AWS SDKs and Tools

Reference Guide

AWS SDKs and Tools: Reference Guide

Table of Contents

AWS SDKs and Tools Reference Guide

Applicable to all SDKs and tools

AWS SDKs and Tools maintenance policy (p. 59) covers the maintenance policy and versioning for AWS Software Development Kits (SDKs) and tools, including Mobile and Internet of Things (IoT) SDKs, and their underlying dependencies.

Applicable to some SDKs and tools

Many SDKs and tools share some common functionality, either through shared design specifications or through a shared library, such as the AWS Common Runtime (CRT) libraries.

This guide includes information regarding:

- Configuration (p. 3) – How to use the shared `config` and `credentials` files or environment variables for the authentication and configuration of an AWS SDK or tool.
- Configuration and authentication settings reference (p. 20) – Reference for all standardized settings available for authentication and configuration.
- AWS Common Runtime (CRT) libraries (p. 58) – Overview of the AWS Common Runtime (CRT) libraries that are available to almost all SDKs.

This AWS SDKs and Tools Reference Guide is intended to be a base of information that is applicable to multiple SDKs and tools. The specific SDK or tool guide for the SDK or tool you are using should be used in addition to any information presented here. The following are the relevant guides which have additional information in this guide:

- AWS Cloud Development Kit (AWS CDK) Developer Guide
- AWS Command Line Interface User Guide
- AWS Serverless Application Model Developer Guide
- AWS SDK for C++ Developer Guide
- AWS SDK for Go Developer Guide
- AWS SDK for Java Developer Guide
- AWS SDK for JavaScript Developer Guide
- AWS SDK for .NET Developer Guide
- AWS SDK for PHP Developer Guide
- AWS SDK for Python (Boto3) Getting Started
- AWS SDK for Ruby Developer Guide
- AWS Toolkit for Eclipse User Guide
- AWS Toolkit for JetBrains User Guide
- AWS Toolkit for Visual Studio User Guide
- AWS Toolkit for Visual Studio Code User Guide

- AWS Tools for Windows PowerShell User Guide

Developer resources

Amazon CodeWhisperer is a machine learning (ML)–powered service that helps improve developer productivity by generating code recommendations based on both code comments and code in the integrated development environment (IDE). To learn more about which languages and IDEs are supported, as well as how to sign up for free preview, see Amazon CodeWhisperer.

Configuration

With AWS SDKs and other AWS developer tools, such as the AWS Command Line Interface (AWS CLI), you can interact with AWS service APIs. Before attempting that, however, you must configure the SDK or tool with the information that it needs to perform the requested operation.

This information includes the following items:

- **Credentials information** that identifies who is calling the API. The credentials are used to encrypt the request to the AWS servers. Using this information, AWS confirms your identity and can retrieve permissions policies associated with it. Then it can determine what actions you're allowed to perform.
- **Other configuration details** that you use to tell the AWS CLI or SDK how to process the request, where to send the request (to which AWS service endpoint), and how to interpret or display the response.

Each SDK or tool supports multiple sources that you can use to supply the required credential and configuration information. Some sources are unique to the SDK or tool, and you must refer to the documentation for that tool or SDK for the details on how to use that method.

However, most of the AWS SDKs and tools support common settings from two primary sources (beyond the code itself):

- **Shared AWS config and credentials files** (p. 3) – The shared `config` and `credentials` files are the most common way that you can specify authentication and configuration to an AWS SDK or tool. Use these files to store settings that your tools and applications can use. Settings within the shared `config` and `credentials` files are associated with a specific profile. With multiple profiles, you can create different settings configurations to apply in different scenarios. When you use an AWS tool to invoke a command or use an SDK to invoke an AWS API, you can specify which profile, and thus which configuration settings, to use for that action. One of the profiles is designated as the `default` profile and is used automatically when you don't explicitly specify a profile to use. The settings that you can store in these files are documented in this reference guide.
- **Environment variables** (p. 8) – Some of the settings can alternatively be stored in the environment variables of your operating system. Although you can have only one set of environment variables in effect at a time, they are easily modified dynamically as your program runs and your requirements change.

Additional topics in this section

- Location of the shared `config` and `credentials` files (p. 6)
- Shared `config` and `credentials` files (p. 3)
- AWS SDKs and tools that use the shared config and credentials files (p. 7)
- Environment variables support (p. 8)

Shared `config` and `credentials` files

The shared AWS `config` and `credentials` files contain a set of profiles. A profile is a set of configuration values that can be referenced from the SDK/tool using its profile name. Configuration values are attached to a profile in order to configure some aspect of the SDK/tool when that profile is used.

As a general rule, any value that you can place in the shared `credentials` file can alternatively be placed in the shared `config` file. The reverse isn't true; only a few settings can be placed in the `credentials` file. However, as a security best practice, we recommend that you keep any sensitive values, such as access key IDs and secret keys, in the separate `credentials` file. This way, you can provide separate permissions for each file, if necessary.

Both the shared `config` and `credentials` files are plaintext files that contain only ASCII characters (UTF-8 encoded). They take the form of what are generally referred to as INI files.

Profiles

Settings within the shared `config` and `credentials` files are associated with a specific profile. With multiple profiles, you can create different settings configurations to apply in different scenarios.

The `[default]` profile contains the values that are used by an SDK or tool operation if a specific named profile is not specified. You can also create separate profiles that you can explicitly reference by name. Each named profile can have a different group of settings.

`[default]` is simply an unnamed profile. This profile is named `default` because it is the default profile used by the SDK if the user does not specify a profile. It does not provide inherited default values to other profiles. For example, if you set something in the `[default]` profile and you don't set it in a named profile, then the value isn't set when you use the named profile.

Optionally, set a named profile that you want to use through your SDK code or AWS CLI commands. Alternatively, you can use the environment variable `AWS_PROFILE` to specify which profile's settings to use.

Linux/macOS example of setting environment variables via command line:

```
export AWS_PROFILE="my_named_profile";
```

Windows example of setting environment variables via command line:

```
setx AWS_PROFILE "my_named_profile"
```

Format of the config file

The `config`file is organized into sections. A section is a named collection of settings, and continues until another section definition line is encountered.

The `config` file is a plaintext file that uses the following format:

- All entries in a section take the general form of `setting-name=value`.
- Lines can be commented out by starting the line with a hashtag character (#).

Section types

A section definition is a line that applies a name to a collection of settings. Section definition lines start and end with square brackets ([ ]). Inside the brackets, there is a section type identifier and a custom name for the section. You can use letters, numbers, hyphens (-), and underscores (_), but no spaces.

Section type: `profile`

Example section definition line: `[profile` *`dev`*`]`

The profile section definition line names a configuration grouping that you can apply in different scenarios. [default] is the only profile that does not require the profile section identifier. To better understand named profiles, see the preceding section on Profiles.

The following example shows a basic config file with a [default] profile. It sets the region (p. 43) setting.

```
[default]
#Full line comment, this text is ignored.
region = us-east-2
```

The following example shows a config file with a profile section definition line. It uses the identifier profile followed by a unique name for the profile. All settings that follow this line, up until another section definition is encountered, will be included with this named profile.

```
[profile developers]
...settings...
```

Some settings have their own nested group of subsettings, such as the s3 setting and subsettings in the following example. Associate the subsettings with the group by indenting them by one or more spaces.

```
[profile testers]
region = us-west-2
s3 =
    max_concurrent_requests=10
    max_queue_size=1000
```

Section type: sso-session

Example section definition line: [sso-session *my-sso*]

The sso-session section definition line names a group of settings that you use to configure a profile to resolve AWS credentials using AWS IAM Identity Center (successor to AWS Single Sign-On). For more information on configuring single sign-on authentication, see IAM Identity Center authentication (p. 10).

The following example configures a profile that will get short-term AWS credentials for the "SampleRole" IAM role in the "111122223333" account using a token from the "dev" sso-session.

```
[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
```

Format of the credentials file

The rules for the credentials file are generally identical to those for the config file, except that profile sections don't begin with the word profile. Use only the unique profile name itself between square brackets.

```
[developers]
```

```
...settings...
```

You can store only a small subset of settings and values in the `credentials` file. Generally, it's only those with values that would be considered "secrets" or sensitive, such as access key IDs and secret keys. The page for each setting in this guide states whether it can be stored in the `credentials` file or only in the `config` file.

The following example shows a basic `credentials` file with a `[default]` profile. It sets the `aws_access_key_id` and `aws_secret_access_key` (p. 37) global settings.

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

Location of the shared `config` and `credentials` files

The shared AWS `config` and `credentials` files are plaintext files that reside by default in a folder named `.aws` that is placed in the "home" folder on your computer.

On Linux and macOS, this is typically shown as `~/.aws`. On Windows, it is `%USERPROFILE%\.aws`.

Operating system	Default location and name of files
Linux and macOS	`~/.aws/config` `~/.aws/credentials`
Windows	`%USERPROFILE%\.aws\config` `%USERPROFILE%\.aws\credentials`

A ~/ or ~ followed by the file system's default path separator at the start of the path is resolved by checking, in order,

1. (All platforms) The `HOME` environment variable
2. (Windows platforms) The `USERPROFILE` environment variable
3. (Windows platforms) The `HOMEDRIVE` environment variable, prepended to the `HOMEPATH` environment variable (for example, `$HOMEDRIVE$HOMEPATH`)
4. (Optional per SDK or tool) An SDK or tool-specific home path resolution function or variable

When possible, if a user's home directory is specified at the start of the path (for example, `~username/`), it is resolved to the requested user name's home directory (for example, `/home/username/.aws/config`).

Changing the default location of these files:

The following environment variables can be set to change the location or name of these files from the default to a custom value:

- `config` file environment variable: **`AWS_CONFIG_FILE`**

- `credentials` file environment variable: **`AWS_SHARED_CREDENTIALS_FILE`**

Linux/macOS

You can specify an alternate location by running the following export commands on Linux or macOS.

```
$ export AWS_CONFIG_FILE=/some/file/path/on/the/system/config-file-name
$ export AWS_SHARED_CREDENTIALS_FILE=/some/other/file/path/on/the/system/credentials-
file-name
```

Windows

You can specify an alternate location by running the following setx commands on Windows.

```
C:\> setx AWS_CONFIG_FILE c:\some\file\path\on\the\system\config-file-name
C:\> setx AWS_SHARED_CREDENTIALS_FILE c:\some\other\file\path\on\the\system
\credentials-file-name
```

AWS SDKs and tools that use the shared config and credentials files

The following is a list of the AWS SDKs and other development tools that can use the shared `config` and `credentials` files to retrieve authentication and other configuration settings.

Each entry includes a link to that SDK's or tool's documentation where the use of these files is discussed in detail.

SDK or tool	Shared config and credentials topic
AWS Cloud Development Kit (AWS CDK)	Getting started with AWS Cloud Development Kit (AWS CDK)
AWS Command Line Interface (AWS CLI)	Configuring the AWS CLI
AWS Serverless Application Model	Setting up AWS credentials
AWS SDK for C++	Providing AWS credentials
AWS SDK for Go	Specifying credentials
AWS SDK for Java	Using credentials
AWS SDK for JavaScript	Getting your credentials
AWS SDK for .NET	Configure AWS credentials
AWS SDK for PHP	Using the AWS Credentials File and Credential Profiles
AWS SDK for Python (Boto3)	Configuration
AWS SDK for Ruby	Setting AWS Credentials
AWS Toolkit for Eclipse	Set up AWS credentials

SDK or tool	Shared config and credentials topic
AWS Toolkit for JetBrains	Setting AWS Credentials for the AWS Toolkit for JetBrains
AWS Toolkit for Visual Studio	Providing AWS Credentials
AWS Toolkit for Visual Studio Code	Setting Up Your AWS Credentials
AWS Tools for PowerShell	Getting Started with AWS Tools for PowerShell

Environment variables support

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. For the list of environment variables supported by most SDKs, see Environment variables list (p. 22).

Precedence of options

- If you specify a setting by using its environment variable, it overrides any value loaded from a profile in the shared AWS `config` and `credentials` files.
- If you specify a setting 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.

How to set environment variables

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

Linux, macOS, or Unix

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

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

Windows Command Prompt

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

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

PowerShell

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

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

Serverless environment variable setup

If you use a serverless architecture for development, you have other options for setting environment variables. Depending on your container, you can use different strategies for code running in those containers to see and access environment variables, similar to non-cloud environments.

For example, with AWS Lambda, you can directly set environment variables. For details, see Using AWS Lambda environment variables in the *AWS Lambda Developer Guide*.

In Serverless Framework, you can often set SDK environment variables in the `serverless.yml` file under the provider key under the environment setting. For information on the `serverless.yml` file, see General function settings in the Serverless Framework documentation.

Regardless of which mechanism you use to set container environment variables, there are some that are reserved by the container, such as those documented for Lambda at Defined runtime environment variables. Always consult the official documentation for the container that you're using to determine how environment variables are treated and whether there are any restrictions.

Authentication and access

You must establish how your code authenticates with AWS when you develop with AWS services. You can configure programmatic access to AWS resources in different ways, depending on the environment and the AWS access available to you.

Authentication options for code running locally (not in AWS)

- IAM Identity Center authentication (p. 10) – As a security best practice, we recommend using AWS Organizations with IAM Identity Center to manage access across all your AWS accounts. You can create users in AWS IAM Identity Center (successor to AWS Single Sign-On), use Microsoft Active Directory, use a SAML 2.0 identity provider (IdP), or individually federate your IdP to AWS accounts. To check if your Region supports IAM Identity Center, see AWS IAM Identity Center (successor to AWS Single Sign-On) endpoints and quotas in the *Amazon Web Services General Reference*.
- Other ways to authenticate (p. 15) – Other options that might be less convenient or might increase the security risk to your AWS resources.

