Reference Guide

AWS SDKs and Tools

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

AWS SDKs and Tools: Reference Guide

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

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

Table of Contents

What is covered in the AWS SDKs and Tools Reference Guide

Many SDKs and tools share some common functionality, either through shared design specifications or through a shared library.

This guide includes information regarding:

- Globally configuring AWS SDKs and tools – How to use the shared `config` and `credentials` files or environment variables to configure your AWS SDKs and tools.
- Authentication and access using AWS SDKs and tools – Establish how your code or tool authenticates with AWS when you develop with AWS services.
- AWS SDKs and tools settings reference – Reference for all standardized settings available for authentication and configuration.
- AWS Common Runtime (CRT) libraries – Overview of the shared AWS Common Runtime (CRT) libraries that are available to almost all SDKs.
- AWS SDKs and Tools maintenance policy 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.

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 guide for the SDK or tool that you are using should be used in addition to any information presented here. The following are the SDK and tools which have relevant sections of material in this guide:

If you are using:	This guide's relevant sections for you are:
• Any SDK or tool	AWS SDKs and Tools maintenance policy
• AWS Cloud9 • AWS CDK • AWS Toolkit for Azure DevOps • AWS Toolkit for JetBrains • AWS Toolkit for Visual Studio	Globally configuring AWS SDKs and tools Authentication and access using AWS SDKs and tools AWS SDKs and Tools maintenance policy

If you are using:	This guide's relevant sections for you are:
• AWS Toolkit for Visual Studio Code • AWS Serverless Application Model • AWS CodeArtifact • AWS CodeBuild • Amazon CodeCatalyst • AWS CodeCommit • AWS CodeDeploy • AWS CodePipeline	
• AWS CLI • AWS SDK for C++ • AWS SDK for Go • AWS SDK for Java • AWS SDK for JavaScript • AWS SDK for Kotlin • AWS SDK for .NET • AWS SDK for PHP • AWS SDK for Python (Boto3) • AWS SDK for Ruby • AWS SDK for Rust • AWS SDK for Swift • AWS Tools for Windows PowerShell	Globally configuring AWS SDKs and tools Authentication and access using AWS SDKs and tools AWS SDKs and tools settings reference AWS Common Runtime (CRT) libraries AWS SDKs and Tools maintenance policy AWS SDKs and Tools version lifecycle

- 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.
- For AWS terminology, see the AWS glossary in the *AWS Glossary Reference*.

Developer resources

Amazon Q Developer is a generative AI-powered conversational assistant that can help you to understand, build, extend, and operate AWS applications. To accelerate your building on AWS, the model that powers Amazon Q is augmented with high-quality AWS content to produce more complete, actionable, and referenced answers. For more information, see What is Amazon Q Developer? in the *Amazon Q Developer User Guide*.

Toolkit telemetry notification

AWS Integrated Development Environment (IDE) Toolkits are plugins and extensions that enable access to AWS services in your IDE. Amazon Q IDE plugins and extensions enable generative AI assistance in your IDE. For detailed information about each of the IDE Toolkits, see the Toolkit User Guides in the preceding table. To learn more about using Amazon Q in your IDE, see the Using Amazon Q in the IDE topic in the *Amazon Q* developer guide.

AWS IDE Toolkits and Amazon Q may collect and store client-side telemetry data to inform decisions regarding future AWS Toolkit and Amazon Q releases. The data collected quantifies your usage of the AWS Toolkit and Amazon Q.

To learn more about the telemetry data collected across all of the AWS IDE Toolkits and Amazon Q, see the commonDefinitions.json document in the `aws-toolkit-common` Github repository.

For detailed information about the telemetry data collected by each of the AWS IDE Toolkits and Amazon Q extensions, reference the resource documents in the following AWS Toolkit GitHub repositories:

- AWS Visual Studio Toolkit with Amazon Q
- AWS Toolkit for Visual Studio Code and Amazon Q extension for VS Code
- AWS Toolkit for JetBrains and Amazon Q plugin for JetBrains
- Amazon Q for Eclipse

Certain AWS services that are accessible in the AWS Toolkits may collect additional client-side telemetry data. For detailed information about the type of data collected by each individual AWS service, see the AWS Documentation topic for the specific service you're interested in.

Globally configuring AWS SDKs and tools

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, the AWS SDKs and tools support common settings from primary sources beyond the code itself. This section covers the following topics:

Topics

Using shared `config` and `credentials` files to globally configure AWS SDKs and tools

The shared AWS `config` and `credentials` files are the most common way that you can specify authentication and configuration to an AWS SDK or tool.

The shared `config` and `credentials` files contain a set of profiles. A profile is a set of configuration settings, in key–value pairs, that is used by AWS SDKs, the AWS Command Line Interface (AWS CLI), and other tools. Configuration values are attached to a profile in order to configure some aspect of the SDK/tool when that profile is used. These files are "shared" in that the values take affect for any applications, processes, or SDKs on the local environment for a user.

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. Multiple profiles can be defined within the file to create different setting configurations to apply in different development environments.

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 profile can use different settings and values as needed by your application and scenario.

> **Note**
>
> `[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. 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.

Set a named profile

The `[default]` profile and multiple named profiles can exist in the same file. Use the following setting to select which profile's settings are used by your SDK or tool when running your code. Profiles can also be selected within code, or per-command when working with the AWS CLI.

Configure this functionality by setting one of the following:

`AWS_PROFILE` - environment variable

When this environment variable is set to a named profile or "default", all SDK code and AWS CLI commands use the settings in that profile.

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

```
export AWS_PROFILE="my_default_profile_name";
```

Windows example of setting environment variables via command line:

```
setx AWS_PROFILE "my_default_profile_name"
```

`aws.profile` - JVM system property

For SDK for Kotlin on the JVM and the SDK for Java 2.x, you can set the `aws.profile` system property. When the SDK creates a service client, it uses the settings in the named profile unless the setting is overridden in code. The SDK for Java 1.x does not support this system property.

> **Note**
>
> If your application is on a server running multiple applications, we recommend you always used named profiles rather than the default profile. The default profile is automatically picked up by any AWS application in the environment and is shared amongst them. Thus, if someone else updates the default profile for their application it can unintentionally impact the others. To safeguard against this, define a named profile in the shared `config` file and then use that named profile in your application by setting the named profile in your code. You can use the environment variable or JVM system property to set the named profile if you know that it's scope only affects your application.

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: `default`

Example section definition line: `[default]`

`[default]` is the only profile that does not require the `profile` section identifier.

The following example shows a basic `config` file with a `[default]` profile. It sets the `region` setting. All settings that follow this line, up until another section definition is encountered, are part of this profile.

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

Section type: `profile`

Example section definition line: `[profile dev]`

The `profile` section definition line is a named configuration grouping that you can apply for different development scenarios. To better understand named profiles, see the preceding section on Profiles.

The following example shows a `config` file with a `profile` section definition line and a named profile called `foo`. All settings that follow this line, up until another section definition is encountered, are part of this named profile.

```
[profile foo]
...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 test]
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. For more information on configuring single sign-on authentication, see Using IAM Identity Center to authenticate AWS SDK and tools. A profile is linked to a `sso-session` section by a key-value pair where `sso-session` is the key and the name of your `sso-session` section is the value, such as `sso-session = <name-of-sso-session-section>`.

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 "my-sso". The "my-sso" `sso-session` section is referenced in the `profile` section by name using the `sso-session` key.

```
[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
```

Section type: `services`

Example section definition line: `[services dev]`

> **Note**
>
> The `services` section supports service-specific endpoint customizations and is only available in SDKs and tools that include this feature. To see if this feature is available for your SDK, see Support by AWS SDKs and tools for service-specific endpoints.

The `services` section definition line names a group of settings that configures custom endpoints for AWS service requests. A profile is linked to a `services` section by a key-value pair where `services` is the key and the name of your `services` section is the value, such as `services = <name-of-services-section>`.

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

The following example uses a `services` definition to configure the endpoint to use for requests made only to the Amazon DynamoDB service. The `"local-dynamodb" services` section is referenced in the `profile` section by name using the `services` key. The AWS service identifier key is dynamodb. The Amazon DynamoDB service subsection begins on the line `dynamodb =` . Any immediately following lines that are indented are included in that subsection and apply to that service.

```
[profile dev]
services = local-dynamodb

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

For more information on custom endpoint configuration, see Service-specific endpoints.

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 profile name itself between

square brackets. The following example shows a `credentials` file with a named profile section called `foo`.

```
[foo]
...credential settings...
```

Only the following settings that are considered "secrets" or sensitive can be stored in the `credentials` file: `aws_access_key_id`,`aws_secret_access_key`, and `aws_session_token`. Although these settings can alternatively be placed in the shared `config` file, we recommend that you keep these sensitive values in the separate `credentials` file. This way, you can provide separate permissions for each file, if necessary.

The following example shows a basic `credentials` file with a `[default]` profile. It sets the `aws_access_key_id`,`aws_secret_access_key`, and `aws_session_token` global settings.

```
[default]
aws_access_key_id=AKIAIOSFODNN7EXAMPLE
aws_secret_access_key=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
aws_session_token=IQoJb3JpZ2luX2IQoJb3JpZ2luX2IQoJb3JpZ2luX2IQoJb3JpZ2luX2IQoJb3JpZVERYLONGSTRI
```

Regardless of whether you use a named profile or "`default`" in your `credentials` file, any settings here will be combined with any settings from your `config` file that uses the same profile name. If there are credentials in both files for a profile sharing the same name, the keys in the credentials file take precedence.

Finding and changing the location of the shared `config` and `credentials` files of AWS SDKs and tools

The shared AWS `config` and `credentials` files are plaintext files that hold configuration information for the AWS SDKs and tools. The files reside locally in your environment and used automatically by SDK code or by AWS CLI commands that you run in that environment. For example, on your own computer or when developing on an Amazon Elastic Compute Cloud instance.

When the SDK or tool runs, it checks for these files and loads any available configuration settings. If the files do not already exist, then a basic file is automatically created by the SDK or tool.

By default, the files are in a folder named `.aws` that is placed in your home or user folder.

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

Home directory resolution

~ is only used for home directory resolution when it:

- Starts the path
- Is followed immediately by / or a platform specific separator. On windows, ~/ and ~\ both resolve to the home directory.

When determining the home directory, the following variables are checked:

- (All platforms) The `HOME` environment variable
- (Windows platforms) The `USERPROFILE` environment variable
- (Windows platforms) The concatenation of `HOMEDRIVE` and `HOMEPATH` environment variables (`$HOMEDRIVE$HOMEPATH`)
- (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`).

Change the default location of these files

You can use any of the following to override where these files are loaded from by the SDK or tool.

Use environment variables

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
```

For more information on configuring your system using environment variables, see Using environment variables to globally configure AWS SDKs and tools.

Use JVM system properties

For the SDK for Kotlin running on the JVM and for SDK for Java 2.x, you can set the following JVM system properties to change the location or name of these files from the default to a custom value:

- `config` file JVM system property: **`aws.configFile`**
- `credentials` file environment variable: **`aws.sharedCredentialsFile`**

For instructions on how to set JVM system properties, see the section called “How to set JVM system properties”. The SDK for Java 1.x does not support these system properties.

Using environment variables to globally configure AWS SDKs and tools

Environment variables provide another way to specify configuration options and credentials when using AWS SDKs and tools. Environment variables 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.

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_SESSION_TOKEN=AQoEXAMPLEH4aoAH0gNCAPy...truncated...zrkuWJOgQs8IZZaIv2BXIa2R4Olgk
$ export AWS_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_SESSION_TOKEN AQoEXAMPLEH4aoAH0gNCAPy...truncated...zrkuWJOgQs8IZZaIv2BXIa2R4Olgk
C:\> setx AWS_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_SESSION_TOKEN="AQoEXAMPLEH4aoAH0gNCAPy...truncated...zrkuWJOgQs8IZZaIv2BXIa2R4Ol
PS C:\> $Env:AWS_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.

Using JVM system properties to globally configure AWS SDK for Java and AWS SDK for Kotlin

JVM system properties provide another way to specify configuration options and credentials for SDKs that run on the JVM such as the AWS SDK for Java and the AWS SDK for Kotlin. For a list of JVM system properties supported by SDKs, see Settings reference.

Precedence of options

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

How to set JVM system properties

You can set JVM system properties several ways.

On the command line

Set JVM system properties on the command-line when invoking the `java` command by using the `-D` switch. The following command configures the AWS Region globally for all service clients unless you explicitly override the value in code.

```
java -Daws.region=us-east-1 -jar <your_application.jar> <other_arguments>
```

If you need to set multiple JVM system properties, specify the `-D` switch multiple times.

With an environment variable

If you can't access the command line to invoke the JVM to run your application, you can use the `JAVA_TOOL_OPTIONS` environment variable to configure command-line options. This approach is useful in situations such as running an AWS Lambda function on the Java runtime or running code in an embedded JVM.

The following example configures the AWS Region globally for all service clients unless you explicitly override the value in code.

Linux, macOS, or Unix

```
$ export JAVA_TOOL_OPTIONS="-Daws.region=us-east-1"
```

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 JAVA_TOOL_OPTIONS -Daws.region=us-east-1
```

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.

At runtime

You can also set JVM system properties at runtime in code by using the `System.setProperty` method as shown in the following example.

```
System.setProperty("aws.region", "us-east-1");
```

> **Important**
>
> Set any JVM system properties *before* you initialize SDK service clients, otherwise service clients may use other values.

Authentication and access using AWS SDKs and tools

When you develop an AWS SDK application or use AWS tools to use AWS services, you must establish how your code or tool authenticates with AWS. You can configure programmatic access to AWS resources in different ways, depending on the environment the code runs in and the AWS access available to you.

The options below are a part of the credential provider chain. This means that by configuring your shared AWS `config` and `credentials` files accordingly, your AWS SDK or tool will automatically discover and use that method of authentication.

Choose a method to authenticate your application code

Choose a method to authenticate the calls made to AWS by your application.

Are you running code INSIDE an AWS service (such as Amazon EC2, Lambda, Amazon ECS, Amazon EKS, CodeBuild)?

If your code runs on AWS, credentials can be made automatically available to your application. For example, if your application is hosted on Amazon Elastic Compute Cloud, and there is an IAM role associated with that resource, the credentials are automatically made available to your application. Likewise, if you use Amazon ECS or Amazon EKS containers, the credentials set for the IAM role can be automatically obtained by the code running inside the container through the SDK's credential provider chain.

Is your code in an Amazon Elastic Compute Cloud instance?

Using IAM roles to authenticate applications deployed to Amazon EC2 – Use IAM roles to securely run your application on an Amazon EC2 instance.

Is your code in an AWS Lambda function?

Lambda creates an execution role with minimal permissions when you create a Lambda function. The AWS SDK or tool then automatically uses the IAM role attached to the Lambda at runtime, via the Lambda execution environment.

Is your code in Amazon Elastic Container Service (on Amazon EC2 or AWS Fargate for Amazon ECS)?

Use IAM Role for Task. You must create a task role and specify that role in your Amazon ECS task definition. The AWS SDK or tool then automatically uses the IAM role assigned to the task at runtime, via the Amazon ECS metadata.

Is your code in Amazon Elastic Kubernetes Service?

We recommend you use Amazon EKS Pod Identities.

Note: If you feel that IAM roles for service accounts (IRSA) might better suit your unique needs, see Comparing EKS Pod Identity and IRSA in the Amazon EKS User Guide.

Is your code running in AWS CodeBuild

See Using identity-based policies for CodeBuild.

Is your code in another AWS service?

See the dedicated guide for your AWS service. When you run code on AWS, the SDK credential provider chain can automatically obtain and refresh credentials for you.

Are you creating mobile applications or client-based web applications?

If you are creating mobile applications or client-based web applications that require access to AWS, build your app so that it requests temporary AWS security credentials dynamically by using web identity federation.

With web identity federation, you don't need to create custom sign-in code or manage your own user identities. Instead, app users can sign in using a well-known external identity provider (IdP), such as Login with Amazon, Facebook, Google, or any other OpenID Connect (OIDC)-compatible IdP. They can receive an authentication token, and then exchange that token for temporary security credentials in AWS that map to an IAM role with permissions to use the resources in your AWS account.

To learn how to configure this for your SDK or tool, see Assuming a role with web identity or OpenID Connect to authenticate AWS SDKs and tools.

For mobile applications, consider using Amazon Cognito. Amazon Cognito acts as an identity broker and does much of the federation work for you. For more information, see Using Amazon Cognito for mobile apps in the *IAM User Guide*.

Are you developing and running the code LOCALLY?

We recommend Using console credentials to authenticate AWS SDKs and tools.

After a quick browser-based authentication flow, AWS automatically generates temporary credentials that work across local development tools like the AWS CLI, AWS Tools for PowerShell and AWS SDKs.

If you use Identity Center for AWS account access

Use IAM Identity Center to authenticate AWS SDK and tools if you already have access to AWS accounts and/or need to manage access for your workforce. 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 IAM Identity Center, 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 Using IAM Identity Center to authenticate AWS SDK and toolsIAM Identity Center endpoints and quotas in the Amazon Web Services General Reference.

If you are looking for other ways to authenticate

Create a least-privileged IAM user with permissions to `sts:AssumeRole` into your target role. Then configure your profile to assume a role using a `source_profile` set up for that user.

You can also use temporary IAM credentials via environment variables or the shared AWS credentials file. See Using short-term credentials to authenticate AWS SDKs and tools.

Note: In sandbox or learning environments only, you can consider Using long-term credentials to authenticate AWS SDKs and tools.

Is this code running on-premise or in a hybrid/on-demand VM (such as server that reads from or writes to Amazon S3, or Jenkins deploying to the cloud)?

Are you using X.509 client certificates?

Yes: See Using IAM Roles Anywhere to authenticate AWS SDKs and tools. You can use IAM Roles Anywhere to obtain temporary security credentials in IAM for workloads such as servers, containers, and applications that run outside of AWS. To use IAM Roles Anywhere, your workloads must use X.509 certificates.

Can the environment securely connect to a federated identity provider (such as Microsoft Entra or Okta) to request temporary AWS credentials?

Yes: Use Process credential provider

Use Process credential provider to retrieve credentials automatically at runtime. These systems might use a helper tool or plugin to obtain the credentials, and might assume an IAM role behind the scenes using `sts:AssumeRole`.

No: Use temporary credentials injected via AWS Secrets Manager

Use temporary credentials injected via AWS Secrets Manager. For options to obtain short-lived access keys, see Request temporary security credentials in the *IAM User Guide*. For options on storing these temporary credentials, see AWS access keys.

You can use these credentials to securely retrieve broader application permissions from Secrets Manager, where your production secrets or long-lived role-based credentials can be stored.

Are you using a third-party tool not in AWS?

Use the documentation written by your third-party provider for best guidance on obtaining credentials.

If your third-party has not provided documentation, can you inject temporary credentials securely?

Yes: Use environment variables and temporary AWS STS credentials.

No: Use static access keys stored in encrypted secret manager (last resort).

Authentication methods

Authentication methods for code running within an AWS environment

If your code runs on AWS, credentials can be made automatically available to your application. For example, if your application is hosted on Amazon Elastic Compute Cloud, and there is an IAM role associated with that resource, the credentials are automatically made available to your application. Likewise, if you use Amazon ECS or Amazon EKS containers, the credentials set for the IAM role can be automatically obtained by the code running inside the container through the SDK's credential provider chain.

- Using IAM roles to authenticate applications deployed to Amazon EC2 – 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.
  - To try cloud-based collaboration space for software development teams, consider using Amazon CodeCatalyst.

Authentication through a web-based identity provider - Mobile or client-based web applications

If you are creating mobile applications or client-based web applications that require access to AWS, build your app so that it requests temporary AWS security credentials dynamically by using web identity federation.

With web identity federation, you don't need to create custom sign-in code or manage your own user identities. Instead, app users can sign in using a well-known external identity provider (IdP), such as Login with Amazon, Facebook, Google, or any other OpenID Connect (OIDC)-compatible IdP. They can receive an authentication token, and then exchange that token for temporary security credentials in AWS that map to an IAM role with permissions to use the resources in your AWS account.

To learn how to configure this for your SDK or tool, see Assuming a role with web identity or OpenID Connect to authenticate AWS SDKs and tools.

For mobile applications, consider using Amazon Cognito. Amazon Cognito acts as an identity broker and does much of the federation work for you. For more information, see Using Amazon Cognito for mobile apps in the *IAM User Guide*.

Authentication methods for code running locally (not in AWS)

- Using console credentials to authenticate AWS SDKs and tools – This feature works with both AWS Command Line Interface and Tools for PowerShell and gives you refreshable credentials that work across local development tools like the AWS CLI, Tools for PowerShell and AWS.
- Using IAM Identity Center to authenticate AWS SDK and tools – 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, 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 endpoints and quotas in the *Amazon Web Services General Reference*.

- Using IAM Roles Anywhere to authenticate AWS SDKs and tools – You can use IAM Roles Anywhere to obtain temporary security credentials in IAM for workloads such as servers, containers, and applications that run outside of AWS. To use IAM Roles Anywhere, your workloads must use X.509 certificates.
- Assuming a role with AWS credentials to authenticate AWS SDKs and tools – You can assume an IAM role to temporarily access AWS resources that you might not have access to otherwise.
- Using AWS access keys to authenticate AWS SDKs and tools – Other options that might be less convenient or might increase the security risk to your AWS resources.

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) – Understand 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 trusted identity propagation (TIP) plugin to access AWS services

- Using the TIP plugin to access AWS services – If you are creating an application for Amazon Q Business or other service that supports trusted identity propagation, and are using the AWS SDK for Java or the AWS SDK for JavaScript, you can use the TIP plugin for a streamlined authorization experience.

AWS Builder ID

Your AWS Builder ID complements any AWS accounts you might already own or want to create. While an AWS account acts as a container for AWS resources you create and provides a security

boundary for those resources, your AWS Builder ID represents you as an individual. You can sign in with your AWS Builder ID to access developer tools and services such as Amazon Q and Amazon CodeCatalyst.

- Sign in with AWS Builder ID in the *AWS Sign-In User Guide* – Learn how to create and use an AWS Builder ID and learn what the Builder ID provides.
- CodeCatalyst concepts - AWS Builder ID in the *Amazon CodeCatalyst User Guide* – Learn how CodeCatalyst uses an AWS Builder ID.

Using console credentials to authenticate AWS SDKs and tools

Using console credentials is the recommended method of providing AWS credentials when developing an AWS application in your local environment or other non-AWS compute service environments. If you are developing on an AWS resource, such as Amazon Elastic Compute Cloud (Amazon EC2) or AWS CloudShell, we recommend getting credentials from that service instead.

You can also authenticate through IAM Identity Center Using IAM Identity Center to authenticate AWS SDK and tools. This option is a common way for organizations to manage access for their workforce and requires Identity Center to be enabled.

How does it work?

Login for AWS local development using console credentials lets you use your existing AWS Management Console sign-in credentials for programmatic access to AWS services. After a browser-based authentication flow, AWS generates temporary credentials that work across local development tools like the AWS CLI, Tools for PowerShell and AWS SDKs. This feature simplifies the process of configuring and managing AWS CLI credentials, especially if you prefer interactive authentication over managing long-term access keys.

With this process, you can authenticate using your root credentials created during initial account setup, IAM users, or a federated identity from your identity provider.

If you use SDKs for development, the SDK clients will use the temporary credentials through the AWS SDKs and Tools standardized credential providers. You can also configure the Login credentials provider.

Authenticating via the login command is supported by both AWS CLI and Tools for PowerShell:

- Login for AWS local development using console credentials

- Login using console credentials in the AWS Tools for PowerShell user guide

Using IAM Identity Center to authenticate AWS SDK and tools

AWS IAM Identity Center can be used to provide AWS credentials when developing an AWS application on a non-AWS compute service environments. 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 instead.

