard to create a new dynamodb table.

```
$ aws dynamodb wizard new-table
```

`aws configure` is the only wizard that does not have a wizard name. When running the wizard, run the `aws configure wizard` command as the following example demonstrates:

```
$ aws configure wizard
```

After calling a wizard, a form in the shell is displayed. For each parameter, you are either provided a list of options to select from or prompted to enter in a string. To select from a list, use your up and down arrow keys and press **ENTER**. To view details on an option, press the right arrow key. When you've finished filling out a parameter, press **ENTER**.

```
$ aws configure wizard
What would you like to configure
> Static Credentials
  Assume Role
  Process Provider
  Additional CLI configuration
Enter the name of the profile:
Enter your Access Key Id:
Enter your Secret Access Key:
```

To edit previous prompts, use **SHIFT + TAB**. For some wizards, after filling in all prompts, you can preview an AWS CloudFormation template or the AWS CLI command filled with your information. This preview mode is useful to learn the AWS CLI, service APIs, and creating templates for scripts.

Press **ENTER** after previewing or the last prompt to run the final command.

```
$ aws configure wizard  
What would you like to configure  
Enter the name of the profile: testWizard  
Enter your Access Key Id: AB1C2D3EF4GH5I678J90K  
Enter your Secret Access Key: ab1c2def34gh5i67j8k90l1mnop2qr3s45tu678v90  
<ENTER>
```

Create and use AWS CLI command shortcuts called 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](#)
- [Step 1: Creating the alias file](#)
- [Step 2: Creating an alias](#)
- [Step 3: Calling an alias](#)
- [Alias repository examples](#)
- [Resources](#)

Prerequisites

To use alias commands, you need to complete the following:

- Install and configure the AWS CLI. For more information, see [the section called “Install/Update”](#) and [Authentication and access credentials](#).
- 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.

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 Account filter and text output options.

```
whoami2 = sts get-caller-identity --query Account --output text
```

Creating a sub-command alias



The sub-command alias feature requires a minimum AWS CLI version of 1.11.24 or 2.0.0

You can create an alias for sub-commands by adding a command using the following syntax in the alias file you created in the previous step.

Syntax

```
[command commandGroup]  
aliasname = command [--options]
```

The *commandGroup* is the command namespace, e.g. The command `aws ec2 describe-regions` is under the `ec2` command group. 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 ec2 regions` using the [`aws ec2 describe-regions`](#) command. Since this alias calls an existing AWS CLI command under the `ec2` command namespace, you can write the command without the `aws ec2` prefix.

```
[command ec2]  
regions = describe-regions --query Regions[].RegionName
```

To create aliases from commands outside of the command namespace, prefix the full command with an exclamation mark. The following example creates an alias named `aws ec2 instance-profiles` using the [`aws iam list-instance-profiles`](#) command.

```
[command ec2]  
instance-profiles = !aws iam list-instance-profiles
```

Note

Aliases only use existing command namespaces and you cannot create new ones. e.g. You can't create an alias with the `[command johnsmith]` section as the `johnsmith` command namespace does not already exist.

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 command alias.

```
$ aws  
  whoami  
{  
    "UserId": "A12BCD34E5FGHI6JKLM",  
    "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
```

The following example uses the aws ec2 regions [sub-command alias](#).

```
$ aws ec2  
  regions  
[  
  "ap-south-1",  
  "eu-north-1",  
  "eu-west-3",  
  "eu-west-2",  
  ...
```

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-2k1mn34567"  
}
```

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.

```
$ aws textalert $phone  
$message  
usage: aws [options] <command> <subcommand> [<subcommand> ...] [parameters]  
To see help text, you can run:  
  
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

```
$ git clone https://github.com/awslabs/awscli-aliases.git
$ mkdir -p ~/.aws/cli
$ cp awscli-aliases/alias ~/.aws/cli/alias
```

Windows

```
C:\> git clone https://github.com/awslabs/awscli-aliases.git
C:\> md %USERPROFILE%\aws\cli
C:\> copy awscli-aliases\alias %USERPROFILE%\aws\cli
```

- 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.
5. To verify the aliases are working, run the following alias.

```
$ aws whoami
```

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* 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*.
- [aws sts get-caller-identity](#)
- [aws ec2 describe-instances](#)
- [aws sns publish](#)

Code examples

This chapter provides a collection of examples that show you how to use the AWS Command Line Interface (AWS CLI) with AWS services.

The AWS CLI has the following types of examples in this guide:

- **Guided command examples** - Guided command examples for the AWS CLI User Guide on how to use the AWS CLI with some AWS services. These are often more detailed examples than the examples from the [AWS CLI version 2 reference guide](#).
- **AWS CLI using Bash scripting code examples** - Open source bash scripting examples. Bash scripting examples are hosted in the [AWS Code Examples Repository](#) on *GitHub*.

Example feedback

Can't find what you need? Request a command example by using the **Provide feedback** link at the bottom of this page or on the relevant command page in the [AWS CLI version 2 reference guide](#).

Want to contribute? Contribute AWS CLI command examples in the [AWS Code Examples Repository](#) on *GitHub*. For more information on contributing, see [AWS CLI code example contribution quick steps](#) on *GitHub pages*.

Guided AWS CLI command examples

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 version 2 reference guide](#), or use the built-in command line help. For more information, see [Get help with the AWS CLI](#).

Services

- [Use Amazon DynamoDB with the AWS CLI](#)

- [Use Amazon EC2 with the AWS CLI](#)
- [Use Amazon S3 Glacier with the AWS CLI](#)
- [Use AWS Identity and Access Management from the AWS CLI](#)
- [Use Amazon S3 with the AWS CLI](#)
- [Use Amazon SNS with the AWS CLI](#)

Use Amazon DynamoDB with the AWS CLI

An introduction to Amazon DynamoDB

[What is Amazon DynamoDB?](#)

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 [dynamodb](#) in the *AWS CLI Command Reference*.

To list the AWS CLI commands for DynamoDB, use the following command.

```
$ aws dynamodb help
```

Topics

- [Prerequisites](#)
- [Creating and using DynamoDB tables](#)
- [Using DynamoDB Local](#)
- [Resources](#)

Prerequisites

To run the dynamodb commands, you need to:

- Install and configure the AWS CLI. For more information, see [the section called “Install/Update”](#) and [Authentication and access credentials](#).

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](#) 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
  AttributeName=SongTitle,AttributeType=S \
  --key-schema AttributeName=Artist,KeyType=HASH
  AttributeName=SongTitle,KeyType=RANGE \
  --provisioned-throughput ReadCapacityUnits=1,WriteCapacityUnits=1
```

You can add new lines to the table with commands similar to those shown in the following example. These examples use a combination of shorthand syntax and JSON.

```
$ aws dynamodb put-item \
  --table-name MusicCollection \
  --item '{
    "Artist": {"S": "No One You Know"},
    "SongTitle": {"S": "Call Me Today"} ,
    "AlbumTitle": {"S": "Somewhat Famous"}
  }' \
  --return-consumed-capacity TOTAL
{
  "ConsumedCapacity": {
    "CapacityUnits": 1.0,
    "TableName": "MusicCollection"
  }
}
```

```
$ aws dynamodb put-item \
  --table-name MusicCollection \
  --item '{
    "Artist": {"S": "Acme Band"},
    "SongTitle": {"S": "Happy Day"} ,
    "AlbumTitle": {"S": "Songs About Life"}
  }' \
  --return-consumed-capacity TOTAL
```

```
{  
    "ConsumedCapacity": {  
        "CapacityUnits": 1.0,  
        "TableName": "MusicCollection"  
    }  
}
```

It can be difficult to compose valid JSON in a single-line command. To make this easier, the AWS CLI can read JSON files. For example, consider the following JSON snippet, which is stored in a file named `expression-attributes.json`.

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

You can use that file to issue a query request using the AWS CLI. In the following example, the content of the `expression-attributes.json` file is used as the value for the `--expression-attribute-values` parameter.

```
$ aws dynamodb query --table-name MusicCollection \  
    --key-condition-expression "Artist = :v1 AND SongTitle = :v2" \  
    --expression-attribute-values file://expression-attributes.json  
{  
    "Count": 1,  
    "Items": [  
        {  
            "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](#)

Service reference:

- [DynamoDB Local](#) in the Amazon DynamoDB Developer Guide
- [Using the AWS CLI with DynamoDB Local](#) in the Amazon DynamoDB Developer Guide

Use 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 [Configure the AWS CLI](#).

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

- [Create, display, and delete Amazon EC2 key pairs](#)
- [Create, configure, and delete security groups for Amazon EC2](#)
- [Launch, list, and terminate Amazon EC2 instances](#)
- [Change an Amazon EC2 instance type with a bash script](#)

Create, display, and delete 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](#)
- [Create a key pair](#)

- [Display your key pair](#)
- [Delete your key pair](#)
- [References](#)

Prerequisites

To run the ec2 commands, you need to:

- Install and configure the AWS CLI. For more information, see [the section called “Install/Update”](#) and [Authentication and access credentials](#).
- 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-----  
EXAMPLEKEYKCAQEAY7WZhaDsrA1W3mR1Qtvhwy0RRX8gnxgDAfRt/gx42kWxsT4rXE/b5CpSgie/  
vBoU7jLxx92pNHoFnByP+Dc21eyyz6CvjTmWA0JwfWiW5/akH7i05dSrvC7dQkW2duV5QuUdE0QW  
Z/aNxMniGQE6XAgfwlnXBwrerrQo+ZWQeqiUwwMkuEbLeJFLhMCvYURpUMSC1oehm449i1x9X1F  
G50TCFe0zf18dqqCP6GzbPaIjiU19xX/az0R9V+tpU0zEL+wmXnZt3/nHPQ5xvD20JH67km6SuPW  
oPzev/D8V+x4+bHthfSjR9Y7DvQFjfBVwHXigBdtZcU2/wei8D/HYwIDAQABoIBAGZ1kaEvnrqu  
/uler7vgIn5m71N5LKw4hJLAIW6tUT/fzvtcHK0SkbQCQXuriHmQ2MQuJX/0kn2NfjLV/ufGxbL1
```

```
mb5qwMGUnEpJaZD6QSSs3kICLwWUYUiGfc0uiSbmJoap/GTLU0W5Mfcv36PaBUNy5p53V6G7hXb2  
bahyWyJNfjLe4M86yd2YK3V2CmK+X/B0sShnJ36+hjrXPPWmV3N9zEmCdJjA+K15DYmhm/tJWSD9  
81oGk9TopEp7CkIfatEATyyZiVqoRq6k64iuM9JkA30zdXzMQexXVJ1TLZVEH0E7bh1Y9d801ozR  
oQs/FiZNAx2iijCWyv0lpjE73+kCgYEA9mZtyhkHkFDpwrSM1APaL8oNAbbjwEy7Z5Mqfq1+lIp1  
Ykril0DbLX1vRAH+yHPRit2hH0jtUNZh4Axv+cpg09qbUI3+43eEy24B7G/Uh+GTfbjsXs0xQx/x  
p9otyVwc7hsQ5TA5PZb+mvkJ50BEKzet9XcKwONBYELGhnEPe7cCgYEA06Vgov6YH1eHui9kHuws  
ayav0elc5zkxjF9nfHFJRRy21R1trw2Vdpn+9g481URrpzWVOEihvm+xTtmaZ1Sp//1kq75XDwnU  
WA8gkn603QE3fq2yN98BURsAKdJfJ5RL1HvGQvTe10HLYYXpJnEkHv+Unl2ajLivWUt5pbBrKbUC  
gYBjb0+0Zk0sCcpZ29sbzjYjpIdxErySIyRX5gV2uNQwAjLdp9Pfn295yQ+BxMBXiIycWVQiwbH  
oMo7yykABY70zd5wQewBQ4AdS1WSX4nGDtsiFxWiI5sKuAAe0CbTosy1s8w8fxoJ5Tz1sdoxNeGs  
Arq6Wv/G16zQuAE9zK9vvwKBgF+09VI/1wJBirsDGz9whVWffPrTkJNvJZzYt69qezexlsjgFKshy  
WBhd4xHZtmCqpBP1AymEjr/T01bxvARmXMnIOWIAAnNXMGB4KGSSy11mzSVAoQ+fqr+cJ3d0dyP11j  
jjb0Ed/NY8fr1NDxAVHE8BSkdsx2f6ELEyBKJSRr9snRAoGAMrTwYneXzvTskF/S5Fyu0i0egLda  
NWUH38v/nDCgEpxID5Hn3qAEcju1IjmbwlvtW+nY2jVhv7UGd8MjwUTNGItdb6nsYqm2asrnF3qs  
VRkAKKKYeGjkpUFVTrW0YfjXkfcrR/V+QFL50ndHAKJXjW7a4ejJLncTzmZSpYzwApc=  
-----END RSA PRIVATE KEY-----
```

Your private key isn't stored in AWS and can be retrieved **only** when it's created. You can't recover it later. Instead, if you lose the private key, you must create a new key pair.

If you're connecting to your instance from a Linux computer, we recommend that you use the following command to set the permissions of your private key file so that only you can read it.

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

Display your key pair

A "fingerprint" is generated from your key pair, and you can use it to verify that the private key that you have on your local machine matches the public key that's stored in AWS.

The fingerprint is an SHA1 hash taken from a DER-encoded copy of the private key. This value is captured when the key pair is created, and is stored in AWS with the public key. You can view the fingerprint in the Amazon EC2 console or by running the AWS CLI command [aws ec2 describe-key-pairs](#).

The following example displays the fingerprint for MyKeyPair.

```
$ aws ec2 describe-key-pairs --key-name MyKeyPair  
{  
  "KeyPairs": [  
    {  
      "KeyName": "MyKeyPair",
```

```
        "KeyFingerprint":  
        "1f:51:ae:28:bf:89:e9:d8:1f:25:5d:37:2d:7d:b8:ca:9f:f5:f1:6f"  
    }  
}  
}
```

For more information about keys and fingerprints, see [Amazon EC2 Key Pairs](#) in the *Amazon EC2 User Guide for Linux Instances*.

Delete your key pair

To delete a key pair, run the `aws ec2 delete-key-pair` command, substituting *MyKeyValuePair* with the name of the pair to delete.

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

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](#).

Create, configure, and delete 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](#)
- [Create a security group](#)
- [Add rules to your security group](#)
- [Delete your security group](#)
- [References](#)

Prerequisites

To run the ec2 commands, you need to:

- Install and configure the AWS CLI. For more information, see [the section called "Install/Update"](#) and [Authentication and access credentials](#).
- 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.

```
$ 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.

```
$ aws ec2 describe-security-groups --group-ids sg-903004f8
{
    "SecurityGroups": [
        {
            "IpPermissionsEgress": [
                {
                    "IpProtocol": "-1",
                    "IpRanges": [
                        {
                            "CidrIp": "0.0.0.0/0"
                        }
                    ],
                    "UserIdGroupPairs": []
                }
            ],
            "Description": "My security group",
            "IpPermissions": [],
            "GroupName": "my-sg",
            "VpcId": "vpc-1a2b3c4d",
            "OwnerId": "123456789012",
            "GroupId": "sg-903004f8"
        }
    ]
}
```

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.

```
$ 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.

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

The following command adds another rule to enable SSH to instances in the same security group.

```
$ aws ec2 authorize-security-group-ingress --group-id sg-903004f8 --protocol tcp --port 22 --cidr x.x.x.x/x
```

To view the changes to the security group, run the [`aws ec2 describe-security-groups`](#) command.

```
$ aws ec2 describe-security-groups --group-ids sg-903004f8
{
    "SecurityGroups": [
        {
            "IpPermissionsEgress": [
                {
```

```
        "IpProtocol": "-1",
        "IpRanges": [
            {
                "CidrIp": "0.0.0.0/0"
            }
        ],
        "UserIdGroupPairs": []
    }
],
"Description": "My security group"
"IpPermissions": [
{
    "ToPort": 22,
    "IpProtocol": "tcp",
    "IpRanges": [
        {
            "CidrIp": "x.x.x.x/x"
        }
    ]
    "UserIdGroupPairs": [],
    "FromPort": 22
}
],
"GroupName": "my-sg",
"OwnerId": "123456789012",
"GroupId": "sg-903004f8"
}
]
}
```

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.

```
$ 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*.

Launch, list, and terminate 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](#)
- [Launch your instance](#)
- [Add a block device to your instance](#)
- [Add a tag to your instance](#)
- [Connect to your instance](#)
- [List your instances](#)

- [Terminate your instance](#)
- [References](#)

Prerequisites

To run the ec2 commands in this topic, you need to:

- Install and configure the AWS CLI. For more information, see [the section called “Install/Update”](#) and [Authentication and access credentials](#).
- 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](#) and a [security group](#).
- 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",  
            "GroupId": "sg-903004f8"  
        }  
    ],  
    "Instances": [
```

```
{  
    "Monitoring": {  
        "State": "disabled"  
    },  
    "PublicDnsName": null,  
    "Platform": "windows",  
    "State": {  
        "Code": 0,  
        "Name": "pending"  
    },  
    "EbsOptimized": false,  
    "LaunchTime": "2013-07-19T02:42:39.000Z",  
    "PrivateIpAddress": "10.0.1.114",  
    "ProductCodes": [],  
    "VpcId": "vpc-1a2b3c4d",  
    "InstanceId": "i-5203422c",  
    "ImageId": "ami-173d747e",  
    "PrivateDnsName": "ip-10-0-1-114.ec2.internal",  
    "KeyName": "MyKeyPair",  
    "SecurityGroups": [  
        {  
            "GroupName": "my-sg",  
            "GroupId": "sg-903004f8"  
        }  
    ],  
    "ClientToken": null,  
    "SubnetId": "subnet-6e7f829e",  
    "InstanceType": "t2.micro",  
    "NetworkInterfaces": [  
        {  
            "Status": "in-use",  
            "SourceDestCheck": true,  
            "VpcId": "vpc-1a2b3c4d",  
            "Description": "Primary network interface",  
            "NetworkInterfaceId": "eni-a7edb1c9",  
            "PrivateIpAddresses": [  
                {  
                    "PrivateDnsName": "ip-10-0-1-114.ec2.internal",  
                    "Primary": true,  
                    "PrivateIpAddress": "10.0.1.114"  
                }  
            ],  
            "PrivateDnsName": "ip-10-0-1-114.ec2.internal",  
            "Attachment": {  
                "DeviceIndex": 0,  
                "DeleteOnTermination": true,  
                "Status": "attached",  
                "VolumeId": "vol-00000000000000000000000000000000",  
                "VolumeSize": 8,  
                "VolumeType": "standard",  
                "AttachmentId": "attachment-i-5203422c-00000000000000000000000000000000",  
                "AttachTime": "2013-07-19T02:42:39.000Z",  
                "DeleteOnTermination": true  
            }  
        }  
    ]  
}
```

```
        "Status": "attached",
        "DeviceIndex": 0,
        "DeleteOnTermination": true,
        "AttachmentId": "eni-attach-52193138",
        "AttachTime": "2013-07-19T02:42:39.000Z"
    },
    "Groups": [
        {
            "GroupName": "my-sg",
            "GroupId": "sg-903004f8"
        }
    ],
    "SubnetId": "subnet-6e7f829e",
    "OwnerId": "123456789012",
    "PrivateIpAddress": "10.0.1.114"
},
],
"SourceDestCheck": true,
"Placement": {
    "Tenancy": "default",
    "GroupName": null,
    "AvailabilityZone": "us-west-2b"
},
"Hypervisor": "xen",
"BlockDeviceMappings": [
{
    "DeviceName": "/dev/sda1",
    "Ebs": {
        "Status": "attached",
        "DeleteOnTermination": true,
        "VolumeId": "vol-877166c8",
        "AttachTime": "2013-07-19T02:42:39.000Z"
    }
}
],
"Architecture": "x86_64",
"StateReason": {
    "Message": "pending",
    "Code": "pending"
},
"RootDeviceName": "/dev/sda1",
"VirtualizationType": "hvm",
"RootDeviceType": "ebs",
"Tags": [
```

```
{  
    "Value": "MyInstance",  
    "Key": "Name"  
}  
,  
"AmiLaunchIndex": 0  
}  
]  
}
```

Add a block device to your instance

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

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

The following example parameter provisions a standard Amazon EBS volume that is 20 GB in size, and maps it to your instance using the identifier `/dev/sdf`.

```
--block-device-mappings "[{\\"DeviceName\\\":\\"/dev/sdf\\",\\"Ebs\\\":{\\"VolumeSize\\\":20,  
\\\"DeleteOnTermination\\\":false}}]"
```

The following example adds an Amazon EBS volume, mapped to `/dev/sdf`, based on an existing snapshot. A snapshot represents an image that is loaded onto the volume for you. When you specify a snapshot, you don't have to specify a volume size; it will be large enough to hold your image. However, if you do specify a size, it must be greater than or equal to the size of the snapshot.

```
--block-device-mappings "[{\\"DeviceName\\\":\\"/dev/sdf\\",\\"Ebs\\\":{\\"SnapshotId\\\":\\"snap-a1b2c3d4\\\"}}]"
```

The following example adds two volumes to your instance. The number of volumes available to your instance depends on its instance type.

```
--block-device-mappings "[{\\"DeviceName\\\":\\"/dev/sdf\\",\\"VirtualName\\\":\\"ephemeral0\\\"},  
{\\"DeviceName\\\":\\"/dev/sdg\\",\\"VirtualName\\\":\\"ephemeral1\\\"}]"
```

The following example creates the mapping (/dev/sdj), but doesn't provision a volume for the instance.

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

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

Add a tag to your instance

A tag is a label that you assign to an AWS resource. It enables you to add metadata to your resources that you can use for a variety of purposes. For more information, see [Tagging Your Resources](#) in the *Amazon EC2 User Guide for Linux Instances*.

The following example shows how to add a tag with the key name "Name" and the value "MyInstance" to the specified instance, by using the [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-05e998023d9c69f9a"  
]
```

The following command lists any of your instances that have the tag Name=MyInstance.

```
$ aws ec2 describe-instances --filters "Name>tag:Name,Values=MyInstance"
```

The following command lists your instances that were launched using any of the following AMIs: ami-x0123456, ami-y0123456, and ami-z0123456.

```
$ aws ec2 describe-instances --filters "Name=image-id,Values=ami-x0123456,ami-  
y0123456,ami-z0123456"
```

Terminate your instance

Terminating an instance deletes it. You can't reconnect to an instance after you've terminated it.

As soon as the state of the instance changes to shutting-down or terminated, you stop incurring charges for that instance. If you want to reconnect to an instance later, use [stop-instances](#) instead of [terminate-instances](#). For more information, see [Terminate Your Instance](#) in the *Amazon EC2 User Guide for Linux Instances*.

To delete an instance, you use the command [aws ec2 terminate-instances](#) to delete it.

```
$ aws ec2 terminate-instances --instance-ids i-5203422c  
{  
  "TerminatingInstances": [  
    {  
      "InstanceId": "i-5203422c",  
      "CurrentState": {  
        "Code": 32,  
        "Name": "shutting-down"  
      },  
      "PreviousState": {  
        "Code": 16,  
        "Name": "running"  
      }  
    }  
  ]
```

{}

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 with 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](#)
- [About this example](#)
- [Parameters](#)
- [Files](#)
- [References](#)

Before you start

Before you can run any of the below examples, the following things need to be completed.

- Install and configure the AWS CLI. For more information, see [the section called “Install/Update”](#) and [Authentication and access credentials](#).
- 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](#)

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*.

Use 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](#)
- [Create an Amazon S3 Glacier vault](#)
- [Prepare a file for uploading](#)
- [Initiate a multipart upload and upload files](#)
- [Complete the upload](#)
- [Resources](#)

Prerequisites

To run the `glacier` commands, you need to:

- Install and configure the AWS CLI. For more information, see [the section called “Install/Update”](#) and [Authentication and access credentials](#).
- 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
```

dd is a utility that copies a number of bytes from an input file to an output file. The previous example uses the system device file /dev/urandom as a source of random data. fsutil performs a similar function in Windows.

Windows

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

Next, split the file into 1 MiB (1,048,576 byte) chunks.

```
$ split -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": "19gaRezEXAMPLES6Ry5YYdqthH0C_kGRCT03L9yetr220UmPtBYKk-OssZtLqyFu7sY1_1R7vgFuJV6NtcV5zpsJ",
    "location": "/123456789012/vaults/myvault/multipart-uploads/19gaRezEXAMPLES6Ry5YYdqthH0C_kGRCT03L9yetr220UmPtBYKk-OssZtLqyFu7sY1_1R7vgFuJV6NtcV5zpsJ"
}
```

S3 Glacier requires the size of each part in bytes (1 MiB in this example), your vault name, and an account ID to configure the multipart upload. The AWS CLI outputs an upload ID when the operation is complete. Save the upload ID to a shell variable for later use.

Linux or macOS

```
$ UPLOADID="19gaRezEXAMPLES6Ry5YYdqthH0C_kGRCT03L9yetr220UmPtBYKk-OssZtLqyFu7sY1_1R7vgFuJV6NtcV5zpsJ"
```

Windows

```
C:\> set UPLOADID="19gaRezEXAMPLES6Ry5YYdqthH0C_kGRCT03L9yetr220UmPtBYKk-OssZtLqyFu7sY1_1R7vgFuJV6NtcV5zpsJ"
```

Next, use the [upload-multipart-part](#) command to upload each of the three parts.

```
$ aws glacier upload-multipart-part --upload-id $UPLOADID --body chunkaa --range 'bytes 0-1048575/*' --account-id - --vault-name myvault
{
    "checksum": "e1f2a7cd6e047fa606fe2f0280350f69b9f8cfa602097a9a026360a7edc1f553"
}
```

```
$ aws glacier upload-multipart-part --upload-id $UPLOADID --body chunkab --range 'bytes 1048576-2097151/*' --account-id - --vault-name myvault
{
    "checksum": "e1f2a7cd6e047fa606fe2f0280350f69b9f8cfa602097a9a026360a7edc1f553"
}
$ aws glacier upload-multipart-part --upload-id $UPLOADID --body chunkac --range 'bytes 2097152-3145727/*' --account-id - --vault-name myvault
{
    "checksum": "e1f2a7cd6e047fa606fe2f0280350f69b9f8cfa602097a9a026360a7edc1f553"
}
```

Note

The previous example uses the dollar sign (\$) to reference the contents of the UPLOADID shell variable on Linux. On the Windows command line, use a percent sign (%) on either side of the variable name (for example, %UPLOADID%).

You must specify the byte range of each part when you upload it so that 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.

To calculate a tree hash

1. If you haven't already, split the original file into 1 MiB parts.

```
$ split --bytes=1048576 --verbose largefile chunk
creating file `chunkaa'
creating file `chunkab'
creating file `chunkac'
```

2. Calculate and store the binary SHA-256 hash of each chunk.

```
$ openssl dgst -sha256 -binary chunkaa > hash1
$ openssl dgst -sha256 -binary chunkab > hash2
$ openssl dgst -sha256 -binary chunkac > hash3
```

3. Combine the first two hashes and take the binary hash of the result.

```
$ cat hash1 hash2 > hash12
$ openssl dgst -sha256 -binary hash12 > hash12hash
```

4. Combine the parent hash of chunks aa and ab with the hash of chunk ac and hash the result, this time outputting hexadecimal. Store the result in a shell variable.

```
$ cat hash12hash hash3 > hash123
$ openssl dgst -sha256 hash123
SHA256(hash123)= 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.

```
$ aws glacier complete-multipart-upload --checksum $TREEHASH --archive-size 3145728 --
upload-id $UPLOADID --account-id - --vault-name myvault
{
    "archiveId": "d3AbWhE0YE1m6f_fI1jPG82F8xzbMEEZmrA1LGAAONJAzo5QdP-
N83MKqd96Unspoia5H51ItWX-sK8-QS0ZhwsyGiu9-R-
kwWUyS1dSB1mgPPWkEbeFfqDSav053rU7FvVLHfRc6hg",
    "checksum": "9628195fcdbcbbe76cdde932d4646fa7de5f219fb39823836d81f0cc0e18aa67",
```

```
"location": "/123456789012/vaults/myvault/archives/  
d3AbWhE0YE1m6f_fI1jPG82F8xzbMEEZmrA1LGAA0NJAz05QdP-N83MKqd96Unspoa5H5lItWX-sK8-  
QS0ZhwsyGiu9-R-kwWUyS1dSB1mgPPWkEbeFfqDSav053rU7FvVLHfRc6hg"  
}
```

You can also check the status of the vault using the [describe-vault](#) command.

```
$ aws glacier describe-vault --account-id - --vault-name myvault  
{  
    "SizeInBytes": 3178496,  
    "VaultARN": "arn:aws:glacier:us-west-2:123456789012:vaults/myvault",  
    "LastInventoryDate": "2018-12-07T00:26:19.028Z",  
    "NumberOfArchives": 1,  
    "CreationDate": "2018-12-06T21:23:45.708Z",  
    "VaultName": "myvault"  
}
```

Note

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

Now it's safe to remove the chunk and hash files that you created.

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

For more information on multipart uploads, see [Uploading Large Archives in Parts](#) and [Computing Checksums](#) in the *Amazon S3 Glacier Developer Guide*.

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*

Use AWS Identity and Access Management from the AWS CLI

An introduction to AWS Identity and Access Management

[Introduction to AWS Identity and Access Management](#)

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 [Configure the AWS CLI](#).

For more information on the IAM service, see the [AWS Identity and Access Management User Guide](#).

Topics

- [Create IAM users and groups](#)
- [Attach an IAM managed policy to a user](#)
- [Set an initial password for an IAM user](#)
- [Create an access key for an IAM user](#)

Create 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 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 [Configure the AWS CLI](#).

To create a group and add a new user to it

1. Use the [create-group](#) command to create the group.

```
$ aws iam create-group --group-name MyIamGroup
{
    "Group": {
        "GroupName": "MyIamGroup",
        "CreateDate": "2018-12-14T03:03:52.834Z",
        "GroupId": "AGPAJNUJ2W4IJVEXAMPLE",
        "Arn": "arn:aws:iam::123456789012:group/MyIamGroup",
        "Path": "/"
    }
}
```

2. Use the [create-user](#) command to create the user.

```
$ aws iam create-user --user-name MyUser
{
    "User": {
        "UserName": "MyUser",
        "Path": "/",
        "CreateDate": "2018-12-14T03:13:02.581Z",
        "UserId": "AIDAJY2PE5XUZ4EXAMPLE",
        "Arn": "arn:aws:iam::123456789012:user/MyUser"
    }
}
```

3. Use the [add-user-to-group](#) command to add the user to the group.

```
$ aws iam add-user-to-group --user-name MyUser --group-name MyIamGroup
```

- To verify that the MyIamGroup group contains the MyUser, use the [get-group](#) command.

```
$ aws iam get-group --group-name MyIamGroup
{
    "Group": {
        "GroupName": "MyIamGroup",
        "CreateDate": "2018-12-14T03:03:52Z",
        "GroupId": "AGPAJNUJ2W4IJVEXAMPLE",
        "Arn": "arn:aws:iam::123456789012:group/MyIamGroup",
        "Path": "/"
    },
    "Users": [
        {
            "UserName": "MyUser",
            "Path": "/",
            "CreateDate": "2018-12-14T03:13:02Z",
            "UserId": "AIDAJY2PE5XUZ4EXAMPLE",
            "Arn": "arn:aws:iam::123456789012:user/MyUser"
        }
    ],
    "IsTruncated": "false"
}
```

Attach an IAM managed policy to a 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 a 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 [Configure the AWS CLI](#).

To attach an IAM managed policy to a user

- Determine the Amazon Resource Name (ARN) of the policy to attach. The following command uses `list-policies` to find the ARN of the policy with the name `PowerUserAccess`. It then stores that ARN in an environment variable.

```
$ export POLICYARN=$(aws iam list-policies --query 'Policies[?PolicyName==`PowerUserAccess`].{ARN:Arn}' --output text)
```

```
$ echo $POLICYARN  
arn:aws:iam::aws:policy/PowerUserAccess
```

- To attach the policy, use the [attach-user-policy](#) command, and reference the environment variable that holds the policy ARN.

```
$ aws iam attach-user-policy --user-name MyUser --policy-arn $POLICYARN
```

- Verify that the policy is attached to the user by running the [list-attached-user-policies](#) command.

```
$ aws iam list-attached-user-policies --user-name MyUser  
{  
    "AttachedPolicies": [  
        {  
            "PolicyName": "PowerUserAccess",  
            "PolicyArn": "arn:aws:iam::aws:policy/PowerUserAccess"  
        }  
    ]  
}
```

For more information, see [Access Management Resources](#). This topic provides links to an overview of permissions and policies, and links to examples of policies for accessing Amazon S3, Amazon EC2, and other services.

