AWS SDK for Java

Developer Guide

AWS SDK for Java: Developer Guide

Table of Contents

AWS SDK for Java 2.0 Developer Guide (Developer Preview)

The AWS SDK for Java provides a Java API for Amazon Web Services. Using the SDK, you can easily build Java applications that work with Amazon S3, Amazon EC2, DynamoDB, and more. We regularly add support for new services to the AWS SDK for Java. For a list of changes and features in a particular version, view the change log.

What's New in Version 2.0

The AWS SDK for Java 2.0 is a major rewrite of the version 1.x code base. It's built on top of Java 8 and adds several frequently requested features. These include support for non-blocking I/O and the ability to plug in a different HTTP implementation at run time. For more information see the AWS Blog.

Important
This is a preview release and is not recommended for production environments.

Support for 1.0

We are not dropping support for the 1.x versions of the AWS SDK for Java currently. As we get closer to the final production release, we will share a detailed plan for continued 1.x support, similar to how we rolled out major versions of other AWS SDKs.

Additional Resources

In addition to this guide, the following are valuable online resources for AWS SDK for Java developers:

- AWS SDK for Java 2.0 Reference
- Java developer blog
- Java developer forums
- GitHub:
  - Documentation source

  - SDK source
- @awsforjava (Twitter)

Contributing to the Developer Preview

Developers can also contribute feedback through the following channels:

- Submit issues on GitHub:
  - Submit documentation issues
  - Submit SDK issues
- Join an informal chat about SDK on the AWS SDK for Java 2.0gitter channel
- Submit feedback anonymously to aws-java-sdk-v2-feedback@amazon.com. This email is monitored by the AWS SDK for Java team.
- Submit pull requests in the documentation or SDK source GitHub repositories to contribute to the SDK development.

Eclipse IDE Support

The AWS Toolkit for Eclipse doesn't currently support the AWS SDK for Java 2.0. To use the AWS Toolkit for Eclipse with the AWS SDK for Java 2.0, you should use Maven tools in Eclipse to add a dependency on the 2.0 SDK.

Developing AWS Applications for Android

If you're an Android developer, Amazon Web Services publishes an SDK made specifically for Android development: the AWS Mobile SDK for Android. See the AWS Mobile SDK for Android Developer Guide for the complete documentation.

Getting Started with AWS SDK for Java 2.0 Developer Preview

This section provides information about how to install, set up, and use the AWS SDK for Java.

Topics

Sign up for AWS and Create an IAM User

To use the AWS SDK for Java to access Amazon Web Services (AWS), you need an AWS account and AWS credentials. To increase the security of your AWS account, we recommend that you use an *IAM user* to provide access credentials instead of using your AWS account credentials.

> **Note**
> For an overview of IAM user and why they are important for the security of your account, see AWS Security Credentials in the *Amazon Web Services General Reference*.

To sign up for AWS

1. Open https://aws.amazon.com/ and click **Sign Up**.
2. Follow the on-screen instructions. Part of the sign-up procedure involves receiving a phone call and entering a PIN using your phone keypad.

Next, create an IAM user and download (or copy) its secret access key.

To create an IAM user

1. Go to the IAM console (you may need to sign in to AWS first).
2. Click **Users** in the sidebar to view your IAM users.
3. If you don't have any IAM users set up, click **Create New Users** to create one.
4. Select the IAM user in the list that you'll use to access AWS.

5. Open the **Security Credentials** tab, and click **Create Access Key**.

   Note
   You can have a maximum of two active access keys for any given IAM user. If your IAM user has two access keys already, then you'll need to delete one of them before creating a new key.

6. On the resulting dialog box, click the **Download Credentials** button to download the credential file to your computer, or click **Show User Security Credentials** to view the IAM user's access key ID and secret access key (which you can copy and paste).

   Important
   There is no way to obtain the secret access key once you close the dialog box. You can, however, delete its associated access key ID and create a new one.

Next, set your credentials (p. 5) in the AWS shared credentials file or in the environment.

Set up the AWS SDK for Java 2.0 Developer Preview

This topic describes how to set up and use the AWS SDK for Java in your project.

Prerequisites

To use the AWS SDK for Java, you must have:

- A suitable Java Development Environment (p. 5).
- An AWS account and access keys. For instructions, see Sign up for AWS and Create an IAM User (p. 3).
- AWS credentials (access keys) set in your environment, or use the shared credentials file used by the AWS CLI and other SDKs. For more information, see Set Up AWS Credentials and Region for Development (p. 5).

Including the SDK in Your Project

Depending on your build system or IDE, use one of the following methods:

- **Apache Maven**– If you use Apache Maven, you can specify only the SDK components you need or the entire SDK (not recommended) as dependencies in your project. See Using the SDK with Apache Maven (p. 7).
- **Gradle**– If you use Gradle, you can import the Maven Bill of Materials (BOM) to your Gradle project to automatically manage SDK dependencies. See Using the SDK with Gradle (p. 9).

Note
Any build system that supports MavenCentral as an artifact source may be used. However we will not provide a downloadable zip for the developer preview.

Compiling the SDK

You can build the AWS SDK for Java using Maven. Maven downloads all necessary dependencies, builds the SDK, and installs the SDK in one step. See http://maven.apache.org/ for installation instructions and more information.

To compile the SDK

1. Open AWS SDK for Java 2.0 (GitHub).

   Note
   Version 1.0 of the SDK is also available in GitHub at AWS SDK for Java 1.x (GitHub).
2. Click the **Clone or download** button to choose your download option.
3. In a terminal window, navigate to the directory where you downloaded the SDK source.
4. Build and install the SDK by using the following command (Maven required).

   ```
   mvn clean install
   ```

   The resulting `.jar` file is built into the `target` directory.
5. (Optional) Build the API Reference documentation using the following command.

   ```
   mvn javadoc:javadoc
   ```

   The documentation is built into the `target/site/apidocs/` directories of each service.

Installing a Java Development Environment

The AWS SDK for Java requires Java SE Development Kit *8.0 or later*. You can download the latest Java software from http://www.oracle.com/technetwork/java/javase/downloads/.

Choosing a JVM

For the best performance of your server-based applications with the AWS SDK for Java, we recommend that you use the *64-bit version* of the Java Virtual Machine (JVM). This JVM runs only in server mode, even if you specify the `-Client` option at run time.

Set Up AWS Credentials and Region for Development

To connect to any of the supported services with the AWS SDK for Java, you must provide AWS credentials. The AWS SDKs and CLIs use *provider chains* to look for AWS credentials in several different places, including system/user environment variables and local AWS configuration files.

This topic provides basic information about setting up your AWS credentials for local application development using the AWS SDK for Java. If you need to set up credentials for use within an Amazon EC2 instance or if you're using the Eclipse IDE for development, see the following topics instead:

- When using an EC2 instance, create an IAM role and then give your EC2 instance access to that role as shown in Configure IAM Roles for Amazon EC2 (p. 23).
- Set up AWS credentials within Eclipse using the AWS Toolkit for Eclipse. See Set up AWS Credentials in the AWS Toolkit for Eclipse User Guide.