Authentication options for code running within an AWS environment

- Using IAM roles for Amazon EC2 instances (p. 18) – Use IAM roles to securely run your application on an Amazon EC2 instance.
- You can programmatically interact with AWS using IAM Identity Center in the following ways:
  - Use AWS CloudShell to run AWS CLI commands from the console.
  - Use AWS Cloud9 to start programming on AWS using an integrated development environment (IDE) with AWS resources.
  - To try cloud-based collaboration space for software development teams, consider using Amazon CodeCatalyst.

More information about access management

The *IAM User Guide* has the following information about securely controlling access to AWS resources:

- IAM Identities (users, user groups, and roles) – Understanding the basics of identities in AWS.
- Security best practices in IAM – Security recommendations to follow when developing AWS applications according to the shared-responsibility model.

The *Amazon Web Services General Reference* has foundational basics on the following:

- Understanding and getting your AWS credentials – Access key options and management practices for both console and programmatic access.

IAM Identity Center authentication

AWS IAM Identity Center (successor to AWS Single Sign-On) is the recommended method of providing AWS credentials when developing on a non-AWS compute service. For example, this would be something like your local development environment. If you are developing on an AWS resource, such as Amazon

Elastic Compute Cloud (Amazon EC2) or AWS Cloud9, we recommend getting credentials from that service.

Configure programmatic access using IAM Identity Center

Step 1: Establish access and select appropriate permission set

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 (successor to AWS Single Sign-On) User Guide*. This process enables IAM Identity Center, creates an administrative user, and adds an appropriate least-privilege permission set.

- **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.

Step 2: Configure SDKs and tools to use IAM Identity Center

1. On your development machine, install the latest AWS CLI. See Installing or updating the latest version of the AWS CLI in the *AWS Command Line Interface User Guide*. You use the AWS CLI to configure and sign in to the IAM Identity Center access portal.
2. (Optional) To verify that the AWS CLI is working, open a command prompt and run the `aws --version` command.

3. In the AWS access portal, select the permission set you use for development, and select the **Command line or programmatic access** link.
4. In the **Get credentials** dialog box, choose either **MacOS and Linux** or **Windows**, depending on your operating system.
5. Choose the **IAM Identity Center** method to get the configuration details that you need for the next step.
6. Follow the procedure to Configure your profile with the `aws configure sso` wizard in the AWS CLI by running the `aws configure sso` command. When prompted, enter the configuration details that you collected in the previous step.

    a. For **CLI profile name**, we recommend entering *`default`* when you are getting started. For information about how to set non-default (named) profiles and their associated environment variable, see Profiles (p. 4).
    b. The `aws configure sso` command also signs you in to the IAM Identity Center access portal and begins your access portal session.
    c. Alternatively, you can create this configuration manually. For `config` file settings, see IAM Identity Center credential provider (p. 33).
7. (Optional) In the AWS CLI, confirm the active session identity by running the `aws sts get-caller-identity` command. The response should show the IAM Identity Center permission set that you configured.
8. If you are using an AWS SDK, create an application for your SDK in your development environment.

    a. For some SDKs, additional packages such as `SSO` and `SSOOIDC` must be added to your application before you can use IAM Identity Center authentication. For details, see your specific SDK.
    b. If you previously configured access to AWS, review your shared AWS `credentials` file for any Static credentials (p. 37). You must remove any static credentials before the SDK or tool can use the IAM Identity Center credentials because of the Credential provider chain (p. 24) precedence.

For a deep dive into how the SDKs and tools use and refresh credentials using this configuration, see Understand IAM Identity Center authentication (p. 12). Depending on your configured session lengths, your access will eventually expire and the SDK or tool will encounter an authentication error.

To refresh the access portal session again when needed, use the AWS CLI to run the `aws sso login` command.

You can extend both the IAM Identity Center access portal session duration and the permission set session duration. This lengthens the amount of time that you can run code before you need to manually sign in again with the AWS CLI. For more information, see the following topics in the *AWS IAM Identity Center (successor to AWS Single Sign-On) User Guide*:

- **IAM Identity Center session duration** – Configure the duration of your users' AWS access portal sessions
- **Permission set session duration** – Set session duration

Understand IAM Identity Center authentication

Relevant IAM Identity Center terms

The following terms help you understand the process and configuration behind AWS IAM Identity Center (successor to AWS Single Sign-On). The documentation for AWS SDK APIs uses different names than IAM Identity Center for some of these authentication concepts. It's helpful to know both names.

The following table shows how alternative names relate to each other.

IAM Identity Center name	SDK API name	Description
Identity Center	`sso`	Although AWS Single Sign-On is renamed, the `sso` API namespaces will keep their original name for backward compatibility purposes. For more information, see IAM Identity Center rename in the *AWS IAM Identity Center (successor to AWS Single Sign-On) User Guide*.
IAM Identity Center console Administrative console		The console you use to configure single sign-on.
AWS access portal URL		A URL unique to your IAM Identity Center account, like `https://xxx.awsapps.com/start`. You sign in to this portal using your IAM Identity Center sign-in credentials.
IAM Identity Center Access Portal session	Authentication session	Provides a bearer access token to the caller.
Permission set session		The IAM session that the SDK uses internally to make the AWS service calls. In informal discussions, you might see this incorrectly referred to as "role session."
Permission set credentials	AWS credentials sigv4 credentials	The credentials the SDK actually uses for most AWS service calls (specifically, all sigv4 AWS service calls). In informal discussions, you might see this incorrectly referred to as "role credentials."
IAM Identity Center credential provider	SSO credential provider	How you get the credentials, such as the class or module providing the functionality.

Understand SDK credential resolution for AWS services

The IAM Identity Center API exchanges bearer token credentials for sigv4 credentials. Most AWS services are sigv4 APIs, with a few exceptions like Amazon CodeWhisperer and Amazon CodeCatalyst. The following describes the credential resolution process for supporting most AWS service calls for your application code through AWS IAM Identity Center (successor to AWS Single Sign-On).

Start an AWS access portal session

- Start the process by signing in to the session with your credentials.

    - Use the `aws sso login` command in the AWS Command Line Interface (AWS CLI). This starts a new IAM Identity Center session if you don't already have an active session.
- When you start a new session, you receive a refresh token and access token from IAM Identity Center. The AWS CLI also updates an SSO cache JSON file with a new access token and refresh token and makes it available for use by SDKs.
- If you already have an active session, the AWS CLI command reuses the existing session and will expire whenever the existing session expires. To learn how to set the length of an IAM Identity Center session, see Configure the duration of your users' AWS access portal sessions in the *AWS IAM Identity Center (successor to AWS Single Sign-On) User Guide*.
    - The maximum session length has been extended to seven days to reduce the need for frequent sign-ins.

How the SDK gets credentials for AWS service calls

SDKs provide access to AWS services when you instantiate a client object per service. When the selected profile of the shared AWS `config` file is configured for IAM Identity Center credential resolution, IAM Identity Center is used to resolve credentials for your application.

- The credential resolution process is completed during runtime when a client is created.

To retrieve credentials for sigv4 APIs using IAM Identity Center single sign-on, the SDK uses the IAM Identity Center access token to get an IAM session. This IAM session is called a permission set session, and it provides AWS access to the SDK by assuming an IAM role.

- The permission set session duration is set independently from the IAM Identity Center session duration.
    - To learn how to set the permission set session duration, see Set session duration in the *AWS IAM Identity Center (successor to AWS Single Sign-On) User Guide*.
- Be aware that the permission set credentials are also referred to as *AWS credentials* and *sigv4 credentials* in most AWS SDK API documentation.

The permission set credentials are returned from a call to getRoleCredentials of the IAM Identity Center API to the SDK. The SDK's client object uses that assumed IAM role to make calls to the AWS service, such as asking Amazon S3 to list the buckets in your account. The client object can continue to operate using those permission set credentials until the permission set session expires.

Session expiration and refresh

When using the SSO token provider configuration (p. 34), the hourly access token obtained from IAM Identity Center is automatically refreshed using the refresh token.

- If the access token is expired when the SDK tries to use it, the SDK uses the refresh token to try to get a new access token. The IAM Identity Center compares the refresh token to your IAM Identity Center access portal session duration. If the refresh token is not expired, the IAM Identity Center responds with another access token.
- This access token can be used to either refresh the permission set session of existing clients, or to resolve credentials for new clients.

However, if the IAM Identity Center access portal session is expired, then no new access token is granted. Therefore, the permission set duration cannot be renewed. It will expire (and access will be lost) whenever the cached permission set session length times out for existing clients.

Any code that creates a new client will fail authentication as soon as the IAM Identity Center session expires. This is because the permission set credentials are not cached. Your code won't be able to create a new client and complete the credential resolution process until you have a valid access token.

To recap, when the SDK needs new permission set credentials, the SDK first checks for any valid, existing credentials and uses those. This applies whether the credentials are for a new client or for an existing client with expired credentials. If credentials aren't found or they're not valid, then the SDK calls the IAM Identity Center API to get new credentials. To call the API, it needs the access token. If the access token is expired, the SDK uses the refresh token to try to get a new access token from the IAM Identity Center service. This token is granted if your IAM Identity Center access portal session is not expired.

Other ways to authenticate

Use short-term credentials

We recommend configuring your SDK or tool to use IAM Identity Center authentication (p. 10) to use extended session duration options.

However, to set up the SDK or tool's temporary credentials directly, see Authenticate using short-term credentials (p. 16).

Use 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 (successor to AWS Single Sign-On).

Manage access across AWS accounts

As a security best practice, we recommend using AWS Organizations with IAM Identity Center to manage access across all your AWS accounts. For more information, see Security best practices in IAM in the *IAM User Guide*.

You can create users in IAM Identity Center, use Microsoft Active Directory, use a SAML 2.0 identity provider (IdP), or individually federate your IdP to AWS accounts. Using one of these approaches, you can provide a single sign-on experience for your users. You can also enforce multi-factor authentication (MFA) and use temporary credentials for AWS account access. This differs from an IAM user, which is a long-term credential that can be shared and which might increase the security risk to your AWS resources.

Create IAM users for sandbox environments only

If you're new to AWS, then you might create an IAM user and then use it to run tutorials and explore what AWS has to offer. It's okay to use this type of credential when you're learning, but we recommend that you avoid using it outside of a sandbox environment.

For the following use cases, it might make sense to get started with IAM users in AWS:

- Getting started with your AWS SDK or tool and exploring AWS services in a sandbox environment.
- Running scheduled scripts, jobs, and other automated processes that don't support a human-attended sign-in process as part of your learning.

If you're using IAM users outside of these use cases, then transition to IAM Identity Center or federate your identity provider to AWS accounts as soon as possible. For more information, see Identity federation in AWS.

Secure IAM user access keys

You should rotate IAM user access keys regularly. Follow the guidance in Rotating access keys in the *IAM User Guide*. If you believe that you have accidentally shared your IAM user access keys, then rotate your access keys.

IAM user access keys should be stored in the shared AWS `credentials` file on the local machine. Don't store the IAM user access keys in your code. Don't include configuration files that contain your IAM user access keys inside of any source code management software. External tools, such as the open source project git-secrets, can help you from inadvertently committing sensitive information to a Git repository. For more information, see IAM Identities (users, user groups, and roles) in the *IAM User Guide*.

To set up an IAM user to get started, see Authenticate using long-term credentials (p. 17).

Authenticate using short-term credentials

We recommend configuring your SDK or tool to use IAM Identity Center authentication (p. 10) 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.

Set up a credentials file using short-term credentials retrieved from AWS access portal

1. Create a shared credentials file.
2. In the credentials file, paste the following placeholder text until you paste in working temporary credentials.

```
[default]
aws_access_key_id=<value from AWS access portal>
aws_secret_access_key=<value from AWS access portal>
aws_session_token=<value from AWS access portal>
```

3. Save the file. The file `~/.aws/credentials` should now exist on your local development system. This file contains the [default] profile that the SDK or tool uses if a specific named profile is not specified.
4. Sign in to the AWS access portal.
5. Follow these instructions to copy IAM role credentials from the AWS access portal.

   Note
   These instructions apply to both the SDKs and the AWS Command Line Interface (AWS CLI).

   a. 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**.
   b. For step 4, select the **Add a profile to your AWS credentials file** option and copy the contents.
6. Paste the copied credentials into your local `credentials` file and remove the generated profile name. Your file should resemble the following.

```
[default]
aws_access_key_id=AKIAIOSFODNN7EXAMPLE
```

```
aws_secret_access_key=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
aws_session_token=IQoJb3JpZ2luX2IQoJb3JpZ2luX2IQoJb3JpZ2luX2IQoJb3JpZ2luX2IQoJb3JpZVERYLONGSTRINGEX
```

7. Save the `credentials` file.

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 5a determine how long the temporary credentials are valid. The maximum duration is twelve hours.

After the temporary credentials expire, repeat steps 4 through 7.

Authenticate using 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 (successor to AWS Single Sign-On).

If you use an IAM user to run your code, then the SDK or tool in your development environment authenticates by using long-term IAM user credentials in the shared AWS `credentials` file. Review the Security best practices in IAM topic and transition to IAM Identity Center or other temporary credentials as soon as possible.

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.
  - You can ignore optional steps.

Step 2: Get your access keys

1. In the navigation pane of the IAM console, select **Users** and then select the **`User name`** of the user that you created previously.
2. On the user's page, select the **Security credentials** page. Then, under **Access keys**, select **Create access key**.
3. For **Create access key Step 1**, choose either **Command Line Interface (CLI)** or **Local code**. Both options generate the same type of key to use with both the AWS CLI and the SDKs.
4. For **Create access key Step 2**, enter an optional tag and select **Next**.
5. 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.
6. Select **Done**.