Set 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 [Configure the AWS CLI](#).

The following command uses [create-login-profile](#) to set an initial password on the specified user. When the user signs in for the first time, the user is required to change the password to something that only the user knows.

```
$ aws iam create-login-profile --user-name MyUser --password My!User1Login8P@ssword --password-reset-required
```

```
{  
    "LoginProfile": {  
        "UserName": "MyUser",  
        "CreateDate": "2018-12-14T17:27:18Z",  
        "PasswordResetRequired": true  
    }  
}
```

You can use the `update-login-profile` command to *change* the password for a user.

```
$ aws iam update-login-profile --user-name MyUser --password My!User1ADifferentP@ssword
```

Create an access key for an IAM user

This topic describes how to use AWS Command Line Interface (AWS CLI) commands to create a set of access keys for an AWS Identity and Access Management (IAM) user. 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 [Configure the AWS CLI](#).

You can use the `create-access-key` command to create an access key for a user. An access key is a set of security credentials that consists of an access key ID and a secret key.

A 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/bPxRfCYEXAMPLEKEY",  
        "CreateDate": "2018-12-14T17:34:16Z"  
    }  
}
```

Use the `delete-access-key` command to delete an access key for a 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
```

Use 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:

- [Use high-level \(s3\) commands with the AWS CLI](#)
- [Use API-Level \(s3api\) commands with the AWS CLI](#)
- [Amazon S3 bucket lifecycle operations scripting example](#)

Use 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](#)
- [Before you start](#)

- [Create a bucket](#)
- [List buckets and objects](#)
- [Delete buckets](#)
- [Delete objects](#)
- [Move objects](#)
- [Copy objects](#)
- [Sync objects](#)
- [Frequently used options for s3 commands](#)
- [Resources](#)

Prerequisites

To run the s3 commands, you need to:

- Install and configure the AWS CLI. For more information, see [the section called “Install/Update”](#) and [Authentication and access credentials](#).
- 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.

By default, the AWS CLI version 2 commands in the `s3` namespace that perform multipart copies transfers all tags and the following set of properties from the source to the destination copy: `content-type`, `content-language`, `content-encoding`, `content-disposition`, `cache-control`, `expires`, and `metadata`.

This can result in additional AWS API calls to the Amazon S3 endpoint that would not have been made if you used AWS CLI version 1. These can include: `HeadObject`, `GetObjectTagging`, and `PutObjectTagging`.

If you need to change this default behavior in AWS CLI version 2 commands, use the `--copy-props` parameter to specify one of the following options:

- **default** – The default value. Specifies that the copy includes all tags attached to the source object and the properties encompassed by the `--metadata-directive` parameter used for non-multipart copies: `content-type`, `content-language`, `content-encoding`, `content-disposition`, `cache-control`, `expires`, and `metadata`.
- **metadata-directive** – Specifies that the copy includes only the properties that are encompassed by the `--metadata-directive` parameter used for non-multipart copies. It doesn't copy any tags.
- **none** – Specifies that the copy includes none of the properties from the source 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](#). 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.

```
$ aws s3 ls  
2018-12-11 17:08:50 my-bucket  
2018-12-14 14:55:44 my-bucket2
```

The following command lists all objects and prefixes in a bucket. In this example output, the prefix example/ has one file named MyFile1.txt.

```
$ 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/*).

```
$ 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

```
$ aws s3 rb <target> [--options]
```

s3 rb examples

The following example removes the s3://bucket-name bucket.

```
$ 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.

```
$ 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

```
$ aws s3 rm <target> [--options]
```

For a few common options to use with this command, and examples, see [Frequently used options for s3 commands](#). 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
```

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](#). 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](#). 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 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](#). 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  
delete: MyFile2.rtf
```

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 version 2 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*.

```
$ 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](#)
- [aws s3 mv](#)
- [aws s3 ls](#)
- [aws s3 rb](#)
- [aws s3 rm](#)
- [aws s3 sync](#)

Service 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*
- [Listing keys hierarchically using a prefix and delimiter](#) in the *Amazon Simple Storage Service User Guide*
- [Abort multipart uploads to an S3 bucket using the AWS SDK for .NET \(low-level\)](#) in the *Amazon Simple Storage Service User Guide*

Use 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 [Use high-level \(s3\) commands with the AWS CLI](#)

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 version 2 reference guide](#).

Topics

- [Prerequisites](#)
- [Apply a custom ACL](#)
- [Configure a logging policy](#)
- [Resources](#)

Prerequisites

To run the s3api commands, you need to:

- Install and configure the AWS CLI. For more information, see [the section called “Install/Update”](#) and [Authentication and access credentials](#).
- 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": "Everyone",  
          "URI": "http://acs.amazonaws.com/groups/global/AllUsers"  
        },  
        "Permission": "READ"  
      }  
    ]  
  }  
}
```

```
{  
    "Grantee": {  
        "Type": "Group",  
        "URI": "http://acs.amazonaws.com/groups/global/AllUsers"  
    },  
    "Permission": "READ"  
}  
]  
}  
}
```

Resources

AWS CLI reference:

- [aws s3api](#)
- [aws s3api put-bucket-acl](#)
- [aws s3api put-bucket-logging](#)

Service 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*
- [Listing keys hierarchically using a prefix and delimiter](#) in the *Amazon Simple Storage Service User Guide*
- [Abort multipart uploads to an S3 bucket using the AWS SDK for .NET \(low-level\)](#) in the *Amazon Simple Storage Service User Guide*

Amazon S3 bucket lifecycle operations scripting example

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](#)
- [About this example](#)

- [Files](#)
- [References](#)

Before you start

Before you can run any of the below examples, the following things need to be completed.

- Install and configure the AWS CLI. For more information, see [the section called “Install/Update”](#) and [Authentication and access credentials](#).
- 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-operations.sh`. 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](#)
- [aws s3api put-object](#)

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*.

Use 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 [Configure the AWS CLI](#).

This topic shows examples of AWS CLI commands that perform common tasks for Amazon SNS.

Topics

- [Create a topic](#)
- [Subscribe to a topic](#)
- [Publish to a topic](#)
- [Unsubscribe from a topic](#)

- [Delete a topic](#)

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

Make a note of the response's TopicArn, which you use later to publish a message.

Subscribe to a topic

To subscribe to a topic, use the [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
{
    "SubscriptionArn": "pending confirmation"
}
```

AWS immediately sends a confirmation message by email to the address you specified in the subscribe command. The email message has the following text.

```
You have chosen to subscribe to the topic:
arn:aws:sns:us-west-2:123456789012:my-topic
To confirm this subscription, click or visit the following link (If this was in error
no action is necessary):
Confirm subscription
```

After the recipient clicks the **Confirm subscription** link, the recipient's browser displays a notification message with information similar to the following.

```
Subscription confirmed!
```

You have subscribed saanvi@example.com to the topic:my-topic.

Your subscription's id is:

arn:aws:sns:us-west-2:123456789012:my-topic:1328f057-de93-4c15-512e-8bb22EXAMPLE

If it was not your intention to subscribe, [click here to unsubscribe](#).

Publish to a topic

To send a message to all subscribers of a topic, use the [sns publish](#) command.

The following example sends the message "Hello World!" to all subscribers of the specified topic.

```
$ aws sns publish --topic-arn arn:aws:sns:us-west-2:123456789012:my-topic --  
message "Hello World!"  
{  
    "MessageId": "4e41661d-5eec-5ddf-8dab-2c867EXAMPLE"  
}
```

In this example, AWS sends an email message with the text "Hello World!" to saanvi@example.com.

Unsubscribe from a topic

To unsubscribe from a topic and stop receiving messages published to that topic, use the [sns unsubscribe](#) command and specify the ARN of the topic you want to unsubscribe from.

```
$ aws sns unsubscribe --subscription-arn arn:aws:sns:us-west-2:123456789012:my-  
topic:1328f057-de93-4c15-512e-8bb22EXAMPLE
```

To verify that you successfully unsubscribed, use the [sns list-subscriptions](#) command to confirm that the ARN no longer appears in the list.

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

Delete a topic

To delete a topic, run the [sns delete-topic](#) command.

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

To verify that AWS successfully deleted the topic, use the [sns list-topics](#) command to confirm that the topic no longer appears in the list.

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

AWS CLI with Bash script code examples

The code examples in this topic show you how to use the AWS Command Line Interface with Bash script with AWS.

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios and cross-service examples.

Scenarios are code examples that show you how to accomplish a specific task by calling multiple functions within the same service.

Cross-service examples are sample applications that work across multiple AWS services.

Examples

- [Actions and scenarios using AWS CLI with Bash script](#)

Actions and scenarios using AWS CLI with Bash script

The following code examples show how to perform actions and implement common scenarios by using the AWS Command Line Interface with Bash script with AWS services.

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios and cross-service examples.

Scenarios are code examples that show you how to accomplish a specific task by calling multiple functions within the same service.

Services

- [DynamoDB examples using AWS CLI with Bash script](#)
- [HealthImaging examples using AWS CLI with Bash script](#)
- [IAM examples using AWS CLI with Bash script](#)

- [Amazon S3 examples using AWS CLI with Bash script](#)
- [AWS STS examples using AWS CLI with Bash script](#)

DynamoDB examples using AWS CLI with Bash script

The following code examples show you how to perform actions and implement common scenarios by using the AWS Command Line Interface with Bash script with DynamoDB.

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios and cross-service examples.

Scenarios are code examples that show you how to accomplish a specific task by calling multiple functions within the same service.

Each example includes a link to GitHub, where you can find instructions on how to set up and run the code in context.

Topics

- [Actions](#)
- [Scenarios](#)

Actions

Create a table

The following code example shows how to create a DynamoDB table.

AWS CLI with Bash script

 **Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
#####
# function dynamodb_create_table
#
```

```
# This function creates an Amazon DynamoDB table.  
#  
# Parameters:  
#     -n table_name -- The name of the table to create.  
#     -a attribute_definitions -- JSON file path of a list of attributes and their  
#       types.  
#     -k key_schema -- JSON file path of a list of attributes and their key types.  
#     -p provisioned_throughput -- Provisioned throughput settings for the table.  
#  
# Returns:  
#     0 - If successful.  
#     1 - If it fails.  
#####  
function dynamodb_create_table() {  
    local table_name attribute_definitions key_schema provisioned_throughput response  
    local option OPTARG # Required to use getopt command in a function.  
  
#####  
# Function usage explanation  
#####  
function usage() {  
    echo "function dynamodb_create_table"  
    echo "Creates an Amazon DynamoDB table."  
    echo " -n table_name -- The name of the table to create."  
    echo " -a attribute_definitions -- JSON file path of a list of attributes and  
    #       their types."  
    echo " -k key_schema -- JSON file path of a list of attributes and their key  
    #       types."  
    echo " -p provisioned_throughput -- Provisioned throughput settings for the  
    #       table."  
    echo ""  
}  
  
# Retrieve the calling parameters.  
while getopt "n:a:k:p:h" option; do  
    case "${option}" in  
        n) table_name="${OPTARG}" ;;  
        a) attribute_definitions="${OPTARG}" ;;  
        k) key_schema="${OPTARG}" ;;  
        p) provisioned_throughput="${OPTARG}" ;;  
        h)  
            usage  
            return 0  
            ;;  
    esac  
done
```

```
\?)  
    echo "Invalid parameter"  
    usage  
    return 1  
;;  
esac  
done  
export OPTIND=1  
  
if [[ -z "$table_name" ]]; then  
    errecho "ERROR: You must provide a table name with the -n parameter."  
    usage  
    return 1  
fi  
  
if [[ -z "$attribute_definitions" ]]; then  
    errecho "ERROR: You must provide an attribute definitions json file path the -a parameter."  
    usage  
    return 1  
fi  
  
if [[ -z "$key_schema" ]]; then  
    errecho "ERROR: You must provide a key schema json file path the -k parameter."  
    usage  
    return 1  
fi  
  
if [[ -z "$provisioned_throughput" ]]; then  
    errecho "ERROR: You must provide a provisioned throughput json file path the -p parameter."  
    usage  
    return 1  
fi  
  
iecho "Parameters:\n"  
iecho "    table_name: $table_name"  
iecho "    attribute_definitions: $attribute_definitions"  
iecho "    key_schema: $key_schema"  
iecho "    provisioned_throughput: $provisioned_throughput"  
iecho ""  
  
response=$(aws dynamodb create-table \  
    --table-name "$table_name" \  
    --attribute-definitions "{\"AttributeName\": \"string\", \"AttributeType\": \"S\"}" \  
    --key-schema "[{\"KeyType\": \"HASH\", \"AttributeName\": \"string\"}, {\"KeyType\": \"RANGE\", \"AttributeName\": \"string\"}]" \  
    --provisioned-throughput {"\"ReadCapacityUnits\": 1, \"WriteCapacityUnits\": 1}" \  
    --region "string" \  
    --stream-specification {"\"StreamEnabled\": false, \"ShardsCount\": 1}" \  
    --tag-specifications "[{\"ResourceType\": \"TABLE\", \"Tags\": [ {\"Key\": \"string\", \"Value\": \"string\"} ]}]" \  
    --table-name "$table_name" \  
    --throughput-mode "ON_DEMAND"
```

```
--attribute-definitions file://"${attribute_definitions}" \
--key-schema file://"${key_schema}" \
--provisioned-throughput "$provisioned_throughput")\n\nlocal error_code=${?}\n\nif [[ $error_code -ne 0 ]]; then\n    aws_cli_error_log $error_code\n    errecho "ERROR: AWS reports create-table operation failed.$response"\n    return 1\nfi\n\nreturn 0\n}
```

The utility functions used in this example.

```
#####\n# function iecho\n#\n# This function enables the script to display the specified text only if\n# the global variable $VERBOSE is set to true.\n#####\nfunction iecho() {\n    if [[ $VERBOSE == true ]]; then\n        echo "$@"\n    fi\n}\n\n#####\n# function errecho\n#\n# This function outputs everything sent to it to STDERR (standard error output).\n#####\nfunction errecho() {\n    printf "%s\n" "$*" 1>&2\n}\n\n#####\n# function aws_cli_error_log()\n#\n# This function is used to log the error messages from the AWS CLI.
```

```
#  
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-help-  
# return-codes.  
#  
# The function expects the following argument:  
#       $1 - The error code returned by the AWS CLI.  
#  
# Returns:  
#       0: - Success.  
#  
#####  
function aws_cli_error_log() {  
    local err_code=$1  
    errecho "Error code : $err_code"  
    if [ "$err_code" == 1 ]; then  
        errecho " One or more S3 transfers failed."  
    elif [ "$err_code" == 2 ]; then  
        errecho " Command line failed to parse."  
    elif [ "$err_code" == 130 ]; then  
        errecho " Process received SIGINT."  
    elif [ "$err_code" == 252 ]; then  
        errecho " Command syntax invalid."  
    elif [ "$err_code" == 253 ]; then  
        errecho " The system environment or configuration was invalid."  
    elif [ "$err_code" == 254 ]; then  
        errecho " The service returned an error."  
    elif [ "$err_code" == 255 ]; then  
        errecho " 255 is a catch-all error."  
    fi  
  
    return 0  
}
```

- For API details, see [CreateTable](#) in *AWS CLI Command Reference*.

Delete a table

The following code example shows how to delete a DynamoDB table.

AWS CLI with Bash script

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
#####
# function dynamodb_delete_table
#
# This function deletes a DynamoDB table.
#
# Parameters:
#     -n table_name -- The name of the table to delete.
#
# Returns:
#     0 - If successful.
#     1 - If it fails.
#####
function dynamodb_delete_table() {
    local table_name response
    local optionOPTARG # Required to use getopt command in a function.

    # bashsupport disable=BP5008
    function usage() {
        echo "function dynamodb_delete_table"
        echo "Deletes an Amazon DynamoDB table."
        echo " -n table_name -- The name of the table to delete."
        echo ""
    }

    # Retrieve the calling parameters.
    while getopt "n:h" option; do
        case "${option}" in
            n) table_name="${OPTARG}" ;;
            h)
                usage
                return 0
                ;;
            \?)
                echo "Invalid parameter"
                ;;
        esac
    done
}
```

```
usage
return 1
;;
esac
done
export OPTIND=1

if [[ -z "$table_name" ]]; then
errecho "ERROR: You must provide a table name with the -n parameter."
usage
return 1
fi

iecho "Parameters:\n"
iecho "    table_name: $table_name"
iecho ""

response=$(aws dynamodb delete-table \
--table-name "$table_name")

local error_code=${?}

if [[ $error_code -ne 0 ]]; then
aws_cli_error_log $error_code
errecho "ERROR: AWS reports delete-table operation failed.$response"
return 1
fi

return 0
}
```

The utility functions used in this example.

```
#####
# function iecho
#
# This function enables the script to display the specified text only if
# the global variable $VERBOSE is set to true.
#####
function iecho() {
if [[ $VERBOSE == true ]]; then
echo "$@"

```

```
    fi
}

#####
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
#####
function errecho() {
    printf "%s\n" "$*" 1>&2
}

#####
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-help-return-codes.
#
# The function expects the following argument:
#       $1 - The error code returned by the AWS CLI.
#
# Returns:
#       0: - Success.
#
#####

function aws_cli_error_log() {
    local err_code=$1
    errecho "Error code : $err_code"
    if [ "$err_code" == 1 ]; then
        errecho " One or more S3 transfers failed."
    elif [ "$err_code" == 2 ]; then
        errecho " Command line failed to parse."
    elif [ "$err_code" == 130 ]; then
        errecho " Process received SIGINT."
    elif [ "$err_code" == 252 ]; then
        errecho " Command syntax invalid."
    elif [ "$err_code" == 253 ]; then
        errecho " The system environment or configuration was invalid."
    elif [ "$err_code" == 254 ]; then
        errecho " The service returned an error."
    elif [ "$err_code" == 255 ]; then
        errecho " 255 is a catch-all error."
    fi
}
```

```
    fi

    return 0
}
```

- For API details, see [DeleteTable](#) in *AWS CLI Command Reference*.

Delete an item from a table

The following code example shows how to delete an item from a DynamoDB table.

AWS CLI with Bash script

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
#####
# function dynamodb_delete_item
#
# This function deletes an item from a DynamoDB table.
#
# Parameters:
#     -n table_name -- The name of the table.
#     -k keys -- Path to json file containing the keys that identify the item to
#               delete.
#
# Returns:
#     0 - If successful.
#     1 - If it fails.
#####
function dynamodb_delete_item() {
    local table_name keys response
    local option OPTARG # Required to use getopt command in a function.

    # #####
    # Function usage explanation
    #####
    function usage() {
```

```
echo "function dynamodb_delete_item"
echo "Delete an item from a DynamoDB table."
echo " -n table_name -- The name of the table."
echo " -k keys -- Path to json file containing the keys that identify the item
to delete."
echo ""
}
while getopts "n:k:h" option; do
  case "${option}" in
    n) table_name="${OPTARG}" ;;
    k) keys="${OPTARG}" ;;
    h)
      usage
      return 0
      ;;
    \?) 
      echo "Invalid parameter"
      usage
      return 1
      ;;
  esac
done
export OPTIND=1

if [[ -z "$table_name" ]]; then
  errecho "ERROR: You must provide a table name with the -n parameter."
  usage
  return 1
fi

if [[ -z "$keys" ]]; then
  errecho "ERROR: You must provide a keys json file path the -k parameter."
  usage
  return 1
fi

iecho "Parameters:\n"
iecho "  table_name: $table_name"
iecho "  keys: $keys"
iecho ""

response=$(aws dynamodb delete-item \
--table-name "$table_name" \
--key file://"$keys")
```

```
local error_code=${?}

if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports delete-item operation failed.$response"
    return 1
fi

return 0

}
```

The utility functions used in this example.

```
#####
# function iecho
#
# This function enables the script to display the specified text only if
# the global variable $VERBOSE is set to true.
#####
function iecho() {
    if [[ $VERBOSE == true ]]; then
        echo "$@"
    fi
}

#####
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
#####
function errecho() {
    printf "%s\n" "$*" 1>&2
}

#####
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
```

```
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-help-return-codes.  
#  
# The function expects the following argument:  
#           $1 - The error code returned by the AWS CLI.  
#  
# Returns:  
#           0: - Success.  
#  
#####  
function aws_cli_error_log() {  
    local err_code=$1  
    errecho "Error code : $err_code"  
    if [ "$err_code" == 1 ]; then  
        errecho " One or more S3 transfers failed."  
    elif [ "$err_code" == 2 ]; then  
        errecho " Command line failed to parse."  
    elif [ "$err_code" == 130 ]; then  
        errecho " Process received SIGINT."  
    elif [ "$err_code" == 252 ]; then  
        errecho " Command syntax invalid."  
    elif [ "$err_code" == 253 ]; then  
        errecho " The system environment or configuration was invalid."  
    elif [ "$err_code" == 254 ]; then  
        errecho " The service returned an error."  
    elif [ "$err_code" == 255 ]; then  
        errecho " 255 is a catch-all error."  
    fi  
  
    return 0  
}  
#####
```

- For API details, see [DeleteItem](#) in *AWS CLI Command Reference*.

Get a batch of items

The following code example shows how to get a batch of DynamoDB items.

AWS CLI with Bash script

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
#####
# function dynamodb_batch_get_item
#
# This function gets a batch of items from a DynamoDB table.
#
# Parameters:
#     -i item -- Path to json file containing the keys of the items to get.
#
# Returns:
#     The items as json output.
# And:
#     0 - If successful.
#     1 - If it fails.
#####
function dynamodb_batch_get_item() {
    local item response
    local option OPTARG # Required to use getopt command in a function.

    #####
    # Function usage explanation
    #####
    function usage() {
        echo "function dynamodb_batch_get_item"
        echo "Get a batch of items from a DynamoDB table."
        echo " -i item -- Path to json file containing the keys of the items to get."
        echo ""
    }

    while getopt "i:h" option; do
        case "${option}" in
            i) item="${OPTARG}" ;;
            h)
                usage
                return 0
            ;;
        esac
    done
}
```

```
;;
\?)  
    echo "Invalid parameter"  
    usage  
    return 1  
;;  
esac  
done  
export OPTIND=1  
  
if [[ -z "$item" ]]; then  
    errecho "ERROR: You must provide an item with the -i parameter."  
    usage  
    return 1  
fi  
  
response=$(aws dynamodb batch-get-item \  
    --request-items file://"$item")  
local error_code=${?}  
  
if [[ $error_code -ne 0 ]]; then  
    aws_cli_error_log $error_code  
    errecho "ERROR: AWS reports batch-get-item operation failed.$response"  
    return 1  
fi  
  
echo "$response"  
  
return 0  
}
```

The utility functions used in this example.

```
#####  
# function errecho  
#  
# This function outputs everything sent to it to STDERR (standard error output).  
#####  
function errecho() {  
    printf "%s\n" "$*" 1>&2  
}
```

```
#####
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-help-  
return-codes.
#
# The function expects the following argument:
#           $1 - The error code returned by the AWS CLI.
#
# Returns:
#           0: - Success.
#
#####
function aws_cli_error_log() {
    local err_code=$1
    errecho "Error code : $err_code"
    if [ "$err_code" == 1 ]; then
        errecho " One or more S3 transfers failed."
    elif [ "$err_code" == 2 ]; then
        errecho " Command line failed to parse."
    elif [ "$err_code" == 130 ]; then
        errecho " Process received SIGINT."
    elif [ "$err_code" == 252 ]; then
        errecho " Command syntax invalid."
    elif [ "$err_code" == 253 ]; then
        errecho " The system environment or configuration was invalid."
    elif [ "$err_code" == 254 ]; then
        errecho " The service returned an error."
    elif [ "$err_code" == 255 ]; then
        errecho " 255 is a catch-all error."
    fi

    return 0
}
```

- For API details, see [BatchGetItem](#) in *AWS CLI Command Reference*.

Get an item from a table

The following code example shows how to get an item from a DynamoDB table.

AWS CLI with Bash script

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
#####
# function dynamodb_get_item
#
# This function gets an item from a DynamoDB table.
#
# Parameters:
#     -n table_name -- The name of the table.
#     -k keys -- Path to json file containing the keys that identify the item to
#               get.
#     [-q query] -- Optional JMESPath query expression.
#
# Returns:
#     The item as text output.
# And:
#     0 - If successful.
#     1 - If it fails.
#####
function dynamodb_get_item() {
    local table_name keys query response
    local option OPTARG # Required to use getopt command in a function.

    # #####
    # Function usage explanation
    #####
    function usage() {
        echo "function dynamodb_get_item"
        echo "Get an item from a DynamoDB table."
        echo " -n table_name -- The name of the table."
        echo " -k keys -- Path to json file containing the keys that identify the item
to get."
        echo " [-q query] -- Optional JMESPath query expression."
        echo ""
    }
    query=""
}
```

```
while getopts "n:k:q:h" option; do
  case "${option}" in
    n) table_name="${OPTARG}" ;;
    k) keys="${OPTARG}" ;;
    q) query="${OPTARG}" ;;
    h)
      usage
      return 0
      ;;
    \?)  
      echo "Invalid parameter"
      usage
      return 1
      ;;
  esac
done
export OPTIND=1

if [[ -z "$table_name" ]]; then
  errecho "ERROR: You must provide a table name with the -n parameter."
  usage
  return 1
fi

if [[ -z "$keys" ]]; then
  errecho "ERROR: You must provide a keys json file path the -k parameter."
  usage
  return 1
fi

if [[ -n "$query" ]]; then
  response=$(aws dynamodb get-item \
    --table-name "$table_name" \
    --key file://"$keys" \
    --output text \
    --query "$query")
else
  response=$((
    aws dynamodb get-item \
    --table-name "$table_name" \
    --key file://"$keys" \
    --output text
  ))
fi
```

```
local error_code=${?}

if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports get-item operation failed.$response"
    return 1
fi

if [[ -n "$query" ]]; then
    echo "$response" | sed "/^\\t/s/\\t//1" # Remove initial tab that the JMSEPath
query inserts on some strings.
else
    echo "$response"
fi

return 0
}
```

The utility functions used in this example.

```
#####
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
#####
function errecho() {
    printf "%s\n" "$*" 1>&2
}

#####
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-help-return-codes.
#
# The function expects the following argument:
#     $1 - The error code returned by the AWS CLI.
#
# Returns:
```

```
#          0: - Success.  
#  
#####  
function aws_cli_error_log() {  
    local err_code=$1  
    errecho "Error code : $err_code"  
    if [ "$err_code" == 1 ]; then  
        errecho " One or more S3 transfers failed."  
    elif [ "$err_code" == 2 ]; then  
        errecho " Command line failed to parse."  
    elif [ "$err_code" == 130 ]; then  
        errecho " Process received SIGINT."  
    elif [ "$err_code" == 252 ]; then  
        errecho " Command syntax invalid."  
    elif [ "$err_code" == 253 ]; then  
        errecho " The system environment or configuration was invalid."  
    elif [ "$err_code" == 254 ]; then  
        errecho " The service returned an error."  
    elif [ "$err_code" == 255 ]; then  
        errecho " 255 is a catch-all error."  
    fi  
  
    return 0  
}  
}
```

- For API details, see [GetItem](#) in *AWS CLI Command Reference*.

Get information about a table

The following code example shows how to get information about a DynamoDB table.

AWS CLI with Bash script

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
#####  
# function dynamodb_describe_table
```

```
#  
# This function returns the status of a DynamoDB table.  
#  
# Parameters:  
#     -n table_name -- The name of the table.  
#  
# Response:  
#     - TableStatus:  
#         And:  
#             0 - Table is active.  
#             1 - If it fails.  
#####  
function dynamodb_describe_table {  
    local table_name  
    local optionOPTARG # Required to use getopt command in a function.  
  
#####  
# Function usage explanation  
#####  
function usage() {  
    echo "function dynamodb_describe_table"  
    echo "Describe the status of a DynamoDB table."  
    echo " -n table_name -- The name of the table."  
    echo ""  
}  
  
# Retrieve the calling parameters.  
while getopt "n:h" option; do  
    case "${option}" in  
        n) table_name="${OPTARG}" ;;  
        h)  
            usage  
            return 0  
            ;;  
        \?)  
            echo "Invalid parameter"  
            usage  
            return 1  
            ;;  
    esac  
done  
export OPTIND=1  
  
if [[ -z "$table_name" ]]; then
```

```
errecho "ERROR: You must provide a table name with the -n parameter."
usage
return 1
fi

local table_status
table_status=$(
    aws dynamodb describe-table \
    --table-name "$table_name" \
    --output text \
    --query 'Table.TableStatus'
)

local error_code=${?}

if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log "$error_code"
    errecho "ERROR: AWS reports describe-table operation failed.$table_status"
    return 1
fi

echo "$table_status"

return 0
}
```

The utility functions used in this example.

```
#####
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
#####
function errecho() {
    printf "%s\n" "$*" 1>&2
}

#####
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
```

```
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-help-return-codes.  
#  
# The function expects the following argument:  
#           $1 - The error code returned by the AWS CLI.  
#  
# Returns:  
#           0: - Success.  
#  
#####  
function aws_cli_error_log() {  
    local err_code=$1  
    errecho "Error code : $err_code"  
    if [ "$err_code" == 1 ]; then  
        errecho " One or more S3 transfers failed."  
    elif [ "$err_code" == 2 ]; then  
        errecho " Command line failed to parse."  
    elif [ "$err_code" == 130 ]; then  
        errecho " Process received SIGINT."  
    elif [ "$err_code" == 252 ]; then  
        errecho " Command syntax invalid."  
    elif [ "$err_code" == 253 ]; then  
        errecho " The system environment or configuration was invalid."  
    elif [ "$err_code" == 254 ]; then  
        errecho " The service returned an error."  
    elif [ "$err_code" == 255 ]; then  
        errecho " 255 is a catch-all error."  
    fi  
  
    return 0  
}  
#####
```

- For API details, see [DescribeTable](#) in *AWS CLI Command Reference*.

List tables

The following code example shows how to list DynamoDB tables.

AWS CLI with Bash script

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
#####
# function dynamodb_list_tables
#
# This function lists all the tables in a DynamoDB.
#
# Returns:
#     0 - If successful.
#     1 - If it fails.
#####
function dynamodb_list_tables() {
    response=$(aws dynamodb list-tables \
        --output text \
        --query "TableNames")

    local error_code=${?}

    if [[ $error_code -ne 0 ]]; then
        aws_cli_error_log $error_code
        errecho "ERROR: AWS reports batch-write-item operation failed.$response"
        return 1
    fi

    echo "$response" | tr -s "[[:space:]]" "\n"

    return 0
}
```

The utility functions used in this example.

```
#####
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
```

```
#####
function errecho() {
    printf "%s\n" "$*" 1>&2
}

#####
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-help-return-codes.
#
# The function expects the following argument:
#       $1 - The error code returned by the AWS CLI.
#
# Returns:
#       0: - Success.
#
#####

function aws_cli_error_log() {
    local err_code=$1
    errecho "Error code : $err_code"
    if [ "$err_code" == 1 ]; then
        errecho " One or more S3 transfers failed."
    elif [ "$err_code" == 2 ]; then
        errecho " Command line failed to parse."
    elif [ "$err_code" == 130 ]; then
        errecho " Process received SIGINT."
    elif [ "$err_code" == 252 ]; then
        errecho " Command syntax invalid."
    elif [ "$err_code" == 253 ]; then
        errecho " The system environment or configuration was invalid."
    elif [ "$err_code" == 254 ]; then
        errecho " The service returned an error."
    elif [ "$err_code" == 255 ]; then
        errecho " 255 is a catch-all error."
    fi

    return 0
}
```

- For API details, see [ListTables](#) in *AWS CLI Command Reference*.

Put an item in a table

The following code example shows how to put an item in a DynamoDB table.

AWS CLI with Bash script

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
#####
# function dynamodb_put_item
#
# This function puts an item into a DynamoDB table.
#
# Parameters:
#   -n table_name -- The name of the table.
#   -i item -- Path to json file containing the item values.
#
# Returns:
#   0 - If successful.
#   1 - If it fails.
#####
function dynamodb_put_item() {
    local table_name item response
    local option OPTARG # Required to use getopt command in a function.