Use IAM Identity Center authentication if you already use Identity Center for AWS account access or need to manage access for an organization.

In this tutorial, you establish IAM Identity Center access and will configure it for your SDK or tool by using the AWS access portal and the AWS CLI.

- The AWS access portal is the web location where you manually sign in to the IAM Identity Center. The format of the URL is `d-xxxxxxxxxx.awsapps.com/start` or *`your_subdomain`*`.awsapps.com/start`. When signed in to the AWS access portal, you can view AWS accounts and roles that have been configured for that user. This procedure uses the AWS access portal to get configuration values you need for the SDK/tool authentication process.
- The AWS CLI is used to configure your SDK or tool to use IAM Identity Center authentication for API calls made by your code. This one-time process updates your shared AWS `config` file, that is then used by your SDK or tool when you run your code.

Prerequisites

Before starting this procedure, you should have completed the following:

- If you do not have an AWS account, sign up for an AWS account.
- If you haven't enabled IAM Identity Center yet, enable IAM Identity Center by following the instructions in the *AWS IAM Identity Center User Guide*.

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

1. Add a user and add administrative permissions by following the Configure user access with the default IAM Identity Center directory procedure in the *AWS IAM Identity Center User Guide*.
2. The `AdministratorAccess` permission set should not be used for regular development. Instead, we recommend using the predefined `PowerUserAccess` permission set, unless your employer has created a custom permission set for this purpose.

   Follow the same Configure user access with the default IAM Identity Center directory procedure again, but this time:

   - Instead of creating the *`Admin team`* group, create a *`Dev team`* group, and substitute this thereafter in the instructions.
   - You can use the existing user, but the user must be added to the new *`Dev team`* group.
   - Instead of creating the *`AdministratorAccess`* permission set, create a *`PowerUserAccess`* permission set, and substitute this thereafter in the instructions.

   When you are done, you should have the following:

   - A `Dev team` group.
   - An attached `PowerUserAccess` permission set to the `Dev team` group.
   - Your user added to the `Dev team` group.
3. Exit the portal and sign in again to see your AWS accounts and options for `Administrator` or `PowerUserAccess`. Select `PowerUserAccess` when working with your tool/SDK.

I already have access to AWS through a federated identity provider managed by my employer (such as Microsoft Entra 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.

   a. See Installing or updating the latest version of the AWS CLI in the *AWS Command Line Interface User Guide*.

   b. (Optional) To verify that the AWS CLI is working, open a command prompt and run the `aws --version` command.

2. Sign in to the AWS access portal. Your employer may provide this URL or you may get it in an email following **Step 1: Establish access**. If not, find your **AWS access portal URL** on the **Dashboard** of https://console.aws.amazon.com/singlesignon/.

   a. In the AWS access portal, in the **Accounts** tab, select the individual account to manage. The roles for your user are displayed. Choose **Access keys** to get credentials for command line or programmatic access for the appropriate permission set. Use the predefined `PowerUserAccess` permission set, or whichever permission set you or your employer has created to apply least-privilege permissions for development.

   b. In the **Get credentials** dialog box, choose either **MacOS and Linux** or **Windows**, depending on your operating system.

   c. Choose the **IAM Identity Center credentials** method to get the `Issuer URL` and `SSO Region` values that you need for the next step. Note: `SSO Start URL` can be used interchangeably with `Issuer URL`.

3. In the AWS CLI command prompt, run the `aws configure sso` command. When prompted, enter the configuration values that you collected in the previous step. For details on this AWS CLI command, see Configure your profile with the `aws configure sso` wizard.

   a. For the prompt `SSO Start URL`, enter the value you obtained for `Issuer URL`.
   b. 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.
4. (Optional) In the AWS CLI command prompt, 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.
5. 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 AWS access keys. You must remove any static credentials before the SDK or tool will use the IAM Identity Center credentials because of the Understand the credential provider chain precedence.

For a deep dive into how the SDKs and tools use and refresh credentials using this configuration, see How IAM Identity Center authentication is resolved for AWS SDKs and tools.

To configure IAM Identity Center provider settings directly in the shared `config` file, see IAM Identity Center credential provider in this guide.

Refreshing portal access sessions

Your access will eventually expire and the SDK or tool will encounter an authentication error. When this expiration occurs depends on your configured session lengths. 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 User Guide*:

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

- **Permission set session duration** – Set session duration

How IAM Identity Center authentication is resolved for AWS SDKs and tools

Relevant IAM Identity Center terms

The following terms help you understand the process and configuration behind AWS IAM Identity Center. 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 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 name	SDK API name	Description
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 credentia ls."
IAM Identity Center credentia l 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.

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 User Guide*.
  - The maximum session length has been extended to 90 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 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, 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.

Using IAM Roles Anywhere to authenticate AWS SDKs and tools

You can use IAM Roles Anywhere to get temporary security credentials in IAM for workloads such as servers, containers, and applications that run outside of AWS. To use IAM Roles Anywhere, your workloads must use X.509 certificates. Your Cloud Administrator should provide the certificate and private key needed to configure IAM Roles Anywhere as your credential provider.

Step 1: Configure IAM Roles Anywhere

IAM Roles Anywhere provides a way to get temporary credentials for a workload or process that runs outside of AWS. A trust anchor is established with the certificate authority to get temporary credentials for the associated IAM role. The role sets the permissions your workload will have when your code authenticates with IAM Roles Anywhere.

For steps to set up the trust anchor, IAM role, and IAM Roles Anywhere profile, see Creating a trust anchor and profile in AWS Identity and Access Management Roles Anywhere in the *IAM Roles Anywhere User Guide*.

> **Note**
>
> A *profile* in the *IAM Roles Anywhere User Guide* refers to a unique concept within the IAM Roles Anywhere service. It's not related to the profiles within the shared AWS `config` file.

Step 2: Use IAM Roles Anywhere

To get temporary security credentials from IAM Roles Anywhere, use the credential helper tool provided by IAM Roles Anywhere. The credential tool implements the signing process for IAM Roles Anywhere.

For instructions to download the credential helper tool, see Obtaining temporary security credentials from AWS Identity and Access Management Roles Anywhere in the *IAM Roles Anywhere User Guide*.

To use temporary security credentials from IAM Roles Anywhere with AWS SDKs and the AWS CLI, you can configure `credential_process` setting in the shared AWS `config` file. The SDKs and AWS CLI support a process credential provider that uses `credential_process` to authenticate. The following shows the general structure to set `credential_process`.

```
credential_process = [path to helper tool] [command] [--parameter1 value] [--
parameter2 value] [...]
```

The `credential-process` command of the helper tool returns temporary credentials in a standard JSON format that is compatible with the `credential_process` setting. Note that the command name contains a hyphen but the setting name contains an underscore. The command requires the following parameters:

- `private-key` – The path to the private key that signed the request.
- `certificate` – The path to the certificate.
- `role-arn` – The ARN of the role to get temporary credentials for.
- `profile-arn` – The ARN of the profile that provides a mapping for the specified role.
- `trust-anchor-arn` – The ARN of the trust anchor used to authenticate.

Your Cloud Administrator should provide the certificate and private key. All three ARN values can be copied from the AWS Management Console. The following example shows a shared `config` file that configures retrieving temporary credentials from the helper tool.

```
[profile dev]
credential_process = ./aws_signing_helper credential-process --certificate /
path/to/certificate --private-key /path/to/private-key --trust-anchor-
arn arn:aws:rolesanywhere:region:account:trust-anchor/TA_ID --profile-
arn arn:aws:rolesanywhere:region:account:profile/PROFILE_ID --role-
arn arn:aws:iam::account:role/ROLE_ID
```

For optional parameters and additional helper tool details, see IAM Roles Anywhere Credential Helper on GitHub.

For details on the SDK configuration setting itself and the process credential provider, see Process credential provider in this guide.

Assuming a role with AWS credentials to authenticate AWS SDKs and tools

Assuming a role involves using a set of temporary security credentials to access AWS resources that you might not have access to otherwise. These temporary credentials consist of an access key ID, a secret access key, and a security token. To learn more about AWS Security Token Service (AWS STS) API requests, see Actions in the *AWS Security Token Service API Reference*.

To set up your SDK or tool to assume a role, you must first create or identify a specific *role* to assume. IAM roles are uniquely identified by a role Amazon Resource Name (ARN). Roles establish trust relationships with another entity. The trusted entity that uses the role might be an AWS service or another AWS account. To learn more about IAM roles, see Using IAM roles in the *IAM User Guide*.

After the IAM role is identified, if you are trusted by that role, you can configure your SDK or tool to use the permissions that are granted by the role.

> **Note**
>
> It is an AWS best practice to use Regional endpoints whenever possible and to configure your AWS Region.

Assume an IAM role

When assuming a role, AWS STS returns a set of temporary security credentials. These credentials are sourced from another profile or from the instance or container that your code is running in. Most commonly this type of assuming a role is used when you have AWS credentials for one account, but your application needs access to resources in another account.

Step 1: Set up an IAM role

To set up your SDK or tool to assume a role, you must first create or identify a specific role to assume. IAM roles are uniquely identified using a role ARN. Roles establish trust relationships with another entity, typically within your account or for cross-account access. To set this up, see Creating IAM roles in the *IAM User Guide*.

Step 2: Configure the SDK or tool

Configure the SDK or tool to source credentials from `credential_source` or `source_profile`.

Use `credential_source` to source credentials from an Amazon ECS container, an Amazon EC2 instance, or from environment variables.

Use `source_profile` to source credentials from another profile. `source_profile` also supports role chaining, which is hierarchies of profiles where an assumed role is then used to assume another role.

When you specify this in a profile, the SDK or tool automatically makes the corresponding AWS STS `AssumeRole` API call for you. To retrieve and use temporary credentials by assuming a role, specify the following configuration values in the shared AWS `config` file. For more details on each of these settings, see the Assume role credential provider settings section.

- `role_arn` - From the IAM role you created in Step 1

- Configure either `credential_source` or `source_profile`
- (Optional) `duration_seconds`
- (Optional) `external_id`
- (Optional) `mfa_serial`
- (Optional) `role_session_name`

The following examples show the configuration of both assume role options in a shared `config` file:

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

```
[profile-with-user-that-can-assume-role]
aws_access_key_id=AKIAIOSFODNN7EXAMPLE
aws_secret_access_key=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
aws_session_token=IQoJb3JpZ2luX2IQoJb3JpZ2luX2IQoJb3JpZ2luX2IQoJb3JpZ2luX2IQoJb3JpZVERYLONGSTRI

[profile dev]
region = us-east-1
output = json
role_arn = arn:aws:iam::123456789012:role/my-role-name
source_profile = profile-with-user-that-can-assume-role
role_session_name = my_session
```

For details on all assume role credential provider settings, see Assume role credential provider in this guide.

Assuming a role with web identity or OpenID Connect to authenticate AWS SDKs and tools

Assuming a role involves using a set of temporary security credentials to access AWS resources that you might not have access to otherwise. These temporary credentials consist of an access key ID, a secret access key, and a security token. To learn more about AWS Security Token Service (AWS STS) API requests, see Actions in the *AWS Security Token Service API Reference*.

To set up your SDK or tool to assume a role, you must first create or identify a specific *role* to assume. IAM roles are uniquely identified by a role Amazon Resource Name (ARN). Roles establish

trust relationships with another entity. The trusted entity that uses the role might be a web identity provider or OpenID Connect(OIDC), or SAML federation. To learn more about IAM roles, see Methods to assume a role in the *IAM User Guide*.

After the IAM role is configured in your SDK, if that role is configured to trust your identity provider, you can further configure your SDK to assume that role in order to get temporary AWS credentials.

> **Note**
>
> It is an AWS best practice to use Regional endpoints whenever possible and to configure your AWS Region.

Federate with web identity or OpenID Connect

You can use the JSON Web Tokens (JWTs) from public identity providers, such as Login With Amazon, Facebook, Google to get temporary AWS credentials using `AssumeRoleWithWebIdentity`. Depending on how they are used, these JWTs may be called ID tokens or access tokens. You may also use JWTs issued from identity providers (IdPs) that are compatible with OIDC's discovery protocol, such as EntraId or PingFederate.

If you are using Amazon Elastic Kubernetes Service, this feature provides the ability to specify different IAM roles for each one of your service accounts in an Amazon EKS cluster. This Kubernetes feature distributes JWTs to your pods which are then used by this credential provider to obtain temporary AWS credentials. For more information on this Amazon EKS configuration, see IAM roles for service accounts in the Amazon EKS User Guide. However, for a simpler option, we recommend you use Amazon EKS Pod Identities instead if your SDK supports it.

Step 1: Set up an identity provider and IAM role

To configure federation with an external IdP, use an IAM identity provider to inform AWS about the external IdP and its configuration. This establishes *trust* between your AWS account and the external IdP. Before configuring the SDK to use the JSON Web Token (JWT) for authentication, you must first set up the identity provider (IdP) and the IAM role used to access it. To set these up, see Creating a role for web identity or OpenID Connect Federation (console) in the *IAM User Guide*.

Step 2: Configure the SDK or tool

Configure the SDK or tool to use a JSON Web Token (JWT) from AWS STS for authentication.

When you specify this in a profile, the SDK or tool automatically makes the corresponding AWS STS `AssumeRoleWithWebIdentity` API call for you. To retrieve and use temporary credentials using web identity federation, specify the following configuration values in the shared AWS `config` file. For more details on each of these settings, see the Assume role credential provider settings section.

- `role_arn` - From the IAM role you created in Step 1
- `web_identity_token_file` - From the external IdP
- (Optional) `duration_seconds`
- (Optional) `role_session_name`

The following is an example of a shared `config` file configuration to assume a role with web identity:

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

> **Note**
>
> For mobile applications, consider using Amazon Cognito. Amazon Cognito acts as an identity broker and does much of the federation work for you. However, the Amazon Cognito identity provider isn't included in the SDKs and tools core libraries like other identity providers. To access the Amazon Cognito API, include the Amazon Cognito service client in the build or libraries for your SDK or tool. For usage with AWS SDKs, see Code Examples in the *Amazon Cognito Developer Guide*.

For details on all assume role credential provider settings, see Assume role credential provider in this guide.

Using AWS access keys to authenticate AWS SDKs and tools

Using AWS access keys is an option for authentication when using AWS SDKs and tools.

Use short-term credentials

We recommend configuring your SDK or tool to use Using IAM Identity Center to authenticate AWS SDK and tools to use extended session duration options.

However, to set up the SDK or tool's temporary credentials directly, see Using short-term credentials to authenticate AWS SDKs and tools.

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.

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, you might create a test 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 Using long-term credentials to authenticate AWS SDKs and tools.

Using short-term credentials to authenticate AWS SDKs and tools

We recommend configuring your AWS SDK or tool to use Using IAM Identity Center to authenticate AWS SDK and tools 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.

Best practice: Instead of manually managing access keys and a token in the credentials file, we recommend your application uses temporary credentials delivered from:

- An AWS compute service, such as running your application on Amazon Elastic Compute Cloud or in AWS Lambda.
- Another option in the credential provider chain, such as Using IAM Identity Center to authenticate AWS SDK and tools.

- Or use the Process credential provider to retrieve temporary credentials.

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 for Manual credential refresh to copy IAM role credentials from the AWS access portal.

   a. For step 4 in the linked instructions, choose the IAM role name that grants access for your development needs. This role typically has a name like **PowerUserAccess** or **Developer**.

   b. For step 7 in the linked instructions, select the **Manually add a profile to your AWS credentials file** option and copy the contents.

6. Paste the copied credentials into your local `credentials` file. The generated profile name is not needed if you are using the `default` profile. Your file should resemble the following.

```
[default]
aws_access_key_id=AKIAIOSFODNN7EXAMPLE
aws_secret_access_key=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
aws_session_token=IQoJb3JpZ2luX2IQoJb3JpZ2luX2IQoJb3JpZ2luX2IQoJb3JpZ2luX2IQoJb3JpZVERYLONG
```

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.

Using long-term credentials to authenticate AWS SDKs and tools

> **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.

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.

Important warnings and guidance for credentials

Warnings for credentials

- ***Do NOT*** use your account's root credentials to access AWS resources. These credentials provide unrestricted account access and are difficult to revoke.
- ***Do NOT*** put literal access keys or credential information in your application files. If you do, you create a risk of accidentally exposing your credentials if, for example, you upload the project to a public repository.
- ***Do NOT*** include files that contain credentials in your project area.
- Be aware that any credentials stored in the shared AWS `credentials` file are stored in plaintext.

Additional guidance for securely managing credentials

For a general discussion of how to securely manage AWS credentials, see Best practices for managing AWS access keys in the AWS General Reference. In addition to that discussion, consider the following:

- Use IAM roles for tasks for Amazon Elastic Container Service (Amazon ECS) tasks.
- Use IAM roles for applications that are running on Amazon EC2 instances.

Prerequisites: Create an AWS account

To use an IAM user to access AWS services, you need an AWS account and AWS credentials.

1. **Create an account.**

   To create an AWS account, see Getting started: Are you a first-time AWS user? in the *AWS Account Management Reference Guide*.

2. **Create an administrative user.**

   Avoid using your root user account (the initial account you create) to access the management console and services. Instead, create an administrative user account, as explained in Create an administrative user in the *IAM User Guide*.

   After you create the administrative user account and record the login details, **be sure to sign out of your root user account** and sign back in using the administrative account.

Neither of these accounts are appropriate for doing development on AWS or for running applications on AWS. As a best practice, you need to create users, permission sets, or service roles that are appropriate for these tasks. For more information, see Apply least-privilege permissions in the *IAM User Guide*.

Step 1: Create your IAM user

- Create your IAM user by following the Creating IAM users (console) procedure in the *IAM User Guide*. When creating your IAM user:
  - We recommend you select **Provide user access to the AWS Management Console**. This allows you to view AWS services related to the code that you are running in a visual environment, such as checking AWS CloudTrail diagnostic logs or uploading files to Amazon Simple Storage Service, which is helpful when debugging your code.
  - For **Set permissions** - **Permission options**, select **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 the optional steps of that procedure regarding setting permission boundaries or tags.

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.

   > **Warning**
   >
   > Use appropriate security measures to keep these credentials safe.

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. These can also be set as environment variables, see AWS access keys for environment variable names. 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 to authenticate applications deployed to Amazon EC2

This example covers setting up an AWS Identity and Access Management role with Amazon S3 access to use in your application deployed to an Amazon Elastic Compute Cloud instance.

In order to run your AWS SDK application on an Amazon Elastic Compute Cloud instance, create an IAM role, and then give your Amazon EC2 instance access to that role. For more information, see IAM Roles for Amazon EC2 in the *Amazon EC2 User Guide* .

Create an IAM role

The AWS SDK application that you develop likely accesses at least one AWS service to perform actions. Create an IAM role that grants the required permissions necessary for your application to run.

This procedure creates a role that grants read-only access to Amazon S3 as an example. Many of the AWS SDK guides have "getting started" tutorials that read from 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 create your Amazon EC2 instance.*

Launch an Amazon EC2 instance and specify your IAM role

You can create and launch an Amazon EC2 instance using your IAM role by doing the following:

- Follow Quickly launch an instance in the *Amazon EC2 User Guide*. However, prior to the final submission step, also do the following:
  - Under **Advanced details**, for **IAM Instance profile**, choose the role that you created in the previous step.

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

Connect to the EC2 instance

Connect to the Amazon EC2 instance so that you can transfer your application to it and then run the application. You'll need the file that contains the private portion of the key pair you used under **Key pair (login)** when you created your instance; that is, the PEM file.

You can do this by following the guidance for your instance type: Connect to your Linux instance or Connect to your Windows instance. When you connect, do so in such a way that you can transfer files from your development machine to your instance.

> **Note**
>
> On Linux or macOS terminal, you can use the secure copy command to copy your application. To use `scp` with a key pair, you can use the following command:
> `scp -i path/to/key file/to/copy ec2-user@ec2-xx-xx-xxx-xxx.compute.amazonaws.com:~ .`
> For more information for Windows, see Transfer files to Windows instances.

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 your application on the EC2 instance

1. Copy your application files from your local drive to your Amazon EC2 instance.
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.
c. Choose **Actions**, **Security**, and then choose **Modify IAM role**.
d. For **IAM role**, detach the IAM role by choosing **No IAM Role**.
e. Choose **Update IAM role**.
f. Run the application again and confirm that it returns an authorization error.

Using the TIP plugin to access AWS services

Trusted identity propagation (TIP) is a feature of AWS IAM Identity Center that enables administrators of AWS services to grant permissions based on user attributes such as group associations. With trusted identity propagation, identity context is added to an IAM role to identify the user requesting access to AWS resources. This context is propagated to other AWS services.

Identity context comprises information that AWS services use to make authorization decisions when they receive access requests. This information includes metadata that identifies the requester (for example, an IAM Identity Center user), the AWS service to which access is requested (for example, Amazon Redshift), and the scope of access (for example, read only access). The receiving AWS service uses this context, and any permissions assigned to the user, to authorize access to its resources. For more information, see in the Trusted identity propagation overview in the AWS IAM Identity Center User Guide.

The TIP plugin can be used with AWS services that support trusted identity propagation. As a reference use case, see Configuring an Amazon Q Business application using AWS IAM Identity Center in the *Amazon Q Business User Guide*.

> **Note**
>
> If you are using Amazon Q Business, see Configuring an Amazon Q Business application using AWS IAM Identity Center for service-specific instructions.

Prerequisites for using the TIP plugin

The following resources are required in order for the plugin to work:

1. You must be using either the AWS SDK for Java or the AWS SDK for JavaScript.
2. Verify that the service you are using supports the trusted identity propagation.

   See the **Enables trusted identity propagation through IAM Identity Center** column of the AWS managed applications that integrate with IAM Identity Center table in the *AWS IAM Identity Center User Guide*.
3. Enable IAM Identity Center and trusted identity propagation.

   See TIP prerequisites and considerations in the *AWS IAM Identity Center User Guide*.
4. You must have an Identity-Center-integrated application.

   See AWS managed applications or Customer managed applications in the *AWS IAM Identity Center User Guide*.
5. You must set up a trusted token issuer (TTI) and connect your service to IAM Identity Center.

   See Prerequisites for trusted token issuers and Tasks for setting up a trusted token issuer in the *AWS IAM Identity Center User Guide*.

To use the TIP plugin in your code

1. Create an instance of the trusted identity propagation plugin.
2. Create a service client instance for interacting with your AWS service and customize the service client by adding the trusted identity propagation plugin.

The TIP plugin takes the following input parameters:

- **`webTokenProvider`**: A function that the customer implements to obtain an OpenID token from their external identity provider.
- **`accessRoleArn`**: The IAM role ARN to be assumed by the plugin with the user's identity context to get the identity-enhanced credentials.
- **`applicationArn`**: The unique identifier string for the client or application. This value is an application ARN that has OAuth grants configured.
- **`ssoOidcClient`**: (Optional) An SSO OIDC client, such as `SsoOidcClient` for Java or `client-sso-oidc` for JavaScript, with customer-defined configurations. If not provided, an OIDC client using `applicationRoleArn` will be instantiated and used.

- **`stsClient`**: (Optional) An AWS STS client with customer-defined configurations, used to assume `accessRoleArn` with the user's identity context. If not provided, an AWS STS client using `applicationRoleArn` will be instantiated and used.
- **`applicationRoleArn`**: (Optional) The IAM role ARN to be assumed with `AssumeRoleWithWebIdentity` so that the OIDC and AWS STS clients can be bootstrapped.
  - If not provided, **both** of the `ssoOidcClient` and `stsClient` parameters must be provided.
  - If provided, `applicationRoleArn` can't be the same value as the `accessRoleArn` parameter. `applicationRoleArn` is used to build the stsClient, which is used to assume accessRole. If the same role is used for both `applicationRole` and `accessRole`, it would mean using a role to assume itself (self-role assumption), which is discouraged by AWS. See the announcement for more details.