Step 3: Update the shared `credentials` file

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 Location of Credentials Files.

2. Add the following text to the shared `credentials` file. Replace the example ID value and example key value with the values in the `.csv` file that you downloaded earlier.

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

3. Save the file.

The shared `credentials` file is the most common way to store credentials. This is a way to get you started, but we recommend you transition to IAM Identity Center or other temporary credentials as soon as possible. After you transition away from using long-term credentials, remember to delete these credentials from the shared `credentials` file.

Using IAM roles for Amazon EC2 instances

This example covers setting up an IAM role with Amazon S3 access to use in your application deployed to an Amazon EC2 instance.

Create an IAM role

Create an IAM role that grants read-only access to Amazon S3.

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, select **Roles**, then select **Create role**.
3. For **Select trusted entity**, under **Trusted entity type**, choose **AWS service**.
4. Under **Use case**, choose **Amazon EC2**, then select **Next**.
5. For **Add permissions**, select the checkbox for **Amazon S3 Read Only Access** from the policy list, then select **Next**.
6. Enter a name for the role, then select **Create role**. *Remember this name because you'll need it when you launch your Amazon EC2 instance.*

Launch an Amazon EC2 instance and specify your IAM role

You can launch an Amazon EC2 instance with an IAM role using the Amazon EC2 console.

Follow the directions to launch an instance in the Amazon EC2 User Guide for Linux Instances or the Amazon EC2 User Guide for Windows Instances.

When you reach the **Review Instance Launch** page, select **Edit instance details**. In **IAM role**, choose the IAM role that you created previously. Complete the procedure as directed.

> **Note**
> You need to create or use an existing security group and key pair to connect to the instance.

With this IAM and Amazon EC2 setup, you can deploy your application to the Amazon EC2 instance and it will have read access to the Amazon S3 service.

Connect to the EC2 instance

Connect to the EC2 instance so that you can transfer the sample application to it and then run the application. You'll need the file that contains the private portion of the key pair you used to launch the instance; that is, the PEM file.

You can do this by following the connect procedure in the Amazon EC2 User Guide for Linux Instances or the Amazon EC2 User Guide for Windows Instances. When you connect, do so in such a way that you can transfer files from your development machine to your instance.

If you're using an AWS Toolkit, you can often also connect to the instance by using the Toolkit. For more information, see the specific user guide for the Toolkit you use.

Run the sample application on the EC2 instance

1. Copy the application files from your local drive to your instance.

   For information about how to transfer files to your instance see the Amazon EC2 User Guide for Linux Instances or the Amazon EC2 User Guide for Windows Instances.
2. Start the application and verify that it runs with the same results as on your development machine.
3. (Optional) Verify that the application uses the credentials provided by the IAM role.

   a. Sign in to the AWS Management Console and open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
   b. Select the instance and detach the IAM role through **Actions**, **Instance Settings**, **Attach/Replace IAM Role**.
   c. Run the application again and confirm that it returns an authorization error.

Configuration and authentication settings reference

- Standardized credential providers (p. 23) – Common credential providers standardized across multiple SDKs.
- Standardized features (p. 39) – Common features standardized across multiple SDKs.

SDKs provide language-specific APIs for AWS services. They take care of some of the heavy lifting necessary in successfully making API calls, including authentication, retry behavior, and more. To do this, the SDKs have flexible strategies to obtain credentials to use for your requests, to maintain settings to use with each service, and to obtain values to use for global settings.

Creating service clients

To programmatically access AWS services, SDKs use a client class/object for each AWS service. For example, if your application needs to access Amazon EC2, your application creates an Amazon EC2 client object to interface with that service. You then use the service client to make requests to that AWS service. A service client object is immutable, so you must create a new client for each service to which you make requests and for making requests to the same service using a different configuration.

Precedence of settings

Global settings configure features, credential providers, and other functionality that are supported by most SDKs and have a broad impact across AWS services. All SDKs have a series of places (or sources) that they check in order to find a value for global settings. The following is the setting lookup precedence:

1. Any explicit setting set in the code or on a service client itself takes precedence over anything else.
   - Some settings can be set on a per-operation basis, and can be changed as needed for each operation that you invoke. For the AWS CLI or AWS Tools for PowerShell, these take the form of per-operation parameters that you enter on the command line. For an SDK, explicit assignments can take the form of a parameter that you set when you instantiate an AWS service client or configuration object, or sometimes when you call an individual API.
2. Java/Kotlin only: Sometimes there is a JVM system property associated with the setting. If it's set, that value is used to configure the client.
3. The environment variable is checked. If it's set, that value is used to configure the client.
4. The SDK checks the shared `credentials` file and then the shared `config` file. If the setting is present, the SDK uses it. The `AWS_PROFILE` environment variable or the `aws.profile` system property can be used to specify which profile that the SDK loads.
5. Any default value provided by the SDK code base is used last.

Note
If a setting exists in both the `config` file and the `credentials` file for the same profile, the value in the `credentials` file is used instead of the value in the `config` file.

Note
Some SDKs and tools might check in a different order. Also, some SDKs and tools support other methods of storing and retrieving parameters. For example, the AWS SDK for .NET supports an additional source called the SDK Store. For more information about providers that are unique to a SDK or tool, see the documentation for that SDK or tool (p. 7).

The order determines which methods take precedence and override others. For example, if you set up a `default` profile in the shared `config` file, it's only found and used after the SDK or tool checks the other places first. This means that if you put a setting in the `credentials` file, it is used instead of one found in the `config` file. If you configure an environment variable with a setting and value, it would override that setting in both the `credentials` and `config` files. And finally, a setting on the individual operation (AWS CLI command-line parameter or API parameter) or in code would override all other values for that one command.

`Config` file settings list

The settings listed in the following table can be assigned in the shared AWS `config` file. They are global and affect all AWS services.

Setting name	Details	
`api_versions`	General configuration settings (p. 48)	
`aws_access_key_id`	Static credentials (p. 37)	
`aws_secret_access_key`	Static credentials (p. 37)	
`aws_session_token`	Static credentials (p. 37)	
`ca_bundle`	General configuration settings (p. 48)	
`credential_process`	Process credentials (p. 31)	
`credential_source`	Assume role credentials (p. 24)	
`defaults_mode`	Smart configuration defaults (p. 55)	
`duration_seconds`	Assume role credentials (p. 24)	
`ec2_metadata_service_endpoint`	IMDS credentials (p. 29)	
`ec2_metadata_service_endpoint_mode`	IMDS credentials (p. 29)	
`endpoint_discovery_enabled`	Endpoint discovery (p. 47)	
`external_id`	Assume role credentials (p. 24)	
`max_attempts`	Retry behavior (p. 52)	
`metadata_service_num_attempts`	Amazon EC2 instance metadata (p. 39)	
`metadata_service_timeout`	Amazon EC2 instance metadata (p. 39)	
`mfa_serial`	Assume role credentials (p. 24)	
`output`	General configuration settings (p. 48)	
`parameter_validation`	General configuration settings (p. 48)	

Setting name	Details	
`region`	AWS Region (p. 43)	
`retry_mode`	Retry behavior (p. 52)	
`role_arn`	Assume role credentials (p. 24)	
`role_session_name`	Assume role credentials (p. 24)	
`s3_disable_multiregion_access_points`	Amazon S3 Multi-Region Access Points (p. 42)	
`s3_use_arn_region`	Amazon S3 access points (p. 41)	
`source_profile`	Assume role credentials (p. 24)	
`sso_account_id`	IAM Identity Center credentials (p. 33)	
`sso_region`	IAM Identity Center credentials (p. 33)	
`sso_registration_scopes`	IAM Identity Center credentials (p. 33)	
`sso_role_name`	IAM Identity Center credentials (p. 33)	
`sso_start_url`	IAM Identity Center credentials (p. 33)	
`sts_regional_endpoints`	AWS STS Regionalized endpoints (p. 45)	
`use_dualstack_endpoint`	Dual-stack and FIPS endpoints (p. 46)	
`use_fips_endpoint`	Dual-stack and FIPS endpoints (p. 46)	
`web_identity_token_file`	Assume role credentials (p. 24)	

`Credentials` file settings list

The settings listed in the following table can be assigned in the shared AWS `credentials` file. They are global and affect all AWS services.

Setting name	Details	
`aws_access_key_id`	Static credentials (p. 37)	
`aws_secret_access_key`	Static credentials (p. 37)	
`aws_session_token`	Static credentials (p. 37)	

Environment variables list

Environment variables supported by most SDKs are listed in the following table. They are global and affect all AWS services.

Setting name	Details	
`AWS_ACCESS_KEY_ID`	Static credentials (p. 37)	

Setting name	Details	
`AWS_CA_BUNDLE`	General configuration settings (p. 48)	
`AWS_CONFIG_FILE`	Location of the shared `config` and `credentials` files (p. 6)	
`AWS_CONTAINER_AUTHORIZATION_TOKEN`	Container credentials (p. 28)	
`AWS_CONTAINER_CREDENTIALS_FULL_URI`	Container credentials (p. 28)	
`AWS_CONTAINER_CREDENTIALS_RELATIVE_URI`	Container credentials (p. 28)	
`AWS_DEFAULTS_MODE`	Smart configuration defaults (p. 55)	
`AWS_EC2_METADATA_DISABLED`	IMDS credentials (p. 29)	
`AWS_EC2_METADATA_SERVICE_ENDPOINT`	IMDS credentials (p. 29)	
`AWS_EC2_METADATA_SERVICE_ENDPOINT_MODE`	IMDS credentials (p. 29)	
`AWS_ENABLE_ENDPOINT_DISCOVERY`	Endpoint discovery (p. 47)	
`AWS_MAX_ATTEMPTS`	Retry behavior (p. 52)	
`AWS_METADATA_SERVICE_NUM_ATTEMPTS`	Amazon EC2 instance metadata (p. 39)	
`AWS_METADATA_SERVICE_TIMEOUT`	Amazon EC2 instance metadata (p. 39)	
`AWS_PROFILE`	Shared AWS`config` and `credentials` files (p. 3)	
`AWS_REGION`	AWS Region (p. 43)	
`AWS_RETRY_MODE`	Retry behavior (p. 52)	
`AWS_S3_DISABLE_MULTIREGION_ACCESS_POINTS`	Amazon S3 Multi-Region Access Points (p. 42)	
`AWS_S3_USE_ARN_REGION`	Amazon S3 access points (p. 41)	
`AWS_SECRET_ACCESS_KEY`	Static credentials (p. 37)	
`AWS_SESSION_TOKEN`	Static credentials (p. 37)	
`AWS_SHARED_CREDENTIALS_FILE`	Location of the shared `config` and `credentials` files (p. 6)	
`AWS_STS_REGIONAL_ENDPOINTS`	AWS STS Regionalized endpoints (p. 45)	
`AWS_USE_DUALSTACK_ENDPOINT`	Dual-stack and FIPS endpoints (p. 46)	
`AWS_USE_FIPS_ENDPOINT`	Dual-stack and FIPS endpoints (p. 46)	

Standardized credential providers

Many credential providers have been standardized to consistent defaults and to work the same way across many SDKs. This consistency increases productivity and clarity when coding across multiple SDKs. All settings can be overridden in code. For details, see your specific SDK API.

Important
Not all SDKs support all providers, or even all aspects within a provider.

Topics

Credential provider chain

All SDKs have a series of places (or sources) that they check in order to find valid credentials to use to make a request to an AWS service. After valid credentials are found, the search is stopped. This systematic search is called the default credential provider chain. Although the distinct chain used by each SDK varies, they most often include sources such as the following:

- Static credentials (such as `AWS_ACCESS_KEY_ID`). For more information, see Static credentials (p. 37).
- Web identity token from AWS Security Token Service (AWS STS). For more information, see Assume role credential provider (p. 24).
- AWS IAM Identity Center (successor to AWS Single Sign-On). For more information, see IAM Identity Center credential provider (p. 33).
- Trusted entity provider (such as `AWS_ROLE_ARN`). For more information, see Assume role credential provider (p. 24).
- Amazon Elastic Container Service (Amazon ECS) credentials. For more information, see Container credential provider (p. 28).
- Custom credential provider. For more information, see Process credential provider (p. 31).
- Amazon Elastic Compute Cloud (Amazon EC2) instance profile credentials (IMDS credential provider). For more information, see IMDS credential provider (p. 29).

For each step in the chain, there are a variety of ways to assign setting values. Setting values specified in code always take precedence, but there are also Environment variables (p. 8) and the Shared `config` and `credentials` files (p. 3). For more information, see Precedence of settings (p. 20).

Assume role credential provider

Assuming a role involves using a set of temporary security credentials that you can use to access AWS resources that you might not normally have access to. These temporary credentials consist of an access key ID, a secret access key, and a security token. Typically, you assume a role within your account or for cross-account access.

To learn more about IAM roles, see Using IAM roles in the *IAM User Guide*.

To learn more about assuming a role, see AssumeRole in the *AWS Security Token Service API Reference*.

Configure this functionality by using the following:

`credential_source` - shared AWS `config` file setting

Used within Amazon EC2 instances or containers to specify where the SDK or development tool can find credentials that have permission to assume the role that you specify with the `role_arn` parameter.

Default value: None

Valid values:

- **Environment** – Specifies that the SDK or tool is to retrieve source credentials from the environment variables AWS_ACCESS_KEY_ID and AWS_SECRET_ACCESS_KEY (p. 37).
- **Ec2InstanceMetadata** – Specifies that the SDK or tool is to use the IAM role attached to the EC2 instance profile to get source credentials.
- **EcsContainer** – Specifies that the SDK or tool is to use the IAM role attached to the ECS container to get source credentials.