    #####
    # Function usage explanation
    #####
    function usage() {
        echo "function dynamodb_put_item"
        echo "Put an item into a DynamoDB table."
        echo " -n table_name -- The name of the table."
        echo " -i item -- Path to json file containing the item values."
        echo ""
    }

    while getopt "n:i:h" option; do
        case "${option}" in
            n) table_name="${OPTARG}" ;;
```

```
i) item="\${OPTARG}" ;;
h)
    usage
    return 0
;;
\?)
    echo "Invalid parameter"
    usage
    return 1
;;
esac
done
export OPTIND=1

if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
    return 1
fi

if [[ -z "$item" ]]; then
    errecho "ERROR: You must provide an item with the -i parameter."
    usage
    return 1
fi

iecho "Parameters:\n"
iecho "    table_name: $table_name"
iecho "    item: $item"
iecho ""
iecho ""

response=$(aws dynamodb put-item \
    --table-name "$table_name" \
    --item file://"$item")

local error_code=\$?

if [[ \$error_code -ne 0 ]]; then
    aws_cli_error_log \$error_code
    errecho "ERROR: AWS reports put-item operation failed.\$response"
    return 1
fi
```

```
    return 0  
  
}
```

The utility functions used in this example.

```
#####  
# function iecho  
#  
# This function enables the script to display the specified text only if  
# the global variable $VERBOSE is set to true.  
#####  
function iecho() {  
    if [[ $VERBOSE == true ]]; then  
        echo "$@"  
    fi  
}  
  
#####  
# function errecho  
#  
# This function outputs everything sent to it to STDERR (standard error output).  
#####  
function errecho() {  
    printf "%s\n" "$*" 1>&2  
}  
  
#####  
# function aws_cli_error_log()  
#  
# This function is used to log the error messages from the AWS CLI.  
#  
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-help-return-codes.  
#  
# The function expects the following argument:  
#       $1 - The error code returned by the AWS CLI.  
#  
# Returns:  
#       0: - Success.  
#  
#####
```

```
function aws_cli_error_log() {  
    local err_code=$1  
    errecho "Error code : $err_code"  
    if [ "$err_code" == 1 ]; then  
        errecho " One or more S3 transfers failed."  
    elif [ "$err_code" == 2 ]; then  
        errecho " Command line failed to parse."  
    elif [ "$err_code" == 130 ]; then  
        errecho " Process received SIGINT."  
    elif [ "$err_code" == 252 ]; then  
        errecho " Command syntax invalid."  
    elif [ "$err_code" == 253 ]; then  
        errecho " The system environment or configuration was invalid."  
    elif [ "$err_code" == 254 ]; then  
        errecho " The service returned an error."  
    elif [ "$err_code" == 255 ]; then  
        errecho " 255 is a catch-all error."  
    fi  
  
    return 0  
}  
}
```

- For API details, see [PutItem](#) in *AWS CLI Command Reference*.

Query a table

The following code example shows how to query a DynamoDB table.

AWS CLI with Bash script

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
#####  
# function dynamodb_query  
#  
# This function queries a DynamoDB table.  
#
```

```
# Parameters:
#   -n table_name -- The name of the table.
#   -k key_condition_expression -- The key condition expression.
#   -a attribute_names -- Path to JSON file containing the attribute names.
#   -v attribute_values -- Path to JSON file containing the attribute values.
#   [-p projection_expression] -- Optional projection expression.
#
# Returns:
#   The items as json output.
# And:
#   0 - If successful.
#   1 - If it fails.
#####
function dynamodb_query() {
    local table_name key_condition_expression attribute_names attribute_values
    projection_expression response
    local optionOPTARG # Required to use getopt command in a function.

    # #####
    # Function usage explanation
    #####
    function usage() {
        echo "function dynamodb_query"
        echo "Query a DynamoDB table."
        echo " -n table_name -- The name of the table."
        echo " -k key_condition_expression -- The key condition expression."
        echo " -a attribute_names -- Path to JSON file containing the attribute names."
        echo " -v attribute_values -- Path to JSON file containing the attribute
values."
        echo " [-p projection_expression] -- Optional projection expression."
        echo ""
    }

    while getopts "n:k:a:v:p:h" option; do
        case "${option}" in
            n) table_name="${OPTARG}" ;;
            k) key_condition_expression="${OPTARG}" ;;
            a) attribute_names="${OPTARG}" ;;
            v) attribute_values="${OPTARG}" ;;
            p) projection_expression="${OPTARG}" ;;
            h)
                usage
                return 0
                ;;
        esac
    done
}
```

```
\?)  
    echo "Invalid parameter"  
    usage  
    return 1  
;;  
esac  
done  
export OPTIND=1  
  
if [[ -z "$table_name" ]]; then  
    errecho "ERROR: You must provide a table name with the -n parameter."  
    usage  
    return 1  
fi  
  
if [[ -z "$key_condition_expression" ]]; then  
    errecho "ERROR: You must provide a key condition expression with the -k parameter."  
    usage  
    return 1  
fi  
  
if [[ -z "$attribute_names" ]]; then  
    errecho "ERROR: You must provide a attribute names with the -a parameter."  
    usage  
    return 1  
fi  
  
if [[ -z "$attribute_values" ]]; then  
    errecho "ERROR: You must provide a attribute values with the -v parameter."  
    usage  
    return 1  
fi  
  
if [[ -z "$projection_expression" ]]; then  
    response=$(aws dynamodb query \  
        --table-name "$table_name" \  
        --key-condition-expression "$key_condition_expression" \  
        --expression-attribute-names file://"$attribute_names" \  
        --expression-attribute-values file://"$attribute_values")  
else  
    response=$(aws dynamodb query \  
        --table-name "$table_name" \  
        --key-condition-expression "$key_condition_expression" \  
        --expression-attribute-names file://"$attribute_names" \  
        --expression-attribute-values file://"$attribute_values")
```

```
--expression-attribute-names file://"$attribute_names" \
--expression-attribute-values file://"$attribute_values" \
--projection-expression "$projection_expression")
fi

local error_code=${?}

if [[ $error_code -ne 0 ]]; then
aws_cli_error_log $error_code
errecho "ERROR: AWS reports query operation failed.$response"
return 1
fi

echo "$response"

return 0
}
```

The utility functions used in this example.

```
#####
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
#####
function errecho() {
    printf "%s\n" "$*" 1>&2
}

#####
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-help-return-codes.
#
# The function expects the following argument:
#     $1 - The error code returned by the AWS CLI.
#
# Returns:
#     0: - Success.
```

```
######
#function aws_cli_error_log() {
    local err_code=$1
    errecho "Error code : $err_code"
    if [ "$err_code" == 1 ]; then
        errecho " One or more S3 transfers failed."
    elif [ "$err_code" == 2 ]; then
        errecho " Command line failed to parse."
    elif [ "$err_code" == 130 ]; then
        errecho " Process received SIGINT."
    elif [ "$err_code" == 252 ]; then
        errecho " Command syntax invalid."
    elif [ "$err_code" == 253 ]; then
        errecho " The system environment or configuration was invalid."
    elif [ "$err_code" == 254 ]; then
        errecho " The service returned an error."
    elif [ "$err_code" == 255 ]; then
        errecho " 255 is a catch-all error."
    fi

    return 0
}
```

- For API details, see [Query](#) in *AWS CLI Command Reference*.

Scan a table

The following code example shows how to scan a DynamoDB table.

AWS CLI with Bash script

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
#####
# function dynamodb_scan
#
```

```
# This function scans a DynamoDB table.  
#  
# Parameters:  
#     -n table_name -- The name of the table.  
#     -f filter_expression -- The filter expression.  
#     -a expression_attribute_names -- Path to JSON file containing the expression  
#                                     attribute names.  
#     -v expression_attribute_values -- Path to JSON file containing the  
#                                     expression attribute values.  
#     [-p projection_expression] -- Optional projection expression.  
#  
# Returns:  
#     The items as json output.  
# And:  
#     0 - If successful.  
#     1 - If it fails.  
#####  
function dynamodb_scan() {  
    local table_name filter_expression expression_attribute_names  
    expression_attribute_values projection_expression response  
    local option OPTARG # Required to use getopt command in a function.  
  
    # #####  
    # Function usage explanation  
    #####  
    function usage() {  
        echo "function dynamodb_scan"  
        echo "Scan a DynamoDB table."  
        echo " -n table_name -- The name of the table."  
        echo " -f filter_expression -- The filter expression."  
        echo " -a expression_attribute_names -- Path to JSON file containing the  
        #                                     expression attribute names."  
        echo " -v expression_attribute_values -- Path to JSON file containing the  
        #                                     expression attribute values."  
        echo " [-p projection_expression] -- Optional projection expression."  
        echo ""  
    }  
  
    while getopt "n:f:a:v:p:h" option; do  
        case "${option}" in  
            n) table_name="${OPTARG}" ;;  
            f) filter_expression="${OPTARG}" ;;  
            a) expression_attribute_names="${OPTARG}" ;;  
            v) expression_attribute_values="${OPTARG}" ;;  
        esac  
    done  
}
```

```
p) projection_expression="${OPTARG}" ;;
h)
    usage
    return 0
;;
\?)
    echo "Invalid parameter"
    usage
    return 1
;;
esac
done
export OPTIND=1

if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
    return 1
fi

if [[ -z "$filter_expression" ]]; then
    errecho "ERROR: You must provide a filter expression with the -f parameter."
    usage
    return 1
fi

if [[ -z "$expression_attribute_names" ]]; then
    errecho "ERROR: You must provide expression attribute names with the -a parameter."
    usage
    return 1
fi

if [[ -z "$expression_attribute_values" ]]; then
    errecho "ERROR: You must provide expression attribute values with the -v parameter."
    usage
    return 1
fi

if [[ -z "$projection_expression" ]]; then
    response=$(aws dynamodb scan \
        --table-name "$table_name" \
        --filter-expression "$filter_expression" \
```

```
--expression-attribute-names file://"${expression_attribute_names}" \
--expression-attribute-values file://"${expression_attribute_values}") \
else
    response=$(aws dynamodb scan \
        --table-name "$table_name" \
        --filter-expression "$filter_expression" \
        --expression-attribute-names file://"${expression_attribute_names}" \
        --expression-attribute-values file://"${expression_attribute_values}" \
        --projection-expression "$projection_expression")
fi

local error_code=${?}

if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports scan operation failed.$response"
    return 1
fi

echo "$response"

return 0
}
```

The utility functions used in this example.

```
#####
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
#####
function errecho() {
    printf "%s\n" "$*" 1>&2
}

#####
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-help-return-codes.
```

```
#  
# The function expects the following argument:  
#      $1 - The error code returned by the AWS CLI.  
#  
# Returns:  
#      0: - Success.  
#  
#####  
function aws_cli_error_log() {  
    local err_code=$1  
    errecho "Error code : $err_code"  
    if [ "$err_code" == 1 ]; then  
        errecho " One or more S3 transfers failed."  
    elif [ "$err_code" == 2 ]; then  
        errecho " Command line failed to parse."  
    elif [ "$err_code" == 130 ]; then  
        errecho " Process received SIGINT."  
    elif [ "$err_code" == 252 ]; then  
        errecho " Command syntax invalid."  
    elif [ "$err_code" == 253 ]; then  
        errecho " The system environment or configuration was invalid."  
    elif [ "$err_code" == 254 ]; then  
        errecho " The service returned an error."  
    elif [ "$err_code" == 255 ]; then  
        errecho " 255 is a catch-all error."  
    fi  
  
    return 0  
}
```

- For API details, see [Scan](#) in *AWS CLI Command Reference*.

Update an item in a table

The following code example shows how to update an item in a DynamoDB table.

AWS CLI with Bash script

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
#####
# function dynamodb_update_item
#
# This function updates an item in a DynamoDB table.
#
#
# Parameters:
#     -n table_name -- The name of the table.
#     -k keys -- Path to json file containing the keys that identify the item to
#               update.
#     -e update expression -- An expression that defines one or more attributes
#               to be updated.
#     -v values -- Path to json file containing the update values.
#
# Returns:
#     0 - If successful.
#     1 - If it fails.
#####
function dynamodb_update_item() {
    local table_name keys update_expression values response
    local option OPTARG # Required to use getopt command in a function.

    #####
    # Function usage explanation
    #####
    function usage() {
        echo "function dynamodb_update_item"
        echo "Update an item in a DynamoDB table."
        echo " -n table_name -- The name of the table."
        echo " -k keys -- Path to json file containing the keys that identify the item
to update."
        echo " -e update expression -- An expression that defines one or more
attributes to be updated."
        echo " -v values -- Path to json file containing the update values."
    }
}
```

```
echo ""
}

while getopts "n:k:e:v:h" option; do
  case "${option}" in
    n) table_name="${OPTARG}" ;;
    k) keys="${OPTARG}" ;;
    e) update_expression="${OPTARG}" ;;
    v) values="${OPTARG}" ;;
    h)
      usage
      return 0
      ;;
    \?) 
      echo "Invalid parameter"
      usage
      return 1
      ;;
  esac
done
export OPTIND=1

if [[ -z "$table_name" ]]; then
  errecho "ERROR: You must provide a table name with the -n parameter."
  usage
  return 1
fi

if [[ -z "$keys" ]]; then
  errecho "ERROR: You must provide a keys json file path the -k parameter."
  usage
  return 1
fi
if [[ -z "$update_expression" ]]; then
  errecho "ERROR: You must provide an update expression with the -e parameter."
  usage
  return 1
fi

if [[ -z "$values" ]]; then
  errecho "ERROR: You must provide a values json file path the -v parameter."
  usage
  return 1
fi
```

```
iecho "Parameters:\n"
iecho "    table_name:    $table_name"
iecho "    keys:    $keys"
iecho "    update_expression:    $update_expression"
iecho "    values:    $values"

response=$(aws dynamodb update-item \
--table-name "$table_name" \
--key file://"$keys" \
--update-expression "$update_expression" \
--expression-attribute-values file://"$values")

local error_code=${?}

if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports update-item operation failed.$response"
    return 1
fi

return 0

}
```

The utility functions used in this example.

```
#####
# function iecho
#
# This function enables the script to display the specified text only if
# the global variable $VERBOSE is set to true.
#####
function iecho() {
    if [[ $VERBOSE == true ]]; then
        echo "$@"
    fi
}

#####
# function errecho
#
```

```
# This function outputs everything sent to it to STDERR (standard error output).
#####
function errecho() {
    printf "%s\n" "$*" 1>&2
}

#####
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-help-return-codes.
#
# The function expects the following argument:
#       $1 - The error code returned by the AWS CLI.
#
# Returns:
#       0: - Success.
#
#####

function aws_cli_error_log() {
    local err_code=$1
    errecho "Error code : $err_code"
    if [ "$err_code" == 1 ]; then
        errecho " One or more S3 transfers failed."
    elif [ "$err_code" == 2 ]; then
        errecho " Command line failed to parse."
    elif [ "$err_code" == 130 ]; then
        errecho " Process received SIGINT."
    elif [ "$err_code" == 252 ]; then
        errecho " Command syntax invalid."
    elif [ "$err_code" == 253 ]; then
        errecho " The system environment or configuration was invalid."
    elif [ "$err_code" == 254 ]; then
        errecho " The service returned an error."
    elif [ "$err_code" == 255 ]; then
        errecho " 255 is a catch-all error."
    fi

    return 0
}
```

- For API details, see [UpdateItem](#) in *AWS CLI Command Reference*.

Write a batch of items

The following code example shows how to write a batch of DynamoDB items.

AWS CLI with Bash script

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
#####
# function dynamodb_batch_write_item
#
# This function writes a batch of items into a DynamoDB table.
#
# Parameters:
#     -i item -- Path to json file containing the items to write.
#
# Returns:
#     0 - If successful.
#     1 - If it fails.
#####
function dynamodb_batch_write_item() {
    local item response
    local option OPTARG # Required to use getopts command in a function.

    #####
    # Function usage explanation
    #####
    function usage() {
        echo "function dynamodb_batch_write_item"
        echo "Write a batch of items into a DynamoDB table."
        echo " -i item -- Path to json file containing the items to write."
        echo ""
    }
    while getopts "i:h" option; do
        case "${option}" in
            i) item="${OPTARG}" ;;
            h) usage
                exit 0
            ;;
        esac
    done
}
```

```
h)
  usage
  return 0
  ;;
\?)
  echo "Invalid parameter"
  usage
  return 1
  ;;
esac
done
export OPTIND=1

if [[ -z "$item" ]]; then
  errecho "ERROR: You must provide an item with the -i parameter."
  usage
  return 1
fi

iecho "Parameters:\n"
iecho "  table_name: $table_name"
iecho "  item: $item"
iecho ""

response=$(aws dynamodb batch-write-item \
--request-items file://"$item")

local error_code=${?}

if [[ $error_code -ne 0 ]]; then
  aws_cli_error_log $error_code
  errecho "ERROR: AWS reports batch-write-item operation failed.$response"
  return 1
fi

return 0
}
```

The utility functions used in this example.

```
#####
# function iecho
```

```
#  
# This function enables the script to display the specified text only if  
# the global variable $VERBOSE is set to true.  
#####
function iecho() {  
    if [[ $VERBOSE == true ]]; then  
        echo "$@"  
    fi  
}  
  
#####  
# function errecho  
#  
# This function outputs everything sent to it to STDERR (standard error output).  
#####  
function errecho() {  
    printf "%s\n" "$*" 1>&2  
}  
  
#####  
# function aws_cli_error_log()  
#  
# This function is used to log the error messages from the AWS CLI.  
#  
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-help-return-codes.  
#  
# The function expects the following argument:  
#           $1 - The error code returned by the AWS CLI.  
#  
# Returns:  
#           0: - Success.  
#  
#####  
function aws_cli_error_log() {  
    local err_code=$1  
    errecho "Error code : $err_code"  
    if [ "$err_code" == 1 ]; then  
        errecho " One or more S3 transfers failed."  
    elif [ "$err_code" == 2 ]; then  
        errecho " Command line failed to parse."  
    elif [ "$err_code" == 130 ]; then  
        errecho " Process received SIGINT."  
    elif [ "$err_code" == 252 ]; then  
        errecho " Process received SIGTERM."  
    else  
        errecho " Unknown error code: $err_code."  
    fi  
}
```

```
errecho "  Command syntax invalid."
elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
fi

return 0
}
```

- For API details, see [BatchWriteItem](#) in *AWS CLI Command Reference*.

Scenarios

Get started with tables, items, and queries

The following code example shows how to:

- Create a table that can hold movie data.
- Put, get, and update a single movie in the table.
- Write movie data to the table from a sample JSON file.
- Query for movies that were released in a given year.
- Scan for movies that were released in a range of years.
- Delete a movie from the table, then delete the table.

AWS CLI with Bash script

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

The DynamoDB getting started scenario.

```
#####
```

```
# function dynamodb_getting_started_movies
#
# Scenario to create an Amazon DynamoDB table and perform a series of operations on
# the table.
#
# Returns:
#     0 - If successful.
#     1 - If an error occurred.
#####
function dynamodb_getting_started_movies() {

    source ./dynamodb_operations.sh

    key_schema_json_file="dynamodb_key_schema.json"
    attribute_definitions_json_file="dynamodb_attr_def.json"
    item_json_file="movie_item.json"
    key_json_file="movie_key.json"
    batch_json_file="batch.json"
    attribute_names_json_file="attribute_names.json"
    attributes_values_json_file="attribute_values.json"

    echo_repeat "*" 88
    echo
    echo "Welcome to the Amazon DynamoDB getting started demo."
    echo
    echo_repeat "*" 88
    echo

    local table_name
    echo -n "Enter a name for a new DynamoDB table: "
    get_input
    table_name=$get_input_result

    local provisioned_throughput="ReadCapacityUnits=5,WriteCapacityUnits=5"

    echo '['
    {"AttributeName": "year", "KeyType": "HASH"},  

    {"AttributeName": "title", "KeyType": "RANGE"}  

    ]' >"$key_schema_json_file"

    echo '['
    {"AttributeName": "year", "AttributeType": "N"},  

    {"AttributeName": "title", "AttributeType": "S"}  

    ]' >"$attribute_definitions_json_file"
```

```
if dynamodb_create_table -n "$table_name" -a "$attribute_definitions_json_file" \
-k "$key_schema_json_file" -p "$provisioned_throughput" 1>/dev/null; then
    echo "Created a DynamoDB table named $table_name"
else
    errecho "The table failed to create. This demo will exit."
    clean_up
    return 1
fi

echo "Waiting for the table to become active...."

if dynamodb_wait_table_active -n "$table_name"; then
    echo "The table is now active."
else
    errecho "The table failed to become active. This demo will exit."
    cleanup "$table_name"
    return 1
fi

echo
echo_repeat "*" 88
echo

echo -n "Enter the title of a movie you want to add to the table: "
get_input
local added_title
added_title=$get_input_result

local added_year
get_int_input "What year was it released? "
added_year=$get_input_result

local rating
get_float_input "On a scale of 1 - 10, how do you rate it? " "1" "10"
rating=$get_input_result

local plot
echo -n "Summarize the plot for me: "
get_input
plot=$get_input_result

echo '{
    "year": {"N" :"""$added_year""},
```

```
"title": {"S" : """$added_title"""},  
"info": {"M" : {"plot": {"S" : """$plot"""}, "rating": {"N" : """$rating"""} } }  
}' >"$item_json_file"  
  
if dynamodb_put_item -n "$table_name" -i "$item_json_file"; then  
    echo "The movie '$added_title' was successfully added to the table  
'$table_name'."  
else  
    errecho "Put item failed. This demo will exit."  
    clean_up "$table_name"  
    return 1  
fi  
  
echo  
echo_repeat "*" 88  
echo  
  
echo "Let's update your movie '$added_title'."  
get_float_input "You rated it $rating, what new rating would you give it? " "1"  
"10"  
rating=$get_input_result  
  
echo -n "You summarized the plot as '$plot'."  
echo "What would you say now? "  
get_input  
plot=$get_input_result  
  
echo '{  
    "year": {"N" : """$added_year"""},  
    "title": {"S" : """$added_title"""}  
' >"$key_json_file"  
  
echo '{  
    ":r": {"N" : """$rating"""},  
    ":p": {"S" : """$plot"""}  
' >"$item_json_file"  
  
local update_expression="SET info.rating = :r, info.plot = :p"  
  
if dynamodb_update_item -n "$table_name" -k "$key_json_file" -e  
"$update_expression" -v "$item_json_file"; then  
    echo "Updated '$added_title' with new attributes."  
else  
    errecho "Update item failed. This demo will exit."
```

```
clean_up "$table_name"
return 1
fi

echo
echo_repeat "*" 88
echo

echo "We will now use batch write to upload 150 movie entries into the table."

local batch_json
for batch_json in movie_files/movies_*.json; do
    echo "{$table_name : $(<"$batch_json") }" >"$batch_json_file"
    if dynamodb_batch_write_item -i "$batch_json_file" 1>/dev/null; then
        echo "Entries in $batch_json added to table."
    else
        errecho "Batch write failed. This demo will exit."
        clean_up "$table_name"
        return 1
    fi
done

local title="The Lord of the Rings: The Fellowship of the Ring"
local year="2001"

if get_yes_no_input "Let's move on...do you want to get info about '$title'? (y/n)"
"; then
    echo '{
"year": {"N" :"'$year'"'},
"title": {"S" : "'"$title"'"}
}' >"$key_json_file"
    local info
    info=$(dynamodb_get_item -n "$table_name" -k "$key_json_file")

    # shellcheck disable=SC2181
    if [[ ${?} -ne 0 ]]; then
        errecho "Get item failed. This demo will exit."
        clean_up "$table_name"
        return 1
    fi

    echo "Here is what I found:"
    echo "$info"
fi
```

```
local ask_for_year=true
while [[ "$ask_for_year" == true ]]; do
    echo "Let's get a list of movies released in a given year."
    get_int_input "Enter a year between 1972 and 2018: " "1972" "2018"
    year=$get_input_result
    echo '{
        "#n": "year"
    }' >"$attribute_names_json_file"

    echo '{
        ":v": {"N" :"""$year"""}
    }' >"$attributes_values_json_file"

    response=$(dynamodb_query -n "$table_name" -k "#n=:v" -a
"$attribute_names_json_file" -v "$attributes_values_json_file")

    # shellcheck disable=SC2181
    if [[ ${?} -ne 0 ]]; then
        errecho "Query table failed. This demo will exit."
        clean_up "$table_name"
        return 1
    fi

    echo "Here is what I found:"
    echo "$response"

    if ! get_yes_no_input "Try another year? (y/n) "; then
        ask_for_year=false
    fi
done

echo "Now let's scan for movies released in a range of years. Enter a year: "
get_int_input "Enter a year between 1972 and 2018: " "1972" "2018"
local start=$get_input_result

get_int_input "Enter another year: " "1972" "2018"
local end=$get_input_result

echo '{
    "#n": "year"
}' >"$attribute_names_json_file"

echo '{
```

```
":v1": {"N" : "'"$start"'"},  
":v2": {"N" : "'"$end"'"}  
}' >"$attributes_values_json_file"  
  
response=$(dynamodb_scan -n "$table_name" -f "#n BETWEEN :v1 AND :v2" -a  
"$attribute_names_json_file" -v "$attributes_values_json_file")  
  
# shellcheck disable=SC2181  
if [[ ${?} -ne 0 ]]; then  
    errecho "Scan table failed. This demo will exit."  
    clean_up "$table_name"  
    return 1  
fi  
  
echo "Here is what I found:"  
echo "$response"  
  
echo  
echo_repeat "*" 88  
echo  
  
echo "Let's remove your movie '$added_title' from the table."  
  
if get_yes_no_input "Do you want to remove '$added_title'? (y/n) "; then  
    echo '{  
"year": {"N" : "'"$added_year"'"},  
"title": {"S" : "'"$added_title"'"}  
' >"$key_json_file"  
  
    if ! dynamodb_delete_item -n "$table_name" -k "$key_json_file"; then  
        errecho "Delete item failed. This demo will exit."  
        clean_up "$table_name"  
        return 1  
    fi  
fi  
  
if get_yes_no_input "Do you want to delete the table '$table_name'? (y/n) "; then  
    if ! clean_up "$table_name"; then  
        return 1  
    fi  
else  
    if ! clean_up; then  
        return 1  
    fi
```

```
    fi

    return 0
}
```

The DynamoDB functions used in this scenario.

```
#####
# function dynamodb_create_table
#
# This function creates an Amazon DynamoDB table.
#
# Parameters:
#   -n table_name -- The name of the table to create.
#   -a attribute_definitions -- JSON file path of a list of attributes and their
#     types.
#   -k key_schema -- JSON file path of a list of attributes and their key types.
#   -p provisioned_throughput -- Provisioned throughput settings for the table.
#
# Returns:
#   0 - If successful.
#   1 - If it fails.
#####
function dynamodb_create_table() {
    local table_name attribute_definitions key_schema provisioned_throughput response
    local option OPTARG # Required to use getopt command in a function.

    #####
    # Function usage explanation
    #####
    function usage() {
        echo "function dynamodb_create_table"
        echo "Creates an Amazon DynamoDB table."
        echo " -n table_name -- The name of the table to create."
        echo " -a attribute_definitions -- JSON file path of a list of attributes and
        their types."
        echo " -k key_schema -- JSON file path of a list of attributes and their key
        types."
        echo " -p provisioned_throughput -- Provisioned throughput settings for the
        table."
        echo ""
    }
}
```

```
# Retrieve the calling parameters.
while getopts "n:a:k:p:h" option; do
    case "${option}" in
        n) table_name="${OPTARG}" ;;
        a) attribute_definitions="${OPTARG}" ;;
        k) key_schema="${OPTARG}" ;;
        p) provisioned_throughput="${OPTARG}" ;;
        h)
            usage
            return 0
            ;;
        \?)
            echo "Invalid parameter"
            usage
            return 1
            ;;
    esac
done
export OPTIND=1

if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
    return 1
fi

if [[ -z "$attribute_definitions" ]]; then
    errecho "ERROR: You must provide an attribute definitions json file path the -a parameter."
    usage
    return 1
fi

if [[ -z "$key_schema" ]]; then
    errecho "ERROR: You must provide a key schema json file path the -k parameter."
    usage
    return 1
fi

if [[ -z "$provisioned_throughput" ]]; then
    errecho "ERROR: You must provide a provisioned throughput json file path the -p parameter."
    usage
```

```
    return 1
fi

iecho "Parameters:\n"
iecho "    table_name: $table_name"
iecho "    attribute_definitions: $attribute_definitions"
iecho "    key_schema: $key_schema"
iecho "    provisioned_throughput: $provisioned_throughput"
iecho ""

response=$(aws dynamodb create-table \
--table-name "$table_name" \
--attribute-definitions file://"$attribute_definitions" \
--key-schema file://"$key_schema" \
--provisioned-throughput "$provisioned_throughput")

local error_code=${?}

if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports create-table operation failed.$response"
    return 1
fi

return 0
}

#####
# function dynamodb_describe_table
#
# This function returns the status of a DynamoDB table.
#
# Parameters:
#     -n table_name -- The name of the table.
#
# Response:
#     - TableStatus:
#         And:
#             0 - Table is active.
#             1 - If it fails.
#####
function dynamodb_describe_table {
    local table_name
    local option OPTARG # Required to use getopt command in a function.
```

```
#####
# Function usage explanation
#####
function usage() {
    echo "function dynamodb_describe_table"
    echo "Describe the status of a DynamoDB table."
    echo " -n table_name -- The name of the table."
    echo ""
}

# Retrieve the calling parameters.
while getopts "n:h" option; do
    case "${option}" in
        n) table_name="${OPTARG}" ;;
        h)
            usage
            return 0
            ;;
        \?)
            echo "Invalid parameter"
            usage
            return 1
            ;;
    esac
done
export OPTIND=1

if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
    return 1
fi

local table_status
table_status=$((
    aws dynamodb describe-table \
        --table-name "$table_name" \
        --output text \
        --query 'Table.TableStatus'
))

local error_code=${?}
```

```
if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log "$error_code"
    errecho "ERROR: AWS reports describe-table operation failed.$table_status"
    return 1
fi

echo "$table_status"

return 0
}

#####
# function dynamodb_put_item
#
# This function puts an item into a DynamoDB table.
#
# Parameters:
#     -n table_name -- The name of the table.
#     -i item -- Path to json file containing the item values.
#
# Returns:
#     0 - If successful.
#     1 - If it fails.
#####
function dynamodb_put_item() {
    local table_name item response
    local option OPTARG # Required to use getopt command in a function.

    #####
    # Function usage explanation
    #####
    function usage() {
        echo "function dynamodb_put_item"
        echo "Put an item into a DynamoDB table."
        echo " -n table_name -- The name of the table."
        echo " -i item -- Path to json file containing the item values."
        echo ""
    }

    while getopt "n:i:h" option; do
        case "${option}" in
            n) table_name="${OPTARG}" ;;
            i) item="${OPTARG}" ;;
            h)

```

```
usage
return 0
;;
\?)
echo "Invalid parameter"
usage
return 1
;;
esac
done
export OPTIND=1

if [[ -z "$table_name" ]]; then
errecho "ERROR: You must provide a table name with the -n parameter."
usage
return 1
fi

if [[ -z "$item" ]]; then
errecho "ERROR: You must provide an item with the -i parameter."
usage
return 1
fi

iecho "Parameters:\n"
iecho "    table_name: $table_name"
iecho "    item: $item"
iecho ""
iecho ""

response=$(aws dynamodb put-item \
--table-name "$table_name" \
--item file://"$item")

local error_code=${?}

if [[ $error_code -ne 0 ]]; then
aws_cli_error_log $error_code
errecho "ERROR: AWS reports put-item operation failed.$response"
return 1
fi

return 0
```

```
}

#####
# function dynamodb_update_item
#
# This function updates an item in a DynamoDB table.
#
#
# Parameters:
#     -n table_name -- The name of the table.
#     -k keys -- Path to json file containing the keys that identify the item to
#     update.
#     -e update expression -- An expression that defines one or more attributes
#     to be updated.
#     -v values -- Path to json file containing the update values.
#
# Returns:
#     0 - If successful.
#     1 - If it fails.
#####
function dynamodb_update_item() {
    local table_name keys update_expression values response
    local option OPTARG # Required to use getopts command in a function.