Considerations for `ssoOidcClient`, `stsClient`, and `applicationRoleArn` parameters

When configuring the TIP plugin, consider the following permission requirements based on which parameters you provide:

- If you are providing `ssoOidcClient` and `stsClient`:
  - Credentials on the `ssoOidcClient` should have `oauth:CreateTokenWithIAM` permission for calling identity center to get the identity center specific user context.
  - Credentials on `stsClient` should have `sts:AssumeRole`, and `sts:SetContext` permissions on `accessRole`. `accessRole` also needs to be configured with a trust relationship with the credentials on `stsClient`.
- If you are providing `applicationRoleArn`:
  - `applicationRole` should have the `oauth:CreateTokenWithIAM`, `sts:AssumeRole` and `sts:SetContext` permissions on the required resources (IdC instance, `accessRole`) as it will be used to build OIDC and STS clients.
  - `applicationRole` should have a trust relationship with the identity provider that is used to generate the `webToken`, as the `webToken` will be used to assume the applicationRole via the AssumeRoleWithWebIdentity call by the plugin.

Example ApplicationRole configuration:

Trust Policy with Web token provider:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Federated": "arn:aws:iam::ACCOUNT_ID:oidc-provider/
IDENTITY_PROVIDER_URL"
            },
            "Action": "sts:AssumeRoleWithWebIdentity",
            "Condition": {
                "StringEquals": {
                    "IDENTITY_PROVIDER_URL:aud": "CLIENT_ID_TO_BE_TRUSTED"
                }
            }
        }
    ]
}
```

Permission Policy:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "sts:AssumeRole",
                "sts:SetContext"
            ],
            "Resource": [
                "accessRoleArn"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "sso-oauth:CreateTokenWithIAM"
            ],
            "Resource": [
                "*"
            ]
        }
```

```
    ]
}
```

Code examples using TIP

The examples below show how to implement the TIP plugin in your code using the AWS SDK for Java or the AWS SDK for JavaScript.

Java

To use the TIP plugin in your AWS SDK for Java project, you need to declare it as a dependency in your project's `pom.xml` file.

```
<dependency>
<groupId>software.amazon.awsidentity.trustedIdentityPropagation</groupId>
<artifactId>aws-sdk-java-trustedIdentityPropagation-java-plugin</artifactId>
   <version>2.0.0</version>
</dependency>
```

In your source code, include the required package statement for `software.amazon.awssdk.trustedidentitypropagation`.

The following examples show two ways to create an instance of the trusted identity propagation plugin and add it to a service client. Both examples use Amazon S3 as the service and utilize `S3AccessGrantsPlugin` to manage user specific permissions, but can be applied to any AWS service that supports trusted identity propagation (TIP).

> **Note**
>
> For these examples, you need to setup the user specific permissions from S3 Access Grants. Refer to the S3 Access Grants documentation for more details.

Option 1: Build and pass OIDC and STS clients

```
SsoOidcClient oidcClient = SsoOidcClient.builder()
    .region(Region.US_EAST_1)
    .credentialsProvider(credentialsProvider).build();

StsClient stsClient = StsClient.builder()
```

```
    .region(Region.US_EAST_1)
    .credentialsProvider(credentialsProvider).build();

TrustedIdentityPropagationPlugin trustedIdentityPropagationPlugin =
 TrustedIdentityPropagationPlugin.builder()
        .webTokenProvider(() -> webToken)
        .applicationArn(idcApplicationArn)
        .accessRoleArn(accessRoleArn)
        .ssoOidcClient(oidcClient)
        .stsClient(stsClient)
        .build();

S3AccessGrantsPlugin accessGrantsPlugin = S3AccessGrantsPlugin.builder()
        .build();

S3Client s3Client =
        S3Client.builder().region(Region.US_EAST_1)
                .crossRegionAccessEnabled(true)
                .addPlugin(trustedIdentityPropagationPlugin)
                .addPlugin(accessGrantsPlugin)
                .build();

final var resp = s3Client.getObject(GetObjectRequest.builder()
        .key("path/to/object/fileName")
        .bucket("bucketName")
        .build());
```

Option 2: Pass applicationRoleArn and defer client creation to the plugin

```
TrustedIdentityPropagationPlugin trustedIdentityPropagationPlugin =
 TrustedIdentityPropagationPlugin.builder()
        .webTokenProvider(() -> webToken)
        .applicationArn(idcApplicationArn)
        .accessRoleArn(accessRoleArn)
        .applicationRoleArn(applicationRoleArn)
        .build();

S3AccessGrantsPlugin accessGrantsPlugin = S3AccessGrantsPlugin.builder()
        .build();

S3Client s3Client =
        S3Client.builder().region(Region.US_EAST_1)
                .crossRegionAccessEnabled(true)
```

```
                .addPlugin(trustedIdentityPropagationPlugin)
                .addPlugin(accessGrantsPlugin)
                .build();

final var resp = s3Client.getObject(GetObjectRequest.builder()
        .key("path/to/object/fileName")
        .bucket("bucketName")
        .build());
```

For additional details and source, see trusted-identity-propagation-java on GitHub.

JavaScript

Run the following command to install the TIP authentication plugin package in your AWS SDK for JavaScript project:

```
$  npm i @aws-sdk-extension/trusted-identity-propagation
```

The final `package.json` should include a dependency similar to the following:

```
  "dependencies": {
"@aws-sdk-extension/trusted-identity-propagation": "^2.0.0"
  },
```

In your source code, import the required `TrustedIdentityPropagationExtension` dependency.

The following examples show two ways to create an instance of the trusted identity propagation plugin and add it to a service client. Both examples use Amazon S3 as the service and utilize Amazon S3 Access Grants to manage user specific permissions, but can be applied to any AWS service that supports trusted identity propagation (TIP).

> **Note**
>
> For these examples, you need to setup the user specific permissions from Amazon S3 Access Grants, refer to the Amazon S3 Access Grants documentation for more details.

Option 1: Build and pass OIDC and STS clients

```
import { S3Client, GetObjectCommand } from "@aws-sdk/client-s3";
```

```
import { S3ControlClient, GetDataAccessCommand } from "@aws-sdk/client-s3-control";
import { TrustedIdentityPropagationExtension } from "@aws-sdk-extension/trusted-
identity-propagation";

const s3ControlClient = new S3ControlClient({
    region: "us-east-1",
    extensions: [
        TrustedIdentityPropagationExtension.create({
            webTokenProvider: async () => {
                return 'ID_TOKEN_FROM_YOUR_IDENTITY_PROVIDER';
            },
            ssoOidcClient: customOidcClient,
            stsClient: customStsClient,
            accessRoleArn: accessRoleArn,
            applicationArn: applicationArn,
        }),
    ],
});

const getDataAccessParams = {
  Target: "S3_URI_PATH",
  Permission: "READ",
  AccountId: ACCOUNT_ID,
  InstanceArn: S3_ACCESS_GRANTS_ARN,
  TargetType: "Object",
};

try {
  const command = new GetDataAccessCommand(getDataAccessParams);
  const response = await s3ControlClient.send(command);

  const credentials = response.Credentials;

  // Create a new S3 client with the temporary credentials
  const temporaryS3Client = new S3Client({
    region: "us-east-1",
    credentials: {
      accessKeyId: credentials.AccessKeyId,
      secretAccessKey: credentials.SecretAccessKey,
      sessionToken: credentials.SessionToken,
    },
  });

  // Use the temporary S3 client to perform the operation
```

```
  const s3Params = {
    Bucket: "BUCKET_NAME",
    Key: "S3_OBJECT_KEY",
  };
  const getObjectCommand = new GetObjectCommand(s3Params);
  const s3Object = await temporaryS3Client.send(getObjectCommand);

  const fileContent = await s3Object.Body.transformToString();

  // Process the S3 object data
  console.log("Successfully retrieved S3 object:", fileContent);
} catch (error) {
  console.error("Error accessing S3 data:", error);
}
```

Option 2: Pass applicationRoleArn and defer client creation to the plugin

```
import { S3Client, GetObjectCommand } from "@aws-sdk/client-s3";
import { S3ControlClient, GetDataAccessCommand } from "@aws-sdk/client-s3-control";
import { TrustedIdentityPropagationExtension } from "@aws-sdk-extension/trusted-
identity-propagation";

const s3ControlClient = new S3ControlClient({
    region: "us-east-1",
    extensions: [
        TrustedIdentityPropagationExtension.create({
            webTokenProvider: async () => {
                return 'ID_TOKEN_FROM_YOUR_IDENTITY_PROVIDER';
            },
            accessRoleArn: accessRoleArn,
            applicationRoleArn: applicationRoleArn,
            applicationArn: applicationArn,
        }),
    ],
});

// Same S3 AccessGrants workflow as Option 1
const getDataAccessParams = {
  Target: "S3_URI_PATH",
  Permission: "READ",
  AccountId: ACCOUNT_ID,
  InstanceArn: S3_ACCESS_GRANTS_ARN,
  TargetType: "Object",
```

```
};

try {
  const command = new GetDataAccessCommand(getDataAccessParams);
  const response = await s3ControlClient.send(command);

  const credentials = response.Credentials;

  const temporaryS3Client = new S3Client({
    region: "us-east-1",
    credentials: {
      accessKeyId: credentials.AccessKeyId,
      secretAccessKey: credentials.SecretAccessKey,
      sessionToken: credentials.SessionToken,
    },
  });

  const s3Params = {
    Bucket: "BUCKET_NAME",
    Key: "S3_OBJECT_KEY",
  };
  const getObjectCommand = new GetObjectCommand(s3Params);
  const s3Object = await temporaryS3Client.send(getObjectCommand);

  const fileContent = await s3Object.Body.transformToString();

  console.log("Successfully retrieved S3 object:", fileContent);
} catch (error) {
  console.error("Error accessing S3 data:", error);
}
```

For additional details and source, see trusted-identity-propagation-js on GitHub.

AWS SDKs and tools settings reference

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.

You can find detailed information about configuration settings in the following sections:

- AWS SDKs and Tools standardized credential providers – Common credential providers standardized across multiple SDKs.
- AWS SDKs and Tools standardized features – Common features standardized across multiple SDKs.

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. In most SDKs, 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: The JVM system property for the setting is checked. 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 for the setting. If it's set, the client uses it.
5. The shared `config` file for the setting. If the setting is present, the SDK uses it.
   - The `AWS_PROFILE` environment variable or the `aws.profile` JVM system property can be used to specify which profile that the SDK loads.
6. Any default value provided by the SDK source code itself is used last.

> **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 specific guide for the SDK or tool that you are using.

The order determines which methods take precedence and override others. For example, if you set up a 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.

Understanding the settings pages of this guide

The pages within the **Settings reference** section of this guide detail the available settings that can be set through various mechanisms. The tables that follow list the config and credential file settings, environment variables, and (for Java and Kotlin SDKs) the JVM settings that can be used outside of your code to configure the feature. Each linked topic in each list takes you to the corresponding settings page.

- `Config` file settings list
- `Credentials` file settings list
- Environment variables list
- JVM system properties list

Each credential provider or feature has a page where the settings that are used to configure that functionality are listed. For each setting, you can often set the value either by adding the setting to a configuration file, or by setting an environment variable, or (for Java and Kotlin only) by setting a JVM system property. Each setting lists all supported methods of setting the value in a block above the details of the description. Although the precedence varies, the resulting functionality is the same regardless of how you set it.

The description will include the default value, if any, that takes effect if you do nothing. It also defines what a valid value is for that setting.

For example, let's look at a setting from the Request compression feature page.

The `disable_request_compression` example setting's information documents the following:

- There are three equivalent ways to control request compression outside of your codebase. You can either:
  - Set it in your config file using `disable_request_compression`
  - Set it as an environment variable using `AWS_DISABLE_REQUEST_COMPRESSION`
  - Or, if you are using the Java or Kotlin SDK, set it as a JVM system property using `aws.disableRequestCompression`

  > **Note**
  >
  > There might also be a way to configure the same functionality directly in your code, but this Reference does not cover this since it is unique to each SDK. If you want to set your configuration in the code itself, see your specific SDK guide or API reference.

- If you do nothing, the value will default to `false`.
- The only valid values for this Boolean setting are `true` and `false`.

At the bottom of each feature page there is a **Support by AWS SDKs and tools** table.

This table shows whether your SDK supports the settings that are listed on the page. The `Supported` column indicates the support level with the following values:

- `Yes` – The settings are fully supported by the SDK as written.
- `Partial` – Some of the settings are supported or the behavior deviates from the description. For `Partial`, an additional note indicates the deviation.
- No – None of the settings are supported. This doesn't make claims as to whether the same functionality might be achieved in code; it only indicates that the listed external configuration settings are not supported.

`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. SDKs and tools may also support unique settings and environment variables. To see the settings and environment variables supported by only an individual SDK or tool, see that specific SDK or tool guide.

Setting name	Details
`account_id_endpoint_mode`	Account-based endpoints
`api_versions`	General configuration settings
`auth_scheme_preference`	Authentication scheme
`aws_access_key_id`	AWS access keys
`aws_account_id`	Account-based endpoints
`aws_secret_access_key`	AWS access keys

Setting name	Details
aws_session_token	AWS access keys
ca_bundle	General configuration settings
credential_process	Process credential provider
credential_source	Assume role credential provider
defaults_mode	Smart configuration defaults
disable_host_prefix_injection	Host prefix injection
disable_request_compression	Request compression
duration_seconds	Assume role credential provider
ec2_metadata_service_endpoint	IMDS credential provider
ec2_metadata_service_endpoint_mode	IMDS credential provider
ec2_metadata_v1_disabled	IMDS credential provider

Setting name	Details
endpoint_discovery_enabled	Endpoint discovery
endpoint_url	Service-specific endpoints
external_id	Assume role credential provider
ignore_configured_endpoint_urls	Service-specific endpoints
max_attempts	Retry behavior
metadata_service_num_attempts	Amazon EC2 instance metadata
metadata_service_timeout	Amazon EC2 instance metadata
mfa_serial	Assume role credential provider
output	General configuration settings
parameter_validation	General configuration settings
region	AWS Region
request_checksum_calculation	Data Integrity Protections for Amazon S3

Setting name	Details
request_min_compression_size_bytes	Request compression
response_checksum_validation	Data Integrity Protections for Amazon S3
retry_mode	Retry behavior
role_arn	Assume role credential provider
role_session_name	Assume role credential provider
s3_disable_express_session_auth	S3 Express One Zone session authentication
s3_disable_multiregion_access_points	Amazon S3 Multi-Region Access Points
s3_use_arn_region	Amazon S3 access points
sdk_ua_app_id	Application ID
sigv4a_signing_region_set	Authentication scheme
source_profile	Assume role credential provider
sso_account_id	IAM Identity Center credential provider

Setting name	Details
sso_region	IAM Identity Center credential provider
sso_registration_scopes	IAM Identity Center credential provider
sso_role_name	IAM Identity Center credential provider
sso_start_url	IAM Identity Center credential provider
sts_regional_endpoints	AWS STS Regional endpoints
use_dualstack_endpoint	Dual-stack and FIPS endpoints
use_fips_endpoint	Dual-stack and FIPS endpoints
web_identity_token_file	Assume role credential provider

`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. SDKs and tools may also support unique settings and environment variables. To see the settings and environment variables supported by only an individual SDK or tool, see that specific SDK or tool guide.

Setting name	Details
aws_access_key_id	AWS access keys
aws_secret_access_key	AWS access keys

Setting name	Details	
aws_session_token	AWS access keys	

Environment variables list

Environment variables supported by most SDKs are listed in the following table. They are global and affect all AWS services. SDKs and tools may also support unique settings and environment variables. To see the settings and environment variables supported by only an individual SDK or tool, see that specific SDK or tool guide.

Setting name	Details	
AWS_ACCESS_KEY_ID	AWS access keys	
AWS_ACCOUNT_ID	Account-based endpoints	
AWS_ACCOUNT_ID_ENDPOINT_MODE	Account-based endpoints	
AWS_AUTH_SCHEME_PREFERENCE	Authentication scheme	
AWS_CA_BUNDLE	General configuration settings	
AWS_CONFIG_FILE	Finding and changing the location of the shared config and credentials files of AWS SDKs and tools	
AWS_CONTAINER_AUTHORIZATION_TOKEN	Container credential provider	

Setting name	Details	
AWS_CONTAINER_AUTHORIZATION_TOKEN_FILE	Container credential provider	
AWS_CONTAINER_CREDENTIALS_FULL_URI	Container credential provider	
AWS_CONTAINER_CREDENTIALS_RELATIVE_URI	Container credential provider	
AWS_DEFAULTS_MODE	Smart configuration defaults	
AWS_DISABLE_HOST_PREFIX_INJECTION	Host prefix injection	
AWS_DISABLE_REQUEST_COMPRESSION	Request compression	
AWS_EC2_METADATA_DISABLED	IMDS credential provider	
AWS_EC2_METADATA_SERVICE_ENDPOINT	IMDS credential provider	

Setting name	Details
AWS_EC2_METADATA_SERVICE_ENDPOINT_MODE	IMDS credential provider
AWS_EC2_METADATA_V1_DISABLED	IMDS credential provider
AWS_ENABLE_ENDPOINT_DISCOVERY	Endpoint discovery
AWS_ENDPOINT_URL	Service-specific endpoints
AWS_ENDPOINT_URL_<SERVICE>	Service-specific endpoints
AWS_IGNORE_CONFIGURED_ENDPOINT_URLS	Service-specific endpoints
AWS_MAX_ATTEMPTS	Retry behavior
AWS_METADATA_SERVICE_NUM_ATTEMPTS	Amazon EC2 instance metadata
AWS_METADATA_SERVICE_TIMEOUT	Amazon EC2 instance metadata

Setting name	Details
AWS_PROFILE	Using shared config and credentials files to globally configure AWS SDKs and tools
AWS_REGION	AWS Region
AWS_REQUEST_CHECKSUM_CALCULATION	Data Integrity Protections for Amazon S3
AWS_REQUEST_MIN_COMPRESSION_SIZE_BYTES	Request compression
AWS_RESPONSE_CHECKSUM_VALIDATION	Data Integrity Protections for Amazon S3
AWS_RETRY_MODE	Retry behavior
AWS_ROLE_ARN	Assume role credential provider
AWS_ROLE_SESSION_NAME	Assume role credential provider
AWS_S3_DISABLE_EXPRESS_SESSION_AUTH	S3 Express One Zone session authentication
AWS_S3_DISABLE_MULTIREGION_ACCESS_POINTS	Amazon S3 Multi-Region Access Points
AWS_S3_USE_ARN_REGION	Amazon S3 access points

Setting name	Details
AWS_SDK_UA_APP_ID	Application ID
AWS_SECRET_ACCESS_KEY	AWS access keys
AWS_SESSION_TOKEN	AWS access keys
AWS_SHARED_CREDENTIALS_FILE	Finding and changing the location of the shared `config` and `credentials` files of AWS SDKs and tools
AWS_SIGV4A_SIGNING_REGION_SET	Authentication scheme
AWS_STS_REGIONAL_ENDPOINTS	AWS STS Regional endpoints
AWS_USE_DUALSTACK_ENDPOINT	Dual-stack and FIPS endpoints
AWS_USE_FIPS_ENDPOINT	Dual-stack and FIPS endpoints
AWS_WEB_IDENTITY_TOKEN_FILE	Assume role credential provider

JVM system properties list

You can use the following JVM system properties for the AWS SDK for Java and the AWS SDK for Kotlin (targeting the JVM). See the section called "How to set JVM system properties" for instructions on how to set JVM system properties.

Setting name	Details
`aws.accessKeyId`	AWS access keys
`aws.accountId`	Account-based endpoints
`aws.accountIdEndpointMode`	Account-based endpoints
`aws.authSchemePreference`	Authentication scheme
`aws.configFile`	Finding and changing the location of the shared `config` and `credentials` files of AWS SDKs and tools
`aws.defaultsMode`	Smart configuration defaults
`aws.disableEc2MetadataV1`	IMDS credential provider
`aws.disableHostPrefixInjection`	Host prefix injection
`aws.disableRequestCompression`	Request compression

Setting name	Details
aws.disableS3ExpressAuth	S3 Express One Zone session authentication
aws.ec2MetadataServiceEndpoint	IMDS credential provider
aws.ec2MetadataServiceEndpointMode	IMDS credential provider
aws.endpointDiscoveryEnabled	Endpoint discovery
aws.endpointUrl	Service-specific endpoints
aws.endpointUrl<ServiceName>	Service-specific endpoints
aws.ignoreConfiguredEndpointUrls	Service-specific endpoints
aws.maxAttempts	Retry behavior
aws.profile	Using shared `config` and `credentials` files to globally configure AWS SDKs and tools
aws.region	AWS Region
aws.requestChecksumCalculation	Data Integrity Protections for Amazon S3

Setting name	Details
aws.requestMinCompressionSizeBytes	Request compression
aws.responseChecksumValidation	Data Integrity Protections for Amazon S3
aws.retryMode	Retry behavior
aws.roleArn	Assume role credential provider
aws.roleSessionName	Assume role credential provider
aws.s3DisableMultiRegionAccessPoints	Amazon S3 Multi-Region Access Points
aws.s3UseArnRegion	Amazon S3 access points
aws.secretAccessKey	AWS access keys
aws.sessionToken	AWS access keys
aws.sharedCredentialsFile	Finding and changing the location of the shared `config` and `credentials` files of AWS SDKs and tools
aws.useDualstackEndpoint	Dual-stack and FIPS endpoints

Setting name	Details
`aws.useFipsEndpoint`	Dual-stack and FIPS endpoints
`aws.webIdentityTokenFile`	Assume role credential provider
`sdk.ua.appId`	Application ID

AWS SDKs and Tools 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

Understand the 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 credential provider chain.

When using one of the standardized credential providers, the AWS SDKs always attempt to renew credentials automatically when they expire. The built-in credential provider chain provides your application with the ability to refresh your credentials regardless of which provider you are using in the chain. No additional code is required for the SDK to do this.

Although the distinct chain used by each SDK varies, they most often include sources such as the following:

Credential provider	Description
AWS access keys	AWS access keys for an IAM user (such as `AWS_ACCESS_KEY_ID` , and `AWS_SECRET_ACCESS_KEY`).
Federate with web identity or OpenID Connect - Assume role credential provider	Sign in using a well-known external identity provider (IdP), such as Login with Amazon, Facebook, Google, or any other OpenID Connect (OIDC)-compatible IdP. Assume the permissions of an IAM role using a JSON Web Token (JWT) from AWS Security Token Service (AWS STS).
Login credentials provider	Get credentials for a new or existing console session that you are logged in to.
IAM Identity Center credential provider	Get credentials from AWS IAM Identity Center.
Assume role credential provider	Get access to other resources by assuming the permissions of an IAM role. (Retrieve and then use temporary credentials for a role).
Container credential provider	Amazon Elastic Container Service (Amazon ECS) and Amazon Elastic Kubernetes Service (Amazon EKS)

Credential provider	Description
	credentials. The container credential provider fetches credentials for the customer's containerized application.
Process credential provider	Custom credential provider. Get your credentials from an external source or process, including IAM Roles Anywhere.
IMDS credential provider	Amazon Elastic Compute Cloud (Amazon EC2) instance profile credentials. Associate an IAM role with each of your EC2 instances. Temporary credentials for that role are made available to code running in the instance. The credentials are delivered through the Amazon EC2 metadata service.

For each step in the chain, there are multiple ways to assign setting values. Setting values that are specified in code always take precedence. However, there are also Environment variables and the Using shared `config` and `credentials` files to globally configure AWS SDKs and tools. For more information, see Precedence of settings.