Setting AWS Credentials

You can set your credentials for use by the AWS SDK for Java in several ways. However, these are the recommended approaches:

- Set credentials in the AWS credentials profile file on your local system, located at:
  - `~/.aws/credentials` on Linux, macOS, or Unix
  - `C:\Users\USERNAME \.aws\credentials` on Windows

  This file should contain lines in the following format:

```
[default]
aws_access_key_id = your_access_key_id
aws_secret_access_key = your_secret_access_key
```

  Substitute your own AWS credentials values for the values *your_access_key_id* and *your_secret_access_key*.
- Set the `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` environment variables.

  To set these variables on Linux, macOS, or Unix, use **`export`** :

```
export AWS_ACCESS_KEY_ID=your_access_key_id
export AWS_SECRET_ACCESS_KEY=your_secret_access_key
```

  To set these variables on Windows, use **`set`** :

```
set AWS_ACCESS_KEY_ID=your_access_key_id
set AWS_SECRET_ACCESS_KEY=your_secret_access_key
```

- For an EC2 instance, specify an IAM role and then give your EC2 instance access to that role. See IAM Roles for Amazon EC2 in the *Amazon EC2 User Guide for Linux Instances* for a detailed discussion about how this works.

Once you set your AWS credentials using one of these methods, the AWS SDK for Java loads them automatically by using the default credential provider chain. For more information about working with AWS credentials in your Java applications, see Working with AWS Credentials (p. 12).

Setting the AWS Region

You should set a default AWS Region to use for accessing AWS services with the AWS SDK for Java. For the best network performance, choose a region that's geographically close to you (or to your customers).

> **Note**
> If you *don't* select a region, service calls that require a region will fail.

You can use techniques similar to those for setting credentials to set your default AWS Region:

- Set the AWS Region in the AWS config file on your local system, located at:
  - `~/.aws/config` on Linux, macOS, or Unix
  - `C:\Users\USERNAME\.aws\config` on Windows

  This file should contain lines in the following format:

```
[default]
region = your_aws_region
```

  Substitute your desired AWS Region (for example, "us-west-2") for *your_aws_region*.
- Set the `AWS_REGION` environment variable.

  On Linux, macOS, or Unix, use **`export`** :

```
export AWS_REGION=your_aws_region
```

On Windows, use **`set`**:

```
set AWS_REGION=your_aws_region
```

Where *your_aws_region* is the desired AWS Region name.

For information about selecting a region, see AWS Region Selection (p. 15).

Using the SDK with Apache Maven

You can use Apache Maven to configure and build AWS SDK for Java projects or to build the SDK itself.

Note
You must have Maven installed to use the guidance in this topic. If it isn't already installed, visit http://maven.apache.org/ to download and install it.

Create a New Maven Package

To create a basic Maven package, open a terminal (command line) window and run the following.

```
mvn -B archetype:generate \
  -DarchetypeGroupId=org.apache.maven.archetypes \
  -DgroupId=org.example.basicapp \
  -DartifactId=myapp
```

Replace *org.example.basicapp* with the full package namespace of your application. Replace *myapp* with your project name (this becomes the name of the directory for your project).

By default, Maven creates a project template for you using the quickstart archetype. This creates a Java 1.5 project. You must update your application to Java 1.8 to be compatible with AWS SDK for Java 2.0. To update to Java 1.8, add the following to your `pom.xml` file.

```
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <configuration>
         <source>1.8</source>
         <target>1.8</target>
      </configuration>
    </plugin>
  </plugins>
</build>
```

You can choose a particular archetype to use by adding the `-DarchetypeArtifactId` argument to the `archetype:generate` command. To skip step to update the `pom.xml` file, you can use the following archetype that creates a Java 1.8 project from the start.

```
mvn archetype:generate -B \
   -DarchetypeGroupId=pl.org.miki \
```

```
  -DarchetypeArtifactId=java8-quickstart-archetype \
  -DarchetypeVersion=1.0.0 \
  -DgroupId=com.example \
  -DartifactId=sdk-sandbox \
  -Dversion=1.0 \
  -Dpackage=com.example
```

There are more archetypes available. See Maven Archetypes for a list of archetypes packaged with Maven.

Note
For much more information about creating and configuring Maven projects, see the Maven Getting Started Guide.

Configure the SDK as a Maven Dependency

To use the AWS SDK for Java in your project, you need to declare it as a dependency in your project's `pom.xml` file. You can import individual components (p. 8) or the entire SDK (p. 9). We strongly recommend that you pull in only the components you need instead of the entire SDK.

Specifying Individual SDK Modules (Recommended)

To select individual SDK modules, use the AWS SDK for Java bill of materials (BOM) for Maven. This ensures that the modules you specify use the same version of the SDK, and that they're compatible with each other.

To use the BOM, add a `<dependencyManagement>` section to your application's `pom.xml` file. Add `bom` as a dependency and specify the version of the SDK to use.

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>software.amazon.awssdk</groupId>
      <artifactId>bom</artifactId>
      <version>2.0.0-preview-1</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
```

To view the latest version of the AWS SDK for Java BOM that is available on Maven Central, see https://mvnrepository.com/artifact/software.amazon.awssdk/bom. This page also shows the modules (dependencies) that are managed by the BOM that you can include within the `<dependencies>` section of your project's `pom.xml` file.

You can now select individual modules from the SDK that you use to your application. Because you already declared the SDK version in the BOM, you don't need to specify the version number for each component.

```
<dependencies>
  <dependency>
    <groupId>software.amazon.awssdk</groupId>
    <artifactId>s3</artifactId>
  </dependency>
  <dependency>
    <groupId>software.amazon.awssdk</groupId>
    <artifactId>dynamodb</artifactId>
  </dependency>
```

```
</dependencies>
```

Importing All SDK Modules (Not Recommended)

To pull in the *entire* SDK as a dependency, don't use the BOM method. Simply declare it in your `pom.xml` as follows.

```
<dependencies>
  <dependency>
    <groupId>software.amazon.awssdk</groupId>
    <artifactId>aws-java-sdk</artifactId>
    <version>2.0.0-preview-1</version>
  </dependency>
</dependencies>
```

Build Your Project

Once you set up your project, you can build it using Maven's `package` command.

```
mvn package
```

This creates your `.jar` file in the `target` directory.

Using the SDK with Gradle

To use the AWS SDK for Java in your Gradle project, use Spring's dependency management plugin for Gradle. You can use this plugin to import the SDK's Maven Bill of Materials (BOM) to manage SDK dependencies for your project.

To configure the SDK for Gradle

1. Add the dependency management plugin to your `build.gradle` file.

```
buildscript {
    repositories {
        mavenCentral()
    }
    dependencies {
        classpath "io.spring.gradle:dependency-management-plugin:1.0.0.RC2"
    }
}

apply plugin: "io.spring.dependency-management"
```

2. Add the BOM to the *dependencyManagement* section of the file.