#####

# Function usage explanation
#####
function usage() {
    echo "function dynamodb_update_item"
    echo "Update an item in a DynamoDB table."
    echo " -n table_name -- The name of the table."
    echo " -k keys -- Path to json file containing the keys that identify the item
to update."
    echo " -e update expression -- An expression that defines one or more
attributes to be updated."
    echo " -v values -- Path to json file containing the update values."
    echo ""
}

while getopts "n:k:e:v:h" option; do
    case "${option}" in
        n) table_name="${OPTARG}" ;;
        k) keys="${OPTARG}" ;;
        e) update_expression="${OPTARG}" ;;
```

```
v) values="\${OPTARG}" ;;
h)
    usage
    return 0
;;
\?)
    echo "Invalid parameter"
    usage
    return 1
;;
esac
done
export OPTIND=1

if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
    return 1
fi

if [[ -z "$keys" ]]; then
    errecho "ERROR: You must provide a keys json file path the -k parameter."
    usage
    return 1
fi
if [[ -z "$update_expression" ]]; then
    errecho "ERROR: You must provide an update expression with the -e parameter."
    usage
    return 1
fi

if [[ -z "$values" ]]; then
    errecho "ERROR: You must provide a values json file path the -v parameter."
    usage
    return 1
fi

iecho "Parameters:\n"
iecho "    table_name:    $table_name"
iecho "    keys:        $keys"
iecho "    update_expression:    $update_expression"
iecho "    values:        $values"

response=$(aws dynamodb update-item \
```

```
--table-name "$table_name" \
--key file://"${keys}" \
--update-expression "$update_expression" \
--expression-attribute-values file://"${values}")\n\nlocal error_code=${?}\n\nif [[ $error_code -ne 0 ]]; then\n    aws_cli_error_log $error_code\n    errecho "ERROR: AWS reports update-item operation failed.$response"\n    return 1\nfi\n\nreturn 0\n}\n\n#####\n# function dynamodb_batch_write_item\n#\n# This function writes a batch of items into a DynamoDB table.\n#\n# Parameters:\n#     -i item -- Path to json file containing the items to write.\n#\n# Returns:\n#     0 - If successful.\n#     1 - If it fails.\n#####\nfunction dynamodb_batch_write_item() {\n    local item response\n    local option OPTARG # Required to use getopt command in a function.\n\n    #####\n    # Function usage explanation\n    #####\n    function usage() {\n        echo "function dynamodb_batch_write_item\"\n        echo "Write a batch of items into a DynamoDB table.\"\n        echo " -i item -- Path to json file containing the items to write.\"\n        echo ""\n    }\n\n    while getopt "i:h" option; do\n        case "${option}" in\n            i:\n                item=\${OPTARG}\n            h:\n                usage\n                exit 1\n        esac\n    done\n\n    if [ -z ${item} ]; then\n        errecho "Error: item path is required.">> ${aws_cli_error_log}\n        return 1\n    fi\n\n    response=$(aws dynamodb batch-write-item --table-name ${table_name} --batch-size 100 --request-items \"\n        {\n            \"PutRequest\": {\n                \"Item\": {\n                    \"id\": {\n                        \"S\": \"${item}\"},\n                    \"name\": {\n                        \"S\": \"${name}\"},\n                    \"age\": {\n                        \"N\": \"${age}\"},\n                    \"is_famous\": {\n                        \"BOOL\": ${is_famous}},\n                    \"last_update\": {\n                        \"S\": \"${last_update}\"}\n                }\n            }\n        }\n    \"\n    )\n\n    if [ -z ${response} ]; then\n        errecho "Error: Failed to write item to DynamoDB.">> ${aws_cli_error_log}\n        return 1\n    fi\n\n    echo ${response}\n}\n\n#####\n# Function usage explanation\n#####\nfunction usage() {\n    echo "function dynamodb_batch_write_item\"\n    echo "Write a batch of items into a DynamoDB table.\"\n    echo " -i item -- Path to json file containing the items to write.\"\n    echo ""\n}\n\nwhile getopt "i:h" option; do\n    case "${option}" in\n        i:\n            item=\${OPTARG}\n        h:\n            usage\n            exit 1\n    esac\ndone\n\nif [ -z ${item} ]; then\n    errecho "Error: item path is required.">> ${aws_cli_error_log}\n    return 1\nfi\n\nresponse=$(aws dynamodb batch-write-item --table-name ${table_name} --batch-size 100 --request-items \"\n    {\n        \"PutRequest\": {\n            \"Item\": {\n                \"id\": {\n                    \"S\": \"${item}\"},\n                \"name\": {\n                    \"S\": \"${name}\"},\n                \"age\": {\n                    \"N\": \"${age}\"},\n                \"is_famous\": {\n                    \"BOOL\": ${is_famous}},\n                \"last_update\": {\n                    \"S\": \"${last_update}\"}\n            }\n        }\n    }\n\"\n)\n\nif [ -z ${response} ]; then\n    errecho "Error: Failed to write item to DynamoDB.">> ${aws_cli_error_log}\n    return 1\nfi\n\necho ${response}
```

```
i) item="${OPTARG}" ;;
h)
    usage
    return 0
;;
\?)
    echo "Invalid parameter"
    usage
    return 1
;;
esac
done
export OPTIND=1

if [[ -z "$item" ]]; then
    errecho "ERROR: You must provide an item with the -i parameter."
    usage
    return 1
fi

iecho "Parameters:\n"
iecho "    table_name: $table_name"
iecho "    item: $item"
iecho ""

response=$(aws dynamodb batch-write-item \
--request-items file:///"$item")

local error_code=${?}

if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports batch-write-item operation failed.$response"
    return 1
fi

return 0
}

#####
# function dynamodb_get_item
#
# This function gets an item from a DynamoDB table.
#
```

```
# Parameters:
#       -n table_name -- The name of the table.
#       -k keys   -- Path to json file containing the keys that identify the item to
#                   get.
#       [-q query] -- Optional JMESPath query expression.
#
# Returns:
#       The item as text output.
# And:
#       0 - If successful.
#       1 - If it fails.
#####
function dynamodb_get_item() {
    local table_name keys query response
    local option OPTARG # Required to use getopt command in a function.

    #####
    # Function usage explanation
    #####
    function usage() {
        echo "function dynamodb_get_item"
        echo "Get an item from a DynamoDB table."
        echo " -n table_name -- The name of the table."
        echo " -k keys   -- Path to json file containing the keys that identify the item
to get."
        echo " [-q query] -- Optional JMESPath query expression."
        echo ""
    }
    query=""
    while getopt "n:k:q:h" option; do
        case "${option}" in
            n) table_name="${OPTARG}" ;;
            k) keys="${OPTARG}" ;;
            q) query="${OPTARG}" ;;
            h)
                usage
                return 0
                ;;
            \?) 
                echo "Invalid parameter"
                usage
                return 1
                ;;
        esac
    done
}
```

```
done
export OPTIND=1

if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
    return 1
fi

if [[ -z "$keys" ]]; then
    errecho "ERROR: You must provide a keys json file path the -k parameter."
    usage
    return 1
fi

if [[ -n "$query" ]]; then
    response=$(aws dynamodb get-item \
        --table-name "$table_name" \
        --key file://"$keys" \
        --output text \
        --query "$query")
else
    response=$((
        aws dynamodb get-item \
            --table-name "$table_name" \
            --key file://"$keys" \
            --output text
    ))
fi

local error_code=${?}

if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports get-item operation failed.$response"
    return 1
fi

if [[ -n "$query" ]]; then
    echo "$response" | sed "/^\\t/s/\\t//1" # Remove initial tab that the JMSEPath
query inserts on some strings.
else
    echo "$response"
fi
```

```
    return 0
}

#####
# function dynamodb_query
#
# This function queries a DynamoDB table.

#
# Parameters:
#   -n table_name -- The name of the table.
#   -k key_condition_expression -- The key condition expression.
#   -a attribute_names -- Path to JSON file containing the attribute names.
#   -v attribute_values -- Path to JSON file containing the attribute values.
#   [-p projection_expression] -- Optional projection expression.

#
# Returns:
#   The items as json output.
# And:
#   0 - If successful.
#   1 - If it fails.

#####
function dynamodb_query() {
    local table_name key_condition_expression attribute_names attribute_values
    projection_expression response
    local option OPTARG # Required to use getopt command in a function.

    #
    # Function usage explanation
    #####
    function usage() {
        echo "function dynamodb_query"
        echo "Query a DynamoDB table."
        echo " -n table_name -- The name of the table."
        echo " -k key_condition_expression -- The key condition expression."
        echo " -a attribute_names -- Path to JSON file containing the attribute names."
        echo " -v attribute_values -- Path to JSON file containing the attribute
values."
        echo " [-p projection_expression] -- Optional projection expression."
        echo ""
    }

    while getopt "n:k:a:v:p:h" option; do
        case "${option}" in

```

```
n) table_name="${OPTARG}" ;;
k) key_condition_expression="${OPTARG}" ;;
a) attribute_names="${OPTARG}" ;;
v) attribute_values="${OPTARG}" ;;
p) projection_expression="${OPTARG}" ;;
h)
    usage
    return 0
    ;;
\?)
    echo "Invalid parameter"
    usage
    return 1
    ;;
esac
done
export OPTIND=1

if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
    return 1
fi

if [[ -z "$key_condition_expression" ]]; then
    errecho "ERROR: You must provide a key condition expression with the -k parameter."
    usage
    return 1
fi

if [[ -z "$attribute_names" ]]; then
    errecho "ERROR: You must provide a attribute names with the -a parameter."
    usage
    return 1
fi

if [[ -z "$attribute_values" ]]; then
    errecho "ERROR: You must provide a attribute values with the -v parameter."
    usage
    return 1
fi

if [[ -z "$projection_expression" ]]; then
```

```
response=$(aws dynamodb query \
--table-name "$table_name" \
--key-condition-expression "$key_condition_expression" \
--expression-attribute-names file://"$attribute_names" \
--expression-attribute-values file://"$attribute_values")  
else  
    response=$(aws dynamodb query \  
    --table-name "$table_name" \  
    --key-condition-expression "$key_condition_expression" \  
    --expression-attribute-names file://"$attribute_names" \  
    --expression-attribute-values file://"$attribute_values" \  
    --projection-expression "$projection_expression")  
fi  
  
local error_code=${?}  
  
if [[ $error_code -ne 0 ]]; then  
    aws_cli_error_log $error_code  
    errecho "ERROR: AWS reports query operation failed.$response"  
    return 1  
fi  
  
echo "$response"  
  
return 0  
}  
  
#####  
# function dynamodb_scan  
#  
# This function scans a DynamoDB table.  
#  
# Parameters:  
#     -n table_name -- The name of the table.  
#     -f filter_expression -- The filter expression.  
#     -a expression_attribute_names -- Path to JSON file containing the expression  
#                                   attribute names.  
#     -v expression_attribute_values -- Path to JSON file containing the  
#                                   expression attribute values.  
#     [-p projection_expression] -- Optional projection expression.  
#  
# Returns:  
#     The items as json output.  
# And:
```

```
#      0 - If successful.
#      1 - If it fails.
#####
function dynamodb_scan() {
    local table_name filter_expression expression_attribute_names
    expression_attribute_values projection_expression response
    local option OPTARG # Required to use getopt command in a function.

    # #####
    # Function usage explanation
    #####
    function usage() {
        echo "function dynamodb_scan"
        echo "Scan a DynamoDB table."
        echo " -n table_name -- The name of the table."
        echo " -f filter_expression -- The filter expression."
        echo " -a expression_attribute_names -- Path to JSON file containing the
        expression attribute names."
        echo " -v expression_attribute_values -- Path to JSON file containing the
        expression attribute values."
        echo "[ -p projection_expression] -- Optional projection expression."
        echo ""
    }

    while getopt "n:f:a:v:p:h" option; do
        case "${option}" in
            n) table_name="${OPTARG}" ;;
            f) filter_expression="${OPTARG}" ;;
            a) expression_attribute_names="${OPTARG}" ;;
            v) expression_attribute_values="${OPTARG}" ;;
            p) projection_expression="${OPTARG}" ;;
            h)
                usage
                return 0
                ;;
            \?)
                echo "Invalid parameter"
                usage
                return 1
                ;;
        esac
    done
    export OPTIND=1
```

```
if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
    return 1
fi

if [[ -z "$filter_expression" ]]; then
    errecho "ERROR: You must provide a filter expression with the -f parameter."
    usage
    return 1
fi

if [[ -z "$expression_attribute_names" ]]; then
    errecho "ERROR: You must provide expression attribute names with the -a
parameter."
    usage
    return 1
fi

if [[ -z "$expression_attribute_values" ]]; then
    errecho "ERROR: You must provide expression attribute values with the -v
parameter."
    usage
    return 1
fi

if [[ -z "$projection_expression" ]]; then
    response=$(aws dynamodb scan \
        --table-name "$table_name" \
        --filter-expression "$filter_expression" \
        --expression-attribute-names file://"$expression_attribute_names" \
        --expression-attribute-values file://"$expression_attribute_values")
else
    response=$(aws dynamodb scan \
        --table-name "$table_name" \
        --filter-expression "$filter_expression" \
        --expression-attribute-names file://"$expression_attribute_names" \
        --expression-attribute-values file://"$expression_attribute_values" \
        --projection-expression "$projection_expression")
fi

local error_code=${?}

if [[ $error_code -ne 0 ]]; then
```

```
aws_cli_error_log $error_code
errecho "ERROR: AWS reports scan operation failed.$response"
return 1
fi

echo "$response"

return 0
}

#####
# function dynamodb_delete_item
#
# This function deletes an item from a DynamoDB table.
#
# Parameters:
#     -n table_name -- The name of the table.
#     -k keys -- Path to json file containing the keys that identify the item to
#               delete.
#
# Returns:
#     0 - If successful.
#     1 - If it fails.
#####
function dynamodb_delete_item() {
    local table_name keys response
    local option OPTARG # Required to use getopt command in a function.

    #####
    # Function usage explanation
    #####
    function usage() {
        echo "function dynamodb_delete_item"
        echo "Delete an item from a DynamoDB table."
        echo " -n table_name -- The name of the table."
        echo " -k keys -- Path to json file containing the keys that identify the item
to delete."
        echo ""
    }
    while getopt "n:k:h" option; do
        case "${option}" in
            n) table_name="${OPTARG}" ;;
            k) keys="${OPTARG}" ;;
            h)

```

```
        usage
        return 0
        ;;
    \?)  
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
done
export OPTIND=1

if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
    return 1
fi

if [[ -z "$keys" ]]; then
    errecho "ERROR: You must provide a keys json file path the -k parameter."
    usage
    return 1
fi

iecho "Parameters:\n"
iecho "    table_name: $table_name"
iecho "    keys: $keys"
iecho ""

response=$(aws dynamodb delete-item \
    --table-name "$table_name" \
    --key file://"$keys")

local error_code=${?}

if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports delete-item operation failed.$response"
    return 1
fi

return 0
}
```

```
#####
# function dynamodb_delete_table
#
# This function deletes a DynamoDB table.
#
# Parameters:
#     -n table_name -- The name of the table to delete.
#
# Returns:
#     0 - If successful.
#     1 - If it fails.
#####
function dynamodb_delete_table() {
    local table_name response
    local option OPTARG # Required to use getopt command in a function.

    # bashsupport disable=BP5008
    function usage() {
        echo "function dynamodb_delete_table"
        echo "Deletes an Amazon DynamoDB table."
        echo " -n table_name -- The name of the table to delete."
        echo ""
    }

    # Retrieve the calling parameters.
    while getopts "n:h" option; do
        case "${option}" in
            n) table_name="${OPTARG}" ;;
            h)
                usage
                return 0
                ;;
            \?)
                echo "Invalid parameter"
                usage
                return 1
                ;;
        esac
    done
    export OPTIND=1

    if [[ -z "$table_name" ]]; then
        errecho "ERROR: You must provide a table name with the -n parameter."
    fi
}
```

```
usage
return 1
fi

iecho "Parameters:\n"
iecho "    table_name: $table_name"
iecho ""

response=$(aws dynamodb delete-table \
--table-name "$table_name")

local error_code=${?}

if [[ $error_code -ne 0 ]]; then
aws_cli_error_log $error_code
errecho "ERROR: AWS reports delete-table operation failed.$response"
return 1
fi

return 0
}
```

The utility functions used in this scenario.

```
#####
# function iecho
#
# This function enables the script to display the specified text only if
# the global variable $VERBOSE is set to true.
#####
function iecho() {
    if [[ $VERBOSE == true ]]; then
        echo "$@"
    fi
}

#####
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
#####
function errecho() {
```

```
printf "%s\n" "$*" 1>&2
}

#####
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-help-return-codes.
#
# The function expects the following argument:
#       $1 - The error code returned by the AWS CLI.
#
# Returns:
#       0: - Success.
#
#####

function aws_cli_error_log() {
    local err_code=$1
    errecho "Error code : $err_code"
    if [ "$err_code" == 1 ]; then
        errecho " One or more S3 transfers failed."
    elif [ "$err_code" == 2 ]; then
        errecho " Command line failed to parse."
    elif [ "$err_code" == 130 ]; then
        errecho " Process received SIGINT."
    elif [ "$err_code" == 252 ]; then
        errecho " Command syntax invalid."
    elif [ "$err_code" == 253 ]; then
        errecho " The system environment or configuration was invalid."
    elif [ "$err_code" == 254 ]; then
        errecho " The service returned an error."
    elif [ "$err_code" == 255 ]; then
        errecho " 255 is a catch-all error."
    fi

    return 0
}
```

- For API details, see the following topics in *AWS CLI Command Reference*.
 - [BatchWriteItem](#)

- [CreateTable](#)
- [DeleteItem](#)
- [DeleteTable](#)
- [DescribeTable](#)
- [GetItem](#)
- [PutItem](#)
- [Query](#)
- [Scan](#)
- [UpdateItem](#)

HealthImaging examples using AWS CLI with Bash script

The following code examples show you how to perform actions and implement common scenarios by using the AWS Command Line Interface with Bash script with HealthImaging.

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios and cross-service examples.

Scenarios are code examples that show you how to accomplish a specific task by calling multiple functions within the same service.

Each example includes a link to GitHub, where you can find instructions on how to set up and run the code in context.

Topics

- [Actions](#)

Actions

Create a data store

The following code example shows how to create a HealthImaging data store.

AWS CLI with Bash script

```
#####
# function errecho
```

```
#  
# This function outputs everything sent to it to STDERR (standard error output).  
#####  
function errecho() {  
    printf "%s\n" "$*" 1>&2  
}  
  
#####  
# function imaging_create_datastore  
#  
# This function creates an AWS HealthImaging data store for importing DICOM P10  
# files.  
#  
# Parameters:  
#     -n data_store_name - The name of the data store.  
#  
# Returns:  
#     The datastore ID.  
# And:  
#     0 - If successful.  
#     1 - If it fails.  
#####  
function imaging_create_datastore() {  
    local datastore_name response  
    local option OPTARG # Required to use getopt command in a function.  
  
    # bashsupport disable=BP5008  
    function usage() {  
        echo "function imaging_create_datastore"  
        echo "Creates an AWS HealthImaging data store for importing DICOM P10 files."  
        echo " -n data_store_name - The name of the data store."  
        echo ""  
    }  
  
    # Retrieve the calling parameters.  
    while getopt "n:h" option; do  
        case "${option}" in  
            n) datastore_name="${OPTARG}" ;;  
            h)  
                usage  
                return 0  
                ;;  
            \?)  
                echo "Invalid parameter"  
        esac  
    done
```

```
usage
return 1
;;
esac
done
export OPTIND=1

if [[ -z "$datastore_name" ]]; then
errecho "ERROR: You must provide a data store name with the -n parameter."
usage
return 1
fi

response=$(aws medical-imaging create-datastore \
--datastore-name "$datastore_name" \
--output text \
--query 'datastoreId')

local error_code=${?}

if [[ $error_code -ne 0 ]]; then
aws_cli_error_log $error_code
errecho "ERROR: AWS reports medical-imaging create-datastore operation failed.
$response"
return 1
fi

echo "$response"

return 0
}
```

- For API details, see [CreateDatastore](#) in *AWS CLI Command Reference*.

 **Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Delete a data store

The following code example shows how to delete a HealthImaging data store.

AWS CLI with Bash script

```
#####
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
#####
function errecho() {
    printf "%s\n" "$*" 1>&2
}

#####
# function imaging_delete_datastore
#
# This function deletes an AWS HealthImaging data store.
#
# Parameters:
#     -i datastore_id - The ID of the data store.
#
# Returns:
#     0 - If successful.
#     1 - If it fails.
#####
function imaging_delete_datastore() {
    local datastore_id response
    local option OPTARG # Required to use getopt command in a function.

    # bashsupport disable=BP5008
    function usage() {
        echo "function imaging_delete_datastore"
        echo "Deletes an AWS HealthImaging data store."
        echo " -i datastore_id - The ID of the data store."
        echo ""
    }

    # Retrieve the calling parameters.
    while getopt "i:h" option; do
        case "${option}" in
            i) datastore_id="${OPTARG}" ;;
            h)

```

```
usage
return 0
;;
\?)
echo "Invalid parameter"
usage
return 1
;;
esac
done
export OPTIND=1

if [[ -z "$datastore_id" ]]; then
errecho "ERROR: You must provide a data store ID with the -i parameter."
usage
return 1
fi

response=$(aws medical-imaging delete-datastore \
--datastore-id "$datastore_id")

local error_code=${?}

if [[ $error_code -ne 0 ]]; then
aws_cli_error_log $error_code
errecho "ERROR: AWS reports medical-imaging delete-datastore operation failed.
$response"
return 1
fi

return 0
}
```

- For API details, see [DeleteDatastore](#) in *AWS CLI Command Reference*.

 **Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Get data store properties

The following code example shows how to get HealthImaging data store properties.

AWS CLI with Bash script

```
#####
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
#####
function errecho() {
    printf "%s\n" "$*" 1>&2
}

#####
# function imaging_get_datastore
#
# Get a data store's properties.
#
# Parameters:
#     -i data_store_id - The ID of the data store.
#
# Returns:
#     [datastore_name, datastore_id, datastore_status, datastore_arn, created_at,
#     updated_at]
#     And:
#         0 - If successful.
#         1 - If it fails.
#####
function imaging_get_datastore() {
    local datastore_id option OPTARG # Required to use getopt command in a function.
    local error_code
    # bashsupport disable=BP5008
    function usage() {
        echo "function imaging_get_datastore"
        echo "Gets a data store's properties."
        echo " -i datastore_id - The ID of the data store."
        echo ""
    }
    #
    # Retrieve the calling parameters.
    while getopt "i:h" option; do
        case "${option}" in
```

```
i) datastore_id="${OPTARG}" ;;
h)
    usage
    return 0
;;
\?)
    echo "Invalid parameter"
    usage
    return 1
;;
esac
done
export OPTIND=1

if [[ -z "$datastore_id" ]]; then
    errecho "ERROR: You must provide a data store ID with the -i parameter."
    usage
    return 1
fi

local response

response=$(
    aws medical-imaging get-datastore \
        --datastore-id "$datastore_id" \
        --output text \
        --query "[ datastoreProperties.datastoreName,
datastoreProperties.datastoreId, datastoreProperties.datastoreStatus,
datastoreProperties.datastoreArn,  datastoreProperties.createdAt,
datastoreProperties.updatedAt]"
)
error_code=${?}

if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports list-datastores operation failed.$response"
    return 1
fi

echo "$response"

return 0
}
```

- For API details, see [GetDatastore](#) in *AWS CLI Command Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

List data stores

The following code example shows how to list HealthImaging data stores.

AWS CLI with Bash script

```
#####
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
#####
function errecho() {
    printf "%s\n" "$*" 1>&2
}

#####
# function imaging_list_datastores
#
# List the HealthImaging data stores in the account.
#
# Returns:
#     [[datastore_name, datastore_id, datastore_status]]
# And:
#     0 - If successful.
#     1 - If it fails.
#####
function imaging_list_datastores() {
    local option OPTARG # Required to use getopt command in a function.
    local error_code
    # bashsupport disable=BP5008
    function usage() {
        echo "function imaging_list_datastores"
```

```
echo "Lists the AWS HealthImaging data stores in the account."
echo ""
}

# Retrieve the calling parameters.
while getopts "h" option; do
    case "${option}" in
        h)
            usage
            return 0
            ;;
        \?)
            echo "Invalid parameter"
            usage
            return 1
            ;;
    esac
done
export OPTIND=1

local response
response=$(aws medical-imaging list-datastores \
--output text \
--query "datastoreSummaries[*][datastoreName, datastoreId, datastoreStatus]")
error_code=${?}

if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports list-datastores operation failed.$response"
    return 1
fi

echo "$response"

return 0
}
```

- For API details, see [ListDatastores](#) in *AWS CLI Command Reference*.

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

IAM examples using AWS CLI with Bash script

The following code examples show you how to perform actions and implement common scenarios by using the AWS Command Line Interface with Bash script with IAM.

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios and cross-service examples.

Scenarios are code examples that show you how to accomplish a specific task by calling multiple functions within the same service.

Each example includes a link to GitHub, where you can find instructions on how to set up and run the code in context.

Topics

- [Actions](#)
- [Scenarios](#)

Actions

Attach a policy to a role

The following code example shows how to attach an IAM policy to a role.

AWS CLI with Bash script

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
#####
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
#####
function errecho() {
    printf "%s\n" "$*" 1>&2
}

#####
# function iam_attach_role_policy
#
# This function attaches an IAM policy to a role.
#
# Parameters:
#     -n role_name -- The name of the IAM role.
#     -p policy_ARN -- The IAM policy document ARN..
#
# Returns:
#     0 - If successful.
#     1 - If it fails.
#####
function iam_attach_role_policy() {
    local role_name policy_arn response
    local option OPTARG # Required to use getopt command in a function.

    # bashsupport disable=BP5008
    function usage() {
        echo "function iam_attach_role_policy"
        echo "Attaches an AWS Identity and Access Management (IAM) policy to an IAM"
        echo "role."
        echo "  -n role_name      The name of the IAM role."
        echo "  -p policy_ARN -- The IAM policy document ARN."
        echo ""
    }

    # Retrieve the calling parameters.
    while getopt "n:p:h" option; do
        case "${option}" in
            n) role_name="${OPTARG}" ;;
            p) policy_arn="${OPTARG}" ;;
            h)
                usage
            ;;
        esac
    done
}
```

```
        return 0
    ;;
\?) echo "Invalid parameter"
usage
return 1
;;
esac
done
export OPTIND=1

if [[ -z "$role_name" ]]; then
    errecho "ERROR: You must provide a role name with the -n parameter."
    usage
    return 1
fi

if [[ -z "$policy_arn" ]]; then
    errecho "ERROR: You must provide a policy ARN with the -p parameter."
    usage
    return 1
fi

response=$(aws iam attach-role-policy \
--role-name "$role_name" \
--policy-arn "$policy_arn")

local error_code=${?}

if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports attach-role-policy operation failed.\n$response"
    return 1
fi

echo "$response"

return 0
}
```

- For API details, see [AttachRolePolicy](#) in *AWS CLI Command Reference*.

Create a policy

The following code example shows how to create an IAM policy.

AWS CLI with Bash script

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
#####
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
#####
function errecho() {
    printf "%s\n" "$*" 1>&2
}

#####
# function iam_create_policy
#
# This function creates an IAM policy.
#
# Parameters:
#     -n policy_name -- The name of the IAM policy.
#     -p policy_json -- The policy document.
#
# Returns:
#     0 - If successful.
#     1 - If it fails.
#####
function iam_create_policy() {
    local policy_name policy_document response
    local option OPTARG # Required to use getopt command in a function.

    # bashsupport disable=BP5008
    function usage() {
        echo "function iam_create_policy"
        echo "Creates an AWS Identity and Access Management (IAM) policy."
        echo "  -n policy_name  The name of the IAM policy."
    }
}
```

```
echo " -p policy_json -- The policy document."
echo ""
}

# Retrieve the calling parameters.
while getopts "n:p:h" option; do
    case "${option}" in
        n) policy_name="${OPTARG}" ;;
        p) policy_document="${OPTARG}" ;;
        h)
            usage
            return 0
            ;;
        \?)
            echo "Invalid parameter"
            usage
            return 1
            ;;
    esac
done
export OPTIND=1

if [[ -z "$policy_name" ]]; then
    errecho "ERROR: You must provide a policy name with the -n parameter."
    usage
    return 1
fi

if [[ -z "$policy_document" ]]; then
    errecho "ERROR: You must provide a policy document with the -p parameter."
    usage
    return 1
fi

response=$(aws iam create-policy \
    --policy-name "$policy_name" \
    --policy-document "$policy_document" \
    --output text \
    --query Policy.Arn)

local error_code=${?}

if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
```

```
errecho "ERROR: AWS reports create-policy operation failed.\n$response"
return 1
fi

echo "$response"
}
```

- For API details, see [CreatePolicy](#) in *AWS CLI Command Reference*.

Create a role

The following code example shows how to create an IAM role.

AWS CLI with Bash script

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
#####
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
#####
function errecho() {
    printf "%s\n" "$*" 1>&2
}

#####
# function iam_create_role
#
# This function creates an IAM role.
#
# Parameters:
#     -n role_name -- The name of the IAM role.
#     -p policy_json -- The assume role policy document.
#
# Returns:
#     The ARN of the role.
```

```
#      And:
#          0 - If successful.
#          1 - If it fails.
#####
function iam_create_role() {
    local role_name policy_document response
    local option OPTARG # Required to use getopts command in a function.

    # bashsupport disable=BP5008
    function usage() {
        echo "function iam_create_user_access_key"
        echo "Creates an AWS Identity and Access Management (IAM) role."
        echo "  -n role_name    The name of the IAM role."
        echo "  -p policy_json -- The assume role policy document."
        echo ""
    }

    # Retrieve the calling parameters.
    while getopts "n:p:h" option; do
        case "${option}" in
            n)
                role_name="${OPTARG}" ;;
            p)
                policy_document="${OPTARG}" ;;
            h)
                usage
                return 0
                ;;
            \?)
                echo "Invalid parameter"
                usage
                return 1
                ;;
        esac
    done
    export OPTIND=1

    if [[ -z "$role_name" ]]; then
        errecho "ERROR: You must provide a role name with the -n parameter."
        usage
        return 1
    fi

    if [[ -z "$policy_document" ]]; then
        errecho "ERROR: You must provide a policy document with the -p parameter."
        usage
    fi
}
```

```
    return 1
fi

response=$(aws iam create-role \
--role-name "$role_name" \
--assume-role-policy-document "$policy_document" \
--output text \
--query Role.Arn)

local error_code=${?}

if [[ $error_code -ne 0 ]]; then
  aws_cli_error_log $error_code
  errecho "ERROR: AWS reports create-role operation failed.\n$response"
  return 1
fi

echo "$response"

return 0
}
```

- For API details, see [CreateRole](#) in *AWS CLI Command Reference*.

Create a user

The following code example shows how to create an IAM user.

Warning

To avoid security risks, don't use IAM users for authentication when developing purpose-built software or working with real data. Instead, use federation with an identity provider such as [AWS IAM Identity Center](#).

AWS CLI with Bash script

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
#####
# function iecho
#
# This function enables the script to display the specified text only if
# the global variable $VERBOSE is set to true.
#####
function iecho() {
    if [[ $VERBOSE == true ]]; then
        echo "$@"
    fi
}

#####
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
#####
function errecho() {
    printf "%s\n" "$*" 1>&2
}

#####
# function iam_create_user
#
# This function creates the specified IAM user, unless
# it already exists.
#
# Parameters:
#     -u user_name -- The name of the user to create.
#
# Returns:
#     The ARN of the user.
#     And:
#     0 - If successful.
```

```
#      1 - If it fails.
#####
function iam_create_user() {
    local user_name response
    local option OPTARG # Required to use getopts command in a function.