You cannot specify both `credential_source` and `source_profile` in the same profile.

Example of setting this in a `config` file to indicate that credentials should be sourced from Amazon EC2:

```
credential_source = Ec2InstanceMetadata
role_arn = arn:aws:iam::123456789012:role/my-role-name
```

`duration_seconds` - shared AWS `config` file setting

Specifies the maximum duration of the role session, in seconds.

This setting applies only when the profile specifies to assume a role.

Default value: 3600 seconds (one hour)

Valid values: The value can range from 900 seconds (15 minutes) up to the maximum session duration setting configured for the role (which can be a maximum of 43200 seconds, or 12 hours). For more information, see View the Maximum Session Duration Setting for a Role in the *IAM User Guide*.

Example of setting this in a `config` file:

```
duration_seconds = 43200
```

`external_id` - shared AWS `config` file setting

Specifies a unique identifier that is used by third parties to assume a role in their customers' accounts.

This setting applies only when the profile specifies to assume a role and the trust policy for the role requires a value for `ExternalId`. The value maps to the `ExternalId` parameter that is passed to the `AssumeRole` operation when the profile specifies a role.

Default value: None.

Valid values: See How to use an External ID When Granting Access to Your AWS Resources to a Third Party in the *IAM User Guide*.

Example of setting this in a `config` file:

```
external_id = unique_value_assigned_by_3rd_party
```

`mfa_serial` - shared AWS `config` file setting

Specifies the identification or serial number of a multi-factor authentication (MFA) device that the user must use when assuming a role.

Required when assuming a role where the trust policy for that role includes a condition that requires MFA authentication.

Default value: None.

Valid values: 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. For more information about MFA, see Configuring MFA-Protected API Access in the *IAM User Guide*.

Example of setting this in a config file:

```
mfa_serial = arn:aws:iam::123456789012:mfa/my-user-name
```

role_arn - shared AWS config file setting

Specifies the Amazon Resource Name (ARN) of an IAM role that you want to use to perform operations requested using this profile.

Default value: None.

Valid values: The value must be the ARN of an IAM role, formatted as follows: `arn:aws:iam::account-id:role/role-name`

In addition, you must also specify **one** of the following settings:

- `source_profile` – To identify another profile to use to find credentials that have permission to assume the role in this profile.
- `credential_source` – To use either credentials identified by the current environment variables or credentials attached to an Amazon EC2 instance profile, or an Amazon ECS container instance.

Example of setting this in a config file:

```
role_arn = arn:aws:iam::123456789012:role/my-role-name
source_profile = profile-name-with-user-that-can-assume-role
```

```
role_arn = arn:aws:iam::123456789012:role/my-role-name
credential_source = Ec2InstanceMetadata
```

role_session_name - shared AWS config file setting

Specifies the name to attach to the role session. This name appears in AWS CloudTrail logs for entries associated with this session.

Default value: An optional parameter. If you don't provide this value, a session name is generated automatically if the profile assumes a role.

Valid values: Provided to the `RoleSessionName` parameter when the AWS CLI calls the `AssumeRole` operation (or operations such as the `AssumeRoleWithWebIdentity` operation) on your behalf. The value becomes part of the assumed role user Amazon Resource Name (ARN) that you can query, and shows up as part of the CloudTrail log entries for operations invoked by this profile.

`arn:aws:sts::123456789012:assumed-role/my-role-name/`**`my-role_session_name`**.

Example of setting this in a config file:

```
role_session_name = my-role-session-name
```

source_profile - shared AWS config file setting

Specifies another profile whose credentials are used to assume the role specified by the `role_arn` setting in the original profile. To understand how profiles are used in the shared AWS `config` and `credentials` files, see Shared config and credentials files (p. 3).

If you specify a profile that is also an assume role profile, each role will be assumed in sequential order to fully resolve the credentials. This chain is stopped when the SDK encounters a profile with static credentials. Role chaining limits your AWS CLI or AWS API role session to a maximum of one hour and can't be increased. For more information, see Roles terms and concepts in the *IAM User Guide*.

Default value: None.

Valid values: A text string that consists of the name of a profile defined in the `config` and `credentials` files. You must also specify a value for `role_arn` in the current profile.

> **Note**
> This setting is an alternative to `credential_source`. You can't specify both `source_profile` and `credential_source` in the same profile.

Example of setting this in a config file:

```
[profile A]
source_profile = B
role_arn =  arn:aws:iam::123456789012:role/RoleA

[profile B]
aws_access_key_id=AKIAIOSFODNN7EXAMPLE
aws_secret_access_key=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
```

`web_identity_token_file` - shared AWS `config` file setting

Specifies the path to a file that contains an access token from a supported OAuth 2.0 provider or OpenID Connect ID identity provider.

This setting enables authentication by using web identity federation providers, such as Google, Facebook, and Amazon, among many others. The SDK or developer tool loads the contents of this file and passes it as the `WebIdentityToken` argument when it calls the `AssumeRoleWithWebIdentity` operation on your behalf.

Default value: None.

Valid values: This value must be a path and file name. The file must contain an OAuth 2.0 access token or an OpenID Connect token that was provided to you by an identity provider.

Example of setting this in a `config` file:

```
[profile web-identity]
role_arn=arn:aws:iam::123456789012:role/my-role-name
web_identity_token_file=/path/to/a/token
```

Compatibility with AWS SDKS

The following SDKs support the features and settings described on this page, any partial exceptions are noted:

SDK	Su	Notes or more information
AWS CLI v2	Yes	
SDK for C++	Partial	`credential_source` not supported. `duration_seconds` not supported. `mfa_serial` not supported.

SDK	Su	Notes or more information
SDK for Go V2 (1.x)	Yes	
SDK for Go 1.x (V1)	Yes	
SDK for Java 2.x	Partial	`mfa_serial` not supported.
SDK for Java 1.x	Partial	`mfa_serial` not supported.
SDK for JavaScript 3.x	Yes	
SDK for JavaScript 2.x	Partial	`credential_source` not supported.
SDK for .NET 3.x	Yes	
SDK for PHP 3.x	Yes	
SDK for Python (Boto3)	Yes	
SDK for Ruby 3.x	Yes	

Container credential provider

The container credential provider fetches credentials for customer's containerized application. This credential provider is useful for Amazon Elastic Container Service (Amazon ECS) customers. SDKs will attempt to load credentials from the specified HTTP endpoint through a GET request. To learn more about the `AWS_CONTAINER_CREDENTIALS_RELATIVE_URI` environment variable, see IAM Roles for Tasks in the *Amazon Elastic Container Service Developer Guide*.

Configure this functionality by using the following:

`AWS_CONTAINER_CREDENTIALS_FULL_URI` - environment variable

Contains the full HTTP URL endpoint for the SDK to use when making a request for credentials. This includes both the scheme and the host.

Default value: None.

Valid values: Valid URI.

Note: This setting is an alternative to `AWS_CONTAINER_CREDENTIALS_RELATIVE_URI` and will only be used if `AWS_CONTAINER_CREDENTIALS_RELATIVE_URI` is not set.

Linux/macOS example of setting environment variables via command line:

```
export AWS_CONTAINER_CREDENTIALS_FULL_URI=http://localhost/get-credentials
```

or

```
export AWS_CONTAINER_CREDENTIALS_FULL_URI=http://localhost:8080/get-credentials
```

`AWS_CONTAINER_CREDENTIALS_RELATIVE_URI` - environment variable

Specifies the relative HTTP URL endpoint for the SDK to use when making a request for credentials.

Default value: None.

Valid values: Valid relative URI.

Linux/macOS example of setting environment variables via command line:

```
export AWS_CONTAINER_CREDENTIALS_RELATIVE_URI=/get-credentials?a=1
```

`AWS_CONTAINER_AUTHORIZATION_TOKEN` - environment variable

If this variable is set, the SDK will set the Authorization header on the HTTP request with the environment variable's value.

Default value: None.

Valid values: String.

Linux/macOS example of setting environment variables via command line:

```
export AWS_CONTAINER_CREDENTIALS_FULL_URI=http://localhost/get-credential
export AWS_CONTAINER_AUTHORIZATION_TOKEN=Basic abcd
```

Compatibility with AWS SDKS

The following SDKs support the features and settings described on this page, any partial exceptions are noted:

SDK	Su	Notes or more information
AWS CLI v2	Yes	
SDK for C++	Yes	
SDK for Go V2 (1.x)	Yes	
SDK for Go 1.x (V1)	Yes	
SDK for Java 2.x	Yes	
SDK for Java 1.x	Yes	
SDK for JavaScript 3.x	Yes	
SDK for JavaScript 2.x	Yes	
SDK for .NET 3.x	Yes	
SDK for PHP 3.x	Yes	
SDK for Python (Boto3)	Yes	
SDK for Ruby 3.x	Yes	

IMDS credential provider

Instance Metadata Service (IMDS) provides data about your instance that you can use to configure or manage the running instance. For more information about the data available, see Instance metadata and user data in the *Amazon EC2 User Guide for Linux Instances* or Instance metadata and user data in the *Amazon EC2 User Guide for Windows Instances*. Amazon EC2 provides a local endpoint available to instances that can provide various bits of information to the instance. If the instance has a role attached, it can provide a set of credentials that are valid for that role. The SDKs can use that endpoint to resolve credentials as part of their default credential provider chain (p. 24). Instance Metadata Service Version 2 (IMDSv2), a more secure version of IMDS that uses a session token, is used by default. If that fails due to a non-retryable condition (HTTP error codes 403, 404, 405), IMDSv1 is used as a fallback.

Configure this functionality by using the following:

`AWS_EC2_METADATA_DISABLED` - environment variable

Whether or not to attempt to use Amazon EC2 Instance Metadata Service (IMDS) to obtain credentials.

Default value: `false`.

Valid values: `true`, `false`.

`ec2_metadata_service_endpoint` - shared AWS `config` file setting, `AWS_EC2_METADATA_SERVICE_ENDPOINT` - environment variable

The endpoint of IMDS.

Default value: If `ec2_metadata_service_endpoint_mode` equals `IPv4`, then default endpoint is `http://169.254.169.254`. If `ec2_metadata_service_endpoint_mode` equals `IPv6`, then default endpoint is `http://[fd00:ec2::254]`.

Valid values: Valid URI.

`ec2_metadata_service_endpoint_mode` - shared AWS `config` file setting, `AWS_EC2_METADATA_SERVICE_ENDPOINT_MODE` - environment variable

The endpoint mode of IMDS.

Default value:`IPv4`.

Valid values: `IPv4`, `IPv6`.

Security for IMDS credentials

By default, when the AWS SDK is not configured with valid credentials the SDK will attempt to use the Amazon EC2 Instance Metadata Service (IMDS) to retrieve credentials for an AWS role. This behavior can be disabled by setting the `AWS_EC2_METADATA_DISABLED` environment variable to `true`. This prevents unnecessary network activity and enhances security on untrusted networks where the Amazon EC2 Instance Metadata Service may be impersonated.

Note
AWS SDK clients configured with valid credentials will never use IMDS to retrieve credentials, regardless of any of these settings.

Disabling use of Amazon EC2 IMDS credentials

How you set this environment variable depends on what operating system is in use as well as whether or not you want the change to be persistent.

Linux and macOS

Customers using Linux or macOS can set this environment variable with the following command:

```
$ export AWS_EC2_METADATA_DISABLED=true
```

If you want this setting to be persistent across multiple shell sessions and system restarts, you can add the above command to your shell profile file, such as `.bash_profile`, `.zsh_profile`, or `.profile`.

Windows

Customers using Windows can set this environment variable with the following command:

```
$ set AWS_EC2_METADATA_DISABLED=true
```

If you want this setting to be persistent across multiple shell sessions and system restarts can use the following command instead:

```
$ setx AWS_EC2_METADATA_DISABLED=true
```

Note
The **setx** command does not apply the value to the current shell session, so you will need to reload or reopen the shell for the change to take effect.

Compatibility with AWS SDKS

The following SDKs support the features and settings described on this page, any partial exceptions are noted:

SDK	Su	Notes or more information
AWS CLI v2	Yes	
SDK for C++	Yes	
SDK for Go V2 (1.x)	Yes	
SDK for Go 1.x (V1)	Yes	
SDK for Java 2.x	Yes	
SDK for Java 1.x	Yes	
SDK for JavaScript 3.x	Yes	
SDK for JavaScript 2.x	Yes	
SDK for .NET 3.x	Yes	
SDK for PHP 3.x	Yes	
SDK for Python (Boto3)	Yes	
SDK for Ruby 3.x	Yes	

Process credential provider

SDKs provide a way to extend the credential provider chain for custom use cases.

Warning
The following describes a method of sourcing credentials from an external process. This can potentially be dangerous, so proceed with caution. Other credential providers should be

preferred if at all possible. If using this option, you should make sure that the `config` file is as locked down as possible using security best practices for your operating system. Confirm 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.

Configure this functionality by using the following:

`credential_process` - shared AWS `config` file setting

Specifies an external command that the SDK or tool runs on your behalf to generate or retrieve authentication credentials to use. The setting specifies the name of a program/command that the SDK will invoke. When the SDK invokes the process, it waits for the process to write JSON data to `stdout`. The custom provider must return information in a specific format. That information contains the credentials that the SDK or tool can use to authenticate you.

Specifying the path to the credentials program

The setting's value is a string that contains a path to a program that the SDK or development tool runs on your behalf:

- The path and file name can consist of only these characters: A-Z, a-z, 0-9, hyphen (-), underscore (_), period (.), forward slash (/), backslash (\ ), and space.
- If the path or file name contains a space, surround the complete path and file name with double-quotation marks (" ").
- 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.
- Don't include any environment variables in the strings. For example, don't include `$HOME` or `%USERPROFILE%`.
- Don't specify the home folder as ~. * You must specify either the full path or a base file name. If there is a base file name, the system attempts to find the program within folders specified by the `PATH` environment variable.