```
dependencyManagement {
    imports {
        mavenBom 'software.amazon.awssdk:bom:2.0.0-preview-1'
    }
}
```

3. Specify the SDK modules you want to use in the *dependencies* section.

```
dependencies {
```

```
    compile 'software.amazon.awssdk:s3'
    testCompile group: 'junit', name: 'junit', version: '4.11'
}
```

Gradle automatically resolves the correct version of your SDK dependencies using the information from the BOM.

Note
For more detail about specifying SDK dependencies using the BOM, see Using the SDK with Apache Maven (p. 7).

Using the AWS SDK for Java 2.0 Developer Preview

This section provides important general information about programming with the AWS SDK for Java that applies to all services you might use with the SDK.

Topics

Creating Service Clients

To make requests to Amazon Web Services, you first create a service client object. In version 2.0 of the SDK, you can create clients only by using service client builders.

Each AWS service has a service interface with methods for each action in the service API. For example, the service interface for Amazon DynamoDB is named DynamoDbClient. Each service interface has a static factory builder method you can use to construct an implementation of the service interface.

Obtaining a Client Builder

To obtain an instance of the client, use the static factory method `builder`. Then customize it by using the setters in the builder, as shown in the following example.

In the AWS SDK for Java 2.0, the setters are named without the `with` prefix.

```
DynamoDBClient client = DynamoDBClient.builder()
                        .region(Region.US_WEST_2)
                        .credentialsProvider(ProfileCredentialsProvider.builder()
                                    .profileName("myProfile")
                                    .build())
                        .build();
```

Note
The fluent setter methods return the `builder` object, so that you can chain the method calls for convenience and for more readable code. After you configure the properties you want, you can call the `build` method to create the client. Once a client is created, it's immutable. The only way to create a client with different settings is to build a new client.

Using DefaultClient

The client builders have another factory method named `create`. This method creates a service client with the default configuration. It uses the default provider chain to load credentials and the AWS Region. If credentials or the region can't be determined from the environment that the application is running in, the call to `create` fails. See Working with AWS Credentials (p. 12) and AWS Region Selection (p. 15) for more information about how credentials and region are determined.

To create a default client

```
DynamoDBClient client = DynamoDBClient.create();
```

Client Lifecycle

Service clients in the SDK are thread-safe. For best performance, treat them as long-lived objects. Each client has its own connection pool resource that is released when the client is garbage collected. The clients in the AWS SDK for Java 2.0 now extend the AutoClosable interface. For best practices, explicitly close a client by calling the `close` method.

To close a client

```
DynamoDBClient client = DynamoDBClient.create();
client.close();
```

Working with AWS Credentials

To make requests to Amazon Web Services, you must supply AWS credentials to the AWS SDK for Java. You can do this in the following ways:

- Use the default credential provider chain *(recommended)*.
- Use a specific credential provider or provider chain (or create your own).
- Supply the credentials yourself. These can be AWS account credentials, IAM credentials, or temporary credentials retrieved from AWS STS.

Important
For security, we *strongly recommend* that you *use IAM account credentials* instead of the AWS account credentials for AWS access. For more information, see AWS Security Credentials in the *Amazon Web Services General Reference*.

Using the Default Credential Provider Chain

When you initialize a new service client without supplying any arguments, the AWS SDK for Java attempts to find AWS credentials by using the *default credential provider chain* implemented by the DefaultCredentialsProvider class. The default credential provider chain looks for credentials in this order:

1. **Java system properties**–`aws.accessKeyId` and `aws.secretKey`. The AWS SDK for Java uses the SystemPropertyCredentialsProvider to load these credentials.
2. **Environment variables**–`AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY`. The AWS SDK for Java uses the EnvironmentVariableCredentialsProvider class to load these credentials.
3. **The default credential profiles file**– typically located at `~/.aws/credentials` (location can vary per platform), and shared by many of the AWS SDKs and by the AWS CLI. The AWS SDK for Java uses the ProfileCredentialsProvider to load these credentials.

   You can create a credentials file by using the `aws configure` command provided by the AWS CLI. Or you can create it by editing the file with a text editor. For information about the credentials file format, see AWS Credentials File Format (p. 14).
4. **Amazon ECS container credentials**– loaded from the Amazon ECS if the environment variable `AWS_CONTAINER_CREDENTIALS_RELATIVE_URI` is set. The AWS SDK for Java uses the ElasticContainerCredentialsProvider to load these credentials.
5. **Instance profile credentials**– used on Amazon EC2 instances, and delivered through the Amazon EC2 metadata service. The AWS SDK for Java uses the InstanceProfileCredentialsProvider to load these credentials.

Setting Credentials

To be able to use AWS credentials, they must be set in *at least one* of the preceding locations. For information about setting credentials, see the following topics:

- To specify credentials in the *environment* or in the default *credential profiles file*, see Set Up AWS Credentials and Region for Development (p. 5).
- To set Java *system properties*, see the System Properties tutorial on the official *Java Tutorials* website.
- To set up and use *instance profile credentials* with your EC2 instances, see Configure IAM Roles for Amazon EC2 (p. 23).

Setting an Alternate Credentials Profile

The AWS SDK for Java uses the *default* profile by default, but there are ways to customize which profile is sourced from the credentials file.

You can use the `AWS_PROFILE` environment variable to change the profile loaded by the SDK.

For example, on Linux, macOS, or Unix, you run the following command to change the profile to *myProfile*.

```
export AWS_PROFILE="myProfile"
```

On Windows, you use the following.

```
set AWS_PROFILE="myProfile"
```

Setting the `AWS_PROFILE` environment variable affects credential loading for all officially supported AWS SDKs and Tools (including the AWS CLI and the AWS CLI for PowerShell). To change only the profile for a Java application, you can use the system property *aws.profile* instead.

Setting an Alternate Credentials File Location

The AWS SDK for Java loads AWS credentials automatically from the default credentials file location. However, you can also specify the location by setting the `AWS_CREDENTIAL_PROFILES_FILE` environment variable with the full path to the credentials file.

You can use this feature to temporarily change the location where the AWS SDK for Java looks for your credentials file (for example, by setting this variable with the command line). Or you can set the environment variable in your user or system environment to change it for the user or systemwide.

To override the default credentials file location

- Set the `AWS_CREDENTIAL_PROFILES_FILE` environment variable to the location of your AWS credentials file.
  - On Linux, macOS, or Unix, use **`export`** :

```
export AWS_CREDENTIAL_PROFILES_FILE=path/to/credentials_file
```

  - On Windows, use **`set`** :

```
set AWS_CREDENTIAL_PROFILES_FILE=path/to/credentials_file
```

AWS Credentials File Format

When you use the `aws configure` command to create an AWS credentials file, the commmand creates a file with the following format.