    # bashsupport disable=BP5008
    function usage() {
        echo "function iam_create_user"
        echo "Creates an AWS Identity and Access Management (IAM) user. You must supply a"
        echo "username:"
        echo "  -u user_name      The name of the user. It must be unique within the"
        echo "account."
        echo ""
    }
}

# Retrieve the calling parameters.
while getopts "u:h" option; do
    case "${option}" in
        u) user_name="${OPTARG}" ;;
        h)
            usage
            return 0
            ;;
        \?)
            echo "Invalid parameter"
            usage
            return 1
            ;;
    esac
done
export OPTIND=1

if [[ -z "$user_name" ]]; then
    errecho "ERROR: You must provide a username with the -u parameter."
    usage
    return 1
fi

iecho "Parameters:\n"
iecho "  User name: $user_name"
iecho ""

# If the user already exists, we don't want to try to create it.
```

```
if (iam_user_exists "$user_name"); then
    errecho "ERROR: A user with that name already exists in the account."
    return 1
fi

response=$(aws iam create-user --user-name "$user_name" \
--output text \
--query 'User.Arn')

local error_code=${?}

if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports create-user operation failed.$response"
    return 1
fi

echo "$response"

return 0
}
```

- For API details, see [CreateUser](#) in *AWS CLI Command Reference*.

Create an access key

The following code example shows how to create an IAM access key.

Warning

To avoid security risks, don't use IAM users for authentication when developing purpose-built software or working with real data. Instead, use federation with an identity provider such as [AWS IAM Identity Center](#).

AWS CLI with Bash script

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
#####
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
#####
function errecho() {
    printf "%s\n" "$*" 1>&2
}

#####
# function iam_create_user_access_key
#
# This function creates an IAM access key for the specified user.
#
# Parameters:
#     -u user_name -- The name of the IAM user.
#     [-f file_name] -- The optional file name for the access key output.
#
# Returns:
#     [access_key_id access_key_secret]
#     And:
#     0 - If successful.
#     1 - If it fails.
#####
function iam_create_user_access_key() {
    local user_name file_name response
    local option OPTARG # Required to use getopts command in a function.

    # bashsupport disable=BP5008
    function usage() {
        echo "function iam_create_user_access_key"
        echo "Creates an AWS Identity and Access Management (IAM) key pair."
        echo "  -u user_name  The name of the IAM user."
        echo "  [-f file_name] Optional file name for the access key output."
    }
}
```

```
echo ""
}

# Retrieve the calling parameters.
while getopts "u:f:h" option; do
  case "${option}" in
    u) user_name="${OPTARG}" ;;
    f) file_name="${OPTARG}" ;;
    h)
      usage
      return 0
      ;;
    \?) 
      echo "Invalid parameter"
      usage
      return 1
      ;;
  esac
done
export OPTIND=1

if [[ -z "$user_name" ]]; then
  errecho "ERROR: You must provide a username with the -u parameter."
  usage
  return 1
fi

response=$(aws iam create-access-key \
--user-name "$user_name" \
--output text)

local error_code=${?}

if [[ $error_code -ne 0 ]]; then
  aws_cli_error_log $error_code
  errecho "ERROR: AWS reports create-access-key operation failed.$response"
  return 1
fi

if [[ -n "$file_name" ]]; then
  echo "$response" >"$file_name"
fi

local key_id key_secret
```

```
# shellcheck disable=SC2086
key_id=$(echo $response | cut -f 2 -d ' ')
# shellcheck disable=SC2086
key_secret=$(echo $response | cut -f 4 -d ' ')

echo "$key_id $key_secret"

return 0
}
```

- For API details, see [CreateAccessKey](#) in *AWS CLI Command Reference*.

Delete a policy

The following code example shows how to delete an IAM policy.

AWS CLI with Bash script

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
#####
# function iecho
#
# This function enables the script to display the specified text only if
# the global variable $VERBOSE is set to true.
#####
function iecho() {
    if [[ $VERBOSE == true ]]; then
        echo "$@"
    fi
}

#####
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
#####
```

```
function errecho() {
    printf "%s\n" "$*" 1>&2
}

#####
# function iam_delete_policy
#
# This function deletes an IAM policy.
#
# Parameters:
#     -n policy_arn -- The name of the IAM policy arn.
#
# Returns:
#     0 - If successful.
#     1 - If it fails.
#####
function iam_delete_policy() {
    local policy_arn response
    local option OPTARG # Required to use getopt command in a function.

    # bashsupport disable=BP5008
    function usage() {
        echo "function iam_delete_policy"
        echo "Deletes an WS Identity and Access Management (IAM) policy"
        echo "  -n policy_arn -- The name of the IAM policy arn."
        echo ""
    }

    # Retrieve the calling parameters.
    while getopt "n:h" option; do
        case "${option}" in
            n) policy_arn="${OPTARG}" ;;
            h)
                usage
                return 0
                ;;
            \?)
                echo "Invalid parameter"
                usage
                return 1
                ;;
        esac
    done
    export OPTIND=1
```

```
if [[ -z "$policy_arn" ]]; then
    errecho "ERROR: You must provide a policy arn with the -n parameter."
    usage
    return 1
fi

iecho "Parameters:\n"
iecho "    Policy arn: $policy_arn"
iecho ""

response=$(aws iam delete-policy \
--policy-arn "$policy_arn")

local error_code=${?}

if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports delete-policy operation failed.\n$response"
    return 1
fi

iecho "delete-policy response:$response"
iecho

return 0
}
```

- For API details, see [DeletePolicy](#) in *AWS CLI Command Reference*.

Delete a role

The following code example shows how to delete an IAM role.

AWS CLI with Bash script

 **Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
#####
# function iecho
#
# This function enables the script to display the specified text only if
# the global variable $VERBOSE is set to true.
#####
function iecho() {
    if [[ $VERBOSE == true ]]; then
        echo "$@"
    fi
}

#####
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
#####
function errecho() {
    printf "%s\n" "$*" 1>&2
}

#####
# function iam_delete_role
#
# This function deletes an IAM role.
#
# Parameters:
#     -n role_name -- The name of the IAM role.
#
# Returns:
#     0 - If successful.
#     1 - If it fails.
#####
function iam_delete_role() {
    local role_name response
    local option OPTARG # Required to use getopt command in a function.

    # bashsupport disable=BP5008
    function usage() {
        echo "function iam_delete_role"
        echo "Deletes an AWS Identity and Access Management (IAM) role"
        echo "  -n role_name -- The name of the IAM role."
        echo ""
    }
}
```

```
}

# Retrieve the calling parameters.
while getopts "n:h" option; do
    case "${option}" in
        n) role_name="${OPTARG}" ;;
        h)
            usage
            return 0
            ;;
        \?)
            echo "Invalid parameter"
            usage
            return 1
            ;;
    esac
done
export OPTIND=1

echo "role_name:$role_name"
if [[ -z "$role_name" ]]; then
    errecho "ERROR: You must provide a role name with the -n parameter."
    usage
    return 1
fi

iecho "Parameters:\n"
iecho "    Role name: $role_name"
iecho ""

response=$(aws iam delete-role \
--role-name "$role_name")

local error_code=${?}

if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports delete-role operation failed.\n$response"
    return 1
fi

iecho "delete-role response:$response"
iecho
```

```
    return 0
}
```

- For API details, see [DeleteRole](#) in *AWS CLI Command Reference*.

Delete a user

The following code example shows how to delete an IAM user.

Warning

To avoid security risks, don't use IAM users for authentication when developing purpose-built software or working with real data. Instead, use federation with an identity provider such as [AWS IAM Identity Center](#).

AWS CLI with Bash script

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
#####
# function iecho
#
# This function enables the script to display the specified text only if
# the global variable $VERBOSE is set to true.
#####
function iecho() {
    if [[ $VERBOSE == true ]]; then
        echo "$@"
    fi
}

#####
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
```

```
#####
function errecho() {
    printf "%s\n" "$*" 1>&2
}

#####
# function iam_delete_user
#
# This function deletes the specified IAM user.
#
# Parameters:
#     -u user_name -- The name of the user to create.
#
# Returns:
#     0 - If successful.
#     1 - If it fails.
#####

function iam_delete_user() {
    local user_name response
    local option OPTARG # Required to use getopt command in a function.

    # bashsupport disable=BP5008
    function usage() {
        echo "function iam_delete_user"
        echo "Deletes an AWS Identity and Access Management (IAM) user. You must supply a"
        echo "username:"
        echo "  -u user_name      The name of the user."
        echo ""
    }

    # Retrieve the calling parameters.
    while getopt "u:h" option; do
        case "${option}" in
            u) user_name="${OPTARG}" ;;
            h)
                usage
                return 0
                ;;
            \?)
                echo "Invalid parameter"
                usage
                return 1
                ;;
        esac
    done
}
```

```
done
export OPTIND=1

if [[ -z "$user_name" ]]; then
    errecho "ERROR: You must provide a username with the -u parameter."
    usage
    return 1
fi

iecho "Parameters:\n"
iecho "    User name: $user_name"
iecho ""

# If the user does not exist, we don't want to try to delete it.
if (! iam_user_exists "$user_name"); then
    errecho "ERROR: A user with that name does not exist in the account."
    return 1
fi

response=$(aws iam delete-user \
--user-name "$user_name")

local error_code=${?}

if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports delete-user operation failed.$response"
    return 1
fi

iecho "delete-user response:$response"
iecho

return 0
}
```

- For API details, see [DeleteUser](#) in *AWS CLI Command Reference*.

Delete an access key

The following code example shows how to delete an IAM access key.

⚠ Warning

To avoid security risks, don't use IAM users for authentication when developing purpose-built software or working with real data. Instead, use federation with an identity provider such as [AWS IAM Identity Center](#).

AWS CLI with Bash script

ⓘ Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
#####
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
#####
function errecho() {
    printf "%s\n" "$*" 1>&2
}

#####
# function iam_delete_access_key
#
# This function deletes an IAM access key for the specified IAM user.
#
# Parameters:
#     -u user_name -- The name of the user.
#     -k access_key -- The access key to delete.
#
# Returns:
#     0 - If successful.
#     1 - If it fails.
#####
function iam_delete_access_key() {
    local user_name access_key response
    local option OPTARG # Required to use getopt command in a function.
```

```
# bashsupport disable=BP5008
function usage() {
    echo "function iam_delete_access_key"
    echo "Deletes an AWS Identity and Access Management (IAM) access key for the
specified IAM user"
    echo " -u user_name      The name of the user."
    echo " -k access_key     The access key to delete."
    echo ""
}

# Retrieve the calling parameters.
while getopts "u:k:h" option; do
    case "${option}" in
        u) user_name="${OPTARG}" ;;
        k) access_key="${OPTARG}" ;;
        h)
            usage
            return 0
            ;;
        \?)
            echo "Invalid parameter"
            usage
            return 1
            ;;
    esac
done
export OPTIND=1

if [[ -z "$user_name" ]]; then
    errecho "ERROR: You must provide a username with the -u parameter."
    usage
    return 1
fi

if [[ -z "$access_key" ]]; then
    errecho "ERROR: You must provide an access key with the -k parameter."
    usage
    return 1
fi

iecho "Parameters:\n"
iecho "    Username: $user_name"
iecho "    Access key: $access_key"
iecho ""
```

```
response=$(aws iam delete-access-key \
--user-name "$user_name" \
--access-key-id "$access_key")

local error_code=${?}

if [[ $error_code -ne 0 ]]; then
aws_cli_error_log $error_code
errecho "ERROR: AWS reports delete-access-key operation failed.\n$response"
return 1
fi

iecho "delete-access-key response:$response"
iecho

return 0
}
```

- For API details, see [DeleteAccessKey](#) in *AWS CLI Command Reference*.

Detach a policy from a role

The following code example shows how to detach an IAM policy from a role.

AWS CLI with Bash script

 **Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
#####
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
#####
function errecho() {
    printf "%s\n" "$*" 1>&2
}
```

```
#####
# function iam_detach_role_policy
#
# This function detaches an IAM policy to a role.
#
# Parameters:
#     -n role_name -- The name of the IAM role.
#     -p policy_ARN -- The IAM policy document ARN..
#
# Returns:
#     0 - If successful.
#     1 - If it fails.
#####
function iam_detach_role_policy() {
    local role_name policy_arn response
    local optionOPTARG # Required to use getopt command in a function.

    # bashsupport disable=BP5008
    function usage() {
        echo "function iam_detach_role_policy"
        echo "Detaches an AWS Identity and Access Management (IAM) policy to an IAM role."
        echo "  -n role_name      The name of the IAM role."
        echo "  -p policy_ARN -- The IAM policy document ARN."
        echo ""
    }

    # Retrieve the calling parameters.
    while getopt "n:p:h" option; do
        case "${option}" in
            n) role_name="${OPTARG}" ;;
            p) policy_arn="${OPTARG}" ;;
            h)
                usage
                return 0
                ;;
            \?)
                echo "Invalid parameter"
                usage
                return 1
                ;;
        esac
    done
```

```
export OPTIND=1

if [[ -z "$role_name" ]]; then
    errecho "ERROR: You must provide a role name with the -n parameter."
    usage
    return 1
fi

if [[ -z "$policy_arn" ]]; then
    errecho "ERROR: You must provide a policy ARN with the -p parameter."
    usage
    return 1
fi

response=$(aws iam detach-role-policy \
--role-name "$role_name" \
--policy-arn "$policy_arn")

local error_code=${?}

if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports detach-role-policy operation failed.\n$response"
    return 1
fi

echo "$response"

return 0
}
```

- For API details, see [DetachRolePolicy](#) in *AWS CLI Command Reference*.

Get a user

The following code example shows how to get an IAM user.

⚠ Warning

To avoid security risks, don't use IAM users for authentication when developing purpose-built software or working with real data. Instead, use federation with an identity provider such as [AWS IAM Identity Center](#).

AWS CLI with Bash script

ⓘ Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
#####
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
#####
function errecho() {
    printf "%s\n" "$*" 1>&2
}

#####
# function iam_user_exists
#
# This function checks to see if the specified AWS Identity and Access Management
# (IAM) user already exists.
#
# Parameters:
#     $1 - The name of the IAM user to check.
#
# Returns:
#     0 - If the user already exists.
#     1 - If the user doesn't exist.
#####
function iam_user_exists() {
    local user_name
    user_name=$1
```

```
# Check whether the IAM user already exists.  
# We suppress all output - we're interested only in the return code.  
  
local errors  
errors=$(aws iam get-user \  
--user-name "$user_name" 2>&1 >/dev/null)  
  
local error_code=${?}  
  
if [[ $error_code -eq 0 ]]; then  
    return 0 # 0 in Bash script means true.  
else  
    if [[ $errors != *"error""*(NoSuchEntity)* ]]; then  
        aws_cli_error_log $error_code  
        errecho "Error calling iam get-user $errors"  
    fi  
  
    return 1 # 1 in Bash script means false.  
fi  
}
```

- For API details, see [GetUser](#) in *AWS CLI Command Reference*.

List a user's access keys

The following code example shows how to list a user's IAM access keys.

Warning

To avoid security risks, don't use IAM users for authentication when developing purpose-built software or working with real data. Instead, use federation with an identity provider such as [AWS IAM Identity Center](#).

AWS CLI with Bash script

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
#####
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
#####
function errecho() {
    printf "%s\n" "$*" 1>&2
}

#####
# function iam_list_access_keys
#
# This function lists the access keys for the specified user.
#
# Parameters:
#     -u user_name -- The name of the IAM user.
#
# Returns:
#     access_key_ids
#     And:
#         0 - If successful.
#         1 - If it fails.
#####
function iam_list_access_keys() {

    # bashsupport disable=BP5008
    function usage() {
        echo "function iam_list_access_keys"
        echo "Lists the AWS Identity and Access Management (IAM) access key IDs for the"
        echo "specified user."
        echo "  -u user_name  The name of the IAM user."
        echo ""
    }

    local user_name response
    local option OPTARG # Required to use getopt command in a function.
    # Retrieve the calling parameters.
    while getopts "u:h" option; do
        case "${option}" in
            u) user_name="${OPTARG}" ;;
            h)
                usage
                return 0
        esac
    done
}
```

```
;;
\?)  
    echo "Invalid parameter"  
    usage  
    return 1  
;;  
esac  
done  
export OPTIND=1  
  
if [[ -z "$user_name" ]]; then  
    errecho "ERROR: You must provide a username with the -u parameter."  
    usage  
    return 1  
fi  
  
response=$(aws iam list-access-keys \  
    --user-name "$user_name" \  
    --output text \  
    --query 'AccessKeyMetadata[].AccessKeyId')  
  
local error_code=${?}  
  
if [[ $error_code -ne 0 ]]; then  
    aws_cli_error_log $error_code  
    errecho "ERROR: AWS reports list-access-keys operation failed.$response"  
    return 1  
fi  
  
echo "$response"  
  
return 0  
}
```

- For API details, see [ListAccessKeys](#) in *AWS CLI Command Reference*.

List users

The following code example shows how to list IAM users.

⚠ Warning

To avoid security risks, don't use IAM users for authentication when developing purpose-built software or working with real data. Instead, use federation with an identity provider such as [AWS IAM Identity Center](#).

AWS CLI with Bash script

ℹ Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
#####
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
#####
function errecho() {
    printf "%s\n" "$*" 1>&2
}

#####
# function iam_list_users
#
# List the IAM users in the account.
#
# Returns:
#     The list of users names
#     And:
#         0 - If the user already exists.
#         1 - If the user doesn't exist.
#####
function iam_list_users() {
    local option OPTARG # Required to use getopt command in a function.
    local error_code
    # bashsupport disable=BP5008
    function usage() {
        echo "function iam_list_users"
```

```
echo "Lists the AWS Identity and Access Management (IAM) user in the account."
echo ""
}

# Retrieve the calling parameters.
while getopts "h" option; do
    case "${option}" in
        h)
            usage
            return 0
            ;;
        \?)
            echo "Invalid parameter"
            usage
            return 1
            ;;
    esac
done
export OPTIND=1

local response

response=$(aws iam list-users \
--output text \
--query "Users[].UserName")
error_code=${?}

if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports list-users operation failed.$response"
    return 1
fi

echo "$response"

return 0
}
```

- For API details, see [ListUsers](#) in *AWS CLI Command Reference*.

Scenarios

Create a user and assume a role

The following code example shows how to create a user and assume a role.

Warning

To avoid security risks, don't use IAM users for authentication when developing purpose-built software or working with real data. Instead, use federation with an identity provider such as [AWS IAM Identity Center](#).

- Create a user with no permissions.
- Create a role that grants permission to list Amazon S3 buckets for the account.
- Add a policy to let the user assume the role.
- Assume the role and list S3 buckets using temporary credentials, then clean up resources.

AWS CLI with Bash script

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
#####
# function iam_create_user_assume_role
#
# Scenario to create an IAM user, create an IAM role, and apply the role to the
# user.
#
# "IAM access" permissions are needed to run this code.
# "STS assume role" permissions are needed to run this code. (Note: It might be
# necessary to
#           create a custom policy).
#
# Returns:
#     0 - If successful.
#     1 - If an error occurred.
```

```
#####
function iam_create_user_assume_role() {
    if [ "$IAM_OPERATIONS_SOURCED" != "True" ]; then

        source ./iam_operations.sh
    fi
}

echo_repeat "*" 88
echo "Welcome to the IAM create user and assume role demo."
echo
echo "This demo will create an IAM user, create an IAM role, and apply the role to
the user."
echo_repeat "*" 88
echo

echo -n "Enter a name for a new IAM user: "
get_input
user_name=$get_input_result

local user_arn
user_arn=$(iam_create_user -u "$user_name")

# shellcheck disable=SC2181
if [[ ${?} == 0 ]]; then
    echo "Created demo IAM user named $user_name"
else
    errecho "$user_arn"
    errecho "The user failed to create. This demo will exit."
    return 1
fi

local access_key_response
access_key_response=$(iam_create_user_access_key -u "$user_name")
# shellcheck disable=SC2181
if [[ ${?} != 0 ]]; then
    errecho "The access key failed to create. This demo will exit."
    clean_up "$user_name"
    return 1
fi

IFS=$'\t ' read -r -a access_key_values <<<"$access_key_response"
local key_name=${access_key_values[0]}
```

```
local key_secret=${access_key_values[1]}

echo "Created access key named $key_name"

echo "Wait 10 seconds for the user to be ready."
sleep 10
echo_repeat "*" 88
echo

local iam_role_name
iam_role_name=$(generate_random_name "test-role")
echo "Creating a role named $iam_role_name with user $user_name as the principal."

local assume_role_policy_document="{
    \"Version\": \"2012-10-17\",
    \"Statement\": [
        {\"Effect\": \"Allow\",
         \"Principal\": {\"AWS\": \"$user_arn\"},
         \"Action\": \"sts:AssumeRole\"
        }
    ]
}"

local role_arn
role_arn=$(iam_create_role -n "$iam_role_name" -p "$assume_role_policy_document")

# shellcheck disable=SC2181
if [ ${?} == 0 ]; then
    echo "Created IAM role named $iam_role_name"
else
    errecho "The role failed to create. This demo will exit."
    clean_up "$user_name" "$key_name"
    return 1
fi

local policy_name
policy_name=$(generate_random_name "test-policy")
local policy_document="{
    \"Version\": \"2012-10-17\",
    \"Statement\": [
        {\"Effect\": \"Allow\",
         \"Action\": \"s3>ListAllMyBuckets\",
         \"Resource\": \"arn:aws:s3:::*\"}]}"
}

local policy_arn
```

```
policy_arn=$(iam_create_policy -n "$policy_name" -p "$policy_document")
# shellcheck disable=SC2181
if [[ ${?} == 0 ]]; then
    echo "Created IAM policy named $policy_name"
else
    errecho "The policy failed to create."
    clean_up "$user_name" "$key_name" "$iam_role_name"
    return 1
fi

if (iam_attach_role_policy -n "$iam_role_name" -p "$policy_arn"); then
    echo "Attached policy $policy_arn to role $iam_role_name"
else
    errecho "The policy failed to attach."
    clean_up "$user_name" "$key_name" "$iam_role_name" "$policy_arn"
    return 1
fi

local assume_role_policy_document="{
    \"Version\": \"2012-10-17\",
    \"Statement\": [
        {
            \"Effect\": \"Allow\",
            \"Action\": \"sts:AssumeRole\",
            \"Resource\": \"$role_arn\"]}]"

local assume_role_policy_name
assume_role_policy_name=$(generate_random_name "test-assume-role-")

# shellcheck disable=SC2181
local assume_role_policy_arn
assume_role_policy_arn=$(iam_create_policy -n "$assume_role_policy_name" -p
"$assume_role_policy_document")
# shellcheck disable=SC2181
if [ ${?} == 0 ]; then
    echo "Created IAM policy named $assume_role_policy_name for sts assume role"
else
    errecho "The policy failed to create."
    clean_up "$user_name" "$key_name" "$iam_role_name" "$policy_arn" "$policy_arn"
    return 1
fi

echo "Wait 10 seconds to give AWS time to propagate these new resources and
connections."
sleep 10
```

```
echo_repeat "*" 88
echo

echo "Try to list buckets without the new user assuming the role."
echo_repeat "*" 88
echo

# Set the environment variables for the created user.
# bashsupport disable=BP2001
export AWS_ACCESS_KEY_ID=$key_name
# bashsupport disable=BP2001
export AWS_SECRET_ACCESS_KEY=$key_secret

local buckets
buckets=$(s3_list_buckets)

# shellcheck disable=SC2181
if [ ${?} == 0 ]; then
    local bucket_count
    bucket_count=$(echo "$buckets" | wc -w | xargs)
    echo "There are $bucket_count buckets in the account. This should not have
happened."
else
    errecho "Because the role with permissions has not been assumed, listing buckets
failed."
fi

echo
echo_repeat "*" 88
echo "Now assume the role $iam_role_name and list the buckets."
echo_repeat "*" 88
echo

local credentials

credentials=$(sts_assume_role -r "$role_arn" -n "AssumeRoleDemoSession")
# shellcheck disable=SC2181
if [ ${?} == 0 ]; then
    echo "Assumed role $iam_role_name"
else
    errecho "Failed to assume role."
    export AWS_ACCESS_KEY_ID=""
    export AWS_SECRET_ACCESS_KEY=""
```

```
    clean_up "$user_name" "$key_name" "$iam_role_name" "$policy_arn" "$policy_arn"
"$assume_role_policy_arn"
    return 1
fi

IFS=$'\t ' read -r -a credentials <<<"$credentials"

export AWS_ACCESS_KEY_ID=${credentials[0]}
export AWS_SECRET_ACCESS_KEY=${credentials[1]}
# bashsupport disable=BP2001
export AWS_SESSION_TOKEN=${credentials[2]}

buckets=$(s3_list_buckets)

# shellcheck disable=SC2181
if [ ${?} == 0 ]; then
    local bucket_count
    bucket_count=$(echo "$buckets" | wc -w | xargs)
    echo "There are $bucket_count buckets in the account. Listing buckets succeeded
because of "
    echo "the assumed role."
else
    errecho "Failed to list buckets. This should not happen."
    export AWS_ACCESS_KEY_ID=""
    export AWS_SECRET_ACCESS_KEY=""
    export AWS_SESSION_TOKEN=""
    clean_up "$user_name" "$key_name" "$iam_role_name" "$policy_arn" "$policy_arn"
"$assume_role_policy_arn"
    return 1
fi

local result=0
export AWS_ACCESS_KEY_ID=""
export AWS_SECRET_ACCESS_KEY=""

echo
echo_repeat "*" 88
echo "The created resources will now be deleted."
echo_repeat "*" 88
echo

clean_up "$user_name" "$key_name" "$iam_role_name" "$policy_arn" "$policy_arn"
"$assume_role_policy_arn"
```

```
# shellcheck disable=SC2181
if [[ ${?} -ne 0 ]]; then
    result=1
fi

return $result
}
```

The IAM functions used in this scenario.

```
#####
# function iam_user_exists
#
# This function checks to see if the specified AWS Identity and Access Management
# (IAM) user already exists.
#
# Parameters:
#     $1 - The name of the IAM user to check.
#
# Returns:
#     0 - If the user already exists.
#     1 - If the user doesn't exist.
#####
function iam_user_exists() {
    local user_name
    user_name=$1

    # Check whether the IAM user already exists.
    # We suppress all output - we're interested only in the return code.

    local errors
    errors=$(aws iam get-user \
        --user-name "$user_name" 2>&1 >/dev/null)

    local error_code=${?}

    if [[ $error_code -eq 0 ]]; then
        return 0 # 0 in Bash script means true.
    else
        if [[ $errors != *"error""*(NoSuchEntity)* ]]; then
            aws_cli_error_log $error_code
            errecho "Error calling iam get-user $errors"
        fi
    fi
}
```

```
fi

    return 1 # 1 in Bash script means false.
fi
}

#####
# function iam_create_user
#
# This function creates the specified IAM user, unless
# it already exists.
#
# Parameters:
#   -u user_name -- The name of the user to create.
#
# Returns:
#   The ARN of the user.
#   And:
#   0 - If successful.
#   1 - If it fails.
#####
function iam_create_user() {
    local user_name response
    local option OPTARG # Required to use getopt command in a function.

    # bashsupport disable=BP5008
    function usage() {
        echo "function iam_create_user"
        echo "Creates an AWS Identity and Access Management (IAM) user. You must supply a"
        echo "username:"
        echo "  -u user_name      The name of the user. It must be unique within the"
        echo "account."
        echo ""
    }

    # Retrieve the calling parameters.
    while getopt "u:h" option; do
        case "${option}" in
            u) user_name="${OPTARG}" ;;
            h)
                usage
                return 0
                ;;
            \?)
```

```
        echo "Invalid parameter"
        usage
        return 1
    ;;
esac
done
export OPTIND=1

if [[ -z "$user_name" ]]; then
    errecho "ERROR: You must provide a username with the -u parameter."
    usage
    return 1
fi

iecho "Parameters:\n"
iecho "    User name: $user_name"
iecho ""

# If the user already exists, we don't want to try to create it.
if (iam_user_exists "$user_name"); then
    errecho "ERROR: A user with that name already exists in the account."
    return 1
fi

response=$(aws iam create-user --user-name "$user_name" \
--output text \
--query 'User.Arn')

local error_code=${?}

if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports create-user operation failed.$response"
    return 1
fi

echo "$response"

return 0
}

#####
# function iam_create_user_access_key
#
```

```
# This function creates an IAM access key for the specified user.  
#  
# Parameters:  
#     -u user_name -- The name of the IAM user.  
#     [-f file_name] -- The optional file name for the access key output.  
#  
# Returns:  
#     [access_key_id access_key_secret]  
#     And:  
#         0 - If successful.  
#         1 - If it fails.  
#####  
function iam_create_user_access_key() {  
    local user_name file_name response  
    local option OPTARG # Required to use getopts command in a function.  
  
    # bashsupport disable=BP5008  
    function usage() {  
        echo "function iam_create_user_access_key"  
        echo "Creates an AWS Identity and Access Management (IAM) key pair."  
        echo " -u user_name    The name of the IAM user."  
        echo " [-f file_name]  Optional file name for the access key output."  
        echo ""  
    }  
  
    # Retrieve the calling parameters.  
    while getopts "u:f:h" option; do  
        case "${option}" in  
            u) user_name="${OPTARG}" ;;  
            f) file_name="${OPTARG}" ;;  
            h)  
                usage  
                return 0  
                ;;  
            \?)  
                echo "Invalid parameter"  
                usage  
                return 1  
                ;;  
        esac  
    done  
    export OPTIND=1  
  
    if [[ -z "$user_name" ]]; then
```

```
errecho "ERROR: You must provide a username with the -u parameter."
usage
return 1
fi

response=$(aws iam create-access-key \
--user-name "$user_name" \
--output text)

local error_code=${?}

if [[ $error_code -ne 0 ]]; then
  aws_cli_error_log $error_code
  errecho "ERROR: AWS reports create-access-key operation failed.$response"
  return 1
fi

if [[ -n "$file_name" ]]; then
  echo "$response" >"$file_name"
fi

local key_id key_secret
# shellcheck disable=SC2086
key_id=$(echo $response | cut -f 2 -d ' ')
# shellcheck disable=SC2086
key_secret=$(echo $response | cut -f 4 -d ' ')

echo "$key_id $key_secret"

return 0
}

#####
# function iam_create_role
#
# This function creates an IAM role.
#
# Parameters:
#   -n role_name -- The name of the IAM role.
#   -p policy_json -- The assume role policy document.
#
# Returns:
#   The ARN of the role.
#   And:
```

```
#      0 - If successful.
#      1 - If it fails.
#####
function iam_create_role() {
    local role_name policy_document response
    local option OPTARG # Required to use getopts command in a function.

    # bashsupport disable=BP5008
    function usage() {
        echo "function iam_create_user_access_key"
        echo "Creates an AWS Identity and Access Management (IAM) role."
        echo " -n role_name   The name of the IAM role."
        echo " -p policy_json -- The assume role policy document."
        echo ""
    }

    # Retrieve the calling parameters.
    while getopts "n:p:h" option; do
        case "${option}" in
            n) role_name="${OPTARG}" ;;
            p) policy_document="${OPTARG}" ;;
            h)
                usage
                return 0
                ;;
            \?)
                echo "Invalid parameter"
                usage
                return 1
                ;;
        esac
    done
    export OPTIND=1

    if [[ -z "$role_name" ]]; then
        errecho "ERROR: You must provide a role name with the -n parameter."
        usage
        return 1
    fi

    if [[ -z "$policy_document" ]]; then
        errecho "ERROR: You must provide a policy document with the -p parameter."
        usage
        return 1
    fi
```

```
fi

response=$(aws iam create-role \
--role-name "$role_name" \
--assume-role-policy-document "$policy_document" \
--output text \
--query Role.Arn)

local error_code=${?}

if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports create-role operation failed.\n$response"
    return 1
fi

echo "$response"

return 0
}

#####
# function iam_create_policy
#
# This function creates an IAM policy.
#
# Parameters:
#     -n policy_name -- The name of the IAM policy.
#     -p policy_json -- The policy document.
#
# Returns:
#     0 - If successful.
#     1 - If it fails.
#####
function iam_create_policy() {
    local policy_name policy_document response
    local option OPTARG # Required to use getopts command in a function.