SDK-specific and tool-specific credential provider chains

To go directly to your SDK's or tool's **specific** credential provider chain details, choose your SDK or tool from the following:

- AWS CLI
- SDK for C++
- SDK for Go
- SDK for Java
- SDK for JavaScript
- SDK for Kotlin
- SDK for .NET
- SDK for PHP
- SDK for Python (Boto3)
- SDK for Ruby

- SDK for Rust
- SDK for Swift
- Tools for PowerShell

AWS access keys

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

AWS access keys for an IAM user can be used as your AWS credentials. 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. It is recommended to always use the `aws_session_token` so that the credentials are temporary and no longer valid after they expire. Using long-term credentials is not recommended.

> **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.

To learn more about AWS credentials and using access keys, see AWS security credentials and Managing access keys for IAM users in the *IAM User Guide*.

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, `aws.accessKeyId` - JVM system property: Java/Kotlin only

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, `aws.secretAccessKey` - JVM system property: Java/Kotlin only

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, `aws.sessionToken` - JVM system property: Java/Kotlin only

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.

For instructions on how to obtain these values, see Using short-term credentials to authenticate AWS SDKs and tools.

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
```

Support by AWS SDKs and tools

The following SDKs support the features and settings described in this topic. Any partial exceptions are noted. Any JVM system property settings are supported by the AWS SDK for Java and the AWS SDK for Kotlin only.

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	To use shared `config` file settings, you must turn on loading from the config file; see Sessions.
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 Kotlin	Yes	
SDK for .NET 4.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	

SDK	Su	Notes or more information
SDK for Rust	Yes	
SDK for Swift	Yes	
Tools for PowerShell V5	Yes	
Tools for PowerShell V4	Yes	Environment variables not supported.

Login credentials provider

You can use your existing AWS Management Console sign-in credentials to acquire short-term credentials that can be used for programmatic access. After you complete the browser-based authentication flow, AWS generates temporary credentials that work across local development tools like the AWS CLI, AWS Tools for PowerShell and AWS SDKs.

To generate these credentials, run the `aws login` command in the AWS CLI, or the `Invoke-AWSLogin` cmdlet in AWS Tools for PowerShell. The resulting short-term credentials will be cached locally, where they can be reused by the AWS SDKs. The short-term credentials expire in 15 minutes, but the CLI and SDKs will automatically refresh them as needed up to 12 hours. When the refresh token expires, you'll be prompted to log in again via the CLI or PowerShell.

The login command will update the profile you specify with the `login_session` setting, which stores the identity of the management console session that you selected during the login workflow.

```
[profile console]
login_session = arn:aws:iam::0123456789012:user/username
region = us-west-2
```

By default, the short-term credentials and refresh token are stored in a JSON file in the `~/.aws/login/cache` directory on Linux and macOS, or `%USERPROFILE%\.aws\login\cache` on Windows. The filename is based on the login session name. You can override the directory by setting the `AWS_LOGIN_CACHE_DIRECTORY` environment variable.

Login Provider Settings

Configure this functionality by using the following:

`AWS_LOGIN_CACHE_DIRECTORY` - environment variable

Alternative directory where the CLI and SDKs will store the cached credentials that map to a login session profile.

Default value: `~/.aws/login/cache` on Linux and macOS, or `%USERPROFILE%\.aws\login\cache` on Windows.

Support by AWS SDKs and tools

The following SDKs support the features and settings described in this topic. Any partial exceptions are noted. Any JVM system property settings are supported by the AWS SDK for Java and the AWS SDK for Kotlin only.

SDK	Su	Notes or more information
AWS CLI v2	Yes	
SDK for C++	Yes	
SDK for Go V2 (1.x)	No	
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	No	
SDK for Kotlin	Yes	
SDK for .NET 4.x	Yes	
SDK for .NET 3.x	Yes	
SDK for PHP 3.x	Yes	
SDK for Python (Boto3)	Yes	Requires CRT

SDK	Su	Notes or more information
SDK for Ruby 3.x	Yes	
SDK for Rust	Yes	
Tools for PowerShell V5	Yes	
Tools for PowerShell V4	No	

Assume role credential provider

> **Note**
>
> For help in understanding the layout of settings pages, or in interpreting the **Support by AWS SDKs and tools** table that follows, see Understanding the settings pages of this guide.

Assuming a role involves using a set of temporary security credentials to access AWS resources that you might not have access to otherwise. These temporary credentials consist of an access key ID, a secret access key, and a security token.

To set up your SDK or tool to assume a role, you must first create or identify a specific *role* to assume. IAM roles are uniquely identified by a role Amazon Resource Name (ARN). Roles establish trust relationships with another entity. The trusted entity that uses the role might be an AWS service, another AWS account, a web identity provider or OIDC, or SAML federation.

After the IAM role is identified, if you are trusted by that role, you can configure your SDK or tool to use the permissions that are granted by the role. To do this, use the following settings.

For guidance on getting started using these settings, see Assuming a role with AWS credentials to authenticate AWS SDKs and tools in this guide.

Assume role credential provider settings

Configure this functionality by using the following:

`credential_source` - shared AWS `config` file setting

Used within Amazon EC2 instances or Amazon Elastic Container Service containers to specify where the SDK or 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`.
- **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 Amazon ECS container or the IAM role attached to the Amazon EKS 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. For more information about MFA, see AWS Multi-factor authentication in IAM in the *IAM User Guide*.

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. The format of the ARN is: `arn:aws:iam::`*`account-id`*`:mfa/`*`mfa-device-name`*

Example of setting this in a `config` file:

This example assumes a virtual MFA device, called `MyMFADevice`, that has been created for the account and enabled for a user.

```
mfa_serial = arn:aws:iam::123456789012:mfa/MyMFADevice
```

`role_arn` - shared AWS `config` file setting, `AWS_ROLE_ARN` - environment variable, `aws.roleArn` - JVM system property: Java/Kotlin only

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.
- `web_identity_token_file` – To use public identity providers or any OpenID Connect (OIDC)-compatible identity provider for users who have been authenticated in a mobile or web application.

`role_session_name` - shared AWS `config` file setting, `AWS_ROLE_SESSION_NAME` - environment variable, `aws.roleSessionName` - JVM system property: Java/Kotlin only

Specifies the name to attach to the role session. This name appears in AWS CloudTrail logs for entries associated with this session, which can be useful when auditing. For details, see CloudTrail userIdentity element in the *AWS CloudTrail User Guide*.

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 or AWS API 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.

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 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.

You cannot specify both `credential_source` and `source_profile` 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
role_session_name = ProfileARoleSession

[profile B]
credential_process = ./aws_signing_helper credential-process --certificate /
path/to/certificate --private-key /path/to/private-key --trust-anchor-
arn arn:aws:rolesanywhere:region:account:trust-anchor/TA_ID --profile-
arn arn:aws:rolesanywhere:region:account:profile/PROFILE_ID --role-arn
 arn:aws:iam::account:role/ROLE_ID
```

In the previous example, the A profile tells the SDK or tool to automatically look up the credentials for the linked B profile. In this case, the B profile uses the credential helper tool provided by Using IAM Roles Anywhere to authenticate AWS SDKs and tools to get credentials for the AWS SDK. Those temporary credentials are then used by your code to access AWS resources. The specified role must have attached IAM permissions policies that allow the

requested code to run, such as the command, AWS service, or API method. Every action that is taken by profile A has the role session name included in CloudTrail logs.

For a second example of role chaining, the following configuration can be used if you have an application on an Amazon Elastic Compute Cloud instance, and you want to have that application assume another role.

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

[profile B]
credential_source=Ec2InstanceMetadata
```

Profile A will use the credentials from the Amazon EC2 instance to assume the specified role and will renew the credentials automatically.

`web_identity_token_file` - shared AWS `config` file setting, `AWS_WEB_IDENTITY_TOKEN_FILE` - environment variable, `aws.webIdentityTokenFile` - JVM system property: Java/Kotlin only

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. Relative paths are treated as relative to the working directory of the process.

Support by AWS SDKs and tools

The following SDKs support the features and settings described in this topic. Any partial exceptions are noted. Any JVM system property settings are supported by the AWS SDK for Java and the AWS SDK for Kotlin only.

SDK	Su	Notes or more information
AWS CLI v2	Yes	
SDK for C++	Parti	`credential_source` not supported. `duration_seconds` not supported. `mfa_serial` not supported.
SDK for Go V2 (1.x)	Yes	
SDK for Go 1.x (V1)	Yes	To use shared `config` file settings, you must turn on loading from the config file; see Sessions.
SDK for Java 2.x	Parti	`mfa_serial` not supported. `duration_seconds` not supported.
SDK for Java 1.x	Parti	`credential_source` not supported. `mfa_serial` not supported. JVM system properties not supported.
SDK for JavaScript 3.x	Yes	
SDK for JavaScript 2.x	Parti	`credential_source` not supported.
SDK for Kotlin	Yes	
SDK for .NET 4.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	
SDK for Rust	Yes	

SDK	Su	Notes or more information
SDK for Swift	Yes	
Tools for PowerShell V5	Yes	
Tools for PowerShell V4	Yes	

Container credential provider

> **Note**
>
> For help in understanding the layout of settings pages, or in interpreting the **Support by AWS SDKs and tools** table that follows, see Understanding the settings pages of this guide.

The container credential provider fetches credentials for customer's containerized application. This credential provider is useful for Amazon Elastic Container Service (Amazon ECS) and Amazon Elastic Kubernetes Service (Amazon EKS) customers. SDKs attempt to load credentials from the specified HTTP endpoint through a GET request.

If you use Amazon ECS, we recommend you use a task IAM Role for improved credential isolation, authorization, and auditability. When configured, Amazon ECS sets the `AWS_CONTAINER_CREDENTIALS_RELATIVE_URI` environment variable that the SDKs and tools use to obtain credentials. To configure Amazon ECS for this functionality, see Task IAM role in the *Amazon Elastic Container Service Developer Guide*.

If you use Amazon EKS, we recommend you use Amazon EKS Pod Identity for improved credential isolation, least privilege, auditability, independent operation, reusability, and scalability. Both your Pod and an IAM role are associated with a Kubernetes service account to manage credentials for your applications. To learn more on Amazon EKS Pod Identity, see Amazon EKS Pod Identities in the Amazon EKS User Guide. When configured, Amazon EKS sets the `AWS_CONTAINER_CREDENTIALS_FULL_URI` and `AWS_CONTAINER_AUTHORIZATION_TOKEN_FILE` environment variables that the SDKs and tools use to obtain credentials. For setup information, see Setting up the Amazon EKS Pod Identity Agent in the Amazon EKS User Guide or Amazon EKS Pod Identity simplifies IAM permissions for applications on Amazon EKS clusters at the AWS Blog website.

Configure this functionality by using the following:

`AWS_CONTAINER_CREDENTIALS_FULL_URI` - environment variable

Specifies 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. The value is appended to the default Amazon ECS hostname of `169.254.170.2`.

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

Specifies an authorization token in plain text. 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.

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

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
```

`AWS_CONTAINER_AUTHORIZATION_TOKEN_FILE` - environment variable

Specifies an absolute file path to a file that contains the authorization token in plain text.

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_FILE=/path/to/token
```

Support by AWS SDKs and tools

The following SDKs support the features and settings described in this topic. Any partial exceptions are noted. Any JVM system property settings are supported by the AWS SDK for Java and the AWS SDK for Kotlin only.

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	When Lambda SnapStart is activated, AWS_CONTAINER_CREDENTIALS_FULL_URI and AWS_CONTA

SDK	Su	Notes or more information
		INER_AUTHORIZATION_TOKEN are automatically used for authentication.
SDK for Java 1.x	Yes	When Lambda SnapStart is activated, AWS_CONTA INER_CREDENTIALS_FULL_URI and AWS_CONTA INER_AUTHORIZATION_TOKEN are automatically used for authentication.
SDK for JavaScript 3.x	Yes	
SDK for JavaScript 2.x	Yes	
SDK for Kotlin	Yes	
SDK for .NET 4.x	Yes	When Lambda SnapStart is activated, AWS_CONTA INER_CREDENTIALS_FULL_URI and AWS_CONTA INER_AUTHORIZATION_TOKEN are automatically used for authentication.
SDK for .NET 3.x	Yes	When Lambda SnapStart is activated, AWS_CONTA INER_CREDENTIALS_FULL_URI and AWS_CONTA INER_AUTHORIZATION_TOKEN are automatically used for authentication.
SDK for PHP 3.x	Yes	
SDK for Python (Boto3)	Yes	When Lambda SnapStart is activated, AWS_CONTA INER_CREDENTIALS_FULL_URI and AWS_CONTA INER_AUTHORIZATION_TOKEN are automatically used for authentication.
SDK for Ruby 3.x	Yes	
SDK for Rust	Yes	
SDK for Swift	Yes	
Tools for PowerShell V5	Yes	

SDK	Su	Notes or more information
Tools for PowerShell V4	Yes	

IAM Identity Center credential provider

> **Note**
>
> For help in understanding the layout of settings pages, or in interpreting the **Support by AWS SDKs and tools** table that follows, see Understanding the settings pages of this guide.

This authentication mechanism uses AWS IAM Identity Center 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 How IAM Identity Center authentication is resolved for AWS SDKs and tools.

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

- **(Recommended) SSO token provider configuration** – Extended session durations. Includes support for custom session durations.
- **Legacy non-refreshable configuration** – Uses a fixed, eight-hour session.

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

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

- AWS IAM Identity Center User Guide
- AWS IAM Identity Center Portal API Reference

For a deep dive on how the SDKs and tools use and refresh credentials using this configuration, see How IAM Identity Center authentication is resolved for AWS SDKs and tools.

Prerequisites

You must first enable IAM Identity Center. For details about enabling IAM Identity Center authentication, see Enabling AWS IAM Identity Center in the *AWS IAM Identity Center User Guide*.

> **Note**
>
> Alternatively, for complete prerequisites **and** the necessary shared `config` file configuration that is detailed on this page, see the guided instructions for setting up Using IAM Identity Center to authenticate AWS SDK and tools.

SSO token provider configuration

When you use the SSO token provider configuration, your AWS SDK or tool automatically refreshes your session up to your extended session period. For more information on session duration and maximum duration, see Configure the session duration of the AWS access portal and IAM Identity Center integrated applications in the *AWS IAM Identity Center User Guide*.

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 on this section within a `config` file, see Format of the config file.

The following shared `config` file example configures the SDK or tool using a `dev` profile to request IAM Identity Center credentials.

```
[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
```

The previous examples shows that you define an `sso-session` section and associate it to a profile. Typically, `sso_account_id` and `sso_role_name` must be set in the `profile` section so that the SDK can request AWS credentials. `sso_region`, `sso_start_url`, and `sso_registration_scopes` must be set within the `sso-session` section.

`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 AWS service guide to determine if the service 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. The previous example sets `sso_registration_scopes` to provide necessary access for listing accounts and roles.

The following example shows how you can reuse the same `sso-session` configuration 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
```

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 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 issuer URL or access portal URL. For more information, see Using the AWS access portal in the *AWS IAM Identity Center User Guide*.

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

- Alternatively, starting with version **2.22.0** of the AWS CLI, you can instead use the value for **AWS Issuer 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 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 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`. An application can request one or more scopes, and the access token issued to the application is limited to the scopes granted. A minimum scope of `sso:account:access` must be granted

to get a refresh token back from the IAM Identity Center service. For the list of available access scope options, see Access scopes in the *AWS IAM Identity Center User Guide*.

These scopes define the permissions requested to be authorized for the registered OIDC client and access tokens retrieved by the client. Scopes authorize access to IAM Identity Center bearer token authorized endpoints.

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.

Support by AWS SDKs and tools

The following SDKs support the features and settings described in this topic. Any partial exceptions are noted. Any JVM system property settings are supported by the AWS SDK for Java and the AWS SDK for Kotlin only.

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	To use shared `config` file settings, you must turn on loading from the config file; see Sessions.
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 Kotlin	Yes	
SDK for .NET 4.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	
SDK for Ruby 3.x	Yes	
SDK for Rust	Parti	Legacy non-refreshable configuration only.
SDK for Swift	Yes	
Tools for PowerShell V5	Yes	
Tools for PowerShell V4	Yes	

IMDS credential provider

> **Note**
>
> For help in understanding the layout of settings pages, or in interpreting the **Support by AWS SDKs and tools** table that follows, see Understanding the settings pages of this guide.

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 Work with instance metadata in the *Amazon EC2 User Guide*. 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. 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`** – Do not use IMDS to obtain credentials.
- **`false`** – Use IMDS to obtain credentials.

`ec2_metadata_v1_disabled` - shared AWS `config` file setting, `AWS_EC2_METADATA_V1_DISABLED` - environment variable, `aws.disableEc2MetadataV1` - JVM system property: Java/Kotlin only

Whether or not to use Instance Metadata Service Version 1 (IMDSv1) as a fallback if IMDSv2 fails.

> **Note**
>
> New SDKs don't support IMDSv1 and, thus, don't support this setting. For details, see table Support by AWS SDKs and tools.

Default value: `false`.

Valid values:

- **`true`** – Do not use IMDSv1 as a fallback.
- **`false`** – Use IMDSv1 as a fallback.

`ec2_metadata_service_endpoint` - shared AWS `config` file setting, `AWS_EC2_METADATA_SERVICE_ENDPOINT` - environment variable, `aws.ec2MetadataServiceEndpoint` - JVM system property: Java/Kotlin only

The endpoint of IMDS. This value overrides the default location that AWS SDKs and tools will search for Amazon EC2 instance metadata.

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, `aws.ec2MetadataServiceEndpointMode` - JVM system property: Java/Kotlin only

The endpoint mode of IMDS.

Default value:`IPv4`.

Valid values: `IPv4`, `IPv6`.

> **Note**
>
> The IMDS credential provider is a part of the Understand the credential provider chain. However, the IMDS credential provider is only checked after several other providers that are in this series. Therefore, if you want your program use this provider's credentials, you must remove other valid credential providers from your configuration or use a different profile. Alternatively, instead of relying on the credential provider chain to automatically discover which provider returns valid credentials, specify the use of the IMDS credential provider in code. You can specify credential sources directly when you create service clients.

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.

Support by AWS SDKs and tools

The following SDKs support the features and settings described in this topic. Any partial exceptions are noted. Any JVM system property settings are supported by the AWS SDK for Java and the AWS SDK for Kotlin only.

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	To use shared `config` file settings, you must turn on loading from the config file; see Sessions.
SDK for Java 2.x	Yes	
SDK for Java 1.x	Parti	JVM system properties: Use `com.amazonaws.sdk.disableEc2MetadataV1` instead of `aws.disableEc2MetadataV1`; `aws.ec2MetadataServiceEndpoint` and `aws.ec2MetadataServiceEndpointMode` not supported.
SDK for JavaScript 3.x	Yes	
SDK for JavaScript 2.x	Yes	
SDK for Kotlin	Yes	Does not use IMDSv1 fallback.
SDK for .NET 4.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	
SDK for Rust	Yes	Does not use IMDSv1 fallback.
SDK for Swift	Yes	

SDK	Su	Notes or more information
Tools for PowerShell V5	Yes	You can disable IMDSv1 fallback explicitly in code using `[Amazon.Util.EC2InstanceMetadata]::EC2MetadataV1Disabled = $true` .
Tools for PowerShell V4	Yes	You can disable IMDSv1 fallback explicitly in code using `[Amazon.Util.EC2InstanceMetadata]::EC2MetadataV1Disabled = $true` .

Process credential provider

> **Note**
>
> For help in understanding the layout of settings pages, or in interpreting the **Support by AWS SDKs and tools** table that follows, see Understanding the settings pages of this guide.

SDKs provide a way to extend the credential provider chain for custom use cases. This provider can be used to provide custom implementations, such as retrieving credentials from an on-premises credentials store or integrating with your on-premises identify provider.

For example, IAM Roles Anywhere uses `credential_process` to get temporary credentials on behalf of your application. To configure `credential_process` for this use, see Using IAM Roles Anywhere to authenticate AWS SDKs and tools.

> **Note**
>
> The following describes a method of sourcing credentials from an external process and might be used if you are running software outside of AWS. If you are building on an AWS compute resource, use other credential providers. 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.

> **Note**
>
> The process credential provider is a part of the Understand the credential provider chain. However, the process credential provider is only checked after several other providers that are in this series. Therefore, if you want your program use this provider's credentials, you must remove other valid credential providers from your configuration or use a different profile. Alternatively, instead of relying on the credential provider chain to automatically discover which provider returns valid credentials, specify the use of the process credential provider in code. You can specify credential sources directly when you create service clients.

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. The path varies depending on the operating system:

The following example shows setting credential_process in the shared `config` file on Linux/macOS.

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

The following example shows setting credential_process in the shared `config` file on Windows.

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

- Can be specified within a dedicated profile:

```
[profile cred_process]
credential_process = /Users/username/process.sh
region = us-east-1
```

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.

Support by AWS SDKs and tools

The following SDKs support the features and settings described in this topic. Any partial exceptions are noted. Any JVM system property settings are supported by the AWS SDK for Java and the AWS SDK for Kotlin only.

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	To use shared `config` file settings, you must turn on loading from the config file; see Sessions.
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	Su	Notes or more information
SDK for Kotlin	Yes	
SDK for .NET 4.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	
SDK for Rust	Yes	
SDK for Swift	Yes	
Tools for PowerShell V5	Yes	
Tools for PowerShell V4	Yes	

AWS SDKs and Tools 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 S3 Multi-Region Access Points
- S3 Express One Zone session authentication
- Authentication scheme
- AWS Region
- AWS STS Regional endpoints
- Data Integrity Protections for Amazon S3
- Dual-stack and FIPS endpoints
- Endpoint discovery
- General configuration settings
- Host prefix injection
- IMDS client
- Retry behavior
- Request compression
- Service-specific endpoints
- Smart configuration defaults

Account-based endpoints

> **Note**
>
> For help in understanding the layout of settings pages, or in interpreting the **Support by AWS SDKs and tools** table that follows, see Understanding the settings pages of this guide.

Account-based endpoints help ensure high performance and scalability by using your AWS account ID to route requests for services that support this feature. When you use an AWS SDK and service that support account-based endpoints, the SDK client constructs and uses an account-based endpoint rather than a regional endpoint. If the account ID isn't visible to the SDK client, the client will use the regional endpoint. Account-based endpoints take the form of `https://<account-id>.ddb.<region>.amazonaws.com`, where *`<account-id>`* and *`<region>`* are your AWS account ID and AWS Region.

Configure this functionality by using the following:

`aws_account_id` - shared AWS `config` file setting, `AWS_ACCOUNT_ID` - environment variable, `aws.accountId` - JVM system property: Java/Kotlin only

The AWS account ID. Used for account-based endpoint routing. An AWS account ID has a format like 111122223333.

Account-based endpoint routing provides better request performance for some services.

`account_id_endpoint_mode` - shared AWS `config` file setting, `AWS_ACCOUNT_ID_ENDPOINT_MODE` - environment variable, `aws.accountIdEndpointMode` - JVM system property: Java/Kotlin only

This setting is used to turn off account-based endpoint routing if necessary, and bypass account-based rules.

Default value: `preferred`

Valid values:

- **`preferred`** – The endpoint should include account ID if available.
- **`disabled`** – A resolved endpoint doesn't include account ID.
- **`required`** – The endpoint must include account ID. If the account ID isn't available, the SDK throws an error.

Support by AWS SDKs and tools

The following SDKs support the features and settings described in this topic. Any partial exceptions are noted. Any JVM system property settings are supported by the AWS SDK for Java and the AWS SDK for Kotlin only.