  Linux/macOS example of setting environment variables via command line:

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

  Windows example of setting environment variables via command line:

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

Valid output from the credentials program

The SDK runs the command as specified in the profile and then reads data from the standard output stream. The command you specify, whether a script or binary program, 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": "RFC3339 timestamp for 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 RFC3339 formatted timestamp. If the `Expiration` key isn't present in the tool's output, the SDK assumes that the credentials are long-term credentials that don't refresh. Otherwise, the credentials are considered temporary credentials, and they are automatically refreshed by rerunning the `credential_process` command before the credentials expire.

Note
The SDK 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.

Compatibility with AWS SDKS

The following SDKs support the features and settings described on this page, any partial exceptions are noted:

SDK	Su	Notes or more information
AWS CLI v2	Yes	
SDK for C++	Yes	
SDK for Go V2 (1.x)	Yes	
SDK for Go 1.x (V1)	Yes	
SDK for Java 2.x	Yes	
SDK for Java 1.x	Yes	
SDK for JavaScript 3.x	Yes	
SDK for JavaScript 2.x	Yes	
SDK for .NET 3.x	Yes	
SDK for PHP 3.x	Yes	
SDK for Python (Boto3)	Yes	
SDK for Ruby 3.x	Yes	

IAM Identity Center credential provider

This authentication mechanism uses AWS IAM Identity Center (successor to AWS Single Sign-On) to get single sign-on (SSO) access to AWS services for your code.

Note
In the AWS SDK API documentation, the IAM Identity Center credential provider is called the SSO credential provider.

After you enable IAM Identity Center, you define a profile for its settings in your shared AWS `config` file. This profile is used to connect to the IAM Identity Center access portal. When a user successfully authenticates with IAM Identity Center, the portal returns short-term credentials for the IAM role associated with that user. To learn how the SDK gets temporary credentials from the configuration and uses them for AWS service requests, see Understand IAM Identity Center authentication (p. 12).

There are two ways to configure IAM Identity Center through the `config` file:

- **SSO token provider configuration (recommended)** – Extend your session period up to a maximum of seven days.
- **Legacy non-refreshable configuration** – Get a fixed, eight-hour session.

In both configurations, you need to sign in again when your session expires.

To set custom session durations, you must use the SSO token provider configuration. For more information on session duration, see Configure AWS access portal session duration in the *AWS IAM Identity Center (successor to AWS Single Sign-On) User Guide*.

The following two guides contain additional information about IAM Identity Center:

- AWS IAM Identity Center (successor to AWS Single Sign-On) User Guide
- AWS IAM Identity Center (successor to AWS Single Sign-On) Portal API Reference

Prerequisites

You must first enable IAM Identity Center. For details about enabling IAM Identity Center authentication, see Getting Started in the *AWS IAM Identity Center (successor to AWS Single Sign-On) User Guide*.

Alternatively, follow the IAM Identity Center authentication (p. 10) instructions in this guide. These instructions provide complete guidance, from enabling IAM Identity Center to completing the necessary shared `config` file configuration that follows here..

SSO token provider configuration

Note
To use the AWS CLI to create this configuration for you, see Configure your profile with the `aws configure sso` wizard in the AWS CLI.

When you use the SSO token provider configuration, your AWS SDK or tool automatically refreshes your session up to your extended session period (seven days maximum).

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. For more details about formatting sections within a `config` file, see Format of the config file (p. 4).

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 AWS credentials.

Note
For a deep dive on how the SDKs and tools use and refresh credentials using this configuration, see Understand IAM Identity Center authentication (p. 12).

The following example configures the SDK to request IAM Identity Center credentials. It also 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
sso_registration_scopes = sso:account:access
```

You can reuse `sso-session` configurations 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
sso_registration_scopes = sso:account:access
```

`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.

Registration scopes are configured as part of an `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 is 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 and 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 file name based on the session name.

Legacy non-refreshable configuration

Automated token refresh isn't supported using the legacy non-refreshable configuration. We recommend using the SSO token provider configuration (p. 34) instead.

To use the legacy non-refreshable configuration, you must specify the following settings within your profile:

- `sso_start_url`
- `sso_region`
- `sso_account_id`
- `sso_role_name`

You specify the user portal for a profile with the `sso_start_url` and `sso_region` settings. You specify permissions with the `sso_account_id` and `sso_role_name` settings.

The following example sets the four required values in the `config` file.

```
[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 = SSOReadOnlyRole
```

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

IAM Identity Center credential provider settings

Configure this functionality by using the following:

`sso_start_url` - shared AWS `config` file setting

The URL that points to your organization's IAM Identity Center access portal. For more information on the IAM Identity Center access portal, see Using the AWS access portal in the *AWS IAM Identity Center (successor to AWS Single Sign-On) User Guide*.

To find this value, open the IAM Identity Center console, view the **Dashboard**, and find **AWS access portal URL**.

`sso_region` - shared AWS `config` file setting

The AWS Region that contains your IAM Identity Center portal host; that is, the Region you selected before enabling IAM Identity Center. This is independent from your default AWS Region, and can be different.

For a complete list of the AWS Regions and their codes, see Regional Endpoints in the *Amazon Web Services General Reference*. To find this value, open the IAM Identity Center console, view the **Dashboard**, and find **Region**.

`sso_account_id` - shared AWS `config` file setting

The numeric ID of the AWS account that was added through the AWS Organizations service to use for authentication.

To see the list of available accounts, go to the IAM Identity Center console and open the **AWS accounts** page. You can also see the list of available accounts using the ListAccounts API method in the *AWS IAM Identity Center (successor to AWS Single Sign-On) Portal API Reference*. For example, you can call the AWS CLI method list-accounts.

`sso_role_name` - shared AWS `config` file setting

The name of a permission set provisioned as an IAM role that defines the user's resulting permissions. The role must exist in the AWS account specified by `sso_account_id`. Use the role name, not the role Amazon Resource Name (ARN).

Permission sets have IAM policies and custom permissions policies attached to them and define the level of access that users have to their assigned AWS accounts.

To see the list of available permission sets per AWS account, go to the IAM Identity Center console and open the **AWS accounts** page. Choose the correct permission set name listed in the AWS accounts table. You can also see the list of available permission sets using the ListAccountRoles API method in the *AWS IAM Identity Center (successor to AWS Single Sign-On) Portal API Reference*. For example, you can call the AWS CLI method list-account-roles.

`sso_registration_scopes` - shared AWS `config` file setting

A comma-delimited list of valid scope strings to be authorized for the `sso-session`. Scopes authorize access to IAM Identity Center bearer token authorized endpoints. A minimum scope of `sso:account:access` must be granted to get a refresh token back from the IAM Identity Center service. This setting doesn't apply to the legacy non-refreshable configuration. Tokens issued using the legacy configuration are limited to scope `sso:account:access` implicitly.

Compatibility with AWS SDKS

The following SDKs support the features and settings described on this page, any partial exceptions are noted:

SDK	Su	Notes or more information
AWS CLI v2	Yes	
SDK for C++	Yes	
SDK for Go V2 (1.x)	Yes	
SDK for Go 1.x (V1)	Yes	
SDK for Java 2.x	Yes	Configuration values also supported in `credentials` file.
SDK for Java 1.x	No	
SDK for JavaScript 3.x	Yes	
SDK for JavaScript 2.x	Yes	
SDK for .NET 3.x	Yes	
SDK for PHP 3.x	Yes	
SDK for Python (Boto3)	Yes	
SDK for Ruby 3.x	Yes	

Static credentials

For basics on static credentials, see AWS account root user credentials and IAM identities and Understanding and getting your AWS credentials in the *Amazon Web Services General Reference*.

The AWS SDK automatically uses these AWS credentials to sign API requests to AWS, so that your workloads can access your AWS resources and data securely and conveniently. If you use an IAM role, these temporary AWS credentials are refreshed multiple times a day.

Note
If AWS becomes unable to refresh these temporary credentials, AWS may extend the validity of the credentials so that your workloads are not impacted.

The shared AWS `credentials` file is the recommended location for storing credentials information because it is safely outside of application source directories and separate from the SDK-specific settings of the shared `config` file.

Configure this functionality by using the following:

`aws_access_key_id` - shared AWS `config` file setting, `aws_access_key_id` - shared AWS `credentials` file setting *(recommended method)*, `AWS_ACCESS_KEY_ID` - environment variable

Specifies the AWS access key used as part of the credentials to authenticate the user.

`aws_secret_access_key` - shared AWS `config` file setting, `aws_secret_access_key` - shared AWS `credentials` file setting *(recommended method)*, `AWS_SECRET_ACCESS_KEY` - environment variable

Specifies the AWS secret key used as part of the credentials to authenticate the user.

`aws_session_token` - shared AWS `config` file setting, `aws_session_token` - shared AWS `credentials` file setting *(recommended method)*, `AWS_SESSION_TOKEN` - environment variable

Specifies an AWS session token used as part of the credentials to authenticate the user. You receive this value as part of the temporary credentials returned by successful requests to assume a role. A session token is required only if you manually specify temporary security credentials. However, we recommend you always use temporary security credentials instead of long-term credentials. For security recommendations, see Security best practices in IAM.

Example of setting these required values in the `config` or `credentials` file:

```
[default]
aws_access_key_id = AKIAIOSFODNN7EXAMPLE
aws_secret_access_key = wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
aws_session_token = AQoEXAMPLEH4aoAH0gNCAPy...truncated...zrkuWJOgQs8IZZaIv2BXIa2R4Olgk
```

Linux/macOS example of setting environment variables via command line:

```
export AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
export AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
export
 AWS_SESSION_TOKEN=AQoEXAMPLEH4aoAH0gNCAPy...truncated...zrkuWJOgQs8IZZaIv2BXIa2R4Olgk
```

Windows example of setting environment variables via command line:

```
setx AWS_ACCESS_KEY_ID AKIAIOSFODNN7EXAMPLE
setx AWS_SECRET_ACCESS_KEY wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
setx AWS_SESSION_TOKEN AQoEXAMPLEH4aoAH0gNCAPy...truncated...zrkuWJOgQs8IZZaIv2BXIa2R4Olgk
```

Compatibility with AWS SDKS

The following SDKs support the features and settings described on this page, any partial exceptions are noted:

SDK	Su	Notes or more information
AWS CLI v2	Yes	
SDK for C++	Yes	shared `config` file not supported.
SDK for Go V2 (1.x)	Yes	
SDK for Go 1.x (V1)	Yes	
SDK for Java 2.x	Yes	
SDK for Java 1.x	Yes	

SDK	Su	Notes or more information
SDK for JavaScript 3.x	Yes	
SDK for JavaScript 2.x	Yes	
SDK for .NET 3.x	Yes	Environment variables not supported.
SDK for PHP 3.x	Yes	
SDK for Python (Boto3)	Yes	
SDK for Ruby 3.x	Yes	

Standardized features

Many features have been standardized to consistent defaults and to work the same way across many SDKs. This consistency increases productivity and clarity when coding across multiple SDKs. All settings can be overridden in code, see your specific SDK API for details.

Important
Not all SDKs support all features, or even all aspects within a feature.

Topics

Amazon EC2 instance metadata

Amazon EC2 provides a service on instances called the Instance Metadata Service (IMDS). To learn more about this service, see Instance metadata and user data in the *Amazon EC2 User Guide for Linux Instances* or Instance metadata and user data in the *Amazon EC2 User Guide for Windows Instances*. When attempting to retrieve credentials on an Amazon EC2 instance that has been configured with an IAM role, the connection to the instance metadata service is adjustable.

Configure this functionality by using the following:

`metadata_service_num_attempts` - shared AWS `config` file setting, `AWS_METADATA_SERVICE_NUM_ATTEMPTS` - environment variable

This setting specifies the number of total attempts to make before giving up when attempting to retrieve data from the instance metadata service.

Default value: 1

Valid values: Number greater than or equal to 1.

`metadata_service_timeout` - shared AWS `config` file setting, `AWS_METADATA_SERVICE_TIMEOUT` - environment variable

Specifies the number of seconds before timing out when attempting to retrieve data from the instance metadata service.

Default value: 1

Valid values: Number greater than or equal to 1.

Example of setting these values in the `config` file:

```
[default]
metadata_service_num_attempts=10
metadata_service_timeout=10
```

Linux/macOS example of setting environment variables via command line:

```
export AWS_METADATA_SERVICE_NUM_ATTEMPTS=10
export AWS_METADATA_SERVICE_TIMEOUT=10
```

Windows example of setting environment variables via command line:

```
setx AWS_METADATA_SERVICE_NUM_ATTEMPTS 10
setx AWS_METADATA_SERVICE_TIMEOUT 10
```

Compatibility with AWS SDKS

The following SDKs support the features and settings described on this page, any partial exceptions are noted:

SDK	Su	Notes or more information
AWS CLI v2	Yes	
SDK for C++	No	
SDK for Go V2 (1.x)	No	
SDK for Go 1.x (V1)	No	
SDK for Java 2.x	No	
SDK for Java 1.x	Partial	`metadata_service_num_attempts` not supported.
SDK for JavaScript 3.x	No	
SDK for JavaScript 2.x	No	
SDK for .NET 3.x	No	
SDK for PHP 3.x	Yes	
SDK for Python (Boto3)	Yes	
SDK for Ruby 3.x	No	

Amazon S3 access points

The Amazon S3 service provides access points as an alternative way to interact with Amazon S3 buckets. Access points have unique policies and configurations that can be applied to them instead of directly to the bucket. With AWS SDKs, you can use access point Amazon Resource Names (ARNs) in the bucket field for API operations instead of specifying the bucket name explicitly. They are used for specific operations such as using an access point ARN with `GetObject` to fetch an object from a bucket, or using an access point ARN with `PutObject` to add an object to a bucket.