```
[default]
aws_access_key_id={YOUR_ACCESS_KEY_ID}
aws_secret_access_key={YOUR_SECRET_ACCESS_KEY}

[profile2]
aws_access_key_id={YOUR_ACCESS_KEY_ID}
aws_secret_access_key={YOUR_SECRET_ACCESS_KEY}
```

The profile name is specified in square brackets (for example, `[default]`), followed by the configurable fields in that profile as key-value pairs. You can have multiple profiles in your credentials file, which can be added or edited using `aws configure --profile` *`PROFILE_NAME`* to select the profile to configure. In addition to the access key and secret access keys, you can specify a session token using the `aws_session_token` field.

Loading Credentials

After you set credentials, you can load them by using the default credential provider chain.

To do this, you instantiate an AWS service client without explicitly providing credentials to the builder, as follows.

```
S3Client s3 = S3Client.builder()
                      .region(Regions.US_WEST_2)
                      .build();
```

Specifying a Credential Provider or Provider Chain

You can specify a credential provider that is different from the *default* credential provider chain by using the client builder.

You provide an instance of a credentials provider or provider chain to a client builder that takes an AwsCredentialsProvider interface as input. The following example shows how to use *environment* credentials specifically.

```
S3Client s3 = S3Client.builder()
                      .credentialsProvider(new EnvironmentVariableCredentialsProvider())
                      .build();
```

For the full list of AWS SDK for Java-supplied credential providers and provider chains, see **All Known Implementing Classes** in AwsCredentialsProvider.

> **Note**
> You can use this technique to supply credential providers or provider chains that you create by using your own credential provider that implements the `AwsCredentialsProvider` interface.

Explicitly Specifying Credentials

If the default credential chain or a specific or custom provider or provider chain doesn't work for your code, you can set credentials that you supply explicitly. If you've retrieved temporary credentials using AWS STS, use this method to specify the credentials for AWS access.

To explicitly supply credentials to an AWS client

1. Instantiate a class that provides the AwsCredentials interface, such as AwsSessionCredentials. Supply it with the AWS access key and secret key you will use for the connection.
2. Create an AwsStaticCredentialsProvider with the `AwsCredentials` object.
3. Configure the client builder with the `AwsStaticCredentialsProvider` and build the client.

The following is an example.

```
AwsSessionCredentials awsCreds = new AwsSessionCredentials(
                                     "access_key_id",
                                     "secret_key_id",
                                     "session_token");
S3Client s3 = S3Client.builder()
                      .credentialsProvider(new AwsStaticCredentialsProvider(awsCreds))
                      .build();
```

More Info

- Sign up for AWS and Create an IAM User (p. 3)
- Set Up AWS Credentials and Region for Development (p. 5)
- Configure IAM Roles for Amazon EC2 (p. 23)

AWS Region Selection

Regions enable you to access AWS services that physically reside in a specific geographic area. This can be useful both for redundancy and to keep your data and applications running close to where you and your users will access them.

In AWS SDK for Java 2.0, all the different region related classes from version 1.x have been collapsed into one Region class. You can use this class for all region-related actions such as retrieving metadata about a region or checking whether a service is available in a region.

Choosing a Region

You can specify a region name and the SDK will automatically choose an appropriate endpoint for you.

To explicitly set a region, we recommend that you use the constants defined in the Region class. This is an enumeration of all publicly available regions. To create a client with a region from the class, use the following code.

```
EC2Client ec2 = EC2Client.builder()
                    .region(Region.US_WEST_2)
                    .build();
```

If the region you are attempting to use isn't one of the constants in the `Region` class, you can create a new region using the `of` method. This feature allows you access to new Regions without upgrading the SDK.

```
Region newRegion = Region.of("us-east-42");
EC2Client ec2 = EC2Client.builder()
                    .region(newRegion)
                    .build();
```

Note
After you build a client with the builder, it's *immutable* and the region *cannot be changed*. If you are working with multiple AWS Regions for the same service, you should create multiple clients —one per region.

Automatically Determine the AWS Region from the Environment

When running on Amazon EC2 or AWS Lambda, you might want to configure clients to use the same region that your code is running on. This decouples your code from the environment it's running in and makes it easier to deploy your application to multiple regions for lower latency or redundancy.

To use the default credential/region provider chain to determine the region from the environment, use the client builder's `create` method.

```
EC2Client ec2 = EC2Client.create();
```

If you don't explicitly set a region using the `region` method, the SDK consults the default region provider chain to try and determine the region to use.

Default Region Provider Chain

The following is the region lookup process:

1. Any explicit region set by using `region` on the builder itself takes precedence over anything else.
2. The `AWS_REGION` environment variable is checked. If it's set, that region is used to configure the client.

   Note
   This environment variable is set by the Lambda container.
3. The SDK checks the AWS shared configuration file (usually located at `~/.aws/config`). If the *region* property is present, the SDK uses it.
   - The `AWS_CONFIG_FILE` environment variable can be used to customize the location of the shared config file.
   - The `AWS_PROFILE` environment variable or the *aws.profile* system property can be used to customize the profile that the SDK loads.
4. The SDK attempts to use the Amazon EC2 instance metadata service to determine the region of the currently running Amazon EC2 instance.

5. If the SDK still hasn't found a region by this point, client creation fails with an exception.

When developing AWS applications, a common approach is to use the *shared configuration file* (described in Using the Default Credential Provider Chain (p. 12)) to set the region for local development, and rely on the default region provider chain to determine the region when running on AWS infrastructure. This greatly simplifies client creation and keeps your application portable.

Checking for Service Availability in an AWS Region

To see if a particular AWS service is available in a region, use the `serviceMetadata` and `region` method on the service that you'd like to check.

```
DynamoDBClient.serviceMetadata().regions().forEach(System.out::println);
```

See the Region class documentation for the regions you can specify, and use the endpoint prefix of the service to query.

Asynchronous Programming

AWS SDK for Java 2.0 features truly non-blocking asychronous clients that implements high concurrency across a few threads. AWS SDK for Java 1.11.x has asynchronous clients that are wrappers around a thread pool and blocking synchronous clients which do not provide the full benefit of non-blocking I/O.

Synchronous methods block your thread's execution until the client receives a response from the service. Asynchronous methods return immediately, giving control back to the calling thread without waiting for a response.

Because an asynchronous method returns before a response is available, you need a way to get the response when it's ready. The AWS SDK for Java 2.0 asynchronous client methods return *CompletableFuture objects* that allows you to access the response when it's ready.

Non-streaming Operations

For non-streaming operations, asychronous method calls are similar to synchronous methods except that the asynchronous methods in the AWS SDK for Java return a CompletableFuture object that contains the results of the asynchronous operation *in the future*.

Call the `CompletableFuture` `whenComplete()` method with an action to complete when the result is available. `CompletableFuture` implements the `Future` interface so you can get the response object by calling the `get()` method as well.

Here is an example of an asynchronous operation that calls a DynamoDB function to get a list of tables, receiving a `CompletableFuture` that can hold a ListTablesResponse object. The action defined in the call to `whenComplete()` is only done when the asynchronous call is complete.