SDK	Supp	Released in SDK version	Notes or more information
AWS CLI v2	Yes	2.25.0	
AWS CLI v1	Yes	1.38.0	
SDK for C++	No		
SDK for Go V2 (1.x)	Yes	v1.35.0	

SDK	Supp	Released in SDK version	Notes or more information
SDK for Go 1.x (V1)	No		
SDK for Java 2.x	Yes	v2.28.4	
SDK for Java 1.x	Yes	v1.12.771	
SDK for JavaScript 3.x	Yes	v3.656.0	
SDK for JavaScript 2.x	No		
SDK for Kotlin	Yes	v1.3.37	
SDK for .NET 4.x	Yes	4.0.0	
SDK for .NET 3.x	No		
SDK for PHP 3.x	Yes	v3.318.0	
SDK for Python (Boto3)	Yes	1.37.0	
SDK for Ruby 3.x	Yes	v1.123.0	
SDK for Rust	Yes	release-2025-04-24	
SDK for Swift	Yes	1.2.0	
Tools for PowerShell V5	No		
Tools for PowerShell V4	No		

Application ID

> **Note**
>
> For help in understanding the layout of settings pages, or in interpreting the **Support by AWS SDKs and tools** table that follows, see Understanding the settings pages of this guide.

A single AWS account can be used by multiple customer applications to make calls to AWS services. Application ID provides a way for customers to identify which source application made a set of calls using an AWS account. AWS SDKs and services don't use or interpret this value other than to surface it back in customer communications. For example, this value can be included in operational emails or in the AWS Health Dashboard to uniquely identify which of your applications is associated with the notification.

Configure this functionality by using the following:

`sdk_ua_app_id` - shared AWS `config` file setting, `AWS_SDK_UA_APP_ID` - environment variable, `sdk.ua.appId` - JVM system property: Java/Kotlin only

This setting is a unique string you assign to your application to identify which of your applications within a particular AWS account makes calls to AWS.

Default value: None

Valid values: String with maximum length of 50. Letters, numbers and the following special characters are allowed: !,#,$,%,&,',*,+,-,.,^,_,`,|,~.

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

```
[default]
sdk_ua_app_id=ABCDEF
```

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

```
export AWS_SDK_UA_APP_ID=ABCDEF
export AWS_SDK_UA_APP_ID="ABC DEF"
```

Windows example of setting environment variables via command line:

```
setx AWS_SDK_UA_APP_ID ABCDEF
setx AWS_SDK_UA_APP_ID="ABC DEF"
```

If you include symbols that have a special meaning to the shell being used, escape the value as appropriate.

Support by AWS SDKs and tools

The following SDKs support the features and settings described in this topic. Any partial exceptions are noted. Any JVM system property settings are supported by the AWS SDK for Java and the AWS SDK for Kotlin only.

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)	No	
SDK for Java 2.x	Parti	Shared `config` file setting not supported; environment variable not supported.
SDK for Java 1.x	No	
SDK for JavaScript 3.x	Yes	
SDK for JavaScript 2.x	No	
SDK for Kotlin	Yes	The JVM system property is `aws.userAgentAppId` .
SDK for .NET 4.x	Yes	
SDK for .NET 3.x	Yes	
SDK for PHP 3.x	Yes	
SDK for Python (Boto3)	Yes	

SDK	Su	Notes or more information
SDK for Ruby 3.x	Yes	
SDK for Rust	Yes	
SDK for Swift	Yes	
Tools for PowerShell V5	Yes	
Tools for PowerShell V4	Yes	

Amazon EC2 instance metadata

> **Note**
>
> For help in understanding the layout of settings pages, or in interpreting the **Support by AWS SDKs and tools** table that follows, see Understanding the settings pages of this guide.

Amazon EC2 provides a service on instances called the Instance Metadata Service (IMDS). To learn more about this service, see Work with instance metadata in the *Amazon EC2 User Guide*. 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
```

Support by AWS SDKs and tools

The following SDKs support the features and settings described in this topic. Any partial exceptions are noted. Any JVM system property settings are supported by the AWS SDK for Java and the AWS SDK for Kotlin only.

SDK	Su	Notes or more information
AWS CLI v2	Yes	
SDK for C++	No	

SDK	Su	Notes or more information
SDK for Go V2 (1.x)	No	
SDK for Go 1.x (V1)	No	
SDK for Java 2.x	Parti	Only `AWS_METADATA_SERVICE_TIMEOUT` is supported.
SDK for Java 1.x	Parti	Only `AWS_METADATA_SERVICE_TIMEOUT` is supported.
SDK for JavaScript 3.x	No	
SDK for JavaScript 2.x	No	
SDK for Kotlin	No	
SDK for .NET 4.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	
SDK for Rust	No	
SDK for Swift	No	
Tools for PowerShell V5	No	
Tools for PowerShell V4	No	

Amazon S3 access points

> **Note**
>
> For help in understanding the layout of settings pages, or in interpreting the **Support by AWS SDKs and tools** table that follows, see Understanding the settings pages of this guide.

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, `aws.s3UseArnRegion` - JVM system property: Java/Kotlin only, 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.

Support by AWS SDKs and tools

The following SDKs support the features and settings described in this topic. Any partial exceptions are noted. Any JVM system property settings are supported by the AWS SDK for Java and the AWS SDK for Kotlin only.

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	To use shared `config` file settings, you must turn on loading from the config file; see Sessions.
SDK for Java 2.x	Yes	
SDK for Java 1.x	Yes	JVM system property not supported.
SDK for JavaScript 3.x	Yes	
SDK for JavaScript 2.x	Yes	
SDK for Kotlin	Yes	
SDK for .NET 4.x	Yes	
SDK for .NET 3.x	Yes	Doesn't follow standard precedence; shared `config` file value takes precedence over environment variable.
SDK for PHP 3.x	Yes	
SDK for Python (Boto3)	Yes	
SDK for Ruby 3.x	Yes	
SDK for Rust	No	
SDK for Swift	No	
Tools for PowerShell V5	Yes	Doesn't follow standard precedence; shared `config` file value takes precedence over environment variable.
Tools for PowerShell V4	Yes	Doesn't follow standard precedence; shared `config` file value takes precedence over environment variable.

Amazon S3 Multi-Region Access Points

> **Note**
>
> For help in understanding the layout of settings pages, or in interpreting the **Support by AWS SDKs and tools** table that follows, see Understanding the settings pages of this guide.

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.

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, `aws.s3DisableMultiRegionAccessPoints` - JVM system property: Java/Kotlin only, 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.

Support by AWS SDKs and tools

The following SDKs support the features and settings described in this topic. Any partial exceptions are noted. Any JVM system property settings are supported by the AWS SDK for Java and the AWS SDK for Kotlin only.

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	No	
SDK for JavaScript 3.x	Yes	
SDK for JavaScript 2.x	No	
SDK for Kotlin	Yes	
SDK for .NET 4.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	
SDK for Rust	Yes	

SDK	Su	Notes or more information
SDK for Swift	No	
Tools for PowerShell V5	Yes	
Tools for PowerShell V4	Yes	

S3 Express One Zone session authentication

> **Note**
>
> For help in understanding the layout of settings pages, or in interpreting the **Support by AWS SDKs and tools** table that follows, see Understanding the settings pages of this guide.

S3 Express One Zone is Amazon S3's high-performance storage class that provides single-digit millisecond latency for frequently accessed data. When you use S3 Express One Zone buckets, AWS SDKs and tools automatically use session-based authentication that is optimized for low-latency authorization of data requests. You use session tokens with Zonal (object-level) operations to distribute the latency that's associated with authorization over a number of requests in a session, reducing the authentication overhead and improving overall request performance.

S3 Express One Zone buckets use a specific naming format that includes the Availability Zone ID, such as `bucket-name--usw2-az1--x-s3`. When the SDK detects this naming pattern, it automatically routes requests to the appropriate S3 Express One Zone endpoints and applies the optimized authentication flow. The session authentication creates temporary, bucket-specific credentials that provide low-latency access to your bucket and are cached and refreshed automatically by the SDK. See S3 Express One Zone in the *Amazon S3 User Guide* to learn more.

By default, session authentication is enabled for S3 Express One Zone buckets.

Configure this functionality by using the following:

`s3_disable_express_session_auth` - shared AWS `config` file setting, `AWS_S3_DISABLE_EXPRESS_SESSION_AUTH` - environment variable, `aws.disableS3ExpressAuth` - JVM system property: Java/Kotlin only

Controls whether S3 Express One Zone session authentication is disabled. When set to `true`, the SDK uses standard SigV4 authentication for S3 Express One Zone buckets instead of session authentication.

Default value: `false`

Valid values:

- **`true`** – Disable S3 Express One Zone session authentication.
- **`false`** – Enable S3 Express One Zone session authentication.

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

```
[default]
s3_disable_express_session_auth=true
```

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

```
export AWS_S3_DISABLE_EXPRESS_SESSION_AUTH=true
```

Windows example of setting environment variables via command line:

```
setx AWS_S3_DISABLE_EXPRESS_SESSION_AUTH true
```

Support by AWS SDKs and tools

The following SDKs support the features and settings described in this topic. Any partial exceptions are noted. Any JVM system property settings are supported by the AWS SDK for Java and the AWS SDK for Kotlin only.

SDK	Supp	Notes or more information
AWS CLI v2	No	
AWS CLI v1	No	

SDK	Supp	Notes or more information
SDK for C++	Yes	
SDK for Go V2 (1.x)	Yes	
SDK for Go 1.x (V1)	No	To use shared `config` file settings, you must turn on loading from the config file; see Sessions.
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 Kotlin	Yes	The JVM system property is `aws.s3DisableExpressSessionAuth` .
SDK for .NET 4.x	Yes	
SDK for .NET 3.x	Yes	
SDK for PHP 3.x	Yes	
SDK for Python (Boto3)	No	
SDK for Ruby 3.x	Yes	
SDK for Rust	Yes	
SDK for Swift	Yes	
Tools for PowerShell V5	Yes	
Tools for PowerShell V4	Yes	

Authentication scheme

> **Note**
>
> For help in understanding the layout of settings pages, or in interpreting the **Support by AWS SDKs and tools** table that follows, see Understanding the settings pages of this guide.

AWS services support multiple authentication schemes, such as AWS Signature Version 4 (SigV4) and AWS Signature Version 4a (SigV4a). By default, SDKs select authentication schemes based on service model definitions and prioritize schemes that provide the best compatibility. However, you can configure your preferred authentication scheme to optimize for specific requirements.

Unlike SigV4, requests signed with SigV4a are valid in multiple AWS Regions. SigV4a provides enhanced availability through cross-region request signing, which enables automatic failover to backup regions during regional disruptions. This is particularly beneficial for global services like AWS Identity and Access Management or Amazon CloudFront.

For more information on these two authentication schemes, see AWS Signature Version 4 for API requests in the *IAM User Guide*.

Configure this functionality by using the following:

`auth_scheme_preference` - shared AWS `config` file setting, `AWS_AUTH_SCHEME_PREFERENCE` - environment variable, `aws.authSchemePreference` - JVM system property: Java/Kotlin only

Specifies a comma-separated list of preferred authentication schemes in priority order. When a service supports multiple authentication schemes, the SDK attempts to use schemes from this list in the specified order, falling back to default behavior if none of the preferred schemes are available.

Default value: None.

Valid values: A comma-separated list of one or more of the following:

- **`sigv4`** – Signature Version 4 (fastest performance, single-region)
- **`sigv4a`** – Signature Version 4a (enhanced availability, cross-region support, has a slower signing performance than SigV4)
- **`httpBearerAuth`** – HTTP Bearer token authentication

Space and tab characters between scheme names are ignored.

Example of setting this value in the `config` file to prefer SigV4a:

```
[default]
auth_scheme_preference=sigv4a,sigv4
```

`sigv4a_signing_region_set` - shared AWS `config` file setting, `AWS_SIGV4A_SIGNING_REGION_SET` - environment variable

Specifies comma-separated list of AWS Regions for SigV4a multi-region signing. This is used as the default Region set for the request if SigV4a is the selected authentication scheme.

Default value: Determined by the request.

Valid values: Comma-separated list of AWS Regions. Space and tab characters between Regions are ignored.

Support by AWS SDKs and tools

The following SDKs support the features and settings described in this topic. Any partial exceptions are noted. Any JVM system property settings are supported by the AWS SDK for Java and the AWS SDK for Kotlin only.

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	No	
SDK for JavaScript 3.x	Yes	
SDK for JavaScript 2.x	No	

SDK	Su	Notes or more information
SDK for Kotlin	Yes	
SDK for .NET 4.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	
SDK for Rust	Yes	
SDK for Swift	Yes	
Tools for PowerShell V5	Yes	
Tools for PowerShell V4	No	

AWS Region

> **Note**
>
> For help in understanding the layout of settings pages, or in interpreting the **Support by AWS SDKs and tools** table that follows, see Understanding the settings pages of this guide.

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 in the code itself, the client defaults to using the Region that is 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, `aws.region` - JVM system property: Java/Kotlin only

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.

Support by AWS SDKs and tools

The following SDKs support the features and settings described in this topic. Any partial exceptions are noted. Any JVM system property settings are supported by the AWS SDK for Java and the AWS SDK for Kotlin only.

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	AWS CLI v1 uses environment variable named `AWS_DEFAULT_REGION` for this purpose.
SDK for C++	Yes	
SDK for Go V2 (1.x)	Yes	

SDK	Su	Notes or more information
SDK for Go 1.x (V1)	Yes	To use shared `config` file settings, you must turn on loading from the config file; see Sessions.
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 Kotlin	Yes	
SDK for .NET 4.x	Yes	
SDK for .NET 3.x	Yes	
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	
SDK for Rust	Yes	
SDK for Swift	Yes	
Tools for PowerShell V5	Yes	
Tools for PowerShell V4	Yes	

AWS STS Regional endpoints

> **Note**
>
> For help in understanding the layout of settings pages, or in interpreting the **Support by AWS SDKs and tools** table that follows, see Understanding the settings pages of this guide.

AWS Security Token Service (AWS STS) is available both as a global and Regional service. Some of AWS SDKs and CLIs use the global service endpoint (`https://sts.amazonaws.com`) by default, while some use the Regional service endpoints (`https://sts.{region_identifier}.{partition_domain}`). In Regions that are enabled by default, requests to the AWS STS global endpoint are automatically served in the same Region where the request originates. In opt-in Regions, requests to the AWS STS global endpoint are served by a single AWS Region, US East (N. Virginia). For more information on AWS STS endpoints, see Endpoints in the *AWS Security Token Service API Reference* or Manage AWS STS in an AWS Region in the *AWS Identity and Access Management User Guide*.

It is an AWS best practice to use Regional endpoints whenever possible and to configure your AWS Region. Customers in partitions other than commercial must use Regional endpoints. Not all SDKs and tools support this setting, but all have defined behavior around global and Regional endpoints. See the following section for more information.

> **Note**
>
> AWS has made changes to the AWS Security Token Service (AWS STS) global endpoint (`https://sts.amazonaws.com`) in Regions enabled by default to enhance its resiliency and performance. AWS STS requests to the global endpoint are automatically served in the same AWS Region as your workloads. These changes will not be deployed to opt-in Regions. We recommend that you use the appropriate AWS STS regional endpoints. For more information, see AWS STS global endpoint changes in the *AWS Identity and Access Management User Guide*.

For SDKs and tools that support this setting, customers can configure the 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: `regional`, see exceptions in the following table.

> **Note**
>
> All new SDK major versions releasing after July 2022 will default to `regional`. New SDK major versions might remove this setting and use `regional` behavior. To reduce future impact regarding this change, we recommend you start using `regional` in your application when possible.

Valid values: *(Recommended value: `regional`)*

- **`legacy`** – Uses the global AWS STS endpoint, `sts.amazonaws.com`.
- **`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
```

Support by AWS SDKs and tools

> **Note**
>
> It is an AWS best practice to use Regional endpoints whenever possible and to configure your AWS Region.

The table that follows summarizes, for your SDK or tool:

- **Supports setting**: Whether the shared `config` file variable and environment variable for STS Regional endpoints are supported.
- **Default setting value**: The default value of the setting if it is supported.
- **Default service client target STS Endpoint**: What default endpoint is used by the client even if the setting to change it is not available.
- **Service client fallback behavior**: What the SDK does when it is supposed to use a Regional endpoint but no Region has been configured. This is the behavior regardless of if it is using a Regional endpoint because of a default or because `regional` has been selected by the setting.

The table also uses the following values:

- **Global endpoint**: `https://sts.amazonaws.com`.
- **Regional endpoint**: Based on the configured AWS Region used by your application.
- **`us-east-1` (Regional)**: Uses the `us-east-1` Region endpoint but with longer session tokens than typical global requests.

SDK	Supports setting	Default setting value	Default service client target STS Endpoint	Service client fallback behavior	Notes or more information
AWS CLI v2	N	N/A	Regional endpoint	Global endpoint	

SDK		Default setting value	Default service client target STS Endpoint	Service client fallback behavior	Notes or more information
AWS CLI v1	Ye	`legacy`	Global endpoint	Global endpoint	
SDK for C++	N	N/A	Regional endpoint	`us-east-1` (Regional)	
SDK for Go V2 (1.x)	N	N/A	Regional endpoint	Request failure	
SDK for Go 1.x (V1)	Ye	`legacy`	Global endpoint	Global endpoint	To use shared `config` file settings, you must turn on loading from the config file; see Sessions.
SDK for Java 2.x	N	N/A	Regional endpoint	Request failure	If no Region is configure d, the `AssumeRole` and `AssumeRoleWithWebI dentity` will use the global STS endpoint.
SDK for Java 1.x	Ye	`legacy`	Global endpoint	Global endpoint	
SDK for JavaScript 3.x	N	N/A	Regional endpoint	`us-east-1` (Regional)	
SDK for JavaScript 2.x	Ye	`legacy`	Global endpoint	Global endpoint	

SDK		Default setting value	Default service client target STS Endpoint	Service client fallback behavior	Notes or more information
SDK for Kotlin	N	N/A	Regional endpoint	Global endpoint	
SDK for .NET 4.x	N	N/A	Regional endpoint	`us-east-1` (Regional)	
SDK for .NET 3.x	Ye	`regional`	Global endpoint	Global endpoint	
SDK for PHP 3.x	Ye	`regional`	Global endpoint	Request failure	
SDK for Python (Boto3)	Ye	`regional`	Global endpoint	Global endpoint	
SDK for Ruby 3.x	Ye	`regional`	Regional endpoint	Request failure	
SDK for Rust	N	N/A	Regional endpoint	Request failure	
SDK for Swift	N	N/A	Regional endpoint	Request failure	
Tools for PowerShell V5	Ye	`regional`	Global endpoint	Global endpoint	
Tools for PowerShell V4	Ye	`regional`	Global endpoint	Global endpoint	

Data Integrity Protections for Amazon S3

> **Note**
>
> For help in understanding the layout of settings pages, or in interpreting the **Support by AWS SDKs and tools** table that follows, see Understanding the settings pages of this guide.

For some time, AWS SDKs have supported data integrity checks when uploading data to or downloading data from Amazon Simple Storage Service. Previously, these checks were opt-in. Now, we've enabled these checks by default, using CRC-based algorithms such as CRC32 or CRC64NVME. Although each SDK or tool has a default algorithm, you can choose a different algorithm. You can also continue to still manually supply a pre-calculated checksum for uploads if you want. Consistent behavior across uploads, multipart uploads, downloads, and encryption modes simplifies client-side integrity checks.

The latest versions of our AWS SDKs and AWS CLI automatically calculate a cyclic redundancy check (CRC)-based checksum for each upload and sends it to Amazon S3. Amazon S3 independently calculates a checksum on the server side and validates it against the provided value before durably storing the object and its checksum in the object's metadata. By storing the checksum in the metadata alongside the object, when the object is downloaded, the same checksum can be automatically returned and used to validate downloads as well. You can also verify the checksum stored in the object's metadata at any time.

To learn more about checksum operations, multipart uploads, or the list of supported checksum algorithms, see Checking object integrity in Amazon S3 in the *Amazon Simple Storage Service User Guide*.

Multipart uploads:

Amazon S3 also provides developers with consistent full object checksums across single part and multipart uploads.

When uploading files in multiple parts, the SDKs calculate checksums for each part. Amazon S3 uses these checksums to verify the integrity of each part through the `UploadPart` API. Additionally, Amazon S3 validates the entire file's size and checksum when you call the `CompleteMultipartUpload` API.

If your SDK has an Amazon S3 Transfer Manager to assist with multipart uploads, the checksums are validated for the parts using the SDK-specific default algorithm found in the Support by AWS

SDKs and tools table. You can opt-in to a full object checksum by setting setting `checksum_type` to `FULL_OBJECT` or by choosing to use the CRC64NVME algorithm.

If you are using an older version of SDK or AWS CLI:

If your application uses a version prior to December 2024 of the SDK or tool, Amazon S3 still computes a CRC64NVME checksum on new objects and stores it in the object metadata for future reference. You can later compare the stored CRC with a CRC computed on your side and verify the network transmission was correct. Also, you can still manually extend the integrity protection by providing your own precomputed checksums with your `PutObject` or `UploadPart` requests, which is the standard technique for addressing this in older versions.

Configure this functionality by using the following:

`request_checksum_calculation` - shared AWS `config` file setting, `AWS_REQUEST_CHECKSUM_CALCULATION` - environment variable, `aws.requestChecksumCalculation` - JVM system property: Java/Kotlin only

By default, users are opted-in to calculating a request checksum when sending a request. The user can choose any of the available checksum algorithms as a part of building the request. Otherwise, an SDK-specific default algorithm is used. See the Support by AWS SDKs and tools table for the default algorithm for each SDK or tool.

Default value: `WHEN_SUPPORTED`

Valid values:

- **`WHEN_SUPPORTED`** – Checksum validation is performed on all request payloads when supported by the API operation, such as data transfers to Amazon S3.
- **`WHEN_REQUIRED`** – Checksum validation is performed only when required by the API operation.

`response_checksum_validation` - shared AWS `config` file setting, `AWS_RESPONSE_CHECKSUM_VALIDATION` - environment variable, `aws.responseChecksumValidation` - JVM system property: Java/Kotlin only

By default, users are opted-in to a response checksum validation when sending a request. A checksum is calculated for the response payload and compared against the checksum response header. If checksum validation fails, an error is raised to the user when the payload is read.

The checksum response header also indicates the algorithm for the checksum. The Amazon S3 client attempts to validate response checksums for all Amazon S3 API operations that support checksums. However, if the SDK has not implemented the specified checksum algorithm then this validation is skipped.

Default value: `WHEN_SUPPORTED`

Valid values:

- **`WHEN_SUPPORTED`** – Checksum validation is performed on all response payloads when supported by the API operation, such as data transfers to Amazon S3.
- **`WHEN_REQUIRED`** – Checksum validation is performed only when supported by the API operation and the caller has explicitly enabled checksum for the operation. For example, when the Amazon S3 `GetObject` API is called and the `ChecksumMode` parameter is set to enabled.

Support by AWS SDKs and tools

The following SDKs support the features and settings described in this topic. Any partial exceptions are noted. Any JVM system property settings are supported by the AWS SDK for Java and the AWS SDK for Kotlin only.

> **Note**
>
> In the following table, 'CRT' refers to the AWS Common Runtime (CRT) libraries and might require adding an additional dependency to your project.

SDK	Supp	Default checksum algorithm	Supported checksum algorithms	Notes or more informati on
AWS CLI v2	Yes	CRC64NVME	CRC64NVME, CRC32, CRC32C, SHA1, SHA256	For AWS CLI v1, the default algorithm and the supported algorithms will be identical to Python (Boto3).