To learn more about Amazon S3 access points and ARNs, see Using access points in the *Amazon S3 User Guide*.

Configure this functionality by using the following:

`s3_use_arn_region` - shared AWS `config` file setting, `AWS_S3_USE_ARN_REGION` - environment variable, To configure value directly in code, consult your specific SDK directly.

This setting controls whether the SDK uses the access point ARN AWS Region to construct the Regional endpoint for the request. The SDK validates that the ARN AWS Region is served by the same AWS partition as the client's configured AWS Region to prevent cross-partition calls that most likely will fail. If multiply defined, the code-configured setting takes precedence, followed by the environment variable setting.

Default value: `false`

Valid values:

- **`true`** – The SDK uses the ARN's AWS Region when constructing the endpoint instead of the client's configured AWS Region. Exception: If the client's configured AWS Region is a FIPS AWS Region, then it must match the ARN's AWS Region. Otherwise, an error will result.
- **`false`** – The SDK uses the client's configured AWS Region when constructing the endpoint.

Compatibility with AWS SDKS

The following SDKs support the features and settings described on this page, any partial exceptions are noted:

SDK	Su	Notes or more information
AWS CLI v2	Yes	
SDK for C++	Yes	
SDK for Go V2 (1.x)	Yes	
SDK for Go 1.x (V1)	Yes	
SDK for Java 2.x	Yes	
SDK for Java 1.x	Yes	
SDK for JavaScript 3.x	Yes	
SDK for JavaScript 2.x	Yes	
SDK for .NET 3.x	Yes	
SDK for PHP 3.x	Partial	

SDK	Su	Notes or more information
SDK for Python (Boto3)	Yes	
SDK for Ruby 3.x	Yes	

Amazon S3 Multi-Region Access Points

Amazon S3 Multi-Region Access Points provide a global endpoint that applications can use to fulfill requests from Amazon S3 buckets located in multiple AWS Regions. You can use Multi-Region Access Points to build multi-Region applications with the same architecture used in a single Region, and then run those applications anywhere in the world.

To learn more about Multi-Region Access Points, see Multi-Region Access Points in Amazon S3 in the *Amazon S3 User Guide*.

To learn more about Multi-Region Access Point Amazon Resource Names (ARNs), see Making requests using a Multi-Region Access Point in the *Amazon S3 User Guide*.

To learn more about creating Multi-Region Access Points, see Managing Multi-Region Access Points in the *Amazon S3 User Guide*.

The SigV4A algorithm is the signing implementation used to sign the global Region requests. This algorithm is obtained by the SDK through a dependency on the AWS Common Runtime (CRT) libraries (p. 58).

Configure this functionality by using the following:

`s3_disable_multiregion_access_points` - shared AWS `config` file setting, `AWS_S3_DISABLE_MULTIREGION_ACCESS_POINTS` - environment variable, To configure value directly in code, consult your specific SDK directly.

This setting controls whether the SDK potentially attempts cross-Region requests. If multiply defined, the code-configured setting takes precedence, followed by the environment variable setting.

Default value: `false`

Valid values:

- **`true`** – Stops the use of cross-Region requests.
- **`false`** – Enables cross-Region requests using Multi-Region Access Points.

Compatibility with AWS SDKS

The following SDKs support the features and settings described on this page, any partial exceptions are noted:

SDK	Su	Notes or more information
AWS CLI v2	Yes	
SDK for C++	Yes	
SDK for Go V2 (1.x)	Yes	
SDK for Go 1.x (V1)	No	

SDK	Su	Notes or more information
SDK for Java 2.x	Yes	
SDK for Java 1.x	No	
SDK for JavaScript 3.x	Yes	
SDK for JavaScript 2.x	No	
SDK for .NET 3.x	Yes	
SDK for PHP 3.x	Yes	
SDK for Python (Boto3)	Yes	
SDK for Ruby 3.x	Yes	

AWS Region

AWS Regions are an important concept to understand when working with AWS services.

With AWS Regions, you can access AWS services that physically reside in a specific geographic area. This can be useful to keep your data and applications running close to where you and your users will access them. Regions provide fault tolerance, stability, and resilience, and can also reduce latency. With Regions, you can create redundant resources that remain available and unaffected by a Regional outage.

Most AWS service requests are associated with a particular geographic region. The resources that you create in one Region do not exist in any other Region unless you explicitly use a replication feature offered by an AWS service. For example, Amazon S3 and Amazon EC2 support cross-Region replication. Some services, such as IAM, do not have Regional resources.

The *AWS General Reference* contains information on the following:

- To understand the relationship between Regions and endpoints, and to view a list of existing Regional endpoints, see AWS service endpoints.
- To view the current list of all supported Regions and endpoints for each AWS service, see Service endpoints and quotas.

Creating service clients

To programmatically access AWS services, SDKs use a client class/object for each AWS service. If your application needs to access Amazon EC2, for example, your application would create an Amazon EC2 client object to interface with that service.

If no Region is explicitly specified for the client, the client defaults to using the Region set through the following `region` setting. However, the active Region for a client can be explicitly set for any individual client object. Setting the Region in this way takes precedence over any global setting for that particular service client. The alternative Region is specified during instantiation of that client, specific to your SDK (check your specific SDK Guide or your SDK's code base).

Configure this functionality by using the following:

`region` - shared AWS `config` file setting, `AWS_REGION` - environment variable

> Specifies the default AWS Region to use for AWS requests. This Region is used for SDK service requests that aren't provided with a specific Region to use.

Default value: None. You must specify this value explicitly.

Valid values:

- Any of the Region codes available for the chosen service, as listed in AWS service endpoints in the *AWS General Reference*. For example, the value `us-east-1` sets the endpoint to the AWS Region US East (N. Virginia).
- `aws-global` specifies the global endpoint for services that support a separate global endpoint in addition to Regional endpoints, such as AWS Security Token Service (AWS STS) and Amazon Simple Storage Service (Amazon S3).

Example of setting this value in the `config` file:

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

Linux/macOS example of setting environment variables via command line:

```
export AWS_REGION=us-west-2
```

Windows example of setting environment variables via command line:

```
setx AWS_REGION us-west-2
```

Most SDKs have a "configuration" object that is available for setting the default Region from within the application code. For details, see your specific AWS SDK developer guide.

Compatibility with AWS SDKS

The following SDKs support the features and settings described on this page, any partial exceptions are noted:

SDK	Su	Notes or more information
AWS CLI v2	Yes	AWS CLI V2 uses any value in `AWS_REGION` before any value in `AWS_DEFAULT_REGION` (both variables are checked).
AWS CLI v1	Yes	This SDK uses environment variable named `AWS_DEFAULT_REGION` for this purpose.
SDK for C++	Yes	
SDK for Go V2 (1.x)	Yes	
SDK for Go 1.x (V1)	Yes	
SDK for Java 2.x	Yes	
SDK for Java 1.x	Yes	
SDK for JavaScript 3.x	Yes	
SDK for JavaScript 2.x	Yes	
SDK for .NET 3.x	Yes	

SDK	Su	Notes or more information
SDK for PHP 3.x	Yes	
SDK for Python (Boto3)	Yes	This SDK uses environment variable named `AWS_DEFAULT_REGION` for this purpose.
SDK for Ruby 3.x	Yes	

AWS STS Regionalized endpoints

By default, AWS Security Token Service (AWS STS) is available as a global service, and all AWS STS requests go to a single endpoint at `https://sts.amazonaws.com`. Global requests map to the US East (N. Virginia) Region. AWS recommends using Regional AWS STS endpoints instead of the global endpoint. For more information on AWS STS endpoints, Endpoints in the *AWS Security Token Service API Reference*.

Configure this functionality by using the following:

`sts_regional_endpoints` - shared AWS `config` file setting, `AWS_STS_REGIONAL_ENDPOINTS` - environment variable

This setting specifies how the SDK or tool determines the AWS service endpoint that it uses to talk to the AWS Security Token Service (AWS STS).

Default value: `legacy`

Note
All new SDK major versions will default to `regional` in the future. New SDK major versions might remove this setting and use `regional` behavior.

Valid values:

- **`legacy`** – Uses the global AWS STS endpoint, `sts.amazonaws.com`, for the following AWS Regions: `ap-northeast-1`, `ap-south-1`, `ap-southeast-1`, `ap-southeast-2`, `aws-global`, `ca-central-1`, `eu-central-1`, `eu-north-1`, `eu-west-1`, `eu-west-2`, `eu-west-3`, `sa-east-1`, `us-east-1`, `us-east-2`, `us-west-1`, and `us-west-2`. All other Regions automatically use their respective Regional endpoint.
- **`regional`** – The SDK or tool always uses the AWS STS endpoint for the currently configured Region. For example, if the client is configured to use `us-west-2`, all calls to AWS STS are made to the Regional endpoint `sts.us-west-2.amazonaws.com`, instead of the global `sts.amazonaws.com` endpoint. To send a request to the global endpoint while this setting is enabled, you can set the Region to `aws-global`.

Example of setting these values in the `config` file:

```
[default]
sts_regional_endpoints = regional
```

Linux/macOS example of setting environment variables via command line:

```
export AWS_STS_REGIONAL_ENDPOINTS=regional
```

Windows example of setting environment variables via command line:

```
setx AWS_STS_REGIONAL_ENDPOINTS regional
```

Compatibility with AWS SDKS

The following SDKs support the features and settings described on this page, any partial exceptions are noted:

SDK	Su	Notes or more information
AWS CLI v2	Partia	Default value is `regional`.
SDK for C++	Partia	Environment variable and `config` file setting not supported. SDK performs with `regional` setting.
SDK for Go V2 (1.x)	Yes	
SDK for Go 1.x (V1)	Yes	
SDK for Java 2.x	Yes	
SDK for Java 1.x	Yes	
SDK for JavaScript 3.x	Yes	
SDK for JavaScript 2.x	Yes	
SDK for .NET 3.x	Yes	
SDK for PHP 3.x	Yes	
SDK for Python (Boto3)	Yes	
SDK for Ruby 3.x	Yes	

Dual-stack and FIPS endpoints

Configure this functionality by using the following:

`use_dualstack_endpoint` - shared AWS `config` file setting, `AWS_USE_DUALSTACK_ENDPOINT` - environment variable

Turns on or off whether the SDK will send requests to dual-stack endpoints. 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.

Default value: `false`

Valid values:

- **`true`** – The SDK or tool will attempt to use dual-stack endpoints to make network requests. If a dual-stack endpoint does not exist for the service and/or AWS Region, the request will fail.
- **`false`** – The SDK or tool will not use dual-stack endpoints to make network requests.

`use_fips_endpoint` - shared AWS `config` file setting, `AWS_USE_FIPS_ENDPOINT` - environment variable

Turns on or off whether the SDK or tool will send requests to FIPS-compliant endpoints. The Federal Information Processing Standards (FIPS) are a set of US Government security requirements for data and its encryption. Government agencies, partners, and those wanting to do business with the federal government are required to adhere to FIPS guidelines. Unlike standard AWS endpoints, FIPS endpoints use a TLS software library that complies with FIPS 140-2. To learn more about other ways to specify FIPS endpoints by AWS Region, see FIPS Endpoints by Service. For more information on Amazon Elastic Compute Cloud service endpoints, see Dual-stack (IPv4 and IPv6) endpoints in the *Amazon EC2 API Reference*.

Default value: `false`

Valid values:

- **`true`** – The SDK or tool will send requests to FIPS-compliant endpoints
- **`false`** – The SDK or tool will not send requests to FIPS-compliant endpoints.

Compatibility with AWS SDKS

The following SDKs support the features and settings described on this page, any partial exceptions are noted:

SDK	Su	Notes or more information
AWS CLI v2	Yes	
SDK for C++	Yes	
SDK for Go V2 (1.x)	Yes	
SDK for Go 1.x (V1)	Yes	
SDK for Java 2.x	Yes	
SDK for Java 1.x	No	
SDK for JavaScript 3.x	Yes	
SDK for JavaScript 2.x	Yes	
SDK for .NET 3.x	Yes	
SDK for PHP 3.x	Yes	
SDK for Python (Boto3)	Yes	
SDK for Ruby 3.x	Yes	

Endpoint discovery

SDKs use endpoint discovery to access service endpoints (URLs to access various resources), while still maintaining flexibility for AWS to alter URLs as needed. This way, your code can automatically detect new endpoints. There are no fixed endpoints for some services. Instead, you get the available endpoints during runtime by making a request to get the endpoints first. After retrieving the available endpoints, the code then uses the endpoint to access other operations. For example, for Amazon Timestream, the SDK makes a `DescribeEndpoints` request to retrieve the available endpoints, and then uses those endpoints to complete specific operations such as `CreateDatabase` or `CreateTable`.

Endpoint discovery is required in some services and optional in others. It defaults to either `true` or `false` depending on whether the service requires endpoint discovery. For example, Timestream defaults to `true`, and Amazon DynamoDB defaults to `false`. For services where endpoint discovery is not required, endpoint discovery is not enabled. Instead, configuration options are available through environment variables, the shared AWS `config` file, or SDK code constructs (for example, configuration classes). For operations where endpoint discovery is required, the SDK automatically attempts to discover an endpoint.

Configure this functionality by using the following:

`endpoint_discovery_enabled` - shared AWS `config` file setting, `AWS_ENABLE_ENDPOINT_DISCOVERY` - environment variable, To configure value directly in code, consult your specific SDK directly.