    # bashsupport disable=BP5008
    function usage() {
        echo "function iam_create_policy"
        echo "Creates an AWS Identity and Access Management (IAM) policy."
        echo "  -n policy_name  The name of the IAM policy."
        echo "  -p policy_json -- The policy document."
    }
}
```

```
echo ""

}

# Retrieve the calling parameters.
while getopts "n:p:h" option; do
  case "${option}" in
    n) policy_name="${OPTARG}" ;;
    p) policy_document="${OPTARG}" ;;
    h)
      usage
      return 0
      ;;
    \?) 
      echo "Invalid parameter"
      usage
      return 1
      ;;
  esac
done
export OPTIND=1

if [[ -z "$policy_name" ]]; then
  errecho "ERROR: You must provide a policy name with the -n parameter."
  usage
  return 1
fi

if [[ -z "$policy_document" ]]; then
  errecho "ERROR: You must provide a policy document with the -p parameter."
  usage
  return 1
fi

response=$(aws iam create-policy \
  --policy-name "$policy_name" \
  --policy-document "$policy_document" \
  --output text \
  --query Policy.Arn)

local error_code=${?}

if [[ $error_code -ne 0 ]]; then
  aws_cli_error_log $error_code
  errecho "ERROR: AWS reports create-policy operation failed.\n$response"
```

```
        return 1
    fi

    echo "$response"
}

#####
# function iam_attach_role_policy
#
# This function attaches an IAM policy to a role.
#
# Parameters:
#     -n role_name -- The name of the IAM role.
#     -p policy_ARN -- The IAM policy document ARN..
#
# Returns:
#     0 - If successful.
#     1 - If it fails.
#####
function iam_attach_role_policy() {
    local role_name policy_arn response
    local option OPTARG # Required to use getopt command in a function.

    # bashsupport disable=BP5008
    function usage() {
        echo "function iam_attach_role_policy"
        echo "Attaches an AWS Identity and Access Management (IAM) policy to an IAM role."
        echo "  -n role_name      The name of the IAM role."
        echo "  -p policy_ARN -- The IAM policy document ARN."
        echo ""
    }

    # Retrieve the calling parameters.
    while getopt "n:p:h" option; do
        case "${option}" in
            n) role_name="${OPTARG}";;
            p) policy_arn="${OPTARG}";;
            h)
                usage
                return 0
                ;;
            \?)
                echo "Invalid parameter"
                exit 1
        esac
    done
}
```

```
usage
return 1
;;
esac
done
export OPTIND=1

if [[ -z "$role_name" ]]; then
errecho "ERROR: You must provide a role name with the -n parameter."
usage
return 1
fi

if [[ -z "$policy_arn" ]]; then
errecho "ERROR: You must provide a policy ARN with the -p parameter."
usage
return 1
fi

response=$(aws iam attach-role-policy \
--role-name "$role_name" \
--policy-arn "$policy_arn")

local error_code=${?}

if [[ $error_code -ne 0 ]]; then
aws_cli_error_log $error_code
errecho "ERROR: AWS reports attach-role-policy operation failed.\n$response"
return 1
fi

echo "$response"

return 0
}

#####
# function iam_detach_role_policy
#
# This function detaches an IAM policy to a role.
#
# Parameters:
#   -n role_name -- The name of the IAM role.
#   -p policy_ARN -- The IAM policy document ARN..
```

```
#  
# Returns:  
#     0 - If successful.  
#     1 - If it fails.  
#####  
function iam_detach_role_policy() {  
    local role_name policy_arn response  
    local optionOPTARG # Required to use getopt command in a function.  
  
    # bashsupport disable=BP5008  
    function usage() {  
        echo "function iam_detach_role_policy"  
        echo "Detaches an AWS Identity and Access Management (IAM) policy to an IAM  
role."  
        echo "  -n role_name  The name of the IAM role."  
        echo "  -p policy_ARN -- The IAM policy document ARN."  
        echo ""  
    }  
  
    # Retrieve the calling parameters.  
    while getopt "n:p:h" option; do  
        case "${option}" in  
            n) role_name="${OPTARG}" ;;  
            p) policy_arn="${OPTARG}" ;;  
            h)  
                usage  
                return 0  
                ;;  
            \?)  
                echo "Invalid parameter"  
                usage  
                return 1  
                ;;  
        esac  
    done  
    export OPTIND=1  
  
    if [[ -z "$role_name" ]]; then  
        errecho "ERROR: You must provide a role name with the -n parameter."  
        usage  
        return 1  
    fi  
  
    if [[ -z "$policy_arn" ]]; then
```

```
errecho "ERROR: You must provide a policy ARN with the -p parameter."
usage
return 1
fi

response=$(aws iam detach-role-policy \
--role-name "$role_name" \
--policy-arn "$policy_arn")

local error_code=${?}

if [[ $error_code -ne 0 ]]; then
  aws_cli_error_log $error_code
  errecho "ERROR: AWS reports detach-role-policy operation failed.\n$response"
  return 1
fi

echo "$response"

return 0
}

#####
# function iam_delete_policy
#
# This function deletes an IAM policy.
#
# Parameters:
#   -n policy_arn -- The name of the IAM policy arn.
#
# Returns:
#   0 - If successful.
#   1 - If it fails.
#####
function iam_delete_policy() {
  local policy_arn response
  local option OPTARG # Required to use getopt command in a function.

  # bashsupport disable=BP5008
  function usage() {
    echo "function iam_delete_policy"
    echo "Deletes an WS Identity and Access Management (IAM) policy"
    echo "  -n policy_arn -- The name of the IAM policy arn."
    echo ""
  }
```

```
}

# Retrieve the calling parameters.
while getopts "n:h" option; do
    case "${option}" in
        n) policy_arn="${OPTARG}" ;;
        h)
            usage
            return 0
            ;;
        \?)
            echo "Invalid parameter"
            usage
            return 1
            ;;
    esac
done
export OPTIND=1

if [[ -z "$policy_arn" ]]; then
    errecho "ERROR: You must provide a policy arn with the -n parameter."
    usage
    return 1
fi

iecho "Parameters:\n"
iecho "    Policy arn: $policy_arn"
iecho ""

response=$(aws iam delete-policy \
--policy-arn "$policy_arn")

local error_code=${?}

if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports delete-policy operation failed.\n$response"
    return 1
fi

iecho "delete-policy response:$response"
iecho

return 0
```

```
}

#####
# function iam_delete_role
#
# This function deletes an IAM role.
#
# Parameters:
#     -n role_name -- The name of the IAM role.
#
# Returns:
#     0 - If successful.
#     1 - If it fails.
#####
function iam_delete_role() {
    local role_name response
    local option OPTARG # Required to use getopt command in a function.

    # bashsupport disable=BP5008
    function usage() {
        echo "function iam_delete_role"
        echo "Deletes an AWS Identity and Access Management (IAM) role"
        echo " -n role_name -- The name of the IAM role."
        echo ""
    }

    # Retrieve the calling parameters.
    while getopts "n:h" option; do
        case "${option}" in
            n) role_name="${OPTARG}" ;;
            h)
                usage
                return 0
                ;;
            \?)
                echo "Invalid parameter"
                usage
                return 1
                ;;
        esac
    done
    export OPTIND=1

    echo "role_name:$role_name"
```

```
if [[ -z "$role_name" ]]; then
    errecho "ERROR: You must provide a role name with the -n parameter."
    usage
    return 1
fi

iecho "Parameters:\n"
iecho "    Role name: $role_name"
iecho ""

response=$(aws iam delete-role \
--role-name "$role_name")

local error_code=${?}

if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports delete-role operation failed.\n$response"
    return 1
fi

iecho "delete-role response:$response"
iecho

return 0
}

#####
# function iam_delete_access_key
#
# This function deletes an IAM access key for the specified IAM user.
#
# Parameters:
#     -u user_name -- The name of the user.
#     -k access_key -- The access key to delete.
#
# Returns:
#     0 - If successful.
#     1 - If it fails.
#####
function iam_delete_access_key() {
    local user_name access_key response
    local option OPTARG # Required to use getopt command in a function.
```

```
# bashsupport disable=BP5008
function usage() {
    echo "function iam_delete_access_key"
    echo "Deletes an AWS Identity and Access Management (IAM) access key for the
specified IAM user"
    echo " -u user_name      The name of the user."
    echo " -k access_key     The access key to delete."
    echo ""
}

# Retrieve the calling parameters.
while getopts "u:k:h" option; do
    case "${option}" in
        u) user_name="${OPTARG}" ;;
        k) access_key="${OPTARG}" ;;
        h)
            usage
            return 0
            ;;
        \?)
            echo "Invalid parameter"
            usage
            return 1
            ;;
    esac
done
export OPTIND=1

if [[ -z "$user_name" ]]; then
    errecho "ERROR: You must provide a username with the -u parameter."
    usage
    return 1
fi

if [[ -z "$access_key" ]]; then
    errecho "ERROR: You must provide an access key with the -k parameter."
    usage
    return 1
fi

iecho "Parameters:\n"
iecho "    Username: $user_name"
iecho "    Access key: $access_key"
iecho ""
```

```
response=$(aws iam delete-access-key \
--user-name "$user_name" \
--access-key-id "$access_key")

local error_code=${?}

if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports delete-access-key operation failed.\n$response"
    return 1
fi

iecho "delete-access-key response:$response"
iecho

return 0
}

#####
# function iam_delete_user
#
# This function deletes the specified IAM user.
#
# Parameters:
#     -u user_name -- The name of the user to create.
#
# Returns:
#     0 - If successful.
#     1 - If it fails.
#####

function iam_delete_user() {
    local user_name response
    local option OPTARG # Required to use getopt command in a function.

    # bashsupport disable=BP5008
    function usage() {
        echo "function iam_delete_user"
        echo "Deletes an AWS Identity and Access Management (IAM) user. You must supply a"
        echo "username:"
        echo "  -u user_name      The name of the user."
        echo ""
    }
}
```

```
# Retrieve the calling parameters.
while getopts "u:h" option; do
    case "${option}" in
        u) user_name="${OPTARG}" ;;
        h)
            usage
            return 0
            ;;
        \?)
            echo "Invalid parameter"
            usage
            return 1
            ;;
    esac
done
export OPTIND=1

if [[ -z "$user_name" ]]; then
    errecho "ERROR: You must provide a username with the -u parameter."
    usage
    return 1
fi

iecho "Parameters:\n"
iecho "    User name: $user_name"
iecho ""

# If the user does not exist, we don't want to try to delete it.
if (! iam_user_exists "$user_name"); then
    errecho "ERROR: A user with that name does not exist in the account."
    return 1
fi

response=$(aws iam delete-user \
    --user-name "$user_name")

local error_code=${?}

if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports delete-user operation failed.$response"
    return 1
fi
```

```
iecho "delete-user response:$response"
iecho

return 0
}
```

- For API details, see the following topics in *AWS CLI Command Reference*.

- [AttachRolePolicy](#)
- [CreateAccessKey](#)
- [CreatePolicy](#)
- [CreateRole](#)
- [CreateUser](#)
- [DeleteAccessKey](#)
- [DeletePolicy](#)
- [DeleteRole](#)
- [DeleteUser](#)
- [DeleteUserPolicy](#)
- [DetachRolePolicy](#)
- [PutUserPolicy](#)

Amazon S3 examples using AWS CLI with Bash script

The following code examples show you how to perform actions and implement common scenarios by using the AWS Command Line Interface with Bash script with Amazon S3.

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios and cross-service examples.

Scenarios are code examples that show you how to accomplish a specific task by calling multiple functions within the same service.

Each example includes a link to GitHub, where you can find instructions on how to set up and run the code in context.

Topics

- [Actions](#)
- [Scenarios](#)

Actions

Copy an object from one bucket to another

The following code example shows how to copy an S3 object from one bucket to another.

AWS CLI with Bash script

 **Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
#####
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
#####
function errecho() {
    printf "%s\n" "$*" 1>&2
}

#####
# function copy_item_in_bucket
#
# This function creates a copy of the specified file in the same bucket.
#
# Parameters:
#     $1 - The name of the bucket to copy the file from and to.
#     $2 - The key of the source file to copy.
#     $3 - The key of the destination file.
#
# Returns:
#     0 - If successful.
#     1 - If it fails.
#####
function copy_item_in_bucket() {
    local bucket_name=$1
```

```
local source_key=$2
local destination_key=$3
local response

response=$(aws s3api copy-object \
--bucket "$bucket_name" \
--copy-source "$bucket_name/$source_key" \
--key "$destination_key")

# shellcheck disable=SC2181
if [[ $? -ne 0 ]]; then
    errecho "ERROR: AWS reports s3api copy-object operation failed.\n$response"
    return 1
fi
}
```

- For API details, see [CopyObject](#) in *AWS CLI Command Reference*.

Create a bucket

The following code example shows how to create an S3 bucket.

AWS CLI with Bash script

 **Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
#####
# function iecho
#
# This function enables the script to display the specified text only if
# the global variable $VERBOSE is set to true.
#####
function iecho() {
    if [[ $VERBOSE == true ]]; then
        echo "$@"
    fi
}
```

```
#####
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
#####
function errecho() {
    printf "%s\n" "$*" 1>&2
}

#####
# function create-bucket
#
# This function creates the specified bucket in the specified AWS Region, unless
# it already exists.
#
# Parameters:
#     -b bucket_name -- The name of the bucket to create.
#     -r region_code -- The code for an AWS Region in which to
#                       create the bucket.
#
# Returns:
#     The URL of the bucket that was created.
#     And:
#         0 - If successful.
#         1 - If it fails.
#####
function create_bucket() {
    local bucket_name region_code response
    local option OPTARG # Required to use getopt command in a function.

    # bashsupport disable=BP5008
    function usage() {
        echo "function create_bucket"
        echo "Creates an Amazon S3 bucket. You must supply a bucket name:"
        echo "  -b bucket_name      The name of the bucket. It must be globally unique."
        echo "  [-r region_code]    The code for an AWS Region in which the bucket is
created."
        echo ""
    }

    # Retrieve the calling parameters.
    while getopt "b:r:h" option; do
        case "${option}" in

```

```
b) bucket_name="${OPTARG}" ;;
r) region_code="${OPTARG}" ;;
h)
    usage
    return 0
    ;;
\?)
    echo "Invalid parameter"
    usage
    return 1
    ;;
esac
done

if [[ -z "$bucket_name" ]]; then
    errecho "ERROR: You must provide a bucket name with the -b parameter."
    usage
    return 1
fi

local bucket_config_arg
# A location constraint for "us-east-1" returns an error.
if [[ -n "$region_code" ]] && [[ "$region_code" != "us-east-1" ]]; then
    bucket_config_arg="--create-bucket-configuration LocationConstraint=
$region_code"
fi

iecho "Parameters:\n"
iecho "    Bucket name:    $bucket_name"
iecho "    Region code:   $region_code"
iecho ""

# If the bucket already exists, we don't want to try to create it.
if (bucket_exists "$bucket_name"); then
    errecho "ERROR: A bucket with that name already exists. Try again."
    return 1
fi

# shellcheck disable=SC2086
response=$(aws s3api create-bucket \
    --bucket "$bucket_name" \
    $bucket_config_arg)

# shellcheck disable=SC2181
```

```
if [[ ${?} -ne 0 ]]; then
    errecho "ERROR: AWS reports create-bucket operation failed.\n$response"
    return 1
fi
}
```

- For API details, see [CreateBucket](#) in *AWS CLI Command Reference*.

Delete an empty bucket

The following code example shows how to delete an empty S3 bucket.

AWS CLI with Bash script

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
#####
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
#####
function errecho() {
    printf "%s\n" "$*" 1>&2
}

#####
# function delete_bucket
#
# This function deletes the specified bucket.
#
# Parameters:
#     $1 - The name of the bucket.

# Returns:
#     0 - If successful.
#     1 - If it fails.
#####
```

```
function delete_bucket() {
    local bucket_name=$1
    local response

    response=$(aws s3api delete-bucket \
        --bucket "$bucket_name")

    # shellcheck disable=SC2181
    if [[ $? -ne 0 ]]; then
        errecho "ERROR: AWS reports s3api delete-bucket failed.\n$response"
        return 1
    fi
}
```

- For API details, see [DeleteBucket](#) in *AWS CLI Command Reference*.

Delete an object

The following code example shows how to delete an S3 object.

AWS CLI with Bash script

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
#####
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
#####
function errecho() {
    printf "%s\n" "$*" 1>&2
}

#####
# function delete_item_in_bucket
#
# This function deletes the specified file from the specified bucket.
```

```
#  
# Parameters:  
#     $1 - The name of the bucket.  
#     $2 - The key (file name) in the bucket to delete.  
  
# Returns:  
#     0 - If successful.  
#     1 - If it fails.  
#####  
function delete_item_in_bucket() {  
    local bucket_name=$1  
    local key=$2  
    local response  
  
    response=$(aws s3api delete-object \  
        --bucket "$bucket_name" \  
        --key "$key")  
  
    # shellcheck disable=SC2181  
    if [[ $? -ne 0 ]]; then  
        errecho "ERROR: AWS reports s3api delete-object operation failed.\n$response"  
        return 1  
    fi  
}  
}
```

- For API details, see [DeleteObject](#) in *AWS CLI Command Reference*.

Delete multiple objects

The following code example shows how to delete multiple objects from an S3 bucket.

AWS CLI with Bash script

 **Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
#####  
# function errecho
```

```
#  
# This function outputs everything sent to it to STDERR (standard error output).  
#####  
function errecho() {  
    printf "%s\n" "$*" 1>&2  
}  
  
#####  
# function delete_items_in_bucket  
#  
# This function deletes the specified list of keys from the specified bucket.  
#  
# Parameters:  
#     $1 - The name of the bucket.  
#     $2 - A list of keys in the bucket to delete.  
  
# Returns:  
#     0 - If successful.  
#     1 - If it fails.  
#####  
function delete_items_in_bucket() {  
    local bucket_name=$1  
    local keys=$2  
    local response  
  
    # Create the JSON for the items to delete.  
    local delete_items  
    delete_items="{\"Objects\":[\"  
    for key in $keys; do  
        delete_items="$delete_items{\"Key\": \"$key\"},"  
    done  
    delete_items=${delete_items%?} # Remove the final comma.  
    delete_items="$delete_items]}"  
  
    response=$(aws s3api delete-objects \  
        --bucket "$bucket_name" \  
        --delete "$delete_items")  
  
    # shellcheck disable=SC2181  
    if [[ $? -ne 0 ]]; then  
        errecho "ERROR: AWS reports s3api delete-object operation failed.\n$response"  
        return 1  
    fi  
}
```

- For API details, see [DeleteObjects](#) in *AWS CLI Command Reference*.

Determine the existence of a bucket

The following code example shows how to determine the existence of an S3 bucket.

AWS CLI with Bash script

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
#####
# function bucket_exists
#
# This function checks to see if the specified bucket already exists.
#
# Parameters:
#     $1 - The name of the bucket to check.
#
# Returns:
#     0 - If the bucket already exists.
#     1 - If the bucket doesn't exist.
#####
function bucket_exists() {
    local bucket_name
    bucket_name=$1

    # Check whether the bucket already exists.
    # We suppress all output - we're interested only in the return code.

    if aws s3api head-bucket \
        --bucket "$bucket_name" \
        >/dev/null 2>&1; then
        return 0 # 0 in Bash script means true.
    else
        return 1 # 1 in Bash script means false.
    fi
}
```

```
    fi  
}
```

- For API details, see [HeadBucket](#) in *AWS CLI Command Reference*.

Get an object from a bucket

The following code example shows how to read data from an object in an S3 bucket.

AWS CLI with Bash script

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
#####
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
#####
function errecho() {
    printf "%s\n" "$*" 1>&2
}

#####
# function download_object_from_bucket
#
# This function downloads an object in a bucket to a file.
#
# Parameters:
#     $1 - The name of the bucket to download the object from.
#     $2 - The path and file name to store the downloaded bucket.
#     $3 - The key (name) of the object in the bucket.
#
# Returns:
#     0 - If successful.
#     1 - If it fails.
#####
```

```
function download_object_from_bucket() {
    local bucket_name=$1
    local destination_file_name=$2
    local object_name=$3
    local response

    response=$(aws s3api get-object \
        --bucket "$bucket_name" \
        --key "$object_name" \
        "$destination_file_name")

    # shellcheck disable=SC2181
    if [[ ${?} -ne 0 ]]; then
        errecho "ERROR: AWS reports put-object operation failed.\n$response"
        return 1
    fi
}
```

- For API details, see [GetObject](#) in *AWS CLI Command Reference*.

List objects in a bucket

The following code example shows how to list objects in an S3 bucket.

AWS CLI with Bash script

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
#####
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
#####
function errecho() {
    printf "%s\n" "$*" 1>&2
}
```

```
#####
# function list_items_in_bucket
#
# This function displays a list of the files in the bucket with each file's
# size. The function uses the --query parameter to retrieve only the key and
# size fields from the Contents collection.
#
# Parameters:
#     $1 - The name of the bucket.
#
# Returns:
#     The list of files in text format.
#     And:
#         0 - If successful.
#         1 - If it fails.
#####
function list_items_in_bucket() {
    local bucket_name=$1
    local response

    response=$(aws s3api list-objects \
        --bucket "$bucket_name" \
        --output text \
        --query 'Contents[].{Key: Key, Size: Size}')

    # shellcheck disable=SC2181
    if [[ ${?} -eq 0 ]]; then
        echo "$response"
    else
        errecho "ERROR: AWS reports s3api list-objects operation failed.\n$response"
        return 1
    fi
}
```

- For API details, see [ListObjectsV2](#) in *AWS CLI Command Reference*.

Upload an object to a bucket

The following code example shows how to upload an object to an S3 bucket.

AWS CLI with Bash script

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
#####
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
#####
function errecho() {
    printf "%s\n" "$*" 1>&2
}

#####
# function copy_file_to_bucket
#
# This function creates a file in the specified bucket.
#
# Parameters:
#     $1 - The name of the bucket to copy the file to.
#     $2 - The path and file name of the local file to copy to the bucket.
#     $3 - The key (name) to call the copy of the file in the bucket.
#
# Returns:
#     0 - If successful.
#     1 - If it fails.
#####
function copy_file_to_bucket() {
    local response bucket_name source_file destination_file_name
    bucket_name=$1
    source_file=$2
    destination_file_name=$3

    response=$(aws s3api put-object \
        --bucket "$bucket_name" \
        --body "$source_file" \
        --key "$destination_file_name")
```

```
# shellcheck disable=SC2181
if [[ ${?} -ne 0 ]]; then
    errecho "ERROR: AWS reports put-object operation failed.\n$response"
    return 1
fi
}
```

- For API details, see [PutObject](#) in *AWS CLI Command Reference*.

Scenarios

Get started with buckets and objects

The following code example shows how to:

- Create a bucket and upload a file to it.
- Download an object from a bucket.
- Copy an object to a subfolder in a bucket.
- List the objects in a bucket.
- Delete the bucket objects and the bucket.

AWS CLI with Bash script

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
#####
# function s3_getting_started
#
# This function creates, copies, and deletes S3 buckets and objects.
#
# Returns:
#     0 - If successful.
#     1 - If an error occurred.
#####
```

```
function s3_getting_started() {
{
    if [ "$BUCKET_OPERATIONS_SOURCED" != "True" ]; then
        cd bucket-lifecycle-operations || exit

        source ./bucket_operations.sh
        cd ..
    fi
}

echo_repeat "*" 88
echo "Welcome to the Amazon S3 getting started demo."
echo_repeat "*" 88

local bucket_name
bucket_name=$(generate_random_name "doc-example-bucket")

local region_code
region_code=$(aws configure get region)

if create_bucket -b "$bucket_name" -r "$region_code"; then
    echo "Created demo bucket named $bucket_name"
else
    errecho "The bucket failed to create. This demo will exit."
    return 1
fi

local file_name
while [ -z "$file_name" ]; do
    echo -n "Enter a file you want to upload to your bucket: "
    get_input
    file_name=$get_input_result

    if [ ! -f "$file_name" ]; then
        echo "Could not find file $file_name. Are you sure it exists?"
        file_name=""
    fi
done

local key
key=$(basename "$file_name")

local result=0
if copy_file_to_bucket "$bucket_name" "$file_name" "$key"; then
```

```
echo "Uploaded file $file_name into bucket $bucket_name with key $key."
else
    result=1
fi

local destination_file
destination_file="$file_name.download"
if yes_no_input "Would you like to download $key to the file $destination_file?
(y/n) "; then
    if download_object_from_bucket "$bucket_name" "$destination_file" "$key"; then
        echo "Downloaded $key in the bucket $bucket_name to the file
$destination_file."
    else
        result=1
    fi
fi

if yes_no_input "Would you like to copy $key a new object key in your bucket? (y/
n) "; then
    local to_key
    to_key="demo/$key"
    if copy_item_in_bucket "$bucket_name" "$key" "$to_key"; then
        echo "Copied $key in the bucket $bucket_name to the $to_key."
    else
        result=1
    fi
fi

local bucket_items
bucket_items=$(list_items_in_bucket "$bucket_name")

# shellcheck disable=SC2181
if [[ $? -ne 0 ]]; then
    result=1
fi

echo "Your bucket contains the following items."
echo -e "Name\t\tSize"
echo "$bucket_items"

if yes_no_input "Delete the bucket, $bucket_name, as well as the objects in it?
(y/n) "; then
    bucket_items=$(echo "$bucket_items" | cut -f 1)
```

```
if delete_items_in_bucket "$bucket_name" "$bucket_items"; then
    echo "The following items were deleted from the bucket $bucket_name"
    echo "$bucket_items"
else
    result=1
fi

if delete_bucket "$bucket_name"; then
    echo "Deleted the bucket $bucket_name"
else
    result=1
fi
fi

return $result
}
```

The Amazon S3 functions used in this scenario.

```
#####
# function create-bucket
#
# This function creates the specified bucket in the specified AWS Region, unless
# it already exists.
#
# Parameters:
#     -b bucket_name -- The name of the bucket to create.
#     -r region_code -- The code for an AWS Region in which to
#                      create the bucket.
#
# Returns:
#     The URL of the bucket that was created.
#     And:
#         0 - If successful.
#         1 - If it fails.
#####
function create_bucket() {
    local bucket_name region_code response
    local option OPTARG # Required to use getopt command in a function.

    # bashsupport disable=BP5008
    function usage() {
```

```
echo "function create_bucket"
echo "Creates an Amazon S3 bucket. You must supply a bucket name:"
echo " -b bucket_name      The name of the bucket. It must be globally unique."
echo " [-r region_code]    The code for an AWS Region in which the bucket is
created."
echo ""
}

# Retrieve the calling parameters.
while getopts "b:r:h" option; do
  case "${option}" in
    b) bucket_name="${OPTARG}" ;;
    r) region_code="${OPTARG}" ;;
    h)
      usage
      return 0
      ;;
    \?) 
      echo "Invalid parameter"
      usage
      return 1
      ;;
  esac
done

if [[ -z "$bucket_name" ]]; then
  errecho "ERROR: You must provide a bucket name with the -b parameter."
  usage
  return 1
fi

local bucket_config_arg
# A location constraint for "us-east-1" returns an error.
if [[ -n "$region_code" ]] && [[ "$region_code" != "us-east-1" ]]; then
  bucket_config_arg="--create-bucket-configuration LocationConstraint=
$region_code"
fi

iecho "Parameters:\n"
iecho "  Bucket name:  $bucket_name"
iecho "  Region code:  $region_code"
iecho ""

# If the bucket already exists, we don't want to try to create it.
```

```
if (bucket_exists "$bucket_name"); then
    errecho "ERROR: A bucket with that name already exists. Try again."
    return 1
fi

# shellcheck disable=SC2086
response=$(aws s3api create-bucket \
    --bucket "$bucket_name" \
    $bucket_config_arg)

# shellcheck disable=SC2181
if [[ ${?} -ne 0 ]]; then
    errecho "ERROR: AWS reports create-bucket operation failed.\n$response"
    return 1
fi
}

#####
# function copy_file_to_bucket
#
# This function creates a file in the specified bucket.
#
# Parameters:
#     $1 - The name of the bucket to copy the file to.
#     $2 - The path and file name of the local file to copy to the bucket.
#     $3 - The key (name) to call the copy of the file in the bucket.
#
# Returns:
#     0 - If successful.
#     1 - If it fails.
#####
function copy_file_to_bucket() {
    local response bucket_name source_file destination_file_name
    bucket_name=$1
    source_file=$2
    destination_file_name=$3

    response=$(aws s3api put-object \
        --bucket "$bucket_name" \
        --body "$source_file" \
        --key "$destination_file_name")

    # shellcheck disable=SC2181
    if [[ ${?} -ne 0 ]]; then
```

```
errecho "ERROR: AWS reports put-object operation failed.\n$response"
return 1
fi
}

#####
# function download_object_from_bucket
#
# This function downloads an object in a bucket to a file.
#
# Parameters:
#   $1 - The name of the bucket to download the object from.
#   $2 - The path and file name to store the downloaded bucket.
#   $3 - The key (name) of the object in the bucket.
#
# Returns:
#   0 - If successful.
#   1 - If it fails.
#####
function download_object_from_bucket() {
    local bucket_name=$1
    local destination_file_name=$2
    local object_name=$3
    local response

    response=$(aws s3api get-object \
        --bucket "$bucket_name" \
        --key "$object_name" \
        "$destination_file_name")

    # shellcheck disable=SC2181
    if [[ ${?} -ne 0 ]]; then
        errecho "ERROR: AWS reports put-object operation failed.\n$response"
        return 1
    fi
}

#####
# function copy_item_in_bucket
#
# This function creates a copy of the specified file in the same bucket.
#
# Parameters:
#   $1 - The name of the bucket to copy the file from and to.
```

```
#      $2 - The key of the source file to copy.
#      $3 - The key of the destination file.
#
# Returns:
#      0 - If successful.
#      1 - If it fails.
#####
function copy_item_in_bucket() {
    local bucket_name=$1
    local source_key=$2
    local destination_key=$3
    local response

    response=$(aws s3api copy-object \
        --bucket "$bucket_name" \
        --copy-source "$bucket_name/$source_key" \
        --key "$destination_key")

    # shellcheck disable=SC2181
    if [[ $? -ne 0 ]]; then
        errecho "ERROR: AWS reports s3api copy-object operation failed.\n$response"
        return 1
    fi
}

#####
# function list_items_in_bucket
#
# This function displays a list of the files in the bucket with each file's
# size. The function uses the --query parameter to retrieve only the key and
# size fields from the Contents collection.
#
# Parameters:
#      $1 - The name of the bucket.
#
# Returns:
#      The list of files in text format.
#      And:
#          0 - If successful.
#          1 - If it fails.
#####
function list_items_in_bucket() {
    local bucket_name=$1
    local response
```

```
response=$(aws s3api list-objects \
--bucket "$bucket_name" \
--output text \
--query 'Contents[].{Key: Key, Size: Size}')

# shellcheck disable=SC2181
if [[ ${?} -eq 0 ]]; then
    echo "$response"
else
    errecho "ERROR: AWS reports s3api list-objects operation failed.\n$response"
    return 1
fi
}

#####
# function delete_items_in_bucket
#
# This function deletes the specified list of keys from the specified bucket.
#
# Parameters:
#     $1 - The name of the bucket.
#     $2 - A list of keys in the bucket to delete.