```
public class DynamoDBAsync {

    public static void main(String[] args) {
     // Creates a default async client with credentials and regions loaded from the
 environment
     DynamoDBAsyncClient client = DynamoDBAsyncClient.create();
     CompletableFuture<ListTablesResponse> response =
 client.listTables(ListTablesRequest.builder()
```

```
 .limit(5)

 .build());
     // Map the response to another CompletableFuture containing just the table names
     CompletableFuture<List<String>> tableNames =
 response.thenApply(ListTablesResponse::tableNames);
     // When future is complete (either successfully or in error) handle the response
     tableNames.whenComplete((tables, err) -> {
         if (tables != null) {
             tables.forEach(System.out::println);
         } else {
             // Handle error
             err.printStackTrace();
         }
     });
    }
}
```

Streaming Operations

For streaming operations, you must provide an AsyncRequestProvider to provide the content incrementally or an AsyncResponseHandler to receive and process the response.

Here is an example that uploads a file to Amazon S3 asynchronously with the `PutObject` operation.

```
public class S3AsyncOps {

    private static final String BUCKET = "sample-bucket";
 private static final String KEY = "testfile.in";

 public static void main(String[] args) {
     S3AsyncClient client = S3AsyncClient.create();
        CompletableFuture<PutObjectResponse> future = client.putObject(
                PutObjectRequest.builder()
                                .bucket(BUCKET)
                                .key(KEY)
                                .build(),
                AsyncRequestProvider.fromFile(Paths.get("myfile.in"))
        );
        future.whenComplete((resp, err) -> {
            try {
                if (resp != null) {
                    System.out.println("my response: " + resp);
                } else {
                    // Handle error
                    err.printStackTrace();
                }
            } finally {
                // Lets the application shut down. Only close the client when you are
 completely done with it.
                FunctionalUtils.invokeSafely(client::close);
            }
        });
    }
}
```

Here is an example that gets a file from Amazon S3 asynchronously with the `GetObject` operation.

```
public class S3AsyncStreamOps {

    private static final String BUCKET = "sample-bucket";
```

```
 private static final String KEY = "testfile.out";

 public static void main(String[] args) {
     S3AsyncClient client = S3AsyncClient.create();
      final CompletableFuture<Void> futureGet = client.getObject(
                GetObjectRequest.builder()
                                .bucket(BUCKET)
                                .key(KEY)
                                .build(),
                AsyncResponseHandler.toFile(Paths.get("myfile.out")));
      futureGet.whenComplete((resp, err) -> {
            try {
                if (resp != null) {
                    System.out.println(resp);
                } else {
                    // Handle error
                    err.printStackTrace();
                }
            } finally {
                // Lets the application shut down. Only close the client when you are
 completely done with it
                FunctionalUtils.invokeSafely(client::close);
            }
        });
    }
}
```

Exception Handling

Understanding how and when the AWS SDK for Java throws exceptions is important to building high-quality applications using the SDK. The following sections describe the different cases of exceptions that are thrown by the SDK and how to handle them appropriately.

Why Unchecked Exceptions?

The AWS SDK for Java uses runtime (or unchecked) exceptions instead of checked exceptions for these reasons:

- To allow developers fine-grained control over the errors they want to handle without forcing them to handle exceptional cases they aren't concerned about (and making their code overly verbose)
- To prevent scalability issues inherent with checked exceptions in large applications

In general, checked exceptions work well on small scales, but can become troublesome as applications grow and become more complex.

AmazonServiceException (and Subclasses)

AmazonServiceException is the most common exception that you'll experience when using the AWS SDK for Java. This exception represents an error response from an AWS service. For example, if you try to terminate an Amazon EC2 instance that doesn't exist, EC2 will return an error response and all the details of that error response will be included in the `AmazonServiceException` that's thrown. For some cases, a subclass of `AmazonServiceException` is thrown to allow developers fine-grained control over handling error cases through catch blocks.

When you encounter an `AmazonServiceException`, you know that your request was successfully sent to the AWS service but couldn't be successfully processed. This can be because of errors in the request's parameters or because of issues on the service side.

`AmazonServiceException` provides you with information such as:

- Returned HTTP status code
- Returned AWS error code
- Detailed error message from the service
- AWS request ID for the failed request

`AmazonServiceException` also includes information about whether the failed request was the caller's fault (a request with illegal values) or the AWS service's fault (an internal service error).

AmazonClientException

AmazonClientException indicates that a problem occurred inside the Java client code, either while trying to send a request to AWS or while trying to parse a response from AWS. An `AmazonClientException` is generally more severe than an `AmazonServiceException`, and indicates a major problem that is preventing the client from making service calls to AWS services. For example, the AWS SDK for Java throws an `AmazonClientException` if no network connection is available when you try to call an operation on one of the clients.

Logging AWS SDK for Java Calls

The AWS SDK for Java is instrumented with Apache Commons Logging, which is an abstraction layer that enables the use of any one of several logging systems at runtime.

Supported logging systems include the Java Logging Framework and Apache Log4j, among others. This topic shows you how to use Log4j. You can use the SDK's logging functionality without making any changes to your application code.

To learn more about Log4j, see the Apache website.

> **Note**
> This topic focuses on Log4j 1.x. Log4j2 doesn't directly support Apache Commons Logging, but provides an adapter that directs logging calls automatically to Log4j2 using the Apache Commons Logging interface. For more information, see Commons Logging Bridge in the Log4j2 documentation.

Download the Log4J JAR

To use Log4j with the SDK, you need to download the Log4j JAR from the Apache website. The SDK doesn't include the JAR. Copy the JAR file to a location that is on your classpath.

Log4j uses a configuration file, log4j.properties. Example configuration files are shown below. Copy this configuration file to a directory on your classpath. The Log4j JAR and the log4j.properties file don't have to be in the same directory.

The log4j.properties configuration file specifies properties such as logging level, where logging output is sent (for example, to a file or to the console), and the format of the output. The logging level is the granularity of output that the logger generates. Log4j supports the concept of multiple logging *hierarchies*. The logging level is set independently for each hierarchy. The following two logging hierarchies are available in the AWS SDK for Java:

- log4j.logger.software.amazon.awssdk
- log4j.logger.org.apache.http.wire

Setting the Classpath

Both the Log4j JAR and the log4j.properties file must be located on your classpath. If you're using Apache Ant, set the classpath in the `path` element in your Ant file. The following example shows a path element from the Ant file for the Amazon S3 example included with the SDK.

```
<path id="aws.java.sdk.classpath">
  <fileset dir="../../third-party" includes="**/*.jar"/>
  <fileset dir="../../lib" includes="**/*.jar"/>
  <pathelement location="."/>
</path>
```

If you're using the Eclipse IDE, you can set the classpath by opening the menu and navigating to **Project | Properties | Java Build Path**.

Service-Specific Errors and Warnings

We recommend that you always leave the "software.amazon.awssdk" logger hierarchy set to "WARN" to catch any important messages from the client libraries. For example, if the Amazon S3 client detects that your application hasn't properly closed an `InputStream` and could be leaking resources, the S3 client reports it through a warning message to the logs. This also ensures that messages are logged if the client has any problems handling requests or responses.

The following log4j.properties file sets the `rootLogger` to WARN, which causes warning and error messages from all loggers in the "software.amazon.awssdk" hierarchy to be included. Alternatively, you can explicitly set the software.amazon.awssdk logger to WARN.