Enables/disables endpoint discovery for services where endpoint discovery is optional. Endpoint discovery is required in some services.

Default value: `false`

Valid values:

- **`true`** – The SDK should automatically attempt to discover an endpoint for services where endpoint discovery is optional.
- **`false`** – The SDK should not automatically attempt to discover an endpoint for services where endpoint discovery is optional.

Compatibility with AWS SDKS

The following SDKs support the features and settings described on this page, any partial exceptions are noted:

SDK	Su	Notes or more information
AWS CLI v2	Yes	
SDK for C++	Yes	
SDK for Go V2 (1.x)	Yes	
SDK for Go 1.x (V1)	Yes	
SDK for Java 2.x	Yes	
SDK for Java 1.x	Yes	
SDK for JavaScript 3.x	Yes	
SDK for JavaScript 2.x	Yes	
SDK for .NET 3.x	Yes	
SDK for PHP 3.x	Yes	
SDK for Python (Boto3)	Yes	
SDK for Ruby 3.x	Yes	

General configuration settings

SDKs support some general settings that configure overall SDK behaviors.

Configure this functionality by using the following:

`api_versions` - shared AWS `config` file setting

Some AWS services maintain multiple API versions to support backward compatibility. By default, SDK and AWS CLI operations use the latest available API version. To require a specific API version to use for your requests, include the `api_versions` setting in your profile.

Default value: None. (Latest API version is used by the SDK.)

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

The example sets a specific API version for two AWS services in the `config` file. These API versions are used only for commands that run under the profile that contains these settings. Commands for any other service use the latest version of that service's API.

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

`ca_bundle` - shared AWS `config` file setting, `AWS_CA_BUNDLE` - environment variable

Specifies the path to a custom certificate bundle (a file with a `.pem` extension) to use when establishing SSL/TLS connections.

Default value: none

Valid values: Specify either the full path or a base file name. If there is a base file name, the system attempts to find the program within folders specified by the `PATH` environment variable.

Example of setting this value in the `config` file:

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

Linux/macOS example of setting environment variables via command line:

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

Windows example of setting environment variables via command line:

```
setx AWS_CA_BUNDLE C:\dev\apps\ca-certs\cabundle-2019mar05.pem
```

`output` - shared AWS `config` file setting

Specifies how results are formatted in the AWS CLI and other AWS SDKs and tools.

Default value: `json`

Valid 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.

`parameter_validation` - shared AWS `config` file setting

Specifies whether the SDK or tool attempts to validate command line parameters before sending them to the AWS service endpoint.

Default value: `true`

Valid values:

- **`true`** – The default. The SDK or tool performs client-side validation of command line parameters. This helps the SDK or tool confirm that parameters are valid, and catches some errors. The SDK or tool can reject requests that aren't valid before sending requests to the AWS service endpoint.
- **`false`** – The SDK or tool doesn't validate command line parameters before sending them to the AWS service endpoint. The AWS service endpoint is responsible for validating all requests and rejecting requests that aren't valid.

Compatibility with AWS SDKS

The following SDKs support the features and settings described on this page, any partial exceptions are noted:

SDK	Su	Notes or more information
AWS CLI v2	Partial	`api_versions` not supported.
SDK for C++	Yes	
SDK for Go V2 (1.x)	Partial	`api_versions` and `parameter_validation` not supported.
SDK for Go 1.x (V1)	Partial	`api_versions` and `parameter_validation` not supported.
SDK for Java 2.x	No	
SDK for Java 1.x	No	
SDK for JavaScript 3.x	Yes	
SDK for JavaScript 2.x	Yes	
SDK for .NET 3.x	No	
SDK for PHP 3.x	Yes	
SDK for Python (Boto3)	Yes	
SDK for Ruby 3.x	Yes	

IMDS client

SDKs implement an Instance Metadata Service Version 2 (IMDSv2) client using session-oriented requests. For more information on IMDSv2, see Use IMDSv2 in the *Amazon EC2 User Guide for Linux Instances* or

Use IMDSv2 in the *Amazon EC2 User Guide for Windows Instances*. The IMDS client is configurable via a client configuration object available in the SDK code base.

Configure this functionality by using the following:

`retries` - client configuration object member

The number of additional retry attempts for any failed request.

Default value: 3

Valid values: Number greater than 0.

`port` - client configuration object member

The port for the endpoint.

Default value: 80

Valid values: Number.

`token_ttl` - client configuration object member

The TTL of the token.

Default value: 21,600 seconds (6 hours, the maximum time allotted).

Valid values: Number.

`endpoint` - client configuration object member

The endpoint of IMDS.

Default value: If `endpoint_mode` equals `IPv4`, then default endpoint is `http://169.254.169.254`. If `endpoint_mode` equals `IPv6`, then default endpoint is `http://[fd00:ec2::254]`.

Valid values: Valid URI.

The following options are supported by most SDKs. See your specific SDK code base for details.

`endpoint_mode` - client configuration object member

The endpoint mode of IMDS.

Default value: `IPv4`

Valid values: `IPv4`, `IPv6`

`http_open_timeout` - client configuration object member (name may vary)

The number of seconds to wait for the connection to open.

Default value: 1 second.

Valid values: Number greater than 0.

`http_read_timeout` - client configuration object member (name may vary)

The number of seconds for one chunk of data to be read.

Default value: 1 second.

Valid values: Number greater than 0.

`http_debug_output` - client configuration object member (name may vary)

Sets an output stream for debugging.

Default value: None.

Valid values: A valid I/O stream, like STDOUT.

`backoff` - client configuration object member (name may vary)

The number of seconds to sleep in between retries or a customer provided backoff function to call. This overrides the default exponential backoff strategy.

Default value: Varies by SDK.

Valid values: Varies by SDK. Can be either a numeric value or a call out to a custom function.

Compatibility with AWS SDKS

The following SDKs support the features and settings described on this page, any partial exceptions are noted:

SDK	Su	Notes or more information
AWS CLI v2	Yes	
SDK for C++	No	IMDSv2 used internally only. See IMDS credential provider (p. 29).
SDK for Go V2 (1.x)	Yes	
SDK for Go 1.x (V1)	Yes	
SDK for Java 2.x	Yes	
SDK for Java 1.x	Yes	
SDK for JavaScript 3.x	Yes	
SDK for JavaScript 2.x	Yes	
SDK for .NET 3.x	Yes	
SDK for PHP 3.x	Yes	
SDK for Python (Boto3)	Yes	
SDK for Ruby 3.x	Yes	

Retry behavior

Retry behavior includes settings regarding how the SDKs attempt to recover from failures resulting from requests made to AWS services.

Configure this functionality by using the following:

`max_attempts` - shared AWS `config` file setting, `AWS_MAX_ATTEMPTS` - environment variable

Specifies the maximum number attempts to make on a request.

Default value: If this value is not specified, its default depends on the value of the `retry_mode` setting:

- If `retry_mode` is `legacy` – Uses a default value specific to your SDK (check your specific SDK guide or your SDK's code base for `max_attempts` default).
- If `retry_mode` is `standard` – Makes three attempts.
- If `retry_mode` is `adaptive` – Makes three attempts.

Valid values: Number greater than 0.

`retry_mode` - shared AWS `config` file setting, `AWS_RETRY_MODE` - environment variable

Specifies how the SDK or developer tool attempts retries.

Default value: `legacy` is the default retry strategy.

Valid values:

- `legacy` – Specific to your SDK (check your specific SDK guide or your SDK's code base).
- `standard` – The standard set of retry rules across AWS SDKs. This mode includes a standard set of errors that are retried, and support for retry quotas. The default maximum number of attempts with this mode is three, unless `max_attempts` is explicitly configured.
- `adaptive` – An experimental retry mode that includes the functionality of standard mode but includes automatic client-side throttling. Because this mode is experimental, it might change behavior in the future.

Following is the high-level pseudocode for both the `standard` and `adaptive` retry modes:

```
MakeSDKRequest() {
  attempts = 0
  loop {
    GetSendToken()
    response = SendHTTPRequest()
    RequestBookkeeping(response)
    if not Retryable(response)
      return response
    attempts += 1
    if attempts >= MAX_ATTEMPTS:
      return response
    if not HasRetryQuota(response)
      return response
    delay = ExponentialBackoff(attempts)
    sleep(delay)
  }
}
```

Following are more details about the components used in the pseudocode:

`GetSendToken`:

Token buckets are only used in `adaptive` retry mode. Token buckets enforce a maximum request rate by requiring a token to be available in order to initiate a request. The SDK client is configurable to either fast fail the request or block until a token becomes available.

Client Side Rate Limiting is an algorithm that initially lets requests be made at any rate up to the token allowance. However, after a throttled response is detected, the client rate-of-request is then limited accordingly. The token allowance is also increased accordingly if successful responses are received.

With adaptive rate limiting, SDKs can slow down the rate at which requests are sent in order to better accommodate the capacity of AWS services.

`SendHTTPRequest`:

Most AWS SDKs use an HTTP library that uses connection pools so that you can reuse an existing connection when making an HTTP request. Generally, connections are reused when retrying requests due to throttling errors. Requests are not reused when retrying due to transient errors.

`RequestBookkeeping`:

The retry quota should be updated if the request is successful. For `adaptive` retry mode only, the state variable `maxsendrate` is updated based on the type of response received.

`Retryable`:

This step determines whether a response can be retried based on the following:

- The HTTP status code.
- The error code returned from the service.
- Connection errors, defined as any error received by the SDK in which an HTTP response from the service is not received.

Transient errors (HTTP status codes 400, 408, 500, 502, 503, and 504) and throttling errors (HTTP status codes 400, 403, 429, 502, 503, and 509) can all potentially be retried. SDK retry behavior is determined in combination with error codes or other data from the service.

`MAX_ATTEMPTS`:

Specified by the `config` file setting or the environment variable.

`HasRetryQuota`

This step throttles retry requests by requiring a token to be available in the retry quota bucket. Retry quota buckets are a mechanism to prevent retries that are unlikely to succeed. These quotas are SDK-dependent, are often client-dependent, and are sometimes even dependent on service endpoints. The available retry quota tokens are removed when requests fail for various reasons, and replenished when they succeed. When no tokens remain, the retry loop is exited.

`ExponentialBackoff`

For an error that can be retried, the retry delay is calculated using truncated exponential backoff. The SDKs use truncated binary exponential backoff with jitter. The following algorithm shows how the amount of time to sleep, in seconds, is defined for a response for request `i`:

```
seconds_to_sleep_i = min(b*r^i, MAX_BACKOFF)
```

In the preceding algorithm, the following values apply:

`b = random number within the range of: 0 <= b <= 1`

`r = 2`

`MAX_BACKOFF = 20 seconds` for most SDKs. See your specific SDK guide or source code for confirmation.

Compatibility with AWS SDKS

The following SDKs support the features and settings described on this page, any partial exceptions are noted:

SDK	Su	Notes or more information
AWS CLI v2	Yes	
SDK for C++	Yes	
SDK for Go V2 (1.x)	Yes	
SDK for Go 1.x (V1)	No	
SDK for Java 2.x	Yes	
SDK for Java 1.x	Yes	
SDK for JavaScript 3.x	Yes	
SDK for JavaScript 2.x	No	
SDK for .NET 3.x	Yes	
SDK for PHP 3.x	Yes	
SDK for Python (Boto3)	Yes	
SDK for Ruby 3.x	Yes	

Smart configuration defaults

With the smart configuration defaults feature, AWS SDKs can provide predefined, optimized default values for other configuration settings.

Configure this functionality by using the following:

`defaults_mode` - shared AWS `config` file setting, `AWS_DEFAULTS_MODE` - environment variable

With this setting, you can choose a mode that aligns with your application architecture, which then provides optimized default values for your application. If an AWS SDK setting has a value explicitly set, then that value always takes precedence. If an AWS SDK setting does not have a value explicitly set, and `defaults_mode` is not equal to legacy, then this feature can provide different default values for various settings optimized for your application. Settings may include the following: HTTP communication settings, retry behavior, service Regional endpoint settings, and, potentially, any SDK-related configuration. Customers who use this feature can get new configuration defaults tailored to common usage scenarios. If your `defaults_mode` is not equal to `legacy`, we recommend performing tests of your application when you upgrade the SDK, because the default values provided might change as best practices evolve.

Default value: `legacy`

Note: New major versions of SDKs will default to `standard`.

Valid values:

- `legacy` – Provides default settings that vary by SDK and existed before establishment of `defaults_mode`.
- `standard` – Provides the latest recommended default values that should be safe to run in most scenarios.
- `in-region` – Builds on the standard mode and includes optimization tailored for applications that call AWS services from within the same AWS Region.

- `cross-region` – Builds on the standard mode and includes optimization tailored for applications that call AWS services in a different Region.
- `mobile` – Builds on the standard mode and includes optimization tailored for mobile applications.
- `auto` – Builds on the standard mode and includes experimental features. The SDK attempts to discover the runtime environment to determine the appropriate settings automatically. The auto detection is heuristics-based and does not provide 100% accuracy. If the runtime environment can't be determined, `standard` mode is used. The auto detection might query Instance metadata and user data, which might introduce latency. If startup latency is critical to your application, we recommend choosing an explicit `defaults_mode` instead.

Example of setting this value in the `config` file:

```
[default]
defaults_mode = standard
```

The following parameters might be optimized based on the selection of `defaults_mode`:

- `retryMode` – Specifies how the SDK attempts retries. See Retry behavior (p. 52).
- `stsRegionalEndpoints` – Specifies how the SDK determines the AWS service endpoint that it uses to talk to the AWS Security Token Service (AWS STS). See AWS STS Regionalized endpoints (p. 45).
- `s3UsEast1RegionalEndpoints` – Specifies how the SDK determines the AWS service endpoint that it uses to talk to the Amazon S3 for the `us-east-1` Region.
- `connectTimeoutInMillis` – After making an initial connection attempt on a socket, the amount of time before timing out. If the client does not receive a completion of the connect handshake, the client gives up and fails the operation.
- `tlsNegotiationTimeoutInMillis` – The maximum amount of time that a TLS handshake can take from the time the CLIENT HELLO message is sent to the time the client and server have fully negotiated ciphers and exchanged keys.