# Returns:
#     0 - If successful.
#     1 - If it fails.
#####
function delete_items_in_bucket() {
    local bucket_name=$1
    local keys=$2
    local response

    # Create the JSON for the items to delete.
    local delete_items
    delete_items="{"Objects":["
for key in $keys; do
    delete_items="$delete_items{"Key": \"$key\"},"
done
    delete_items=${delete_items%?} # Remove the final comma.
    delete_items="$delete_items"]}"

    response=$(aws s3api delete-objects \
--bucket "$bucket_name" \
```

```
--delete "$delete_items")\n\n# shellcheck disable=SC2181\nif [[ $? -ne 0 ]]; then\n    errecho "ERROR: AWS reports s3api delete-object operation failed.\n$response"\n    return 1\nfi\n}\n\n#####\n# function delete_bucket\n#\n# This function deletes the specified bucket.\n#\n# Parameters:\n#     $1 - The name of the bucket.\n\n# Returns:\n#     0 - If successful.\n#     1 - If it fails.\n#####\nfunction delete_bucket() {\n    local bucket_name=$1\n    local response\n\n    response=$(aws s3api delete-bucket \\ \n        --bucket "$bucket_name")\n\n    # shellcheck disable=SC2181\n    if [[ $? -ne 0 ]]; then\n        errecho "ERROR: AWS reports s3api delete-bucket failed.\n$response"\n        return 1\n    fi\n}
```

- For API details, see the following topics in *AWS CLI Command Reference*.

- [CopyObject](#)
- [CreateBucket](#)
- [DeleteBucket](#)
- [DeleteObjects](#)

- [GetObject](#)
- [ListObjectsV2](#)
- [PutObject](#)

AWS STS examples using AWS CLI with Bash script

The following code examples show you how to perform actions and implement common scenarios by using the AWS Command Line Interface with Bash script with AWS STS.

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios and cross-service examples.

Scenarios are code examples that show you how to accomplish a specific task by calling multiple functions within the same service.

Each example includes a link to GitHub, where you can find instructions on how to set up and run the code in context.

Topics

- [Actions](#)

Actions

Assume a role

The following code example shows how to assume a role with AWS STS.

AWS CLI with Bash script

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
#####
# function iecho
#
```

```
# This function enables the script to display the specified text only if
# the global variable $VERBOSE is set to true.
#####
function iecho() {
    if [[ $VERBOSE == true ]]; then
        echo "$@"
    fi
}

#####
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
#####
function errecho() {
    printf "%s\n" "$*" 1>&2
}

#####
# function sts_assume_role
#
# This function assumes a role in the AWS account and returns the temporary
# credentials.
#
# Parameters:
#     -n role_session_name -- The name of the session.
#     -r role_arn -- The ARN of the role to assume.
#
# Returns:
#     [access_key_id, secret_access_key, session_token]
#     And:
#         0 - If successful.
#         1 - If an error occurred.
#####
function sts_assume_role() {
    local role_session_name role_arn response
    local option OPTARG # Required to use getopts command in a function.

    # bashsupport disable=BP5008
    function usage() {
        echo "function sts_assume_role"
        echo "Assumes a role in the AWS account and returns the temporary credentials:"
        echo "  -n role_session_name -- The name of the session."
        echo "  -r role_arn -- The ARN of the role to assume."
    }
}
```

```
echo ""
}

while getopts n:r:h option; do
  case "${option}" in
    n) role_session_name=${OPTARG} ;;
    r) role_arn=${OPTARG} ;;
    h)
      usage
      return 0
      ;;
    \?) 
      echo "Invalid parameter"
      usage
      return 1
      ;;
  esac
done

response=$(aws sts assume-role \
--role-session-name "$role_session_name" \
--role-arn "$role_arn" \
--output text \
--query "Credentials.[AccessKeyId, SecretAccessKey, SessionToken]")

local error_code=${?}

if [[ $error_code -ne 0 ]]; then
  aws_cli_error_log $error_code
  errecho "ERROR: AWS reports create-role operation failed.\n$response"
  return 1
fi

echo "$response"

return 0
}
```

- For API details, see [AssumeRole](#) in *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](#)
- [Identity and Access Management](#)
- [Compliance Validation for this AWS Product or Service](#)
- [Resilience for this AWS Product or Service](#)
- [Infrastructure Security for this AWS Product or Service](#)
- [Enforce a minimum version of TLS](#)

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. You are also responsible for 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 users with AWS IAM Identity Center or 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 require TLS 1.2 and recommend TLS 1.3.
- 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 sensitive 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 text 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 text 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

AWS Identity and Access Management (IAM) is an AWS service that helps an administrator securely control access to AWS resources. IAM administrators control who can be *authenticated* (signed in) and *authorized* (have permissions) to use AWS resources. IAM is an AWS service that you can use with no additional charge.

Topics

- [Audience](#)
- [Authenticating with identities](#)
- [Managing access using policies](#)
- [How AWS services work with IAM](#)
- [Troubleshooting AWS identity and access](#)

Audience

How you use AWS Identity and Access Management (IAM) differs, depending on the work that you do in AWS.

Service user – If you use AWS services to do your job, then your administrator provides you with the credentials and permissions that you need. As you use more AWS features to do your work, you might need additional permissions. Understanding how access is managed can help you request the right permissions from your administrator. If you cannot access a feature in AWS, see [Troubleshooting AWS identity and access](#) or the user guide of the AWS service you are using.

Service administrator – If you're in charge of AWS resources at your company, you probably have full access to AWS. It's your job to determine which AWS features and resources your service users should access. You must then submit requests to your IAM administrator to change the permissions of your service users. Review the information on this page to understand the basic concepts of IAM. To learn more about how your company can use IAM with AWS, see the user guide of the AWS service you are using.

IAM administrator – If you're an IAM administrator, you might want to learn details about how you can write policies to manage access to AWS. To view example AWS identity-based policies that you can use in IAM, see the user guide of the AWS service you are using.

Authenticating with identities

Authentication is how you sign in to AWS using your identity credentials. You must be *authenticated* (signed in to AWS) as the AWS account root user, as an IAM user, or by assuming an IAM role.

You can sign in to AWS as a federated identity by using credentials provided through an identity source. AWS IAM Identity Center (IAM Identity Center) users, your company's single sign-on authentication, and your Google or Facebook credentials are examples of federated identities. When you sign in as a federated identity, your administrator previously set up identity federation using IAM roles. When you access AWS by using federation, you are indirectly assuming a role.

Depending on the type of user you are, you can sign in to the AWS Management Console or the AWS access portal. For more information about signing in to AWS, see [How to sign in to your AWS account](#) in the *AWS Sign-In User Guide*.

If you access AWS programmatically, AWS provides a software development kit (SDK) and a command line interface (CLI) to cryptographically sign your requests by using your credentials. If you don't use AWS tools, you must sign requests yourself. For more information about using the recommended method to sign requests yourself, see [Signing AWS API requests](#) in the *IAM User Guide*.

Regardless of the authentication method that you use, you might be required to provide additional security information. For example, AWS recommends that you use multi-factor authentication (MFA) to increase the security of your account. To learn more, see [Multi-factor authentication](#) in the *AWS IAM Identity Center User Guide* and [Using multi-factor authentication \(MFA\) in AWS](#) in the *IAM User Guide*.

AWS account root user

When you create an AWS account, you begin with one sign-in identity that has complete access to all AWS services and resources in the account. This identity is called the AWS account *root user* and is accessed by signing in with the email address and password that you used to create the account. We strongly recommend that you don't use the root user for your everyday tasks. Safeguard your root user credentials and use them to perform the tasks that only the root user can perform. For the complete list of tasks that require you to sign in as the root user, see [Tasks that require root user credentials](#) in the *IAM User Guide*.

Federated identity

As a best practice, require human users, including users that require administrator access, to use federation with an identity provider to access AWS services by using temporary credentials.

A *federated identity* is a user from your enterprise user directory, a web identity provider, the AWS Directory Service, the Identity Center directory, or any user that accesses AWS services by using credentials provided through an identity source. When federated identities access AWS accounts, they assume roles, and the roles provide temporary credentials.

For centralized access management, we recommend that you use AWS IAM Identity Center. You can create users and groups in IAM Identity Center, or you can connect and synchronize to a set of users and groups in your own identity source for use across all your AWS accounts and applications. For information about IAM Identity Center, see [What is IAM Identity Center?](#) in the *AWS IAM Identity Center User Guide*.

IAM users and groups

An [*IAM user*](#) is an identity within your AWS account that has specific permissions for a single person or application. Where possible, we recommend relying on temporary credentials instead of creating IAM users who have long-term credentials such as passwords and access keys. However, if you have specific use cases that require long-term credentials with IAM users, we recommend that you rotate access keys. For more information, see [Rotate access keys regularly for use cases that require long-term credentials](#) in the *IAM User Guide*.

An [*IAM group*](#) is an identity that specifies a collection of IAM users. You can't sign in as a group. You can use groups to specify permissions for multiple users at a time. Groups make permissions easier to manage for large sets of users. For example, you could have a group named *IAMAAdmins* and give that group permissions to administer IAM resources.

Users are different from roles. A user is uniquely associated with one person or application, but a role is intended to be assumable by anyone who needs it. Users have permanent long-term credentials, but roles provide temporary credentials. To learn more, see [When to create an IAM user \(instead of a role\)](#) in the *IAM User Guide*.

IAM roles

An [IAM role](#) is an identity within your AWS account that has specific permissions. It is similar to an IAM user, but is not associated with a specific person. You can temporarily assume an IAM role in the AWS Management Console by [switching roles](#). You can assume a role by calling an AWS CLI or AWS API operation or by using a custom URL. For more information about methods for using roles, see [Using IAM roles](#) in the *IAM User Guide*.

IAM roles with temporary credentials are useful in the following situations:

- **Federated user access** – To assign permissions to a federated identity, you create a role and define permissions for the role. When a federated identity authenticates, the identity is associated with the role and is granted the permissions that are defined by the role. For information about roles for federation, see [Creating a role for a third-party Identity Provider](#) in the *IAM User Guide*. If you use IAM Identity Center, you configure a permission set. To control what your identities can access after they authenticate, IAM Identity Center correlates the permission set to a role in IAM. For information about permissions sets, see [Permission sets](#) in the *AWS IAM Identity Center User Guide*.
- **Temporary IAM user permissions** – An IAM user or role can assume an IAM role to temporarily take on different permissions for a specific task.
- **Cross-account access** – You can use an IAM role to allow someone (a trusted principal) in a different account to access resources in your account. Roles are the primary way to grant cross-account access. However, with some AWS services, you can attach a policy directly to a resource (instead of using a role as a proxy). To learn the difference between roles and resource-based policies for cross-account access, see [How IAM roles differ from resource-based policies](#) in the *IAM User Guide*.
- **Cross-service access** – Some AWS services use features in other AWS services. For example, when you make a call in a service, it's common for that service to run applications in Amazon EC2 or store objects in Amazon S3. A service might do this using the calling principal's permissions, using a service role, or using a service-linked role.
 - **Forward access sessions (FAS)** – When you use an IAM user or role to perform actions in AWS, you are considered a principal. When you use some services, you might perform an

action that then initiates another action in a different service. FAS uses the permissions of the principal calling an AWS service, combined with the requesting AWS service to make requests to downstream services. FAS requests are only made when a service receives a request that requires interactions with other AWS services or resources to complete. In this case, you must have permissions to perform both actions. For policy details when making FAS requests, see [Forward access sessions](#).

- **Service role** – A service role is an [IAM role](#) that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see [Creating a role to delegate permissions to an AWS service](#) in the *IAM User Guide*.
- **Service-linked role** – A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your AWS account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.
- **Applications running on Amazon EC2** – You can use an IAM role to manage temporary credentials for applications that are running on an EC2 instance and making AWS CLI or AWS API requests. This is preferable to storing access keys within the EC2 instance. To assign an AWS role to an EC2 instance and make it available to all of its applications, you create an instance profile that is attached to the instance. An instance profile contains the role and enables programs that are running on the EC2 instance to get temporary credentials. For more information, see [Using an IAM role to grant permissions to applications running on Amazon EC2 instances](#) in the *IAM User Guide*.

To learn whether to use IAM roles or IAM users, see [When to create an IAM role \(instead of a user\)](#) in the *IAM User Guide*.

Managing access using policies

You control access in AWS by creating policies and attaching them to AWS identities or resources. A policy is an object in AWS that, when associated with an identity or resource, defines their permissions. AWS evaluates these policies when a principal (user, root user, or role session) makes a request. Permissions in the policies determine whether the request is allowed or denied. Most policies are stored in AWS as JSON documents. For more information about the structure and contents of JSON policy documents, see [Overview of JSON policies](#) in the *IAM User Guide*.

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

By default, users and roles have no permissions. To grant users permission to perform actions on the resources that they need, an IAM administrator can create IAM policies. The administrator can then add the IAM policies to roles, and users can assume the roles.

IAM policies define permissions for an action regardless of the method that you use to perform the operation. For example, suppose that you have a policy that allows the `iam:GetRole` action. A user with that policy can get role information from the AWS Management Console, the AWS CLI, or the AWS API.

Identity-based policies

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see [Creating IAM policies](#) in the *IAM User Guide*.

Identity-based policies can be further categorized as *inline policies* or *managed policies*. Inline policies are embedded directly into a single user, group, or role. Managed policies are standalone policies that you can attach to multiple users, groups, and roles in your AWS account. Managed policies include AWS managed policies and customer managed policies. To learn how to choose between a managed policy or an inline policy, see [Choosing between managed policies and inline policies](#) in the *IAM User Guide*.

Resource-based policies

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must [specify a principal](#) in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.

Resource-based policies are inline policies that are located in that service. You can't use AWS managed policies from IAM in a resource-based policy.

Access control lists (ACLs)

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

Amazon S3, AWS WAF, and Amazon VPC are examples of services that support ACLs. To learn more about ACLs, see [Access control list \(ACL\) overview](#) in the *Amazon Simple Storage Service Developer Guide*.

Other policy types

AWS supports additional, less-common policy types. These policy types can set the maximum permissions granted to you by the more common policy types.

- **Permissions boundaries** – A permissions boundary is an advanced feature in which you set the maximum permissions that an identity-based policy can grant to an IAM entity (IAM user or role). You can set a permissions boundary for an entity. The resulting permissions are the intersection of an entity's identity-based policies and its permissions boundaries. Resource-based policies that specify the user or role in the Principal field are not limited by the permissions boundary. An explicit deny in any of these policies overrides the allow. For more information about permissions boundaries, see [Permissions boundaries for IAM entities](#) in the *IAM User Guide*.
- **Service control policies (SCPs)** – SCPs are JSON policies that specify the maximum permissions for an organization or organizational unit (OU) in AWS Organizations. AWS Organizations is a service for grouping and centrally managing multiple AWS accounts that your business owns. If you enable all features in an organization, then you can apply service control policies (SCPs) to any or all of your accounts. The SCP limits permissions for entities in member accounts, including each AWS account root user. For more information about Organizations and SCPs, see [How SCPs work](#) in the *AWS Organizations User Guide*.
- **Session policies** – Session policies are advanced policies that you pass as a parameter when you programmatically create a temporary session for a role or federated user. The resulting session's permissions are the intersection of the user or role's identity-based policies and the session policies. Permissions can also come from a resource-based policy. An explicit deny in any of these policies overrides the allow. For more information, see [Session policies](#) in the *IAM User Guide*.

Multiple policy types

When multiple types of policies apply to a request, the resulting permissions are more complicated to understand. To learn how AWS determines whether to allow a request when multiple policy types are involved, see [Policy evaluation logic](#) in the *IAM User Guide*.

How AWS services work with IAM

To get a high-level view of how AWS services work with most IAM features, see [AWS services that work with IAM](#) in the *IAM User Guide*.

To learn how to use a specific AWS service with IAM, see the security section of the relevant service's User Guide.

Troubleshooting AWS identity and access

Use the following information to help you diagnose and fix common issues that you might encounter when working with AWS and IAM.

Topics

- [I am not authorized to perform an action in AWS](#)
- [I am not authorized to perform iam:PassRole](#)
- [I want to allow people outside of my AWS account to access my AWS resources](#)

I am not authorized to perform an action in AWS

If you receive an error that you're not authorized to perform an action, your policies must be updated to allow you to perform the action.

The following example error occurs when the mateojackson IAM user tries to use the console to view details about a fictional *my-example-widget* resource but doesn't have the fictional awes:*GetWidget* permissions.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:  
awes:GetWidget on resource: my-example-widget
```

In this case, the policy for the mateojackson user must be updated to allow access to the *my-example-widget* resource by using the awes:*GetWidget* action.

If you need help, contact your AWS administrator. Your administrator is the person who provided you with your sign-in credentials.

I am not authorized to perform iam:PassRole

If you receive an error that you're not authorized to perform the `iam:PassRole` action, your policies must be updated to allow you to pass a role to AWS.

Some AWS services allow you to pass an existing role to that service instead of creating a new service role or service-linked role. To do this, you must have permissions to pass the role to the service.

The following example error occurs when an IAM user named `marymajor` tries to use the console to perform an action in AWS. However, the action requires the service to have permissions that are granted by a service role. Mary does not have permissions to pass the role to the service.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:  
    iam:PassRole
```

In this case, Mary's policies must be updated to allow her to perform the `iam:PassRole` action.

If you need help, contact your AWS administrator. Your administrator is the person who provided you with your sign-in credentials.

I want to allow people outside of my AWS account to access my AWS resources

You can create a role that users in other accounts or people outside of your organization can use to access your resources. You can specify who is trusted to assume the role. For services that support resource-based policies or access control lists (ACLs), you can use those policies to grant people access to your resources.

To learn more, consult the following:

- To learn whether AWS supports these features, see [How AWS services work with IAM](#).
- To learn how to provide access to your resources across AWS accounts that you own, see [Providing access to an IAM user in another AWS account that you own](#) in the *IAM User Guide*.
- To learn how to provide access to your resources to third-party AWS accounts, see [Providing access to AWS accounts owned by third parties](#) in the *IAM User Guide*.
- To learn how to provide access through identity federation, see [Providing access to externally authenticated users \(identity federation\)](#) in the *IAM User Guide*.

- To learn the difference between using roles and resource-based policies for cross-account access, see [How IAM roles differ from resource-based policies](#) in the *IAM User Guide*.

Compliance Validation for this AWS Product or Service

To learn whether an AWS service is within the scope of specific compliance programs, see [AWS services in Scope by Compliance Program](#) and choose the compliance program that you are interested in. For general information, see [AWS Compliance Programs](#).

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

Your compliance responsibility when using AWS services 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 baseline environments on AWS that are security and compliance focused.
- [Architecting for HIPAA Security and Compliance on Amazon Web Services](#) – This whitepaper describes how companies can use AWS to create HIPAA-eligible applications.

 **Note**

Not all AWS services are HIPAA eligible. For more information, see the [HIPAA Eligible Services Reference](#).

- [AWS Compliance Resources](#) – This collection of workbooks and guides might apply to your industry and location.
- [AWS Customer Compliance Guides](#) – Understand the shared responsibility model through the lens of compliance. The guides summarize the best practices for securing AWS services and map the guidance to security controls across multiple frameworks (including National Institute of Standards and Technology (NIST), Payment Card Industry Security Standards Council (PCI), and International Organization for Standardization (ISO)).
- [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. Security Hub uses security controls to evaluate your AWS resources and to check your compliance against security industry standards and best practices. For a list of supported services and controls, see [Security Hub controls reference](#).
- [AWS Audit Manager](#) – This AWS service helps you continuously audit your AWS usage to simplify how you manage risk and compliance with regulations and industry standards.

This AWS product or service follows the [shared responsibility model](#) through the specific Amazon Web Services (AWS) services it supports. For AWS service security information, see the [AWS service security documentation page](#) and [AWS services that are in scope of AWS compliance efforts by compliance program](#).

Resilience for this AWS Product or Service

The AWS global infrastructure is built around AWS Regions and Availability Zones.

AWS Regions provide multiple physically separated and isolated Availability Zones, which are connected with low-latency, high-throughput, and highly redundant networking.

With Availability Zones, you can design and operate applications and databases that automatically fail over between zones without interruption. Availability Zones are more highly available, fault tolerant, and scalable than traditional single or multiple data center infrastructures.

For more information about AWS Regions and Availability Zones, see [AWS Global Infrastructure](#).

This AWS product or service follows the [shared responsibility model](#) through the specific Amazon Web Services (AWS) services it supports. For AWS service security information, see the [AWS service security documentation page](#) and [AWS services that are in scope of AWS compliance efforts by compliance program](#).

Infrastructure Security for this AWS Product or Service

This AWS product or service uses managed services, and therefore is protected by the AWS global network security. For information about AWS security services and how AWS protects infrastructure, see [AWS Cloud Security](#). To design your AWS environment using the best practices for infrastructure security, see [Infrastructure Protection in Security Pillar AWS Well-Architected Framework](#).

You use AWS published API calls to access this AWS Product or Service through the network. Clients must support the following:

- Transport Layer Security (TLS). We require TLS 1.2 and recommend TLS 1.3.
- Cipher suites with perfect forward secrecy (PFS) such as DHE (Ephemeral Diffie-Hellman) or ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). Most modern systems such as Java 7 and later support these modes.

Additionally, requests must be signed by using an access key ID and a secret access key that is associated with an IAM principal. Or you can use the [AWS Security Token Service](#) (AWS STS) to generate temporary security credentials to sign requests.

This AWS product or service follows the [shared responsibility model](#) through the specific Amazon Web Services (AWS) services it supports. For AWS service security information, see the [AWS service security documentation page](#) and [AWS services that are in scope of AWS compliance efforts by compliance program](#).

Enforce 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.

AWS CLI version 2 uses an internal Python script that's compiled to use a minimum of TLS 1.2 when the service it's talking to supports it. As long as you use version 2 of the AWS CLI, no further steps are needed to enforce this minimum.

Troubleshoot AWS CLI errors

This section covers common errors and troubleshooting steps to follow to resolve your issue. We suggest following the [general troubleshooting](#) first.

Contents

- [General troubleshooting to try first](#)
 - [Check your AWS CLI command formatting](#)
 - [Confirm that you're running a recent version of the AWS CLI](#)
 - [Use the --debug option](#)
 - [Enable and review the AWS CLI command history logs](#)
 - [Confirm that your AWS CLI is configured](#)
- [Command not found errors](#)
- [The "aws --version" command returns a different version than you installed](#)
- [The "aws --version" command returns a version after uninstalling the AWS CLI](#)
- [The AWS CLI processed a command with an incomplete parameter name](#)
- [Access denied errors](#)
- [Invalid credentials and key errors](#)
- [Signature does not match errors](#)
- [SSL certificate errors](#)
- [Invalid JSON errors](#)
- [Additional resources](#)

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.

[Back to top](#)

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](#) is correct in your command.
- Generate an [AWS CLI skeleton](#) to confirm your command structure.
- For JSON, see the additional [troubleshooting for JSON values](#). 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](#).

For more information on how a specific command should be structured, see the [AWS CLI version 2 reference guide](#).

[Back to top](#)

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 version 2 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 [the section called "Install/Update"](#).

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](#)

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.

```
$ 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
2019-08-12 12:36:18,305 - MainThread - awscli.clidriver - DEBUG - CLI version: aws-
cli/1.16.215 Python/3.7.3 Linux/4.14.133-113.105.amzn2.x86_64 botocore/1.12.205
2019-08-12 12:36:18,305 - MainThread - awscli.clidriver - DEBUG - Arguments entered to
CLI: ['iam', 'list-groups', '--debug']
```

```
2019-08-12 12:36:18,305 - MainThread - botocore.hooks - DEBUG - Event session-  
initialized: calling handler <function add_scalar_parsers at 0x7fdf173161e0>  
2019-08-12 12:36:18,305 - MainThread - botocore.hooks - DEBUG - Event session-  
initialized: calling handler <function register_uri_param_handler at 0x7fdf17dec400>  
2019-08-12 12:36:18,305 - MainThread - botocore.hooks - DEBUG - Event session-  
initialized: calling handler <function inject_assume_role_provider_cache at  
0x7fdf17da9378>  
2019-08-12 12:36:18,307 - MainThread - botocore.credentials - DEBUG - Skipping  
environment variable credential check because profile name was explicitly set.  
2019-08-12 12:36:18,307 - MainThread - botocore.hooks - DEBUG - Event session-  
initialized: calling handler <function attach_history_handler at 0x7fdf173ed9d8>  
2019-08-12 12:36:18,308 - MainThread - botocore.loaders - DEBUG - Loading JSON  
file: /home/ec2-user/venv/lib/python3.7/site-packages/botocore/data/iam/2010-05-08/  
service-2.json  
2019-08-12 12:36:18,317 - MainThread - botocore.hooks - DEBUG - Event building-command-  
table.iam: calling handler <function add_waiters at 0x7fdf1731a840>  
2019-08-12 12:36:18,320 - MainThread - botocore.loaders - DEBUG - Loading JSON  
file: /home/ec2-user/venv/lib/python3.7/site-packages/botocore/data/iam/2010-05-08/  
waiters-2.json  
2019-08-12 12:36:18,321 - MainThread - awscli.clidriver - DEBUG - OrderedDict([('path-  
prefix', <awscli.arguments.CLIAArgument object at 0x7fdf171ac780>), ('marker',  
<awscli.arguments.CLIAArgument object at 0x7fdf171b09e8>), ('max-items',  
<awscli.arguments.CLIAArgument object at 0x7fdf171b09b0>)])  
2019-08-12 12:36:18,322 - MainThread - botocore.hooks - DEBUG - Event building-  
argument-table.iam.list-groups: calling handler <function add_streaming_output_arg at  
0x7fdf17316510>  
2019-08-12 12:36:18,322 - MainThread - botocore.hooks - DEBUG - Event building-  
argument-table.iam.list-groups: calling handler <function add_cli_input_json at  
0x7fdf17da9d90>  
2019-08-12 12:36:18,322 - MainThread - botocore.hooks - DEBUG - Event building-  
argument-table.iam.list-groups: calling handler <function unify.paging_params at  
0x7fdf17328048>  
2019-08-12 12:36:18,326 - MainThread - botocore.loaders - DEBUG - Loading JSON  
file: /home/ec2-user/venv/lib/python3.7/site-packages/botocore/data/iam/2010-05-08/  
paginators-1.json  
2019-08-12 12:36:18,326 - MainThread - awscli.customizations.paginate - DEBUG -  
Modifying paging parameters for operation: ListGroups  
2019-08-12 12:36:18,326 - MainThread - botocore.hooks - DEBUG - Event building-  
argument-table.iam.list-groups: calling handler <function add_generate_skeleton at  
0x7fdf1737eae8>  
2019-08-12 12:36:18,326 - MainThread - botocore.hooks - DEBUG - Event  
before-building-argument-table-parser.iam.list-groups: calling handler  
<bound method OverrideRequiredArgsArgument.override_required_args of  
<awscli.customizations.cliinputjson.CliInputJSONArgument object at 0x7fdf171b0a58>>
```

```
2019-08-12 12:36:18,327 - MainThread - botocore.hooks - DEBUG - Event
before-building-argument-table-parser.iam.list-groups: calling handler
<bound method GenerateCliSkeletonArgument.override_required_args of
<awscli.customizations.generatecliskeleton.GenerateCliSkeletonArgument object at
0x7fdf171c5978>>
2019-08-12 12:36:18,327 - MainThread - botocore.hooks - DEBUG - Event operation-
args-parsed.iam.list-groups: calling handler functools.partial(<function
check_should_enable_pagination at 0x7fdf17328158>, ['marker', 'max-items'], {'max-
items': <awscli.arguments.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.cliinputjson.CliInputJSONArgument object at
0x7fdf171b0a58>), ('starting-token', <awscli.customizations.paginate.PageArgument
object at 0x7fdf171b0a20>), ('page-size', <awscli.customizations.paginate.PageArgument
object at 0x7fdf171c5828>), ('generate-cli-skeleton',
<awscli.customizations.generatecliskeleton.GenerateCliSkeletonArgument object at
0x7fdf171c5978>)]))
2019-08-12 12:36:18,328 - MainThread - botocore.hooks - DEBUG - Event load-cli-
arg.iam.list-groups.path-prefix: calling handler <awscli.paramfile.URIArgumentHandler
object at 0x7fdf1725c978>
2019-08-12 12:36:18,328 - MainThread - botocore.hooks - DEBUG - Event load-cli-
arg.iam.list-groups.marker: calling handler <awscli.paramfile.URIArgumentHandler object
at 0x7fdf1725c978>
2019-08-12 12:36:18,328 - MainThread - botocore.hooks - DEBUG - Event load-cli-
arg.iam.list-groups.max-items: calling handler <awscli.paramfile.URIArgumentHandler
object at 0x7fdf1725c978>
2019-08-12 12:36:18,328 - MainThread - botocore.hooks - DEBUG -
Event load-cli-arg.iam.list-groups.cli-input-json: calling handler
<awscli.paramfile.URIArgumentHandler object at 0x7fdf1725c978>
2019-08-12 12:36:18,328 - MainThread - botocore.hooks - DEBUG -
Event load-cli-arg.iam.list-groups.starting-token: calling handler
<awscli.paramfile.URIArgumentHandler object at 0x7fdf1725c978>
2019-08-12 12:36:18,328 - MainThread - botocore.hooks - DEBUG - Event load-cli-
arg.iam.list-groups.page-size: calling handler <awscli.paramfile.URIArgumentHandler
object at 0x7fdf1725c978>
2019-08-12 12:36:18,328 - MainThread - botocore.hooks - DEBUG - Event
load-cli-arg.iam.list-groups.generate-cli-skeleton: calling handler
<awscli.paramfile.URIArgumentHandler object at 0x7fdf1725c978>
2019-08-12 12:36:18,329 - MainThread - botocore.hooks - DEBUG
- Event calling-command.iam.list-groups: calling handler
<bound method CliInputJSONArgument.add_to_call_parameters of
<awscli.customizations.cliinputjson.CliInputJSONArgument object at 0x7fdf171b0a58>>
```

```
2019-08-12 12:36:18,329 - MainThread - botocore.hooks - DEBUG -  
Event calling-command.iam.list-groups: calling handler <bound  
method GenerateCliSkeletonArgument.generate_json_skeleton of  
<awscli.customizations.generatecliskeleton.GenerateCliSkeletonArgument object at  
0x7fdf171c5978>>  
2019-08-12 12:36:18,329 - MainThread - botocore.credentials - DEBUG - Looking for  
credentials via: assume-role  
2019-08-12 12:36:18,329 - MainThread - botocore.credentials - DEBUG - Looking for  
credentials via: assume-role-with-web-identity  
2019-08-12 12:36:18,329 - MainThread - botocore.credentials - DEBUG - Looking for  
credentials via: shared-credentials-file  
2019-08-12 12:36:18,329 - MainThread - botocore.credentials - INFO - Found credentials  
in shared credentials file: ~/.aws/credentials  
2019-08-12 12:36:18,330 - MainThread - botocore.loaders - DEBUG - Loading JSON file: /  
home/ec2-user/venv/lib/python3.7/site-packages/botocore/data/endpoints.json  
2019-08-12 12:36:18,334 - MainThread - botocore.hooks - DEBUG - Event choose-service-  
name: calling handler <function handle_service_name_alias at 0x7fdf1898eb70>  
2019-08-12 12:36:18,337 - MainThread - botocore.hooks - DEBUG - Event creating-client-  
class.iam: calling handler <function add_generate_presigned_url at 0x7fdf18a028c8>  
2019-08-12 12:36:18,337 - MainThread - botocore.regions - DEBUG - Using partition  
endpoint for iam, us-west-2: aws-global  
2019-08-12 12:36:18,337 - MainThread - botocore.args - DEBUG - The s3 config key is not  
a dictionary type, ignoring its value of: None  
2019-08-12 12:36:18,340 - MainThread - botocore.endpoint - DEBUG - Setting iam timeout  
as (60, 60)  
2019-08-12 12:36:18,341 - MainThread - botocore.loaders - DEBUG - Loading JSON file: /  
home/ec2-user/venv/lib/python3.7/site-packages/botocore/data/_retry.json  
2019-08-12 12:36:18,341 - MainThread - botocore.client - DEBUG - Registering retry  
handlers for service: iam  
2019-08-12 12:36:18,342 - MainThread - botocore.hooks - DEBUG - Event before-  
parameter-build.iam.ListGroups: calling handler <function generate_idempotent_uuid at  
0x7fdf189b10d0>  
2019-08-12 12:36:18,342 - MainThread - botocore.hooks - DEBUG - Event before-  
call.iam.ListGroups: calling handler <function inject_api_version_header_if_needed at  
0x7fdf189b2a60>  
2019-08-12 12:36:18,343 - MainThread - botocore.endpoint - DEBUG - Making  
request for OperationModel(name=ListGroups) with params: {'url_path': '/',  
'query_string': '', 'method': 'POST', 'headers': {'Content-Type': 'application/x-  
www-form-urlencoded; charset=utf-8', 'User-Agent': 'aws-cli/1.16.215 Python/3.7.3  
Linux/4.14.133-113.105.amzn2.x86_64 botocore/1.12.205'}, 'body': {'Action':  
'ListGroups', 'Version': '2010-05-08'}, 'url': 'https://iam.amazonaws.com/',  
'context': {'client_region': 'aws-global', 'client_config': <botocore.config.Config  
object at 0x7fdf16e9a4a8>, 'has_streaming_input': False, 'auth_type': None}}}
```

```
2019-08-12 12:36:18,343 - MainThread - botocore.hooks - DEBUG - Event request-created.iam.ListGroups: calling handler <bound method RequestSigner.handler of <botocore.singers.RequestSigner object at 0x7fdf16e9a470>>
2019-08-12 12:36:18,343 - MainThread - botocore.hooks - DEBUG - Event choose-signer.iam.ListGroups: calling handler <function set_operation_specific_signer at 0x7fdf18996f28>
2019-08-12 12:36:18,343 - MainThread - botocore.auth - DEBUG - Calculating signature using v4 auth.
2019-08-12 12:36:18,343 - MainThread - botocore.auth - DEBUG - CanonicalRequest: POST
/
content-type:application/x-www-form-urlencoded; charset=utf-8
host:iam.amazonaws.com
x-amz-date:20190812T193618Z

content-type;host;x-amz-date
5f776d91EXAMPLE9b8cb5eb5d6d4a787a33ae41c8cd6eEXAMPLEca69080e1e1f
2019-08-12 12:36:18,344 - MainThread - botocore.auth - DEBUG - StringToSign: AWS4-HMAC-SHA256
20190812T193618Z
20190812/us-east-1/iam/aws4_request
ab7e367eEXAMPLE2769f178ea509978cf8bfa054874b3EXAMPLE8d043fab6cc9
2019-08-12 12:36:18,344 - MainThread - botocore.auth - DEBUG - Signature: d85a0EXAMPLEb40164f2f539cdc76d4f294fe822EXAMPLE18ad1ddf58a1a3ce7
2019-08-12 12:36:18,344 - MainThread - botocore.endpoint - DEBUG - Sending http request: <AWSPreparedRequest stream_output=False, method=POST, url=https://iam.amazonaws.com/, headers={'Content-Type': b'application/x-www-form-urlencoded; charset=utf-8', 'User-Agent': b'aws-cli/1.16.215 Python/3.7.3 Linux/4.14.133-113.105.amzn2.x86_64 botocore/1.12.205', 'X-Amz-Date': b'20190812T193618Z', 'Authorization': b'AWS4-HMAC-SHA256 Credential=AKIA01234567890EXAMPLE-east-1/iam/aws4_request, SignedHeaders=content-type;host;x-amz-date, Signature=d85a07692aceb401EXAMPLEa1b18ad1ddf58a1a3ce7EXAMPLE', 'Content-Length': '36'}>
2019-08-12 12:36:18,344 - MainThread - urllib3.util.retry - DEBUG - Converted retries value: False -> Retry(total=False, connect=None, read=None, redirect=0, status=None)
2019-08-12 12:36:18,344 - MainThread - urllib3.connectionpool - DEBUG - Starting new HTTPS connection (1): iam.amazonaws.com:443
2019-08-12 12:36:18,664 - MainThread - urllib3.connectionpool - DEBUG - https://iam.amazonaws.com:443 "POST / HTTP/1.1" 200 570
2019-08-12 12:36:18,664 - MainThread - botocore.parsers - DEBUG - Response headers: {'x-amzn-RequestId': '74c11606-bd38-11e9-9c82-559da0adb349', 'Content-Type': 'text/xml', 'Content-Length': '570', 'Date': 'Mon, 12 Aug 2019 19:36:18 GMT'}
2019-08-12 12:36:18,664 - MainThread - botocore.parsers - DEBUG - Response body:
```

```
b'<ListGroupsResponse xmlns="https://iam.amazonaws.com/doc/2010-05-08/">\n<ListGroupsResult>\n    <IsTruncated>false</IsTruncated>\n    <Groups>\n        <member>\n            <Path>/</Path>\n            <GroupName>MyTestGroup</GroupName>\n        <Arn>arn:aws:iam::123456789012:group/MyTestGroup</Arn>\n        <GroupId>AGPA1234567890EXAMPLE</GroupId>\n        <CreateDate>2019-08-12T19:34:04Z</CreateDate>\n    </member>\n</Groups>\n</ListGroupsResult>\n<ResponseMetadata>\n    <RequestId>74c11606-bd38-11e9-9c82-559da0adb349</RequestId>\n</ResponseMetadata>\n</ListGroupsResponse>\n'\n2019-08-12 12:36:18,665 - MainThread - botocore.hooks - DEBUG - Event needs-retry.iam.ListGroups: calling handler <botocore.retryhandler.RetryHandler object at 0x7fdf16e9a780>\n2019-08-12 12:36:18,665 - MainThread - botocore.retryhandler - DEBUG - No retry needed.\n2019-08-12 12:36:18,665 - MainThread - botocore.hooks - DEBUG - Event after-call.iam.ListGroups: calling handler <function json_decode_policies at 0x7fdf189b1d90>\n{\n    "Groups": [\n        {\n            "Path": "/",
            "GroupName": "MyTestGroup",
            "GroupId": "AGPA123456789012EXAMPLE",
            "Arn": "arn:aws:iam::123456789012:group/MyTestGroup",
            "CreateDate": "2019-08-12T19:34:04Z"
        }
    ]
}
```

[Back to top](#)

Enable and review the AWS CLI command history logs

You can enable the AWS CLI command history logs using the [cli_history](#) 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.