```
log4j.rootLogger=WARN, A1
log4j.appender.A1=org.apache.log4j.ConsoleAppender
log4j.appender.A1.layout=org.apache.log4j.PatternLayout
log4j.appender.A1.layout.ConversionPattern=%d [%t] %-5p %c -  %m%n
# Or you can explicitly enable WARN and ERROR messages for the AWS Java clients
log4j.logger.software.amazon.awssdk=WARN
```

Request/Response Summary Logging

Every request to an AWS service generates a unique AWS request ID that is useful if you run into an issue with how an AWS service is handling a request. AWS request IDs are accessible programmatically through Exception objects in the SDK for any failed service call, and can also be reported through the DEBUG log level in the "software.amazon.awssdk.request" logger.

The following log4j.properties file enables a summary of requests and responses, including AWS request IDs.

```
log4j.rootLogger=WARN, A1
log4j.appender.A1=org.apache.log4j.ConsoleAppender
log4j.appender.A1.layout=org.apache.log4j.PatternLayout
log4j.appender.A1.layout.ConversionPattern=%d [%t] %-5p %c -  %m%n
# Turn on DEBUG logging in software.amazon.awssdk.request to log
# a summary of requests/responses with AWS request IDs
log4j.logger.software.amazon.awssdk.request=DEBUG
```

Here is an example of the log output.

```
2009-12-17 09:53:04,269 [main] DEBUG software.amazon.awssdk.request - Sending
Request: POST https://rds.amazonaws.com / Parameters: (MaxRecords: 20,
```

```
Action: DescribeEngineDefaultParameters, SignatureMethod: HmacSHA256,
AWSAccessKeyId: ACCESSKEYID, Version: 2009-10-16, SignatureVersion: 2,
Engine: mysql5.1, Timestamp: 2009-12-17T17:53:04.267Z, Signature:
q963XH63Lcovl5Rr71APlzlye99rmWwT9DfuQaNznkD, ) 2009-12-17 09:53:04,464
[main] DEBUG software.amazon.awssdk.request - Received successful response: 200, AWS
Request ID: 694d1242-cee0-c85e-f31f-5dab1ea18bc6 2009-12-17 09:53:04,469
[main] DEBUG software.amazon.awssdk.request - Sending Request: POST
https://rds.amazonaws.com / Parameters: (ResetAllParameters: true, Action:
ResetDBParameterGroup, SignatureMethod: HmacSHA256, DBParameterGroupName:
java-integ-test-param-group-0000000000000, AWSAccessKeyId: ACCESSKEYID,
Version: 2009-10-16, SignatureVersion: 2, Timestamp:
2009-12-17T17:53:04.467Z, Signature:
9WcgfPwTobvLVcpyhbrdN7P7l3uH0oviYQ4yZ+TQjsQ=, )

2009-12-17 09:53:04,646 [main] DEBUG software.amazon.awssdk.request - Received
successful response: 200, AWS Request ID:
694d1242-cee0-c85e-f31f-5dab1ea18bc6
```

Verbose Wire Logging

In some cases, it can be useful to see the exact requests and responses that the AWS SDK for Java sends and receives. If you really need access to this information, you can temporarily enable it through the Apache HttpClient 4 logger. Enabling the DEBUG level on the `apache.http.wire` logger enables logging for all request and response data.

> **Warning**
> We recommend you only use wire logging for debugging purposes. Disable it in your production environments because it can log sensitive data. It logs the full request or response without encryption, even for an HTTPS call. For large requests (e.g., to upload a file to Amazon S3) or responses, verbose wire logging can also significantly impact your application's performance.

The following log4j.properties file turns on full wire logging in Apache HttpClient 4.

```
log4j.rootLogger=WARN, A1
log4j.appender.A1=org.apache.log4j.ConsoleAppender
log4j.appender.A1.layout=org.apache.log4j.PatternLayout
log4j.appender.A1.layout.ConversionPattern=%d [%t] %-5p %c -  %m%n
# Log all HTTP content (headers, parameters, content, etc)  for
# all requests and responses. Use caution with this since it can
# be very expensive to log such verbose data!
log4j.logger.org.apache.http.wire=DEBUG
```

Advanced Topics for AWS SDK for Java 2.0

This section provides more advanced topics about programming with the AWS SDK for Java 2.0 that applies to specific use cases.

Topics

Configure IAM Roles for Amazon EC2

All requests to AWS services must be cryptographically signed using credentials issued by AWS. You can use *IAM roles* to conveniently grant secure access to AWS resources from your Amazon EC2 instances.

This topic provides information about how to use IAM roles with AWS SDK for Java applications running on Amazon EC2. For more information about IAM instances, see IAM Roles for Amazon EC2 in the *Amazon EC2 User Guide for Linux Instances*.

Default Provider Chain and Amazon EC2 Instance Profiles

If your application creates an AWS client using the `create` method, the client searches for credentials using the *default credentials provider chain*, in the following order:

1. In the Java system properties: `aws.accessKeyId` and `aws.secretKey`.
2. In system environment variables: `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY`.
3. In the default credentials file (the location of this file varies by platform).
4. In the Amazon ECS environment variable: `AWS_CONTAINER_CREDENTIALS_RELATIVE_URI`.
5. In the *instance profile credentials*, which exist within the instance metadata associated with the IAM role for the EC2 instance.

The final step in the default provider chain is available only when running your application on an Amazon EC2 instance. However, it provides the greatest ease of use and best security when working with Amazon EC2 instances. You can also pass an InstanceProfileCredentialsProvider instance directly to the client constructor to get instance profile credentials without proceeding through the entire default provider chain.

For example:

```
S3Client s3 = S3Client.builder()
              .credentialsProvider(InstanceProfileCredentialsProvider.builder().build())
              .build();
```

When you use this approach, the SDK retrieves temporary AWS credentials that have the same permissions as those associated with the IAM role that is associated with the Amazon EC2 instance in its instance profile. Although these credentials are temporary and would eventually expire, `InstanceProfileCredentialsProvider` periodically refreshes them for you so that the obtained credentials continue to allow access to AWS.

Walkthrough: Using IAM roles for Amazon EC2 Instances

This walkthrough shows you how to retrieve an object from Amazon S3 using an IAM role to manage access.

Create an IAM Role

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

To create the IAM role

1. Open the IAM console.
2. In the navigation pane, choose **Roles**, then **Create New Role**.
3. On the **Select Role Type** page, under **AWS Service Roles**, choose **Amazon EC2**.
4. On the **Attach Policy** page, choose **Amazon S3 Read Only Access** from the policy list, then choose **Next Step**.
5. **Enter a name for the role, then select Next Step. Remember this name**

   because you'll need it when you launch your Amazon EC2 instance.
6. On the **Review** page, choose **Create Role**.

Launch an EC2 Instance and Specify Your IAM Role

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

To launch an Amazon EC2 instance using the console, follow the directions in Getting Started with Amazon EC2 Linux Instances in the *Amazon EC2 User Guide for Linux Instances*.

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

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

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

Amazon S3 Examples

This section provides examples of programming Amazon S3 using the AWS SDK for Java 2.0.

> **Note**
> The examples include only the code needed to demonstrate each technique. The complete example code is available on GitHub. From there, you can download a single source file or clone the repository locally to get all the examples to build and run.

Topics

Creating, Listing, and Deleting Amazon S3 Buckets

Every object (file) in Amazon S3 must reside within a *bucket*. A bucket represents a collection (container) of objects. Each bucket must have a unique *key* (name). For detailed information about buckets and their configuration, see Working with Amazon S3 Buckets in the *Amazon S3 Developer Guide*.
<admonition>
<title>Best Practice</title>

We recommend that you enable the AbortIncompleteMultipartUpload lifecycle rule on your Amazon S3 buckets.

This rule directs Amazon S3 to abort multipart uploads that don't complete within a specified number of days after being initiated. When the set time limit is exceeded, Amazon S3 aborts the upload and then deletes the incomplete upload data.

For more information, see Lifecycle Configuration for a Bucket with Versioning in the *Amazon S3 User Guide*.
</admonition>

> **Note**
> These code snippets assume that you understand the material in Using the AWS SDK for Java 2.0 Developer Preview (p. 11), and have configured default AWS credentials using the information in Set Up AWS Credentials and Region for Development (p. 5).

Create a Bucket

Build a CreateBucketRequest and provide a bucket name. Pass it to the S3Client`createBucket` method. Use the S3Client to do additional operations such as listing or deleting buckets as shown in later examples.

Imports

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.s3.S3Client;
import software.amazon.awssdk.services.s3.model.CreateBucketConfiguration;
import software.amazon.awssdk.services.s3.model.CreateBucketRequest;
```

Code

```
// Create bucket
CreateBucketRequest createBucketRequest = CreateBucketRequest
        .builder()
        .bucket(bucket)
```

```
        .createBucketConfiguration(CreateBucketConfiguration.builder()

 .locationConstraint(region.value())
                                                                .build())
        .build();
s3.createBucket(createBucketRequest);
```

See the complete example.

List the Buckets

Build a ListBucketRequest. Use the S3Client`listBuckets` method to retrieve the list of buckets. If the request succeeds a ListBucketsResponse is returned. Use this response object to retrieve the list of buckets.

Imports

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.s3.S3Client;
import software.amazon.awssdk.services.s3.model.CreateBucketConfiguration;
import software.amazon.awssdk.services.s3.model.CreateBucketRequest;
import software.amazon.awssdk.services.s3.model.DeleteBucketRequest;
import software.amazon.awssdk.services.s3.model.ListBucketsRequest;
import software.amazon.awssdk.services.s3.model.ListBucketsResponse;
```

Code

```
ListBucketsRequest listBucketsRequest = ListBucketsRequest.builder().build();
ListBucketsResponse listBucketsResponse = s3.listBuckets(listBucketsRequest);
System.out.println(listBucketsResponse.buckets());
```

See the complete example.

Delete a Bucket

Before you can delete an Amazon S3 bucket, you must ensure that the bucket is empty or the service will return an error. If you have a versioned bucket, you must also delete any versioned objects that are in the bucket.

Topics

Delete Objects in a Bucket

Build a ListObjectsV2Request and use the S3Client`listObjects` method to retrieve the list of objects in the bucket. Then use the `deleteObject` method on each object to delete it.

Imports

```
import software.amazon.awssdk.services.s3.S3Client;
import software.amazon.awssdk.services.s3.model.CreateBucketConfiguration;
import software.amazon.awssdk.services.s3.model.CreateBucketRequest;
import software.amazon.awssdk.services.s3.model.DeleteBucketRequest;
import software.amazon.awssdk.services.s3.model.DeleteObjectRequest;
import software.amazon.awssdk.services.s3.model.ListObjectsV2Request;
```

```
import software.amazon.awssdk.services.s3.model.ListObjectsV2Response;
```

Code

```
ListObjectsV2Response listObjectsV2Response;
do {
    listObjectsV2Response = s3.listObjectsV2(listObjectsV2Request);
    for (S3Object s3Object : listObjectsV2Response.contents()) {

 s3.deleteObject(DeleteObjectRequest.builder().bucket(bucket2).key(s3Object.key()).build());
    }

    listObjectsV2Request = ListObjectsV2Request.builder().bucket(bucket2)

 .continuationToken(listObjectsV2Response.nextContinuationToken())
                                                .build();

} while (listObjectsV2Response.isTruncated());
```

See the complete example.

Delete an Empty Bucket

Build a DeleteBucketRequest with a bucket name and pass it to the S3Client`deleteBucket` method.

Imports

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.s3.S3Client;
import software.amazon.awssdk.services.s3.model.CreateBucketConfiguration;
import software.amazon.awssdk.services.s3.model.CreateBucketRequest;
import software.amazon.awssdk.services.s3.model.DeleteBucketRequest;
```

Code

```
DeleteBucketRequest deleteBucketRequest =
 DeleteBucketRequest.builder().bucket(bucket).build();
s3.deleteBucket(deleteBucketRequest);
```

See the complete example.

Performing Operations on an Amazon S3 Object

An Amazon S3 object represents a file or collection of data. Every object must be contained in a bucket (p. 25).

> **Note**
> These code snippets assume that you understand the material in Using the AWS SDK for Java 2.0 Developer Preview (p. 11), and have configured default AWS credentials using the information in Set Up AWS Credentials and Region for Development (p. 5).

Topics

Upload an Object

Build a PutObjectRequest and supply a bucket name and key name. Then use the S3Client`putObject` method with a `RequestBody` RequestBody that contains the object content and the `PutObjectRequest` object. *The bucket must exist, or the service will return an error.*

Imports

```
import software.amazon.awssdk.services.s3.model.PutObjectRequest;
import software.amazon.awssdk.services.s3.model.UploadPartRequest;
import software.amazon.awssdk.sync.RequestBody;
```

Code

```
// Put Object
s3.putObject(PutObjectRequest.builder().bucket(bucket).key(key)
                            .build(),
             RequestBody.of(getRandomByteBuffer(10_000)));
```

See the complete example.

Upload Objects in Multiple Parts

Use the S3Client`createMultipartUpload` method to get an upload ID. Then use the `uploadPart` method to upload each part. Finally, use the S3Client`completeMultipartUpload` method to tell Amazon S3 to merge all the uploaded parts and finish the upload operation.