SDK	Supp	Default checksum algorithm	Supported checksum algorithms	Notes or more informati on
SDK for C++	Yes	CRC64NVME	CRC64NVME, CRC32, CRC32C, SHA1, SHA256	
SDK for Go V2 (1.x)	Yes	CRC32	CRC64NVME, CRC32, CRC32C, SHA1, SHA256	
SDK for Go 1.x (V1)	No			
SDK for Java 2.x	Yes	CRC32	CRC64NVME (via CRT only), CRC32, CRC32C, SHA1, SHA256	
SDK for Java 1.x	No			
SDK for JavaScript 3.x	Yes	CRC32	CRC32, CRC32C, SHA1, SHA256	
SDK for JavaScript 2.x	No			
SDK for Kotlin	Yes	CRC32	CRC32, CRC32C, SHA1, SHA256	
SDK for .NET 4.x	Yes	CRC32	CRC32, CRC32C, SHA1, SHA256	
SDK for .NET 3.x	Yes	CRC32	CRC32, CRC32C, SHA1, SHA256	

SDK	Supp	Default checksum algorithm	Supported checksum algorithms	Notes or more informati on
SDK for PHP 3.x	Yes	CRC32	CRC32, CRC32C (via CRT only), SHA1, SHA256	`awscrt` extension is required in order to use CRC32C.
SDK for Python (Boto3)	Yes	CRC32	CRC64NVME (via CRT only), CRC32, CRC32C (via CRT only), SHA1, SHA256	
SDK for Ruby 3.x	Yes	CRC32	CRC64NVME (via CRT only), CRC32, CRC32C (via CRT only), SHA1, SHA256	
SDK for Rust	Yes	CRC32	CRC64NVME, CRC32, CRC32C, SHA1, SHA256	
SDK for Swift	Yes	CRC32	CRC64NVME, CRC32, CRC32C, SHA1, SHA256	CRT dependency required for all algorithms.
Tools for PowerShell V5	Yes	CRC32	CRC32, CRC32C, SHA1, SHA256	
Tools for PowerShell V4	Yes	CRC32	CRC32, CRC32C, SHA1, SHA256	

Dual-stack and FIPS endpoints

> **Note**
>
> For help in understanding the layout of settings pages, or in interpreting the **Support by AWS SDKs and tools** table that follows, see Understanding the settings pages of this guide.

Configure this functionality by using the following:

`use_dualstack_endpoint` - shared AWS `config` file setting, `AWS_USE_DUALSTACK_ENDPOINT` - environment variable, `aws.useDualstackEndpoint` - JVM system property: Java/Kotlin only

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, `aws.useFipsEndpoint` - JVM system property: Java/Kotlin only

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 is validated against FIPS 140. If this setting is enabled and a FIPS endpoint does not exist for the service in your AWS Region, the AWS call may fail. Service-specific endpoints and the `--endpoint-url` option for the AWS Command Line Interface override this setting.

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.

Support by AWS SDKs and tools

The following SDKs support the features and settings described in this topic. Any partial exceptions are noted. Any JVM system property settings are supported by the AWS SDK for Java and the AWS SDK for Kotlin only.

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	To use shared `config` file settings, you must turn on loading from the config file; see Sessions.
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 Kotlin	Yes	
SDK for .NET 4.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	
SDK for Ruby 3.x	Yes	
SDK for Rust	Yes	
SDK for Swift	Yes	
Tools for PowerShell V5	Yes	
Tools for PowerShell V4	Yes	

Endpoint discovery

> **Note**
>
> For help in understanding the layout of settings pages, or in interpreting the **Support by AWS SDKs and tools** table that follows, see Understanding the settings pages of this guide.

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`.

Configure this functionality by using the following:

`endpoint_discovery_enabled` - shared AWS `config` file setting, `AWS_ENABLE_ENDPOINT_DISCOVERY` - environment variable, `aws.endpointDiscoveryEnabled` - JVM system property: Java/Kotlin only, To configure value directly in code, consult your specific SDK directly.

Turns on or off endpoint discovery for DynamoDB.

Endpoint discovery is required in Timestream and optional in Amazon DynamoDB. This setting defaults to either `true` or `false` depending on whether the service requires endpoint discovery. Timestream requests default to `true`, and Amazon DynamoDB requests default to `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.

Support by AWS SDKs and tools

The following SDKs support the features and settings described in this topic. Any partial exceptions are noted. Any JVM system property settings are supported by the AWS SDK for Java and the AWS SDK for Kotlin only.

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	To use shared `config` file settings, you must turn on loading from the config file; see Sessions.
SDK for Java 2.x	Yes	The SDK for Java 2.x uses `AWS_ENDPOINT_DISCOVERY_ENABLED` for the environment variable name.
SDK for Java 1.x	Parti	JVM system property not supported.

SDK	Su	Notes or more information
SDK for JavaScript 3.x	Yes	
SDK for JavaScript 2.x	Yes	
SDK for Kotlin	Yes	
SDK for .NET 4.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	
SDK for Rust	Parti	Supported for Timestream only.
SDK for Swift	No	
Tools for PowerShell V5	Yes	
Tools for PowerShell V4	Yes	

General configuration settings

> **Note**
>
> For help in understanding the layout of settings pages, or in interpreting the **Support by AWS SDKs and tools** table that follows, see Understanding the settings pages of this guide.

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
```

Due to differences in how operating systems handle paths and escaping of path characters, the following is an example of setting this value in the `config` file on Windows:

```
[default]
```

```
ca_bundle = C:\\Users\\username\\.aws\\aws-custom-bundle.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.

Support by AWS SDKs and tools

The following SDKs support the features and settings described in this topic. Any partial exceptions are noted. Any JVM system property settings are supported by the AWS SDK for Java and the AWS SDK for Kotlin only.

SDK	Su	Notes or more information
AWS CLI v2	Parti	`api_versions` not supported.
SDK for C++	Yes	
SDK for Go V2 (1.x)	Parti	`api_versions` and `parameter_validation` not supported.
SDK for Go 1.x (V1)	Parti	`api_versions` and `parameter_validation` not supported. To use shared `config` file settings, you must turn on loading from the config file; see Sessions.
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 Kotlin	No	
SDK for .NET 4.x	No	
SDK for .NET 3.x	No	
SDK for PHP 3.x	Yes	
SDK for Python (Boto3)	Yes	

SDK	Su	Notes or more information
SDK for Ruby 3.x	Yes	
SDK for Rust	No	
SDK for Swift	No	
Tools for PowerShell V5	No	
Tools for PowerShell V4	No	

Host prefix injection

> **Note**
>
> For help in understanding the layout of settings pages, or in interpreting the **Support by AWS SDKs and tools** table that follows, see Understanding the settings pages of this guide.

Host prefix injection is a feature where AWS SDKs automatically prepend a prefix to the hostname of service endpoints for certain API operations. This prefix can be either a static string or a dynamic value that includes data from your request parameters.

For example, when using Amazon Simple Storage Service to perform actions on Amazon S3 objects or buckets, the SDK replaces your bucket name and AWS account ID in the final API endpoint.

While this behavior is required for normal AWS service endpoints, it can cause problems when using custom endpoints such as VPC endpoints or local testing tools. In these cases, you might need to disable host prefix injection.

Configure this functionality by using the following:

`disable_host_prefix_injection` - shared AWS `config` file setting,
`AWS_DISABLE_HOST_PREFIX_INJECTION` - environment variable,
`aws.disableHostPrefixInjection` - JVM system property: Java/Kotlin only

This setting controls whether the SDK or tool will modify the endpoint hostname by prepending a host prefix as defined in your SDK's client object or variable.

Default value: `false`

Valid values:

- **`true`** – Disable host prefix injection. The SDK will not modify the endpoint hostname.
- **`false`** – Enable host prefix injection. The SDK will prepend the host prefix to the endpoint hostname.

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

```
[default]
disable_host_prefix_injection = true
```

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

```
export AWS_DISABLE_HOST_PREFIX_INJECTION=true
```

Windows example of setting environment variables via command line:

```
setx AWS_DISABLE_HOST_PREFIX_INJECTION true
```

Examples of host prefix injection

The following table of examples show how SDKs modify the final endpoint when host prefix injection is enabled and disabled.

- **Host prefix**: The template of the host prefix property string set on the SDK's client object or variable in code.
- **Inputs**: Additional inputs set on the SDK's client object or variable in code.
- **Client endpoint**: The client's derived endpoint.
- **Setting value**: Resolved value for the previous setting.
- **Resulting endpoint**: The resulting endpoint the SDK client uses to make the API call.

Host prefix	Inputs	Client endpoint	Setting value	Resulting endpoint
"data."	{}	"https://service.us-west-2.amazonaws.com"	false	"https://data.service.us-west-2.amazonaws.com"
"{Bucket}-{AccountId}."	Bucket: "amzn-s3-demo-bucket1", AccountId:"123456789012"	"https://service.us-west-2.amazonaws.com"	false	"https://amzn-s3-demo-bucket1-123456789012.service.us-west-2.amazonaws.com"
"data."	{}	"https://override.us-west-2.amazonaws.com" (as an override endpoint)	true	"https://override.us-west-2.amazonaws.com"

Support by AWS SDKs and tools

The following SDKs support the features and settings described in this topic. Any partial exceptions are noted. Any JVM system property settings are supported by the AWS SDK for Java and the AWS SDK for Kotlin only.

SDK	Su	Notes or more information
AWS CLI v2	Yes	
SDK for C++	No	Setting not supported, but can be configured in code on the client using: `enableHostPrefixInjection` .

SDK	Su	Notes or more information
SDK for Go V2 (1.x)	No	Can be disabled using middleware.
SDK for Go 1.x (V1)	No	
SDK for Java 2.x	No	Setting not supported, but can be configured in code on the client using: SdkAdvancedClientOption.DISABLE_HOST_PREFIX_INJECTION.
SDK for Java 1.x	No	Setting not supported, but can be configured in code on the client using: withDisableHostPrefixInjection.
SDK for JavaScript 3.x	No	Setting not supported, but can be configured in code on the client using: disableHostPrefix.
SDK for JavaScript 2.x	No	Setting not supported, but can be configured in code on the client using: hostPrefixEnabled.
SDK for Kotlin	No	
SDK for .NET 4.x	No	Setting not supported, but can be configured in code on the client using: DisableHostPrefixInjection.
SDK for .NET 3.x	No	Setting not supported, but can be configured in code on the client using: DisableHostPrefixInjection.
SDK for PHP 3.x	No	Setting not supported, but can be configured in code on the client using: disable_host_prefix_injection.
SDK for Python (Boto3)	Yes	Can be configured in code on the client using: inject_host_prefix.
SDK for Ruby 3.x	No	Setting not supported, but can be configured in code on the client using: disable_host_prefix_injection.
SDK for Rust	No	
SDK for Swift	No	

SDK	Su	Notes or more information
Tools for PowerShell V5	No	Setting not supported, but can be included in specific cmdlets using parameter `-ClientConfig @{DisableHostPrefixInjection = $true}` .
Tools for PowerShell V4	No	Setting not supported, but can be included in specific cmdlets using parameter `-ClientConfig @{DisableHostPrefixInjection = $true}` .

IMDS client

> **Note**
>
> For help in understanding the layout of settings pages, or in interpreting the **Support by AWS SDKs and tools** table that follows, see Understanding the settings pages of this guide.

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*. 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.

Support by AWS SDKs and tools

The following SDKs support the features and settings described in this topic. Any partial exceptions are noted. Any JVM system property settings are supported by the AWS SDK for Java and the AWS SDK for Kotlin only.

SDK	Su	Notes or more information
AWS CLI v2	Yes	
SDK for C++	No	
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 Kotlin	No	

SDK	Su	Notes or more information
SDK for .NET 4.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	
SDK for Rust	Yes	
SDK for Swift	Yes	
Tools for PowerShell V5	Yes	
Tools for PowerShell V4	Yes	

Retry behavior

> **Note**
>
> For help in understanding the layout of settings pages, or in interpreting the **Support by AWS SDKs and tools** table that follows, see Understanding the settings pages of this guide.

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:

`retry_mode` - shared AWS `config` file setting, `AWS_RETRY_MODE` - environment variable, `aws.retryMode` - JVM system property: Java/Kotlin only

Specifies how the SDK or developer tool attempts retries.

Default value: This value is specific to your SDK. Check your specific SDK guide or your SDK's code base for its default `retry_mode`.

Valid values:

- `standard` – (Recommended) The recommended set of retry rules across AWS SDKs. This mode includes a standard set of errors that are retried, and automatically adjusts the number of retries to maximize availability and stability. This mode is safe for use in multi-tenant applications. The default maximum number of attempts with this mode is three, unless `max_attempts` is explicitly configured.
- `adaptive` – A retry mode, appropriate only for specialized use-cases, that includes the functionality of standard mode as well as automatic client-side rate limiting. This retry mode is not recommended for multi-tenant applications, unless you take care to isolate application tenants. See Choosing between `standard` and `adaptive` retry modes for more information. This mode is experimental and it might change behavior in the future.
- `legacy` – (Not Recommended) Specific to your SDK (check your specific SDK guide or your SDK's code base).

`max_attempts` - shared AWS `config` file setting, `AWS_MAX_ATTEMPTS` - environment variable, `aws.maxAttempts` - JVM system property: Java/Kotlin only

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.

Choosing between `standard` and `adaptive` retry modes

We recommend you use the `standard` retry mode unless you are certain that your usage is better suited for `adaptive`.

Note

The `adaptive` mode assumes that you are pooling clients based on the scope at which the backend service may throttle requests. If you don't do this, throttles in one resource could delay requests for an unrelated resource if you are using the same client for both resources.

Standard	Adaptive
Application use-cases: All.	Application use-cases: 1. Not sensitive to latency. 2. Client only accesses a single resource, or, you are providing logic to pool your clients separately by the service resource that is being accessed.
Supports circuit-breaking to prevent the SDK from retrying during outages.	Supports circuit-breaking to prevent the SDK from retrying during outages.
Uses jittered exponential backoff in the event of failures.	Uses dynamic backoff durations to attempt to minimize the number of failed requests, in exchange for the potential for increased latency.
Never delays the first request attempt, only the retries.	Can throttle or delay the initial request attempt.

If you choose to use `adaptive` mode, your application must construct clients that are designed around each resource that might be throttled. A resource, in this case, is finer-tuned than just thinking of each AWS service. AWS services can have additional dimensions that they use to throttle requests. Let's use the Amazon DynamoDB service as an example. DynamoDB uses AWS Region plus the table being accessed to throttle requests. This means that one table that your code is accessing might be throttled more than others. If your code used the same client to access all the tables, and requests to one of those tables is throttled, then adaptive retry mode will reduce the request rate for all tables. Your code should be designed to have one client per Region-and-

table pair. If you experience unexpected latency when using `adaptive` mode, see the specific AWS documentation guide for the service you are using.

Retry mode implementation details

The AWS SDKs make use of token buckets to decide whether a request should be retried and (in the case of the `adaptive` retry mode) how quickly requests should be sent. Two token buckets are used by the SDK: a retry token bucket and a request rate token bucket.

- The retry token bucket is used to determine whether the SDK should temporarily disable retries in order to protect the upstream and downstream services during outages. Tokens are acquired from the bucket before retries are attempted, and tokens are returned to the bucket when requests succeed. If the bucket is empty when a retry is attempted, the SDK will not retry the request.
- The request rate token bucket is used only in the `adaptive` retry mode to determine the rate at which to send requests. Tokens are acquired from the bucket before a request is sent, and tokens are returned to the bucket at a dynamically-determined rate based on throttling responses returned by the service.

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`:

This step is only used in `adaptive` retry mode. This step acquires a token from the request rate token bucket. If a token is not available, it will wait for one to become available. Your SDK might have configuration options available to fail the request instead of wait. Tokens in the bucket are refilled at a rate that is determined dynamically, based on the number of throttling responses received by the client.

`SendHTTPRequest`:

This step sends the request to AWS. Most AWS SDKs use an HTTP library that uses connection pools to reuse an existing connection when making an HTTP request. Generally, connections are reused if a request failed due to throttling errors but not if a request fails due to a transient error.

`RequestBookkeeping`:

Tokens are added to the token bucket if the request is successful. For `adaptive` retry mode only, the fill rate of the request rate token bucket 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`:

The default number of maximum attempts is set by the `retry_mode` setting, unless overridden by the `max_attempts` setting.

`HasRetryQuota`

This step acquires a token from the retry token bucket. If the retry token bucket is empty, the request will not be retried.

`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.

Support by AWS SDKs and tools

The following SDKs support the features and settings described in this topic. Any partial exceptions are noted. Any JVM system property settings are supported by the AWS SDK for Java and the AWS SDK for Kotlin only.

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	JVM system properties: use `com.amazonaws.sdk.maxAttempts` instead of `aws.maxAttempts`; use `com.amazonaws.sdk.retryMode` instead of `aws.retryMode`.
SDK for JavaScript 3.x	Yes	

SDK	Su	Notes or more information
SDK for JavaScript 2.x	No	Supports a maximum number of retries, exponential backoff with jitter, and an option for a custom method for retry backoff.
SDK for Kotlin	Yes	
SDK for .NET 4.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	
SDK for Rust	Yes	
SDK for Swift	Yes	
Tools for PowerShell V5	Yes	
Tools for PowerShell V4	Yes	

Request compression

> **Note**
>
> For help in understanding the layout of settings pages, or in interpreting the **Support by AWS SDKs and tools** table that follows, see Understanding the settings pages of this guide.

AWS SDKs and tools can automatically compress payloads when sending requests to AWS services that support receiving compressed payloads. Compressing the payload on the client prior to sending it to a service may reduce the overall number of requests and bandwidth required to send data to the service, as well as reduce unsuccessful requests due to service limitations on the payload size. For compression, the SDK or tool selects an encoding algorithm that is supported by

both the service and the SDK. However, the current list of possible encodings consists only of gzip, but it may expand in the future.

Request compression can be especially useful if your application is using Amazon CloudWatch. CloudWatch is a monitoring and observability service that collects monitoring and operational data in the form of logs, metrics, and events. One example of a service operation that supports compression is CloudWatch's PutMetricDataAPI method.

Configure this functionality by using the following:

`disable_request_compression` - shared AWS `config` file setting, `AWS_DISABLE_REQUEST_COMPRESSION` - environment variable, `aws.disableRequestCompression` - JVM system property: Java/Kotlin only

Turns on or off whether the SDK or tool will compress a payload prior to sending a request.

Default value: `false`

Valid values:

- **`true`** – Turn off request compression.
- **`false`** – Use request compression when possible.

`request_min_compression_size_bytes` - shared AWS `config` file setting, `AWS_REQUEST_MIN_COMPRESSION_SIZE_BYTES` - environment variable, `aws.requestMinCompressionSizeBytes` - JVM system property: Java/Kotlin only

Sets the minimum size in bytes of the request body that the SDK or tool should compress. Small payloads may become longer when compressed, thus, there is a lower limit where it makes sense to perform compression. This value is inclusive, a request size greater than or equal to the value is compressed.

Default value: 10240 bytes

Valid values: Integer value between 0 and 10485760 bytes inclusive.

Support by AWS SDKs and tools

The following SDKs support the features and settings described in this topic. Any partial exceptions are noted. Any JVM system property settings are supported by the AWS SDK for Java and the AWS SDK for Kotlin only.

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	No	
SDK for JavaScript 3.x	Yes	
SDK for JavaScript 2.x	No	
SDK for Kotlin	Yes	
SDK for .NET 4.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	
SDK for Rust	Yes	
SDK for Swift	No	
Tools for PowerShell V5	Yes	
Tools for PowerShell V4	Yes	

Service-specific endpoints

> **Note**
>
> For help in understanding the layout of settings pages, or in interpreting the **Support by AWS SDKs and tools** table that follows, see Understanding the settings pages of this guide.

Service-specific endpoint configuration provides the option to use an endpoint of your choosing for API requests and to have that choice persist. These settings provide flexibility to support local endpoints, VPC endpoints, and third-party local AWS development environments. Different endpoints can be used for testing and production environments. You can specify an endpoint URL for individual AWS services.

Configure this functionality by using the following:

`endpoint_url` - shared AWS `config` file setting, `AWS_ENDPOINT_URL` - environment variable, `aws.endpointUrl` - JVM system property: Java/Kotlin only

When specified directly within a profile or as an environment variable, this setting specifies the endpoint that is used for all service requests. This endpoint is overridden by any configured service-specific endpoint.

You can also use this setting within a `services` section of a shared AWS `config` file to set a custom endpoint for a specific service. For a list of all service identifier keys to use for subsections within the `services` section, see Identifiers for service-specific endpoints.

Default value: none

Valid values: A URL including the scheme and host for the endpoint. The URL can optionally contain a path component that contains one or more path segments.

`AWS_ENDPOINT_URL_<SERVICE>` - environment variable, `aws.endpointUrl<ServiceName>` - JVM system property: Java/Kotlin only

`AWS_ENDPOINT_URL_<SERVICE>`, where `<SERVICE>` is the AWS service identifier, sets a custom endpoint for a specific service. For a list of all service-specific environment variables, see Identifiers for service-specific endpoints.

This service-specific endpoint overrides any global endpoint set in `AWS_ENDPOINT_URL`.

Default value: none

Valid values: A URL including the scheme and host for the endpoint. The URL can optionally contain a path component that contains one or more path segments.

`ignore_configured_endpoint_urls` - shared AWS `config` file setting, `AWS_IGNORE_CONFIGURED_ENDPOINT_URLS` - environment variable, `aws.ignoreConfiguredEndpointUrls` - JVM system property: Java/Kotlin only

This setting is used to ignore all custom endpoints configurations.

Note that any explicit endpoint set in the code or on a service client itself is used regardless of this setting. For example, including the `--endpoint-url` command line parameter with an AWS CLI command or passing an endpoint URL into a client constructor will always take effect.

Default value: `false`

Valid values:

- **`true`** – The SDK or tool does not read any custom configuration options from the shared `config` file or from environment variables for setting an endpoint URL.
- **`false`** – The SDK or tool uses any available user-provided endpoints from the shared `config` file or from environment variables.

Configure endpoints using environment variables

To route requests for all services to a custom endpoint URL, set the `AWS_ENDPOINT_URL` global environment variable.

```
export AWS_ENDPOINT_URL=http://localhost:4567
```

To route requests for a specific AWS service to a custom endpoint URL, use the `AWS_ENDPOINT_URL_<SERVICE>` environment variable. Amazon DynamoDB has a `serviceId` of DynamoDB. For this service, the endpoint URL environment variable is `AWS_ENDPOINT_URL_DYNAMODB`. This endpoint takes precedence over the global endpoint set in `AWS_ENDPOINT_URL` for this service.

```
export AWS_ENDPOINT_URL_DYNAMODB=http://localhost:5678
```

As another example, AWS Elastic Beanstalk has a `serviceId` of `Elastic Beanstalk`. The AWS service identifier is based on the API model's `serviceId` by replacing all spaces with underscores and uppercasing all letters. To set the endpoint for this service, the corresponding environment variable is `AWS_ENDPOINT_URL_ELASTIC_BEANSTALK`. For a list of all service-specific environment variables, see Identifiers for service-specific endpoints.

```
export AWS_ENDPOINT_URL_ELASTIC_BEANSTALK=http://localhost:5567
```

Configure endpoints using the shared `config` file

In the shared `config` file, `endpoint_url` is used in different places for different functionality.

- `endpoint_url` specified directly within a `profile` makes that endpoint the global endpoint.
- `endpoint_url` nested under a service identifier key within a `services` section makes that endpoint apply to requests made only to that service. For details on defining a `services` section in your shared `config` file, see Format of the config file.