[Back to top](#)

Confirm that your AWS CLI is configured

Various errors can occur if your config and credentials files or your IAM user or role is 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”](#) and [the section called “Invalid credentials and key errors”](#).

[Back to top](#)

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> [<subcommand> ...] [parameters]
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
cp                                         | mv
....
```

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”](#) and [the section called “Confirm that you’re running a recent version of the AWS CLI”](#).

[Back to top](#)

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.

[Back to top](#)

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 [the section called “Install/Update”](#).

[Back to top](#)

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/
```

[Back to top](#)

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.

```
$ 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. The following command adds your local bin to the current PATH variable.

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

3. Reload the updated profile into your current session.

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

Windows

1. In a Windows Command Prompt, use the where command with the /R *path* parameter to find the aws file location. The results return all folders containing aws.

```
C:\> where /R c:\ aws  
c:\Program Files\Amazon\AWSCLIV2\aws.exe
```

...

By default, the AWS CLI version 2 is located in:

c:\Program Files\Amazon\AWSCLIV2\aws.exe

2. Press the Windows key and enter **environment variables**.
3. From the list of suggestions, choose **Edit environment variables for your account**.
4. Choose **PATH**, and then choose **Edit**.
5. Add the path you found in the first step into the **Variable value** field, for example, **C:\Program Files\Amazon\AWSCLIV2\aws.exe**.
6. Choose **OK** twice to apply the new settings.
7. Close any running command prompts and reopen the command prompt window.

[Back to top](#)

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. All open terminals needs to be closed, not just your active terminal.

[Back to top](#)

Possible cause: The system needs to be restarted after installing

If the aws command shows the wrong version and restarting the terminal didn't work, you might need to restart your system for it to recognize your PATH updates.

[Back to top](#)

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](#) 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 [the section called "Migration instructions"](#).

[Back to top](#)

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](#)

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.

[Back to top](#)

The AWS CLI processed a command with an incomplete parameter name

Possible cause: You used a recognized abbreviation of the AWS CLI parameter

Since the AWS CLI is built using Python, the AWS CLI uses the Python argparse library, including the [allow_abbrev](#) argument. Abbreviations of parameters are recognized by the AWS CLI and processed.

The following [create-change-set](#) command example changes the CloudFormation stack name. The parameter `--change-set-n` is recognized as an abbreviation of `--change-set-name`, and the AWS CLI processes the command.

```
$ aws cloudformation create-change-set --stack-name my-stack --change-set-n my-change-set
```

When your abbreviation could be multiple commands, the parameter will not be recognized as an abbreviation.

The following [create-change-set](#) command example changes the CloudFormation stack name. The parameter `--change-set-` is **not** recognized as an abbreviation, as there are multiple parameters it could be an abbreviation of, such as `--change-set-name` and `--change-set-type`. Therefore the AWS CLI does **not** process the command.

```
$ aws cloudformation create-change-set --stack-name my-stack --change-set- my-change-set
```

Warning

Do not purposefully use parameter abbreviations. They are unreliable and are not backwards compatible. If any new parameters are added to a command that confuse your abbreviations, it will break your commands.

Additionally, if the parameter is a single-value argument, it can cause unexpected behavior with your commands. If multiple instances of a single-value argument is passed, only the last instance will run. In the following example, the parameter `--filters` is a single-valued argument. The parameters `--filters` and `--filter` are specified. The `--filter` parameter is an abbreviation of `--filters`. This cause two instances of `--filters` being applied and only the last `--filter` argument applies.

```
$ aws ec2 describe-vpc-peering-connections \
  --filters Name>tag:TagName,Values=VpcPeeringConnection \
  --filter Name=status-code,Values=active
```

Confirm you are using valid parameters before running a command to prevent unexpected behavior.

[Back to top](#)

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
```

[Back to top](#)

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 account or role. The policies attached 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](#) to verify that the credentials the AWS CLI is using are the ones that you expect.

For information about assigning IAM permissions, see [Overview of Access Management: Permissions and Policies](#) in the *IAM User Guide*.

[Back to top](#)

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
included in the request is invalid.
```

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”](#). For more information on credentials and authentication, including credentials precedence, see [Authentication and access credentials](#).

[Back to top](#)

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](#)

Signature does not match errors

Example error text:

```
$ aws s3 ls  
An error occurred (SignatureDoesNotMatch) when calling the ListBuckets operation: The  
request signature we  
calculated does not match the signature you provided. Check your key and signing  
method.
```

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.

```
$ 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](#)

Possible cause: Your operating system is mishandling AWS keys that contain certain special characters

If your AWS keys include certain special characters, such as -, +, /, or %, some operating system variants process the string improperly and cause the key string to be interpreted incorrectly.

If you process your 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](#)

SSL certificate errors

Possible cause: The AWS CLI doesn't trust your proxy's certificate

Example error text:

```
$ aws s3 ls  
[SSL: CERTIFICATE_VERIFY_FAILED] certificate verify failed
```

When you use a 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 companies CA root certificate in the local CA registry.

To fix this, instruct the AWS CLI where to find your companies .pem file using the [ca_bundle](#) configuration file setting, [--ca-bundle](#) command line option, or the [AWS_CA_BUNDLE](#) environment variable.

[Back to top](#)

Possible cause: Your configuration isn't pointing to the correct CA root certificate location

Example error text:

```
$ 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 companies .pem file is located and update the AWS CLI configuration by using the [ca_bundle](#) configuration file setting, [--ca-bundle](#) command line option, or the [AWS_CA_BUNDLE](#) environment variable.

[Back to top](#)

Invalid JSON errors

Example error text:

```
$ 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)
JSON received: {"ReadCapacityUnits":15,WriteCapacityUnits":10}
```

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](#).

[Back to top](#)

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 its 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](#).

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
2022-07-19 22:25:07,741 - MainThread - awscli.clidriver - DEBUG - CLI version: aws-
cli/1.18.147
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”](#).

 **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 [Blob](#).

[Back to top](#)

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](#)

Migrate from AWS CLI version 1 to version 2

This section contains instructions for updating the AWS CLI version 1 to AWS CLI version 2. The AWS CLI version 2 builds on AWS CLI version 1 and includes features and enhancements based on community feedback.

Before you migrate to version 2, [learn about the differences between the versions](#). The AWS CLI version 2 includes new features and changes that might require you to update your scripts or commands for backwards compatibility.

AWS CLI versions 1 and 2 use the same aws command name. If you have both versions installed, your computer uses the first one found in your search path.

If you previously installed AWS CLI version 1, follow our [migration instructions to start using version 2](#).

If you have not previously installed AWS CLI version 1, follow the instructions in [Get started](#).

Topics

- [New features and changes in AWS CLI version 2](#)
- [AWS CLI version 2 migration instructions](#)

New features and changes in AWS CLI version 2

This topic describes new features and changes in behavior between AWS CLI version 1 and AWS CLI version 2. These changes might require you to update your scripts or commands to get the same behavior in version 2 as you did in version 1.

Topics

- [AWS CLI version 2 new features](#)
- [Breaking changes between AWS CLI version 1 and AWS CLI version 2](#)

AWS CLI version 2 new features

The AWS CLI version 2 is the most recent major version of the AWS CLI and supports all of the latest features. Some features introduced in version 2 are not backported to version 1 and you must upgrade to access those features. These features include the following:

Python interpreter not needed

The AWS CLI version 2 doesn't need a separate install of Python. It includes an embedded version.

Wizards

You can use a wizard with the AWS CLI version 2. The wizard guides you through constructing certain commands.

IAM Identity Center authentication

If your organization uses AWS IAM Identity Center (IAM Identity Center), your users can sign in to Active Directory, a built-in IAM Identity Center directory, or [another IdP connected to IAM Identity Center](#). Then, they are mapped to an AWS Identity and Access Management (IAM) role that allows you to run AWS CLI commands.

Auto-prompt

When enabled, the AWS CLI version 2 can prompt you for commands, parameters, and resources when you run an aws command.

Run the AWS CLI from the official Amazon ECR Public or Docker images

The official Docker image for the AWS CLI provides isolation, portability, and security that AWS directly supports and maintains. This way, you can use the AWS CLI version 2 in a container-based environment without having to manage the installation yourself.

Client-side pager

The AWS CLI version 2 provides the use of a client-side pager program for output. By default, this feature is turned on and returns all output through your operating system's default pager program.

aws configure import

Import .csv credentials generated from the AWS Management Console. A .csv file is imported with the profile name matching the IAM user name.

aws configure list-profiles

Lists the names of all profiles you have configured.

the section called “YAML stream output format”

The `yaml` and `yaml-stream` format takes advantage of the [YAML](#) format while providing more responsive viewing of large datasets by streaming the data to you. You can start viewing and using YAML data before the entire query downloads.

New high-level ddb commands for DynamoDB

The AWS CLI version 2 has the high-level Amazon DynamoDB commands [`ddb put`](#) and [`ddb select`](#). These commands provide a simplified interface for putting items in DynamoDB tables and searching in a DynamoDB table or index.

`aws logs tail`

The AWS CLI version 2 has a custom `aws logs tail` command that tails the logs for an Amazon CloudWatch Logs group. By default, the command returns logs from all associated CloudWatch Logs streams during the past ten minutes.

Added metadata support for high-level s3 commands

The AWS CLI version 2 adds the `--copy-props` parameter to the high-level `s3` commands. With this parameter, you can configure additional metadata and tags for Amazon Simple Storage Service (Amazon S3).

AWS_REGION

The AWS CLI version 2 has an AWS SDK-compatible environment variable called `AWS_REGION`. This variable specifies the AWS Region to send requests to. It overrides the `AWS_DEFAULT_REGION` environment variable, which is only applicable in the AWS CLI.

Breaking changes between AWS CLI version 1 and AWS CLI version 2

This sections describes all of the changes in behavior between AWS CLI version 1 and AWS CLI version 2. These changes might require you to update your scripts or commands to get the same behavior in version 2 as you did in version 1.

Topics

- [Environment variable added to set text file encoding](#)
- [Binary parameters are passed as base64-encoded strings by default](#)
- [Improved Amazon S3 handling of file properties and tags for multipart copies](#)
- [No automatic retrieval of `http://` or `https://` URLs for parameters](#)

- [Pager used for all output by default](#)
- [Timestamp output values are standardized to ISO 8601 format](#)
- [Improved handling of CloudFormation deployments that result in no changes](#)
- [Changed default behavior for Regional Amazon S3 endpoint for us-east-1 Region](#)
- [Changed default behavior for Regional AWS STS endpoints](#)
- [ecr get-login removed and replaced with ecr get-login-password](#)
- [AWS CLI version 2 support for plugins is changing](#)
- [Hidden alias support removed](#)
- [The api_versions configuration file setting is not supported](#)
- [AWS CLI version 2 uses only Signature v4 to authenticate Amazon S3 requests](#)
- [AWS CLI version 2 is more consistent with paging parameters](#)
- [AWS CLI version 2 provides more consistent return codes across all commands](#)

Environment variable added to set text file encoding

By default, text files for [the section called “Blob”](#) use the same encoding as the installed locale. Because the AWS CLI version 2 uses an embedded version of Python, the PYTHONUTF8 and PYTHONIOENCODING environment variables are not supported. To set encoding for text files to be different from the locale, use the AWS_CLI_FILE_ENCODING environment variable. The following example sets the AWS CLI to open text files using UTF-8 on Windows.

```
AWS_CLI_FILE_ENCODING=UTF-8
```

For more information, see [Environment variables to configure the AWS CLI](#).

Binary parameters are passed as base64-encoded strings by default

In the AWS CLI, some commands required [base64](#)-encoded strings, while others required UTF-8-encoded byte strings. In the AWS CLI version 1, passing data between two encoded string types often required some intermediate processing. The AWS CLI version 2 makes handling binary parameters more consistent, which helps pass values from one command to another more reliably.

By default, the AWS CLI version 2 passes all binary input and binary output parameters as the base64-encoded string blobs (binary large object). For more information, see [the section called “Blob”](#).

To revert to the AWS CLI version 1 behavior, use the [cli_binary_format](#) file configuration or the [--cli-binary-format](#) parameter.

Improved Amazon S3 handling of file properties and tags for multipart copies

When you use the AWS CLI version 1 commands in the `aws s3` namespace to copy a file from one S3 bucket location to another, and that operation uses [multipart copy](#), no file properties from the source object are copied to the destination object.

By default, the corresponding commands in the AWS CLI version 2 transfer all tags and some of the properties from the source to the destination copy. Compared to the AWS CLI version 1, this can result in more AWS API calls being made to the Amazon S3 endpoint. To change the default behavior for `s3` commands in AWS CLI version 2 , use the `--copy-props` parameter.

For more information, see [the section called “File properties and tags in multipart copies”](#).

No automatic retrieval of `http://` or `https://` URLs for parameters

The AWS CLI version 2 does not perform a GET operation when a parameter value begins with `http://` or `https://`, and does not use the returned content as the parameter value. As a result, the associated command line option `cli_follow_urlparam` is removed from the AWS CLI version 2.

If you need to retrieve a URL and pass the URL contents into a parameter value, we recommend that you use `curl` or a similar tool to download the contents of the URL to a local file. Then, use the `file://` syntax to read the contents of that file and use it as the parameter value.

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

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

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

```
$ curl https://my.example.com/mypolicyfile.json -o mypolicyfile.json
```

```
$ aws iam put-role-policy \
--policy-document file://./mypolicyfile.json \
--role-name MyRole \
--policy-name MyReadOnlyPolicy
```

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

Pager used for all output by default

By default, the AWS CLI version 2 returns all output through your operating system's default pager program. This program is the [less](#) program on Linux or macOS, and the [more](#) program on Windows. This can help you navigate a large amount of output from a service by displaying that output one page at a time.

You can configure the AWS CLI version 2 to use a different paging program or none at all. For more information, see [the section called “Client-side pager”](#).

Timestamp output values are standardized to ISO 8601 format

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

To see timestamps in the format returned by the HTTP API response, use the `wire` value in your config file. For more information, see [cli_timestamp_format](#).

Improved handling of CloudFormation deployments that result in no changes

By default in the AWS CLI version 1, if you deploy a AWS CloudFormation template that results in no changes, the AWS CLI returns a failed error code. This causes problems if you don't consider that to be an error and you want your script to continue. You can work around this in the AWS CLI version 1 by adding the flag `--no-fail-on-empty-changeset`, which returns `0`.

Since this is a common use case, the AWS CLI version 2 defaults to returning a successful exit code of `0` when there is no change caused by a deployment and the operation returns an empty changeset.

To revert to the original behavior, add the flag `--fail-on-empty-changeset`.

Changed default behavior for Regional Amazon S3 endpoint for us-east-1 Region

When you configure the AWS CLI version 1 to use the us-east-1 Region, the AWS CLI uses the global s3.amazonaws.com endpoint that is physically hosted in the us-east-1 Region. The AWS CLI version 2 uses the true Regional endpoint s3.us-east-1.amazonaws.com when that Region is specified. To force the AWS CLI version 2 to use the global endpoint, you can set the Region for a command to aws-global.

Changed default behavior for Regional AWS STS endpoints

By default, the AWS CLI version 2 sends all AWS Security Token Service (AWS STS) API requests to the Regional endpoint for the currently configured AWS Region.

By default, the AWS CLI version 1 sends AWS STS requests to the global AWS STS endpoint. You can control this default behavior in version 1 by using the [stsRegionalEndpoints](#) setting.

ecr get-login removed and replaced with ecr get-login-password

The AWS CLI version 2 replaces the command aws ecr get-login with the aws ecr get-login-password command that improves automated integration with container authentication.

The aws ecr get-login-password command reduces the risk of exposing your credentials in the process list, shell history, or other log files. It also improves compatibility with the docker login command for better automation.

The aws ecr get-login-password command is available in the AWS CLI version 1.17.10 and later, and the AWS CLI version 2. The earlier aws ecr get-login command is still available in the AWS CLI version 1 for backward compatibility.

With the aws ecr get-login-password command, you can replace the following code that retrieves a password.

```
$ (aws ecr get-login --no-include-email)
```

To reduce the risk of exposing the password to the shell history or logs, use the following example command instead. In this example, the password is piped directly to the docker login command, where it is assigned to the password parameter by the --password-stdin option.

```
$ aws ecr get-login-password | docker login --username AWS --password-stdin MY-REGISTRY-URL
```

For more information, see [aws ecr get-login-password](#) in the *AWS CLI version 2 Reference Guide*.

AWS CLI version 2 support for plugins is changing

Plugin support in the AWS CLI version 2 is completely provisional and is intended to help users migrate from AWS CLI version 1 until a stable, updated plugin interface is released. There are no guarantees that a particular plugin or even the AWS CLI plugin interface will be supported in future versions of the AWS CLI version 2. If you rely on plugins, be sure to lock into a particular version of the AWS CLI and test the functionality of your plugin when you do upgrade.

To enable plugin support, create a [plugins] section in your `~/.aws/config`.

```
[plugins]
cli_legacy_plugin_path = <path-to-plugins>/python3.7/site-packages
<plugin-name> = <plugin-module>
```

In the [plugins] section, define the `cli_legacy_plugin_path` variable and set its value to the Python site packages path where your plugin module is. Then, you can configure a plugin by providing a name for the plugin (`plugin-name`) and the file name of the Python module (`plugin-module`) that contains the source code for your plugin. The AWS CLI loads each plugin by importing its `plugin-module` and calling its `awscli_initialize` function.

Hidden alias support removed

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

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

Working service, command, and parameter	Obsolete alias
cognito-identity create-identity-pool open-id-connect-provider-arns	open-id-connect-provider-arns

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

The `api_versions` configuration file setting is not supported

The AWS CLI version 2 doesn't support calling earlier versions of AWS service APIs by using the `api_versions` configuration file setting. All AWS CLI commands now call the latest version of the service APIs that are currently supported by the endpoint.

AWS CLI version 2 uses only Signature v4 to authenticate Amazon S3 requests

The AWS CLI version 2 doesn't support earlier signature algorithms to cryptographically authenticate service requests sent to Amazon S3 endpoints. This signing happens automatically with every Amazon S3 request and only the [Signature Version 4 Signing Process](#) is supported. You can't configure the signature version. All Amazon S3 bucket presigned URLs now use only SigV4 and have a maximum expiration duration of one week.

AWS CLI version 2 is more consistent with paging parameters

In the AWS CLI version 1, if you specify pagination parameters on the command line, then automatic pagination is turned off as expected. However, when you specify pagination parameters by using a file with the `--cli-input-json` parameter, automatic pagination was not turned off, which could result in unexpected output. The AWS CLI version 2 turns off automatic pagination regardless of how you provide the parameters.

AWS CLI version 2 provides more consistent return codes across all commands

The AWS CLI version 2 is more consistent across all commands and properly returns an appropriate exit code compared to the AWS CLI version 1. We also added exit codes 252, 253, and 254. For more information on exit codes, see [the section called "Return Codes"](#).

If you have a dependency on how the AWS CLI version 1 uses return code values, we recommend checking the exit codes to make sure that you're getting the values you expect.

AWS CLI version 2 migration instructions

This topic provides instructions for migrating from AWS CLI version 1 to AWS CLI version 2.

AWS CLI versions 1 and 2 use the same `aws` command name. If you have both versions installed, your computer uses the first one found in your search path. If you previously installed AWS CLI version 1, we recommend that you do one of the following to use AWS CLI version 2:

- **Recommended** – [Uninstall AWS CLI version 1 and use only AWS CLI version 2.](#)

- [To have both version installed](#), use your operating system's ability to create a symbolic link (symlink) or alias with a different name for one of the two aws commands.

For information on breaking changes between version 1 and version 2, see [the section called "New features and changes"](#).

Replacing version 1 with version 2

Perform the following steps to replace AWS CLI version 1 with AWS CLI version 2.

To replace AWS CLI version 1 with AWS CLI version 2

1. Prepare any existing scripts you have for the migration by confirming any breaking changes between version 1 and version 2 in [the section called "New features and changes"](#).
2. Uninstall the AWS CLI version 1 by following the uninstall instructions for your operating system in [Installing, updating, and uninstalling the AWS CLI version 1](#).
3. Confirm that the AWS CLI is completely uninstalled by using the following command.

```
$ aws --version
```

Complete one of the following based on the output:

- **No version returned:** You've successfully uninstalled the AWS CLI version 1 and can proceed to the next step.
 - **A version is returned:** You still have an install of the AWS CLI version 1. For troubleshooting steps, see [the section called "The "aws --version" command returns a version after uninstalling the AWS CLI"](#). Perform troubleshooting steps until no version output is received.
4. Install the AWS CLI version 2 by following the appropriate install instructions for your operating system in [Install or update to the latest version of the AWS CLI](#).

Side-by-side install

To have both versions installed, use your operating system's ability to create a symbolic link (symlink) or alias with a different name for one of the two aws commands.

1. Install the AWS CLI version 2 by following the appropriate install instructions for your operating system in [Install or update to the latest version of the AWS CLI](#).

2. Use your operating system's ability to create a symlink or alias with a different name for one of the two aws commands, such as using `aws2` for AWS CLI version 2. The following are symlink examples for AWS CLI version 2. Replace the `PATH` with your install location.

Linux and macOS

You can use a [symbolic link](#) or [alias](#) on Linux and macOS.

```
$ alias aws2='PATH'
```

Windows command prompt

[DOSKEY](#) on Windows.

```
C:\> doskey aws2=PATH
```

Uninstall the AWS CLI version 2

This topic describes how to uninstall the AWS Command Line Interface version 2 (AWS CLI version 2).

AWS CLI version 2 uninstallation instructions:

Linux

To uninstall the AWS CLI version 2, run the following commands.

1. Locate the symlink and install paths.

- Use the `which` command to find the symlink. This shows the path you used with the `--bin-dir` parameter.

```
$ which aws  
/usr/local/bin/aws
```

- Use the `ls` command to find the directory that the symlink points to. This gives you the path you used with the `--install-dir` parameter.

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

2. Delete the two symlinks in the `--bin-dir` directory. If your user has write permission to these directories, you don't need to use `sudo`.

```
$ sudo rm /usr/local/bin/aws  
$ sudo rm /usr/local/bin/aws_completer
```

3. Delete the `--install-dir` directory. If your user has write permission to this directory, you don't need to use `sudo`.

```
$ sudo rm -rf /usr/local/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 `~/.aws/`. If your user has write permission to this directory, you don't need to use sudo.

```
$ sudo rm -rf ~/.aws/
```

macOS

To uninstall the AWS CLI version 2, run the following commands, substituting the paths you used to install. The example commands use the default installation paths.

1. Find the folder that contains the symlinks to the main program and the completer.

```
$ which aws  
/usr/local/bin/aws
```

2. Using that information, run the following command to find the installation folder that the symlinks point to.

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

3. Delete the two symlinks in the first folder. If your user already has write permission to these folders, you don't need to use sudo.

```
$ sudo rm /usr/local/bin/aws  
$ sudo rm /usr/local/bin/aws_completer
```

4. Delete the main installation folder. Use sudo to gain write access to the /usr/local folder.

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

5. **(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 has write permission to this directory, you don't need to use sudo.

```
$ sudo rm -rf ~/.aws/
```

Windows

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 then 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`.

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

Troubleshooting AWS CLI install and uninstall errors

If you come across issues after installing or uninstalling the AWS CLI, see [Troubleshoot errors](#) for troubleshooting steps. For the most relevant troubleshooting steps, see [the section called "Command not found errors"](#), [the section called "The "aws --version" command returns a different version than you installed"](#), and [the section called "The "aws --version" command returns a version after uninstalling the AWS CLI"](#).

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.

Change	Description	Date
<u>Updated credential and authentication information.</u>	<p>Updated credential and authentication method instructions and examples. This includes updating relevant Getting started pages and configuration pages. To accommodate this increase in documentation, relevant credential topics were moved to the new <u>Authentication and access credentials</u> section.</p>	March 31, 2023
<u>Token provider configuration with automatic authentication refresh for AWS IAM Identity Center added</u>	<p>The new process to configure the AWS CLI to authenticate users with AWS IAM Identity Center (IAM Identity Center) using the SSO token provider configuration, which can automatically retrieve refreshed authentication tokens.</p>	December 7, 2022
<u>Official Amazon ECR Public image for the AWS CLI version 2 released</u>	<p>The official supported Amazon ECR Public image for the AWS CLI version 2 is released for Linux, macOS, and Windows.</p>	November 18, 2022

<u>Updated the guide for migrating from AWS CLI V1 to V2</u>	Expanded the breaking changes guide to include migration instructions to going from AWS CLI version 1 to the AWS CLI version 2. Includes updates to the Troubleshooting page to help with installation issues.	May 13, 2022
<u>New process to build a AWS CLI installer from source.</u>	New process to install or update from source to the latest release of the AWS CLI on supported operating systems.	February 17, 2022
<u>Content for the AWS CLI V1 and V2 are now separated into their respective guides</u>	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 1, see the AWS CLI version 1 User Guide .	November 2, 2021
<u>Added AWS CLI alias information</u>	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.	March 11, 2021
<u>Updated filter output information</u>	Updated information for filters and moved to their own page.	February 1, 2021
<u>Added information for Wizards</u>	Added AWS CLI version 2 wizard information.	November 20, 2020

<u>Updated auto-prompt</u>	Updated the AWS CLI version 2 auto-prompt information with current features.	November 10, 2020
<u>Added Amazon S3 scripting example</u>	Added an Amazon S3 lifecycle scripting example.	October 15, 2020
<u>Added Amazon EC2 scripting example</u>	Added an Amazon EC2 instance type scripting example.	October 15, 2020
<u>Added retries information</u>	Added a retries page for features and behavior of retries in the AWS CLI.	September 17, 2020
<u>Server-side and client-side pagination page</u>	Updated pagination information and centralized on a single page.	August 17, 2020
<u>Updated s3 commands page</u>	Updated the high-level s3 commands page with new examples and resources.	July 30, 2020
<u>Updated installation information</u>	The install, update, and uninstall information for Linux, macOS, and Windows are updated.	May 19, 2020
<u>Added information for text file encoding on the AWS CLI version 2</u>	By default, AWS CLI version 2 uses the same text file encoding as the local. You can now use environment variables to set text file encoding.	May 14, 2020

<u>Official Docker image for the AWS CLI version 2 released</u>	The official support Docker image for the AWS CLI version 2 is released for all Linux, macOS, and Windows.	March 31, 2020
<u>Added information regarding client-side pagers for AWS CLI version 2</u>	By default, AWS CLI version 2 uses the pager program less for all client-side output.	February 19, 2020
<u>AWS Command Line Interface (AWS CLI) Version 2 is officially released</u>	The AWS CLI version 2 is generally available and is the recommended version for customers to install.	February 10, 2020
<u>macOS installer for AWS CLI version 2 is now an Apple Package installer .pkg file.</u>	The macOS installer for AWS CLI version 2 has been updated from a .zip file with a shell script to full macOS Installer package. This simplifies installation and makes it compatible with the newest macOS releases.	February 3, 2020
<u>Added content for AWS CLI version 2's improved default handling of S3 and STS Regional endpoints</u>	By default, AWS CLI version 2 now directs requests for the Amazon S3 and AWS STS services to the currently configured Regional endpoint instead of the global endpoint.	January 13, 2020
<u>Developer preview release for AWS CLI version 2</u>	Announcing preview release of AWS CLI version 2. Added instructions about installing version 2. Add Migration topic to discuss differences between versions 1 and 2.	November 7, 2019

<u>Added support for AWS IAM Identity Center to AWS CLI named profiles</u>	AWS CLI version 2 adds support for creating a named profile that can directly login to IAM Identity Center and get AWS temporary credentials for use in subsequent AWS CLI commands.	November 7, 2019
<u>New MFA section</u>	Added a new section describing how to access the CLI using multi-factor authentication and roles.	May 3, 2019
<u>Update to "Using the CLI" section</u>	Major improvements and additions to the usage instructions and procedures.	March 7, 2019
<u>Update to "Installing the CLI" section</u>	Major improvements and additions to the AWS CLI installation instructions and procedures.	March 7, 2019
<u>Update to "Configuring the CLI" section</u>	Major improvements and additions to the AWS CLI configuration instructions and procedures.	March 7, 2019

AWS Glossary

For the latest AWS terminology, see the [AWS glossary](#) in the *AWS Glossary Reference*.