The default value for each setting changes depending on the `defaults_mode` selected for your application. These values are currently set as follows (subject to change):

Parameter	`standard` mode	`in-region` mode	`cross-region` mode	`mobile` mode
`retryMode`	`standard`	`standard`	`standard`	`standard`
`stsRegionalEndpoints`	`regional`	`regional`	`regional`	`regional`
`s3UsEast1RegionalEndpoints`	`regional`	`regional`	`regional`	`regional`
`connectTimeoutInMillis`	3100	1100	3100	30000
`tlsNegotiationTimeoutInMillis`	3100	1100	3100	30000

For example, if the `defaults_mode` you selected was `standard`, then the value of `standard` would be assigned for `retry_mode` (from the valid `retry_mode` options) and the value of `regional` would be assigned for `stsRegionalEndpoints` (from the valid `stsRegionalEndpoints` options).

Compatibility with AWS SDKS

The following SDKs support the features and settings described on this page, any partial exceptions are noted:

SDK	Supported	Notes or more information
AWS CLI v2	No	
SDK for C++	Yes	Parameters not optimized: `stsRegionalEndpoints`, `s3UsEast1RegionalEndpoints`, `tlsNegotiationTimeoutInMillis`.
SDK for Go V2 (1.x)	Yes	Parameters not optimized: `retryMode`, `stsRegionalEndpoints`, `s3UsEast1RegionalEndpoints`.
SDK for Go 1.x (V1)	No	
SDK for Java 2.x	Yes	Parameters not optimized: `stsRegionalEndpoints`.
SDK for Java 1.x	No	
SDK for JavaScript 3.x	Yes	Parameters not optimized: `stsRegionalEndpoints`, `s3UsEast1RegionalEndpoints`, `tlsNegotiationTimeoutInMillis`. `connectTimeoutInMillis` is called `connectionTimeout`.
SDK for JavaScript 2.x	No	
SDK for .NET 3.x	Yes	Parameters not optimized: `connectTimeoutInMillis`, `tlsNegotiationTimeoutInMillis`.
SDK for PHP 3.x	Yes	Parameters not optimized: `tlsNegotiationTimeoutInMillis`.
SDK for Python (Boto3)	Yes	Parameters not optimized: `tlsNegotiationTimeoutInMillis`.
SDK for Ruby 3.x	Yes	

AWS Common Runtime (CRT) libraries

The AWS Common Runtime (CRT) libraries are a base library of the SDKs. The CRT is a modular family of independent packages, written in C. Each package provides good performance and minimal footprint for different required functionalities. These functionalities are common and shared across all SDKs providing better code reuse, optimization, and accuracy. The packages are:

- `awslabs/aws-c-auth`: AWS client-side authentication (standard credential providers and signing (sigv4))
- `awslabs/aws-c-cal`: Cryptographic primitive types, hashes (MD5, SHA256, SHA256 HMAC), signers, AES
- `awslabs/aws-c-common`: Basic data structures, threading/synchronization primitive types, buffer management, stdlib-related functions
- `awslabs/aws-c-compression`: Compression algorithms (Huffman encoding/decoding)
- `awslabs/aws-c-event-stream`: Event stream message processing (headers, prelude, payload, crc/trailer), remote procedure call (RPC) implementation over event streams
- `awslabs/aws-c-http`: C99 implementation of the HTTP/1.1 and HTTP/2 specifications
- `awslabs/aws-c-io`: Sockets (TCP, UDP), DNS, pipes, event loops, channels, SSL/TLS
- `awslabs/aws-c-iot`: C99 implementation of AWS IoT cloud services integration with devices
- `awslabs/aws-c-mqtt`: Standard, lightweight messaging protocol for the Internet of Things (IoT)
- `awslabs/aws-c-s3`: C99 library implementation for communicating with the Amazon S3 service, designed for maximizing throughput on high bandwidth Amazon EC2 instances
- `awslabs/aws-c-sdkutils`: A utilities library for parsing and managing AWS profiles
- `awslabs/aws-checksums`: Cross-platform hardware-accelerated CRC32c and CRC32 with fallback to efficient software implementations
- `awslabs/aws-lc`: General-purpose cryptographic library maintained by the AWS Cryptography team for AWS and their customers, based on code from the Google BoringSSL project and the OpenSSL project
- `awslabs/s2n`: C99 implementation of the TLS/SSL protocols, designed to be small and fast with security as a priority

The CRT is available through all SDKs except Go.

CRT dependencies

The CRT libraries form a complex net of relationships and dependencies. Knowing these relationships is helpful if you need to build the CRT directly from source. However, most users access CRT functionality through their language SDK (such as AWS SDK for C++ or AWS SDK for Java) or their language IoT device SDK (such as AWS IoT SDK for C++ or AWS IoT SDK for Java). In the following diagram, the Language CRT Bindings box refers to the package wrapping the CRT libraries for a specific language SDK. This is a collection of packages of the form `aws-crt-*`, where '*' is an SDK language (such as `aws-crt-cpp` or `aws-crt-java`).

The following is an illustration of the hierarchical dependencies of the CRT libraries.

Maintenance and support

For an overview of tools that can help you develop applications on AWS, see Tools to Build on AWS. For information on support, see the AWS Knowledge Center.

The following topics cover the maintenance and version support policies for AWS SDKs.

Topics

AWS SDKs and Tools maintenance policy

Overview

This document outlines the maintenance policy for AWS Software Development Kits (SDKs) and Tools, including Mobile and IoT SDKs, and their underlying dependencies. AWS regularly provides the AWS SDKs and Tools with updates that may contain support for new or updated AWS APIs, new features, enhancements, bug fixes, security patches, or documentation updates. Updates may also address changes with dependencies, language runtimes, and operating systems. AWS SDK releases are published to package managers (e.g. Maven, NuGet, PyPI), and are available as source code on GitHub.

We recommend users to stay up-to-date with SDK releases to keep up with the latest features, security updates, and underlying dependencies. Continued use of an unsupported SDK version is not recommended and is done at the user's discretion.

Versioning

The AWS SDK release versions are in the form of X.Y.Z where X represents the major version. Increasing the major version of an SDK indicates that this SDK underwent significant and substantial changes to support new idioms and patterns in the language. Major versions are introduced when public interfaces (e.g. classes, methods, types, etc.), behaviors, or semantics have changed. Applications need to be updated in order for them to work with the newest SDK version. It is important to update major versions carefully and in accordance with the upgrade guidelines provided by AWS.

SDK major version life-cycle

The life-cycle for major SDKs and Tools versions consists of 5 phases, which are outlined below.

- *Developer Preview (Phase 0)* - During this phase, SDKs are not supported, should not be used in production environments, and are meant for early access and feedback purposes only. It is possible for future releases to introduce breaking changes. Once AWS identifies a release to be a stable product, it may mark it as a Release Candidate. Release Candidates are ready for GA release unless significant bugs emerge, and will receive full AWS support.
- *General Availability (GA) (Phase 1)* - During this phase, SDKs are fully supported. AWS will provide regular SDK releases that include support for new services, API updates for existing services, as well as bug and security fixes. For Tools, AWS will provide regular releases that include new feature updates and bug fixes. AWS will support the GA version of an SDK for *at least 24 months*.

- *Maintenance Announcement (Phase 2)* - AWS will make a public announcement at least 6 months before an SDK enters maintenance mode. During this period, the SDK will continue to be fully supported. Typically, maintenance mode is announced at the same time as the next major version is transitioned to GA.
- *Maintenance (Phase 3)* - During the maintenance mode, AWS limits SDK releases to address critical bug fixes and security issues only. An SDK will not receive API updates for new or existing services, or be updated to support new regions. Maintenance mode has a *default duration of 12 months*, unless otherwise specified.
- *End-of-Support (Phase 4)* - When an SDK reaches end-of support, it will no longer receive updates or releases. Previously published releases will continue to be available via public package managers and the code will remain on GitHub. The GitHub repository may be archived. Use of an SDK which has reached end-of-support is done at the user's discretion. We recommend users upgrade to the new major version.

The following is a visual illustration of the SDK major version life-cycle. Please note that the timelines shown below are illustrative and not binding.

Dependency life-cycle

Most AWS SDKs have underlying dependencies, such as language runtimes, operating systems, or third party libraries and frameworks. These dependencies are typically tied to the language community or the vendor who owns that particular component. Each community or vendor publishes their own end-of-support schedule for their product.

The following terms are used to classify underlying third party dependencies:

- *Operating System (OS):* Examples include Amazon Linux AMI, Amazon Linux 2, Windows 2008, Windows 2012, Windows 2016, etc.
- **Language Runtime:** Examples include Java 7, Java 8, Java 11, .NET Core, .NET Standard, .NET PCL, etc.
- **Third party Library / Framework:* Examples include OpenSSL, .NET Framework 4.5, Java EE, etc.

Our policy is to continue supporting SDK dependencies for* at least 6 months* after the community or vendor ends support for the dependency. This policy, however, could vary depending on the specific dependency.

> **Note**
> AWS reserves the right to stop support for an underlying dependency without increasing the major SDK version

Communication methods

Maintenance announcements are communicated in several ways:

- An email announcement is sent to affected accounts, announcing our plans to end support for the specific SDK version. The email will outline the path to end-of-support, specify the campaign timelines, and provide upgrade guidance.
- AWS SDK documentation, such as API reference documentation, user guides, SDK product marketing pages, and GitHub readme(s) are updated to indicate the campaign timeline and provide guidance on upgrading affected applications.
- An AWS blog post is published that outlines the path to end-of-support, as well as reiterates the campaign timelines.
- Deprecation warnings are added to the SDKs, outlining the path to end-of-support and linking to the SDK documentation.

To see the list of available major versions of AWS SDKs and Tools and where they are in their maintenance life cycle, see the section called “Version support matrix” (p. 61).

AWS SDKs and Tools version support matrix

The matrix below shows the list of available AWS SDK major versions and where they are in the maintenance life cycle with associated timelines. For detailed information on the life-cycle for the major versions of Software Development Kits (SDKs) and Tools and their underlying dependencies see the section called “Maintenance policy” (p. 59)

SDK	Major version	Current Phase	General Availability Date	Notes
AWS CLI	1.x	General Availability	9/2/2013	
AWS CLI	2.x	General Availability	2/10/2020	
SDK for C++	1.x	General Availability	9/2/2015	
SDK for Go V2	V2 1.x	General Availability	1/19/2021	
SDK for Go	1.x	General Availability	11/19/2015	
SDK for Java	1.x	General Availability	3/25/2010	
SDK for Java	2.x	General Availability	11/20/2018	
SDK for JavaScript	1.x	End-of-Support	5/6/2013	
SDK for JavaScript	2.x	General Availability	6/19/2014	
SDK for JavaScript	3.x	General Availability	12/15/2020	
SDK for Kotlin	1.x	Developer Preview		
SDK for .NET	1.x	End-of-Support	11/2009	
SDK for .NET	2.x	End-of-Support	11/8/2013	
SDK for .NET	3.x	General Availability	7/28/2015	
SDK for PHP	2.x	End-of-Support	11/2/2012	
SDK for PHP	3.x	General Availability	5/27/2015	
SDK for Python (Boto2)	1.x	End-of-Support	7/13/2011	

SDK	Major version	Current Phase	General Availability Date	Notes
SDK for Python (Boto3)	1.x	General Availability	6/22/2015	
SDK for Python (Botocore)	1.x	General Availability	6/22/2015	
SDK for Ruby	1.x	End-of-Support	7/14/2011	
SDK for Ruby	2.x	End-of-Support	2/15/2015	
SDK for Ruby	3.x	General Availability	8/29/2017	
SDK for Rust	1.x	Developer Preview		
SDK for Swift	1.x	Developer Preview		
Tools for PowerShell	2.x	End-of-Support	11/8/2013	
Tools for PowerShell	3.x	End-of-Support	7/29/2015	
Tools for PowerShell	4.x	General Availability	11/21/2019	

Document history for AWS SDKs and Tools Reference Guide

The following table describes important additions and updates to the *AWS SDKs and Tools Reference Guide*. For notification about updates to this documentation, you can subscribe to the RSS feed.

Change	Description	Date
IAM best practices updates (p. 63)	Updated guide to align with the IAM best practices. For more information, see Security best practices in IAM.	February 27, 2023
SSO updates (p. 63)	Updates to SSO credentials for the new SSO token configuration.	November 19, 2022
Settings updates (p. 63)	Updates to support table for General configuration and for Amazon S3 Multi-Region Access Points.	November 17, 2022
Settings updates (p. 63)	Updates to clarity of IMDS client and IMDS credentials. Updates to Environment variables.	November 4, 2022
Updating welcome page (p. 63)	Announcing Amazon CodeWhisperer.	September 22, 2022
Service name change for single sign-on (p. 63)	Updates to reflect that AWS SSO is now referred to as AWS IAM Identity Center (successor to AWS Single Sign-On).	July 26, 2022
Settings update (p. 63)	Minor updates to config file details and to supported settings.	June 15, 2022
Update (p. 63)	Massive update of almost all parts of this guide.	February 1, 2022
Initial release (p. 63)	The first release of this guide is released to the public.	March 13, 2020

AWS glossary

For the latest AWS terminology, see the AWS glossary in the *AWS General Reference*.