The following example uses a `services` definition to configure a service-specific endpoint URL to be used for Amazon S3 and a custom global endpoint to be used for all other services:

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

[services s3-specific]
s3 =
  endpoint_url = https://play.min.io:9000
```

A single profile can configure endpoints for multiple services. This example shows how to set the service-specific endpoint URLs for Amazon S3 and AWS Elastic Beanstalk in the same profile. AWS Elastic Beanstalk has a `serviceId` of `Elastic Beanstalk`. The AWS service identifier is based on the API model's `serviceId` by replacing all spaces with underscores and lowercasing all letters. Thus, the service identifier key becomes `elastic_beanstalk` and settings for this service begin on the line `elastic_beanstalk =` . For a list of all service identifier keys to use in the `services` section, see Identifiers for service-specific endpoints.

```
[services testing-s3-and-eb]
s3 =
```

```
  endpoint_url = http://localhost:4567
elastic_beanstalk =
  endpoint_url = http://localhost:8000

[profile dev]
services = testing-s3-and-eb
```

The service configuration section can be used from multiple profiles. For example, two profiles can use the same `services` definition while altering other profile properties:

```
[services testing-s3]
s3 =
  endpoint_url = https://localhost:4567

[profile testing-json]
output = json
services = testing-s3

[profile testing-text]
output = text
services = testing-s3
```

Configure endpoints in profiles using role-based credentials

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

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

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

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

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

Precedence of settings

The settings for this feature can be used at the same time but only one value will take priority per service. For API calls made to a given AWS service, the following order is used to select a value:

1. Any explicit setting set in the code or on a service client itself takes precedence over anything else.
   - For the AWS CLI, this is the value provided by the `--endpoint-url` command line parameter. 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.
2. The value provided by a service-specific environment variable such as `AWS_ENDPOINT_URL_DYNAMODB`.
3. The value provided by the `AWS_ENDPOINT_URL` global endpoint environment variable.
4. The value provided by the `endpoint_url` setting nested under a service identifier key within a `services` section of the shared `config` file.
5. The value provided by the `endpoint_url` setting specified directly within a `profile` of the shared `config` file.
6. Any default endpoint URL for the respective AWS service is used last.

Support by AWS SDKs and tools

The following SDKs support the features and settings described in this topic. Any partial exceptions are noted. Any JVM system property settings are supported by the AWS SDK for Java and the AWS SDK for Kotlin only.

SDK	Su	Notes or more information
AWS CLI v2	Yes	

SDK	Su	Notes or more information
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	No	
SDK for JavaScript 3.x	Yes	
SDK for JavaScript 2.x	No	
SDK for Kotlin	Yes	
SDK for .NET 4.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	
SDK for Rust	Yes	
SDK for Swift	Yes	
Tools for PowerShell V5	Yes	
Tools for PowerShell V4	Yes	

Identifiers for service-specific endpoints

For information on how and where to use the identifiers in the following table, see Service-specific endpoints.

serviceId	Se id r ke fo sh AV co fil	AWS_ENDPOINT_URL_<SERVICE> environment variable
AccessAnalyzer	ac ly	AWS_ENDPOINT_URL_ACCESSANALYZER
Account	ac	AWS_ENDPOINT_URL_ACCOUNT
ACM	ac	AWS_ENDPOINT_URL_ACM
ACM PCA	ac	AWS_ENDPOINT_URL_ACM_PCA
Alexa For Business	a _I	AWS_ENDPOINT_URL_ALEXA_FOR_BUSINESS
amp	ar	AWS_ENDPOINT_URL_AMP
Amplify	ar	AWS_ENDPOINT_URL_AMPLIFY
AmplifyBackend	ar cl	AWS_ENDPOINT_URL_AMPLIFYBACKEND
AmplifyUIBuilder	ar bu	AWS_ENDPOINT_URL_AMPLIFYUIBUILDER
API Gateway	ap ay	AWS_ENDPOINT_URL_API_GATEWAY
ApiGatewayManagementApi	ap yn nt	AWS_ENDPOINT_URL_APIGATEWAYMANAGEMENTAPI

serviceId	Se id r ke fo sh AV co fil	AWS_ENDPOINT_URL_<SERVICE> environment variable
ApiGatewayV2	ap yv	AWS_ENDPOINT_URL_APIGATEWAYV2
AppConfig	ap	AWS_ENDPOINT_URL_APPCONFIG
AppConfigData	ap da	AWS_ENDPOINT_URL_APPCONFIGDATA
AppFabric	ap	AWS_ENDPOINT_URL_APPFABRIC
Appflow	ap	AWS_ENDPOINT_URL_APPFLOW
AppIntegrations	ap at	AWS_ENDPOINT_URL_APPINTEGRATIONS
Application Auto Scaling	ap on ca	AWS_ENDPOINT_URL_APPLICATION_AUTO_SCALING
Application Insights	ap on ts	AWS_ENDPOINT_URL_APPLICATION_INSIGHTS
ApplicationCostProfiler	ap on fi	AWS_ENDPOINT_URL_APPLICATIONCOSTPROFILER

serviceId	Se id r ke fo sh A c fil	AWS_ENDPOINT_URL_<SERVICE> environment variable
App Mesh	ap	AWS_ENDPOINT_URL_APP_MESH
AppRunner	ap	AWS_ENDPOINT_URL_APPRUNNER
AppStream	ap	AWS_ENDPOINT_URL_APPSTREAM
AppSync	ap	AWS_ENDPOINT_URL_APPSYNC
ARC Zonal Shift	a: _	AWS_ENDPOINT_URL_ARC_ZONAL_SHIFT
Artifact	a:	AWS_ENDPOINT_URL_ARTIFACT
Athena	at	AWS_ENDPOINT_URL_ATHENA
AuditManager	au ge	AWS_ENDPOINT_URL_AUDITMANAGER
Auto Scaling	au in	AWS_ENDPOINT_URL_AUTO_SCALING
Auto Scaling Plans	au in	AWS_ENDPOINT_URL_AUTO_SCALING_PLANS
b2bi	b2	AWS_ENDPOINT_URL_B2BI

serviceId	Se id r ke fo sh AV co fil	AWS_ENDPOINT_URL_<SERVICE> environment variable
Backup	ba	AWS_ENDPOINT_URL_BACKUP
Backup Gateway	ba te	AWS_ENDPOINT_URL_BACKUP_GATEWAY
BackupStorage	ba ra	AWS_ENDPOINT_URL_BACKUPSTORAGE
Batch	ba	AWS_ENDPOINT_URL_BATCH
BCM Data Exports	bc ex	AWS_ENDPOINT_URL_BCM_DATA_EXPORTS
Bedrock	be	AWS_ENDPOINT_URL_BEDROCK
Bedrock Agent	be ge	AWS_ENDPOINT_URL_BEDROCK_AGENT
Bedrock Agent Runtime	be ge ir	AWS_ENDPOINT_URL_BEDROCK_AGENT_RUNTIME
Bedrock Runtime	be ur	AWS_ENDPOINT_URL_BEDROCK_RUNTIME
billingconductor	bi nc	AWS_ENDPOINT_URL_BILLINGCONDUCTOR
Braket	bi	AWS_ENDPOINT_URL_BRAKET

serviceId	Se id r ke fo sh AV co fil	AWS_ENDPOINT_URL_<SERVICE> environment variable
Budgets	bu	AWS_ENDPOINT_URL_BUDGETS
Cost Explorer	co o:	AWS_ENDPOINT_URL_COST_EXPLORER
chatbot	ch	AWS_ENDPOINT_URL_CHATBOT
Chime	ch	AWS_ENDPOINT_URL_CHIME
Chime SDK Identity	ch _:	AWS_ENDPOINT_URL_CHIME_SDK_IDENTITY
Chime SDK Media Pipelines	ch _r pe	AWS_ENDPOINT_URL_CHIME_SDK_MEDIA_PIPELINES
Chime SDK Meetings	ch _r	AWS_ENDPOINT_URL_CHIME_SDK_MEETINGS
Chime SDK Messaging	ch _r g	AWS_ENDPOINT_URL_CHIME_SDK_MESSAGING
Chime SDK Voice	ch _v	AWS_ENDPOINT_URL_CHIME_SDK_VOICE

serviceId	Se id r ke fo sh A\ co fil	AWS_ENDPOINT_URL_<SERVICE> environment variable
CleanRooms	cl s	AWS_ENDPOINT_URL_CLEANROOMS
CleanRoomsML	cl sr	AWS_ENDPOINT_URL_CLEANROOMSML
Cloud9	cl	AWS_ENDPOINT_URL_CLOUD9
CloudControl	cl rc	AWS_ENDPOINT_URL_CLOUDCONTROL
CloudDirectory	cl ct	AWS_ENDPOINT_URL_CLOUDDIRECTORY
CloudFormation	cl at	AWS_ENDPOINT_URL_CLOUDFORMATION
CloudFront	cl t	AWS_ENDPOINT_URL_CLOUDFRONT
CloudFront KeyValueStore	cl t_ es	AWS_ENDPOINT_URL_CLOUDFRONT_KEYVALUESTORE
CloudHSM	cl	AWS_ENDPOINT_URL_CLOUDHSM
CloudHSM V2	cl v2	AWS_ENDPOINT_URL_CLOUDHSM_V2

serviceId	Se id r ke fo sh AV co fil	AWS_ENDPOINT_URL_<SERVICE> environment variable
CloudSearch	cl ch	AWS_ENDPOINT_URL_CLOUDSEARCH
CloudSearch Domain	cl ch	AWS_ENDPOINT_URL_CLOUDSEARCH_DOMAIN
CloudTrail	cl l	AWS_ENDPOINT_URL_CLOUDTRAIL
CloudTrail Data	cl l_	AWS_ENDPOINT_URL_CLOUDTRAIL_DATA
CloudWatch	cl h	AWS_ENDPOINT_URL_CLOUDWATCH
codeartifact	co ac	AWS_ENDPOINT_URL_CODEARTIFACT
CodeBuild	co	AWS_ENDPOINT_URL_CODEBUILD
CodeCatalyst	co ys	AWS_ENDPOINT_URL_CODECATALYST
CodeCommit	co t	AWS_ENDPOINT_URL_CODECOMMIT

serviceId	Se id r ke fo sh AV co fil	AWS_ENDPOINT_URL_<SERVICE> environment variable
CodeDeploy	co y	AWS_ENDPOINT_URL_CODEDEPLOY
CodeGuru Reviewer	co re	AWS_ENDPOINT_URL_CODEGURU_REVIEWER
CodeGuru Security	co se	AWS_ENDPOINT_URL_CODEGURU_SECURITY
CodeGuruProfiler	co ro	AWS_ENDPOINT_URL_CODEGURUPROFILER
CodePipeline	co in	AWS_ENDPOINT_URL_CODEPIPELINE
CodeStar	co	AWS_ENDPOINT_URL_CODESTAR
CodeStar connections	co co ns	AWS_ENDPOINT_URL_CODESTAR_CONNECTIONS
codestar notifications	co no ti	AWS_ENDPOINT_URL_CODESTAR_NOTIFICATIONS
Cognito Identity	co de	AWS_ENDPOINT_URL_COGNITO_IDENTITY

serviceId	Se id r ke fo sh A c fil	AWS_ENDPOINT_URL_<SERVICE> environment variable
Cognito Identity Provider	c de r	AWS_ENDPOINT_URL_COGNITO_IDENTITY_PROVIDER
Cognito Sync	c y	AWS_ENDPOINT_URL_COGNITO_SYNC
Comprehend	c d	AWS_ENDPOINT_URL_COMPREHEND
ComprehendMedical	c dr	AWS_ENDPOINT_URL_COMPREHENDMEDICAL
Compute Optimizer	c p	AWS_ENDPOINT_URL_COMPUTE_OPTIMIZER
Config Service	c r	AWS_ENDPOINT_URL_CONFIG_SERVICE
Connect	c	AWS_ENDPOINT_URL_CONNECT
Connect Contact Lens	c o n	AWS_ENDPOINT_URL_CONNECT_CONTACT_LENS
ConnectCampaigns	c m	AWS_ENDPOINT_URL_CONNECTCAMPAIGNS

serviceId	Se id r ke fo sh AV co fil	AWS_ENDPOINT_URL_<SERVICE> environment variable
ConnectCases	co se	AWS_ENDPOINT_URL_CONNECTCASES
ConnectParticipant	co rt	AWS_ENDPOINT_URL_CONNECTPARTICIPANT
ControlTower	co we	AWS_ENDPOINT_URL_CONTROLTOWER
Cost Optimization Hub	co m: hu	AWS_ENDPOINT_URL_COST_OPTIMIZATION_HUB
Cost and Usage Report Service	co us o: ce	AWS_ENDPOINT_URL_COST_AND_USAGE_REPO RT_SERVICE
Customer Profiles	cu p:	AWS_ENDPOINT_URL_CUSTOMER_PROFILES
DataBrew	da	AWS_ENDPOINT_URL_DATABREW
DataExchange	da ng	AWS_ENDPOINT_URL_DATAEXCHANGE

serviceId	Se id r ke fo sh AV co fil	AWS_ENDPOINT_URL_<SERVICE> environment variable
Data Pipeline	da l:	AWS_ENDPOINT_URL_DATA_PIPELINE
DataSync	da	AWS_ENDPOINT_URL_DATASYNC
DataZone	da	AWS_ENDPOINT_URL_DATAZONE
DAX	da	AWS_ENDPOINT_URL_DAX
Detective	de	AWS_ENDPOINT_URL_DETECTIVE
Device Farm	de rn	AWS_ENDPOINT_URL_DEVICE_FARM
DevOps Guru	de ru	AWS_ENDPOINT_URL_DEVOPS_GURU
Direct Connect	d: nı	AWS_ENDPOINT_URL_DIRECT_CONNECT
Application Discovery Service	ap or e: ce	AWS_ENDPOINT_URL_APPLICATION_DISCOVERY_SERVICE
DLM	dl	AWS_ENDPOINT_URL_DLM

serviceId	Se id r ke fo sh A co fil	AWS_ENDPOINT_URL_<SERVICE> environment variable
Database Migration Service	da m _	AWS_ENDPOINT_URL_DATABASE_MIGRATION_ SERVICE
DocDB	do	AWS_ENDPOINT_URL_DOCDB
DocDB Elastic	do st	AWS_ENDPOINT_URL_DOCDB_ELASTIC
drs	dr	AWS_ENDPOINT_URL_DRS
Directory Service	di _	AWS_ENDPOINT_URL_DIRECTORY_SERVICE
DynamoDB	dy	AWS_ENDPOINT_URL_DYNAMODB
DynamoDB Streams	dy st	AWS_ENDPOINT_URL_DYNAMODB_STREAMS
EBS	eb	AWS_ENDPOINT_URL_EBS
EC2	ec	AWS_ENDPOINT_URL_EC2
EC2 Instance Connect	ec ne ct	AWS_ENDPOINT_URL_EC2_INSTANCE_CONNECT
ECR	ec	AWS_ENDPOINT_URL_ECR

serviceId	Se id r ke fo sh AV co fil	AWS_ENDPOINT_URL_<SERVICE> environment variable
ECR PUBLIC	ec c	AWS_ENDPOINT_URL_ECR_PUBLIC
ECS	ec	AWS_ENDPOINT_URL_ECS
EFS	ef	AWS_ENDPOINT_URL_EFS
EKS	el	AWS_ENDPOINT_URL_EKS
EKS Auth	el	AWS_ENDPOINT_URL_EKS_AUTH
Elastic Inference	el nf	AWS_ENDPOINT_URL_ELASTIC_INFERENCE
ElastiCache	el he	AWS_ENDPOINT_URL_ELASTICACHE
Elastic Beanstalk	el ea	AWS_ENDPOINT_URL_ELASTIC_BEANSTALK
Elastic Transcoder	el ra	AWS_ENDPOINT_URL_ELASTIC_TRANSCODER
Elastic Load Balancing	el oa c:	AWS_ENDPOINT_URL_ELASTIC_LOAD_BALANCING

serviceId	Se id r ke fo sh AV co fil	AWS_ENDPOINT_URL_<SERVICE> environment variable
Elastic Load Balancing v2	el oa c:	AWS_ENDPOINT_URL_ELASTIC_LOAD_BALANCING_V2
EMR	er	AWS_ENDPOINT_URL_EMR
EMR containers	er in	AWS_ENDPOINT_URL_EMR_CONTAINERS
EMR Serverless	er r:	AWS_ENDPOINT_URL_EMR_SERVERLESS
EntityResolution	er o:	AWS_ENDPOINT_URL_ENTITYRESOLUTION
Elasticsearch Service	el a: ic	AWS_ENDPOINT_URL_ELASTICSEARCH_SERVICE
EventBridge	ev ge	AWS_ENDPOINT_URL_EVENTBRIDGE
Evidently	ev	AWS_ENDPOINT_URL_EVIDENTLY
finspace	fi	AWS_ENDPOINT_URL_FINSPACE
finspace data	fi da	AWS_ENDPOINT_URL_FINSPACE_DATA

serviceId	Se id r ke fo sh AV co fil	AWS_ENDPOINT_URL_<SERVICE> environment variable
Firehose	f:	AWS_ENDPOINT_URL_FIREHOSE
fis	f:	AWS_ENDPOINT_URL_FIS
FMS	fr	AWS_ENDPOINT_URL_FMS
forecast	fo	AWS_ENDPOINT_URL_FORECAST
forecastquery	fo ue	AWS_ENDPOINT_URL_FORECASTQUERY
FraudDetector	f: ct	AWS_ENDPOINT_URL_FRAUDDETECTOR
FreeTier	f:	AWS_ENDPOINT_URL_FREETIER
FSx	fs	AWS_ENDPOINT_URL_FSX
GameLift	ga	AWS_ENDPOINT_URL_GAMELIFT
Glacier	gl	AWS_ENDPOINT_URL_GLACIER
Global Accelerator	gl ce	AWS_ENDPOINT_URL_GLOBAL_ACCELERATOR
Glue	gl	AWS_ENDPOINT_URL_GLUE
grafana	gr	AWS_ENDPOINT_URL_GRAFANA

serviceId	Se id r ke fo sh AV co fil	AWS_ENDPOINT_URL_<SERVICE> environment variable
Greengrass	g: s	AWS_ENDPOINT_URL_GREENGRASS
GreengrassV2	g: sv	AWS_ENDPOINT_URL_GREENGRASSV2
GroundStation	g: t:	AWS_ENDPOINT_URL_GROUNDSTATION
GuardDuty	gu	AWS_ENDPOINT_URL_GUARDDUTY
Health	he	AWS_ENDPOINT_URL_HEALTH
HealthLake	he e	AWS_ENDPOINT_URL_HEALTHLAKE
Honeycode	ho	AWS_ENDPOINT_URL_HONEYCODE
IAM	ia	AWS_ENDPOINT_URL_IAM
identitystore	id to	AWS_ENDPOINT_URL_IDENTITYSTORE
imagebuilder	im de	AWS_ENDPOINT_URL_IMAGEBUILDER

serviceId	Se id r ke fo sh AV co fil	AWS_ENDPOINT_URL_<SERVICE> environment variable
ImportExport	ir o:	AWS_ENDPOINT_URL_IMPORTEXPORT
Inspector	ir	AWS_ENDPOINT_URL_INSPECTOR
Inspector Scan	ir _s	AWS_ENDPOINT_URL_INSPECTOR_SCAN
Inspector2	ir 2	AWS_ENDPOINT_URL_INSPECTOR2
InternetMonitor	ir or	AWS_ENDPOINT_URL_INTERNETMONITOR
IoT	io	AWS_ENDPOINT_URL_IOT
IoT Data Plane	io p:	AWS_ENDPOINT_URL_IOT_DATA_PLANE
IoT Jobs Data Plane	io da e	AWS_ENDPOINT_URL_IOT_JOBS_DATA_PLANE
IoT 1Click Devices Service	io k_ _s	AWS_ENDPOINT_URL_IOT_1CLICK_DEVICES_ SERVICE

serviceId	Se id r ke fo sh AV co fil	AWS_ENDPOINT_URL_<SERVICE> environment variable
IoT 1Click Projects	io k_ s	AWS_ENDPOINT_URL_IOT_1CLICK_PROJECTS
IoTAnalytics	io io	AWS_ENDPOINT_URL_IOTANALYTICS
IotDeviceAdvisor	io ad	AWS_ENDPOINT_URL_IOTDEVICEADVISOR
IoT Events	io s	AWS_ENDPOINT_URL_IOT_EVENTS
IoT Events Data	io s_	AWS_ENDPOINT_URL_IOT_EVENTS_DATA
IoTFleetHub	io ub	AWS_ENDPOINT_URL_IOTFLEETHUB
IoTFleetWise	io is	AWS_ENDPOINT_URL_IOTFLEETWISE
IoTSecureTunneling	io tu	AWS_ENDPOINT_URL_IOTSECURETUNNELING
IoTSiteWise	io se	AWS_ENDPOINT_URL_IOTSITEWISE

serviceId	Se id r ke fo sh AV co fil	AWS_ENDPOINT_URL_<SERVICE> environment variable
IoTThingsGraph	ic g:	AWS_ENDPOINT_URL_IOTTHINGSGRAPH
IoTTwinMaker	ic ke	AWS_ENDPOINT_URL_IOTTWINMAKER
IoT Wireless	ic es	AWS_ENDPOINT_URL_IOT_WIRELESS
ivs	iv	AWS_ENDPOINT_URL_IVS
IVS RealTime	iv ir	AWS_ENDPOINT_URL_IVS_REALTIME
ivschat	iv	AWS_ENDPOINT_URL_IVSCHAT
Kafka	ka	AWS_ENDPOINT_URL_KAFKA
KafkaConnect	ka ec	AWS_ENDPOINT_URL_KAFKACONNECT
kendra	ke	AWS_ENDPOINT_URL_KENDRA
Kendra Ranking	ke nl	AWS_ENDPOINT_URL_KENDRA_RANKING
Keyspaces	ke	AWS_ENDPOINT_URL_KEYSPACES

serviceId	Se id r ke fo sh AV co fil	AWS_ENDPOINT_URL_<SERVICE> environment variable
Kinesis	k:	AWS_ENDPOINT_URL_KINESIS
Kinesis Video Archived Media	k: ic iv a	AWS_ENDPOINT_URL_KINESIS_VIDEO_ARCHI VED_MEDIA
Kinesis Video Media	k: ic a	AWS_ENDPOINT_URL_KINESIS_VIDEO_MEDIA
Kinesis Video Signaling	k: ic a:	AWS_ENDPOINT_URL_KINESIS_VIDEO_SIGNALING
Kinesis Video WebRTC Storage	k: ic tc e	AWS_ENDPOINT_URL_KINESIS_VIDEO_WEBRT C_STORAGE
Kinesis Analytics	k: na	AWS_ENDPOINT_URL_KINESIS_ANALYTICS
Kinesis Analytics V2	k: na v2	AWS_ENDPOINT_URL_KINESIS_ANALYTICS_V2

serviceId	Se id r ke fo sh AV co fil	AWS_ENDPOINT_URL_<SERVICE> environment variable
Kinesis Video	k i	AWS_ENDPOINT_URL_KINESIS_VIDEO
KMS	kr	AWS_ENDPOINT_URL_KMS
LakeFormation	la t	AWS_ENDPOINT_URL_LAKEFORMATION
Lambda	la	AWS_ENDPOINT_URL_LAMBDA
Launch Wizard	la za	AWS_ENDPOINT_URL_LAUNCH_WIZARD
Lex Model Building Service	le _ _	AWS_ENDPOINT_URL_LEX_MODEL_BUILDING_ SERVICE
Lex Runtime Service	le me e	AWS_ENDPOINT_URL_LEX_RUNTIME_SERVICE
Lex Models V2	le s_	AWS_ENDPOINT_URL_LEX_MODELS_V2
Lex Runtime V2	le me	AWS_ENDPOINT_URL_LEX_RUNTIME_V2
License Manager	l ar	AWS_ENDPOINT_URL_LICENSE_MANAGER

serviceId	Se id r ke fo sh AV co fil	AWS_ENDPOINT_URL_<SERVICE> environment variable
License Manager Linux Subscriptions	l: ar nu r:	AWS_ENDPOINT_URL_LICENSE_MANAGER_LIN UX_SUBSCRIPTIONS
License Manager User Subscriptions	l: ar e: ip	AWS_ENDPOINT_URL_LICENSE_MANAGER_USE R_SUBSCRIPTIONS
Lightsail	l:	AWS_ENDPOINT_URL_LIGHTSAIL
Location	lo	AWS_ENDPOINT_URL_LOCATION
CloudWatch Logs	c: h_	AWS_ENDPOINT_URL_CLOUDWATCH_LOGS
LookoutEquipment	lo u:	AWS_ENDPOINT_URL_LOOKOUTEQUIPMENT
LookoutMetrics	lo t:	AWS_ENDPOINT_URL_LOOKOUTMETRICS
LookoutVision	lo s:	AWS_ENDPOINT_URL_LOOKOUTVISION
m2	m:	AWS_ENDPOINT_URL_M2

serviceId	Se id r ke fo sh AV ce fil	AWS_ENDPOINT_URL_<SERVICE> environment variable
Machine Learning	ma ea	AWS_ENDPOINT_URL_MACHINE_LEARNING
Macie2	ma	AWS_ENDPOINT_URL_MACIE2
ManagedBlockchain	ma oc	AWS_ENDPOINT_URL_MANAGEDBLOCKCHAIN
ManagedBlockchain Query	ma oc qu	AWS_ENDPOINT_URL_MANAGEDBLOCKCHAIN_QUERY
Marketplace Agreement	ma ce er	AWS_ENDPOINT_URL_MARKETPLACE_AGREEMENT
Marketplace Catalog	ma ce g	AWS_ENDPOINT_URL_MARKETPLACE_CATALOG
Marketplace Deployment	ma ce me	AWS_ENDPOINT_URL_MARKETPLACE_DEPLOYMENT
Marketplace Entitlement Service	ma ce er vi	AWS_ENDPOINT_URL_MARKETPLACE_ENTITLEMENT_SERVICE

serviceId	Se id r ke fo sh AV co fil	AWS_ENDPOINT_URL_<SERVICE> environment variable
Marketplace Commerce Analytics	ma ce ce ic	AWS_ENDPOINT_URL_MARKETPLACE_COMMERCE_ANALYTICS
MediaConnect	me ec	AWS_ENDPOINT_URL_MEDIACONNECT
MediaConvert	me e:	AWS_ENDPOINT_URL_MEDIACONVERT
MediaLive	me	AWS_ENDPOINT_URL_MEDIALIVE
MediaPackage	me ag	AWS_ENDPOINT_URL_MEDIAPACKAGE
MediaPackage Vod	me ag	AWS_ENDPOINT_URL_MEDIAPACKAGE_VOD
MediaPackageV2	me ag	AWS_ENDPOINT_URL_MEDIAPACKAGEV2
MediaStore	me e	AWS_ENDPOINT_URL_MEDIASTORE
MediaStore Data	me e_	AWS_ENDPOINT_URL_MEDIASTORE_DATA

serviceId	Se id r ke fo sh AV co fil	AWS_ENDPOINT_URL_<SERVICE> environment variable
MediaTailor	me o:	AWS_ENDPOINT_URL_MEDIATAILOR
Medical Imaging	me ma	AWS_ENDPOINT_URL_MEDICAL_IMAGING
MemoryDB	me	AWS_ENDPOINT_URL_MEMORYDB
Marketplace Metering	ma ce ng	AWS_ENDPOINT_URL_MARKETPLACE_METERING
Migration Hub	m: _I	AWS_ENDPOINT_URL_MIGRATION_HUB
mgn	mg	AWS_ENDPOINT_URL_MGN
Migration Hub Refactor Spaces	m: _I ct es	AWS_ENDPOINT_URL_MIGRATION_HUB_REFAC TOR_SPACES
MigrationHub Config	m: hu g	AWS_ENDPOINT_URL_MIGRATIONHUB_CONFIG

serviceId	Se id r ke fo sh AV co fil	AWS_ENDPOINT_URL_<SERVICE> environment variable
MigrationHubOrches trator	m: hu t:	AWS_ENDPOINT_URL_MIGRATIONHUBORCHESTRATOR
MigrationHubStrategy	m: hu gy	AWS_ENDPOINT_URL_MIGRATIONHUBSTRATEGY
Mobile	mo	AWS_ENDPOINT_URL_MOBILE
mq	mo	AWS_ENDPOINT_URL_MQ
MTurk	mt	AWS_ENDPOINT_URL_MTURK
MWAA	mw	AWS_ENDPOINT_URL_MWAA
Neptune	ne	AWS_ENDPOINT_URL_NEPTUNE
Neptune Graph	ne ra	AWS_ENDPOINT_URL_NEPTUNE_GRAPH
neptunedata	ne ta	AWS_ENDPOINT_URL_NEPTUNEDATA
Network Firewall	ne i:	AWS_ENDPOINT_URL_NETWORK_FIREWALL
NetworkManager	ne na	AWS_ENDPOINT_URL_NETWORKMANAGER

serviceId	Se id r ke fo sh A c fil	AWS_ENDPOINT_URL_<SERVICE> environment variable
NetworkMonitor	ne n:	AWS_ENDPOINT_URL_NETWORKMONITOR
nimble	n:	AWS_ENDPOINT_URL_NIMBLE
OAM	oa	AWS_ENDPOINT_URL_OAM
Omics	or	AWS_ENDPOINT_URL_OMICS
OpenSearch	op h	AWS_ENDPOINT_URL_OPENSEARCH
OpenSearchServerless	op hs ss	AWS_ENDPOINT_URL_OPENSEARCHSERVERLESS
OpsWorks	op	AWS_ENDPOINT_URL_OPSWORKS
OpsWorksCM	op m	AWS_ENDPOINT_URL_OPSWORKSCM
Organizations	or ic	AWS_ENDPOINT_URL_ORGANIZATIONS
OSIS	os	AWS_ENDPOINT_URL_OSIS
Outposts	ou	AWS_ENDPOINT_URL_OUTPOSTS
p8data	p8	AWS_ENDPOINT_URL_P8DATA

serviceId	Se id r ke fo sh AV co fil	AWS_ENDPOINT_URL_<SERVICE> environment variable
p8data	p8	AWS_ENDPOINT_URL_P8DATA
Panorama	pa	AWS_ENDPOINT_URL_PANORAMA
Payment Cryptography	pa ry hy	AWS_ENDPOINT_URL_PAYMENT_CRYPTOGRAPHY
Payment Cryptography Data	pa ry hy	AWS_ENDPOINT_URL_PAYMENT_CRYPTOGRAPHY_DATA
Pca Connector Ad	pc ct	AWS_ENDPOINT_URL_PCA_CONNECTOR_AD
Personalize	pe ze	AWS_ENDPOINT_URL_PERSONALIZE
Personalize Events	pe ze	AWS_ENDPOINT_URL_PERSONALIZE_EVENTS
Personalize Runtime	pe ze e	AWS_ENDPOINT_URL_PERSONALIZE_RUNTIME
PI	pi	AWS_ENDPOINT_URL_PI
Pinpoint	pi	AWS_ENDPOINT_URL_PINPOINT

serviceId	Se id r ke fo sh AV co fil	AWS_ENDPOINT_URL_<SERVICE> environment variable
Pinpoint Email	p: er	AWS_ENDPOINT_URL_PINPOINT_EMAIL
Pinpoint SMS Voice	p: sr	AWS_ENDPOINT_URL_PINPOINT_SMS_VOICE
Pinpoint SMS Voice V2	p: sr _\	AWS_ENDPOINT_URL_PINPOINT_SMS_VOICE_V2
Pipes	p:	AWS_ENDPOINT_URL_PIPES
Polly	p(	AWS_ENDPOINT_URL_POLLY
Pricing	p:	AWS_ENDPOINT_URL_PRICING
PrivateNetworks	p: tv	AWS_ENDPOINT_URL_PRIVATENETWORKS
Proton	p:	AWS_ENDPOINT_URL_PROTON
QBusiness	qb	AWS_ENDPOINT_URL_QBUSINESS
QConnect	q(	AWS_ENDPOINT_URL_QCONNECT
QLDB	q:	AWS_ENDPOINT_URL_QLDB

serviceId	Se id r ke fo sh AV co fil	AWS_ENDPOINT_URL_<SERVICE> environment variable
QLDB Session	q ic	AWS_ENDPOINT_URL_QLDB_SESSION
QuickSight	qu t	AWS_ENDPOINT_URL_QUICKSIGHT
RAM	ra	AWS_ENDPOINT_URL_RAM
rbin	rb	AWS_ENDPOINT_URL_RBIN
RDS	rd	AWS_ENDPOINT_URL_RDS
RDS Data	rd	AWS_ENDPOINT_URL_RDS_DATA
Redshift	re	AWS_ENDPOINT_URL_REDSHIFT
Redshift Data	re da	AWS_ENDPOINT_URL_REDSHIFT_DATA
Redshift Serverless	re se s	AWS_ENDPOINT_URL_REDSHIFT_SERVERLESS
Rekognition	re or	AWS_ENDPOINT_URL_REKOGNITION
repostspace	re ce	AWS_ENDPOINT_URL_REPOSTSPACE

serviceId	Se id r ke fo sh AV co fil	AWS_ENDPOINT_URL_<SERVICE> environment variable
resiliencehub	re eh	AWS_ENDPOINT_URL_RESILIENCEHUB
Resource Explorer 2	re ex 2	AWS_ENDPOINT_URL_RESOURCE_EXPLORER_2
Resource Groups	re gr	AWS_ENDPOINT_URL_RESOURCE_GROUPS
Resource Groups Tagging API	re gr gg	AWS_ENDPOINT_URL_RESOURCE_GROUPS_TAG GING_API
RoboMaker	ro	AWS_ENDPOINT_URL_ROBOMAKER
RolesAnywhere	ro he	AWS_ENDPOINT_URL_ROLESANYWHERE
Route 53	ro	AWS_ENDPOINT_URL_ROUTE_53
Route53 Recovery Cluster	ro ec lu	AWS_ENDPOINT_URL_ROUTE53_RECOVERY_CLUSTER

serviceId	Se id r ke fo sh AV co fil	AWS_ENDPOINT_URL_<SERVICE> environment variable
Route53 Recovery Control Config	rc ec or nt	AWS_ENDPOINT_URL_ROUTE53_RECOVERY_CO NTROL_CONFIG
Route53 Recovery Readiness	rc ec ea	AWS_ENDPOINT_URL_ROUTE53_RECOVERY_RE ADINESS
Route 53 Domains	rc do	AWS_ENDPOINT_URL_ROUTE_53_DOMAINS
Route53Resolver	rc so	AWS_ENDPOINT_URL_ROUTE53RESOLVER
RUM	ru	AWS_ENDPOINT_URL_RUM
S3	s3	AWS_ENDPOINT_URL_S3
S3 Control	s3 l	AWS_ENDPOINT_URL_S3_CONTROL
S3Outposts	s3 s	AWS_ENDPOINT_URL_S3OUTPOSTS
SageMaker	sa	AWS_ENDPOINT_URL_SAGEMAKER

serviceId	Se id r ke fo sh AV co fil	AWS_ENDPOINT_URL_<SERVICE> environment variable
SageMaker A2I Runtime	sa _a in	AWS_ENDPOINT_URL_SAGEMAKER_A2I_RUNTIME
Sagemaker Edge	sa _e	AWS_ENDPOINT_URL_SAGEMAKER_EDGE
SageMaker FeatureSt ore Runtime	sa _f to in	AWS_ENDPOINT_URL_SAGEMAKER_FEATUREST ORE_RUNTIME
SageMaker Geospatial	sa _g a	AWS_ENDPOINT_URL_SAGEMAKER_GEOSPATIAL
SageMaker Metrics	sa _m	AWS_ENDPOINT_URL_SAGEMAKER_METRICS
SageMaker Runtime	sa _r	AWS_ENDPOINT_URL_SAGEMAKER_RUNTIME
savingsplans	sa ar	AWS_ENDPOINT_URL_SAVINGSPLANS
Scheduler	sc	AWS_ENDPOINT_URL_SCHEDULER

serviceId	Se id r ke fo sh AV co fil	AWS_ENDPOINT_URL_<SERVICE> environment variable
schemas	sc	AWS_ENDPOINT_URL_SCHEMAS
SimpleDB	si	AWS_ENDPOINT_URL_SIMPLEDB
Secrets Manager	se ar	AWS_ENDPOINT_URL_SECRETS_MANAGER
SecurityHub	se ub	AWS_ENDPOINT_URL_SECURITYHUB
SecurityLake	se al	AWS_ENDPOINT_URL_SECURITYLAKE
ServerlessApplicationRepository	se sa io to	AWS_ENDPOINT_URL_SERVERLESSAPPLICATIONREPOSITORY
Service Quotas	se uo	AWS_ENDPOINT_URL_SERVICE_QUOTAS
Service Catalog	se at	AWS_ENDPOINT_URL_SERVICE_CATALOG
Service Catalog AppRegistry	se at pi	AWS_ENDPOINT_URL_SERVICE_CATALOG_APPREGISTRY

serviceId	Se id r ke fo sh AV co fil	AWS_ENDPOINT_URL_<SERVICE> environment variable
ServiceDiscovery	se sc	AWS_ENDPOINT_URL_SERVICEDISCOVERY
SES	se	AWS_ENDPOINT_URL_SES
SESv2	se	AWS_ENDPOINT_URL_SESV2
Shield	sh	AWS_ENDPOINT_URL_SHIELD
signer	si	AWS_ENDPOINT_URL_SIGNER
SimSpaceWeaver	si ea	AWS_ENDPOINT_URL_SIMSPACEWEAVER
SMS	sn	AWS_ENDPOINT_URL_SMS
Snow Device Management	sn ce me	AWS_ENDPOINT_URL_SNOW_DEVICE_MANAGEMENT
Snowball	sn	AWS_ENDPOINT_URL_SNOWBALL
SNS	sn	AWS_ENDPOINT_URL_SNS
SQS	sq	AWS_ENDPOINT_URL_SQS
SSM	ss	AWS_ENDPOINT_URL_SSM
SSM Contacts	ss ct	AWS_ENDPOINT_URL_SSM_CONTACTS

serviceId	Se id r ke fo sh AV co fil	AWS_ENDPOINT_URL_<SERVICE> environment variable
SSM Incidents	ss er	AWS_ENDPOINT_URL_SSM_INCIDENTS
Ssm Sap	ss	AWS_ENDPOINT_URL_SSM_SAP
SSO	ss	AWS_ENDPOINT_URL_SSO
SSO Admin	ss	AWS_ENDPOINT_URL_SSO_ADMIN
SSO OIDC	ss	AWS_ENDPOINT_URL_SSO_OIDC
SFN	st	AWS_ENDPOINT_URL_SFN
Storage Gateway	st at	AWS_ENDPOINT_URL_STORAGE_GATEWAY
STS	st	AWS_ENDPOINT_URL_STS
SupplyChain	su in	AWS_ENDPOINT_URL_SUPPLYCHAIN
Support	su	AWS_ENDPOINT_URL_SUPPORT
Support App	su pp	AWS_ENDPOINT_URL_SUPPORT_APP
SWF	sw	AWS_ENDPOINT_URL_SWF

serviceId	Se id r ke fo sh AV co fil	AWS_ENDPOINT_URL_<SERVICE> environment variable
synthetics	sy s	AWS_ENDPOINT_URL_SYNTHETICS
Textract	te	AWS_ENDPOINT_URL_TEXTRACT
Timestream InfluxDB	ti m_ b	AWS_ENDPOINT_URL_TIMESTREAM_INFLUXDB
Timestream Query	ti m_	AWS_ENDPOINT_URL_TIMESTREAM_QUERY
Timestream Write	ti m_	AWS_ENDPOINT_URL_TIMESTREAM_WRITE
tnb	tr	AWS_ENDPOINT_URL_TNB
Transcribe	ti e	AWS_ENDPOINT_URL_TRANSCRIBE
Transfer	ti	AWS_ENDPOINT_URL_TRANSFER
Translate	ti	AWS_ENDPOINT_URL_TRANSLATE
TrustedAdvisor	ti vi	AWS_ENDPOINT_URL_TRUSTEDADVISOR

serviceId	Se id r ke fo sh AV co fil	AWS_ENDPOINT_URL_<SERVICE> environment variable
VerifiedPermissions	ve e: s	AWS_ENDPOINT_URL_VERIFIEDPERMISSIONS
Voice ID	vo	AWS_ENDPOINT_URL_VOICE_ID
VPC Lattice	vp ce	AWS_ENDPOINT_URL_VPC_LATTICE
WAF	wa	AWS_ENDPOINT_URL_WAF
WAF Regional	wa na	AWS_ENDPOINT_URL_WAF_REGIONAL
WAFV2	wa	AWS_ENDPOINT_URL_WAFV2
WellArchitected	we te	AWS_ENDPOINT_URL_WELLARCHITECTED
Wisdom	wi	AWS_ENDPOINT_URL_WISDOM
WorkDocs	wo	AWS_ENDPOINT_URL_WORKDOCS
WorkLink	wo	AWS_ENDPOINT_URL_WORKLINK
WorkMail	wo	AWS_ENDPOINT_URL_WORKMAIL

serviceId	Se id r ke fo sh A\ c fil	AWS_ENDPOINT_URL_<SERVICE> environment variable
WorkMailMessageFlow	wo es w	AWS_ENDPOINT_URL_WORKMAILMESSAGEFLOW
WorkSpaces	wo s	AWS_ENDPOINT_URL_WORKSPACES
WorkSpaces Thin Client	wo s_ ie	AWS_ENDPOINT_URL_WORKSPACES_THIN_CLIENT
WorkSpaces Web	wo s_	AWS_ENDPOINT_URL_WORKSPACES_WEB
XRay	x:	AWS_ENDPOINT_URL_XRAY

Smart configuration defaults

> **Note**
>
> For help in understanding the layout of settings pages, or in interpreting the **Support by AWS SDKs and tools** table that follows, see Understanding the settings pages of this guide.

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, `aws.defaultsMode` - JVM system property: Java/Kotlin only

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, 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.
- `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 Regional endpoints.
- `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

Parameter	standard mode	in-region mode	cross-region mode	mobile mode
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).

Support by AWS SDKs and tools

The following SDKs support the features and settings described in this topic. Any partial exceptions are noted. Any JVM system property settings are supported by the AWS SDK for Java and the AWS SDK for Kotlin only.

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	Supported	Notes or more information
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 Kotlin	No	
SDK for .NET 4.x	Yes	Parameters not optimized: `connectTimeoutInMillis` , `tlsNegotiationTimeoutInMillis` .
SDK for .NET 3.x	Yes	Parameters not optimized: `connectTimeoutInMillis` , `tlsNegotiationTimeoutInMillis` .
SDK for PHP 3.x	Yes	Parameters not optimized: `tlsNegotiationTimeoutInMillis` .

SDK	Supported	Notes or more information
SDK for Python (Boto3)	Yes	Parameters not optimized: `tlsNegotiationTimeoutInMillis` .
SDK for Ruby 3.x	Yes	
SDK for Rust	No	
SDK for Swift	No	
Tools for PowerShell V5	Yes	Parameters not optimized: `connectTimeoutInMillis` , `tlsNegotiationTimeoutInMillis` .
Tools for PowerShell V4	Yes	Parameters not optimized: `connectTimeoutInMillis` , `tlsNegotiationTimeoutInMillis` .

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 and Rust.

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.
CRT dependency diagram showing how the individual CRT libraries interrelate with each other.

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 lifecycle

The lifecycle 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 lifecycle. Please note that the timelines shown below are illustrative and not binding.
Maintenance policy timelines

Dependency lifecycle

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 lifecycle, see *Version lifecycle*.

AWS SDKs and Tools version lifecycle

The table below shows the list of available AWS Software Development Kit (SDK) major versions and where they are in the maintenance lifecycle with associated timelines. For detailed information on the lifecycle for the major versions of AWS SDKs and Tools and their underlying dependencies, see *Maintenance policy*.

SDK	Major version	Current Phase	General Availability Date	Notes
AWS CLI	1.x	Maintenance Announcement	9/2/2013	See announcement for details and dates
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	End-of-Support	11/19/2015	
SDK for Java	1.x	End-of-Support	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	End-of-Support	6/19/2014	

SDK	Major version	Current Phase	General Availability Date	Notes
SDK for JavaScript	3.x	General Availability	12/15/2020	
SDK for Kotlin	1.x	General Availability	11/27/2023	
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 .NET	4.x	General Availability	4/28/2025	
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 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	Major version	Current Phase	General Availability Date	Notes
SDK for Rust	1.x	General Availability	11/27/2023	
SDK for Swift	1.x	General Availability	9/17/2024	
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	
Tools for PowerShell	5.x	General Availability	6/23/2025	

Searching for an SDK or tool not mentioned? Encryption SDKs, IoT Device SDKs, and Mobile SDKs, for example, are not included in this guide. To find documentation on these other tools, see Tools to Build on AWS.

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
Adding new S3 Express One Zone setting	Adding new S3 Express One Zone setting to disable of session authentication.	October 13, 2025
Adding new authentication decision tree	Adding new decision tree to assist in authentication decisions between options.	September 23, 2025
Adding new authentication scheme feature	Adding new authentication scheme feature. Updates to AWS STS Regional endpoints.	August 18, 2025
Adding new version of Tools for PowerShell	Adding latest version of Tools for PowerShell support to all Setting reference Compatibi lity with AWS SDKs tables. Added Host prefix injection feature.	June 23, 2025
Page title updates	More titles, table titles, abstracts, and SEO updates.	March 5, 2025
Page title updates	Updating content to use more descriptive titles.	February 24, 2025
Adding Swift SDK to Settings reference	Adding Swift SDK support to all Setting reference	September 17, 2024

	Compatibility with AWS SDKs tables.	
SDK for Java 1.x system properties	Add details about supported JVM system configuration settings by the AWS SDK for Java 1.x.	May 30, 2024
Settings updates	Add JVM system configura tion settings.	March 27, 2024
Compatibility table updates	Updates to compatibility for SDK support, updates to IAM Identity Center procedures.	February 20, 2024
Container credential update. IMDS update.	Adding support for Amazon EKS. Adding setting to disable IMDSv1 fallback.	December 29, 2023
Request compression	Adding settings for request compression feature.	December 27, 2023
Compatibility tables	Compatibility tables for SDK and tool features updated to include SDK for Kotlin, SDK for Rust, and AWS Tools for PowerShell.	December 10, 2023
Authentication updates	Updates to supported methods of authentication for SDKs and tools.	July 1, 2023
IAM best practices updates	Updated guide to align with the IAM best practices . For more information, see Security best practices in IAM.	February 27, 2023

SSO updates	Updates to SSO credentials for the new SSO token configuration.	November 19, 2022
Settings updates	Updates to support table for General configuration and for Amazon S3 Multi-Region Access Points.	November 17, 2022
Settings updates	Updates to clarity of IMDS client and IMDS credentials. Updates to Environment variables.	November 4, 2022
Updating welcome page	Announcing Amazon CodeWhisperer.	September 22, 2022
Service name change for single sign-on	Updates to reflect that AWS SSO is now referred to as AWS IAM Identity Center.	July 26, 2022
Settings update	Minor updates to config file details and to supported settings.	June 15, 2022
Update	Massive update of almost all parts of this guide.	February 1, 2022
Initial release	The first release of this guide is released to the public.	March 13, 2020