Imports

```
import software.amazon.awssdk.services.s3.model.CompleteMultipartUploadRequest;
import software.amazon.awssdk.services.s3.model.CompletedMultipartUpload;
import software.amazon.awssdk.services.s3.model.CompletedPart;
import software.amazon.awssdk.services.s3.model.CreateBucketConfiguration;
import software.amazon.awssdk.services.s3.model.CreateBucketRequest;
import software.amazon.awssdk.services.s3.model.CreateMultipartUploadRequest;
import software.amazon.awssdk.services.s3.model.CreateMultipartUploadResponse;
import software.amazon.awssdk.services.s3.model.DeleteBucketRequest;
import software.amazon.awssdk.services.s3.model.DeleteObjectRequest;
import software.amazon.awssdk.services.s3.model.GetObjectRequest;
import software.amazon.awssdk.services.s3.model.PutObjectRequest;
import software.amazon.awssdk.services.s3.model.UploadPartRequest;
```

Code

```
// First create a multipart upload and get upload id
CreateMultipartUploadRequest createMultipartUploadRequest =
 CreateMultipartUploadRequest.builder()

 .bucket(bucketName).key(key)

 .build();
CreateMultipartUploadResponse response =
 s3.createMultipartUpload(createMultipartUploadRequest);
String uploadId = response.uploadId();
System.out.println(uploadId);

// Upload all the different parts of the object
UploadPartRequest uploadPartRequest1 =
 UploadPartRequest.builder().bucket(bucketName).key(key)
```

```
                                                        .uploadId(uploadId)
                                                        .partNumber(1).build();
String etag1 = s3.uploadPart(uploadPartRequest1, RequestBody.of(getRandomByteBuffer(5 *
 MB))).eTag();
CompletedPart part1 = CompletedPart.builder().partNumber(1).eTag(etag1).build();

UploadPartRequest uploadPartRequest2 =
 UploadPartRequest.builder().bucket(bucketName).key(key)
                                                        .uploadId(uploadId)
                                                        .partNumber(2).build();
String etag2 = s3.uploadPart(uploadPartRequest2, RequestBody.of(getRandomByteBuffer(3 *
 MB))).eTag();
CompletedPart part2 = CompletedPart.builder().partNumber(2).eTag(etag2).build();


// Finally call completeMultipartUpload operation to tell S3 to merge all uploaded
// parts and finish the multipart operation.
CompletedMultipartUpload completedMultipartUpload =
 CompletedMultipartUpload.builder().parts(part1, part2).build();
CompleteMultipartUploadRequest completeMultipartUploadRequest =

 CompleteMultipartUploadRequest.builder().bucket(bucketName).key(key).uploadId(uploadId)
                                      .multipartUpload(completedMultipartUpload).build();
s3.completeMultipartUpload(completeMultipartUploadRequest);
```

See the complete example.

Download an Object

Build a GetObjectRequest and supply a bucket name and key name. Use the S3Client`getObject` method, passing it the `GetObjectRequest` object and a `StreamingResponseHandler` object. The `StreamingResponseHandler` creates a response handler that writes the response content to the specified file or stream.

The following example specifies a file name to write the object content to.

Imports

```
import java.io.IOException;
import java.nio.ByteBuffer;
import java.nio.file.Paths;
import java.util.Random;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.s3.S3Client;
import software.amazon.awssdk.services.s3.model.CompleteMultipartUploadRequest;
```

Code

```
// Get Object
s3.getObject(GetObjectRequest.builder().bucket(bucket).key(key).build(),
             StreamingResponseHandler.toFile(Paths.get("myfile.out")));
```

See the complete example.

Delete an Object

Build a DeleteObjectRequest and supply a bucket name and key name. Use the S3Client`deleteObject` method, and pass it the name of a bucket and object to delete. *The specified bucket and object key must exist, or the service will return an error.*

Imports

```
import software.amazon.awssdk.services.s3.model.DeleteObjectRequest;
import software.amazon.awssdk.services.s3.model.GetObjectRequest;
```

Code

```
// Delete Object
DeleteObjectRequest deleteObjectRequest =
 DeleteObjectRequest.builder().bucket(bucket).key(key).build();
s3.deleteObject(deleteObjectRequest);
```

See the complete example.

Amazon SQS Examples

This section provides examples of programming Amazon SQS using the AWS SDK for Java 2.0.

> **Note**
> The examples include only the code needed to demonstrate each technique. The complete example code is available on GitHub. From there, you can download a single source file or clone the repository locally to get all the examples to build and run.

Topics

Working with Amazon SQS Message Queues

A *message queue* is the logical container used for sending messages reliably in Amazon SQS. There are two types of queues: *standard* and *first-in, first-out* (FIFO). To learn more about queues and the differences between these types, see the Amazon SQS Developer Guide.

This topic describes how to create, list, delete, and get the URL of an Amazon SQS queue by using the AWS SDK for Java.

Create a Queue

Use the SQSClient`createQueue` method, and provide a CreateQueueRequest object that describes the queue parameters.

Imports

```
import software.amazon.awssdk.services.sqs.SQSClient;
import software.amazon.awssdk.services.sqs.model.CreateQueueRequest;
```

Code

```
System.out.println("\nCreate Queue");
CreateQueueRequest createQueueRequest =
 CreateQueueRequest.builder().queueName(queueName).build();
sqsClient.createQueue(createQueueRequest);
```

See the complete sample.

List Queues

To list the Amazon SQS queues for your account, call the SQSClient`listQueues` method with a ListQueuesRequest object.

Using the `listQueues` overload without any parameters returns *all queues*, up to 1,000 queues. You can supply a queue name prefix to the `ListQueuesRequest` object to limit the results to queues that match that prefix.

Imports

```
import software.amazon.awssdk.services.sqs.model.ListQueuesRequest;
import software.amazon.awssdk.services.sqs.model.ListQueuesResponse;
```

Code

```
System.out.println("\nList Queues");
String prefix = "que";
ListQueuesRequest listQueuesRequest =
 ListQueuesRequest.builder().queueNamePrefix(prefix).build();
ListQueuesResponse listQueuesResponse = sqsClient.listQueues(listQueuesRequest);
for (String url : listQueuesResponse.queueUrls()) {
 System.out.println(url);
}
```

See the complete sample.

Get the URL for a Queue

Call the SQSClient`getQueueUrl` method. with a GetQueueUrlRequest object.

Imports

```
import software.amazon.awssdk.services.sqs.model.GetQueueUrlRequest;
import software.amazon.awssdk.services.sqs.model.GetQueueUrlResponse;
```

Code

```
System.out.println("\nGet queue url");
GetQueueUrlResponse getQueueUrlResponse =
        sqsClient.getQueueUrl(GetQueueUrlRequest.builder().queueName(queueName).build());
String queueUrl = getQueueUrlResponse.queueUrl();
System.out.println(queueUrl);
```

See the complete sample.

Delete a Queue

Provide the queue's URL (p. 31) to the DeleteMessageRequest object. Then call the SQSClient`deleteQueue` method.

Imports

```
import software.amazon.awssdk.services.sqs.model.DeleteMessageRequest;
```

```
import software.amazon.awssdk.services.sqs.model.DeleteQueueRequest;
```

Code

```
System.out.println("\nDelete Queue");
DeleteQueueRequest deleteQueueRequest =
 DeleteQueueRequest.builder().queueUrl(queueUrl).build();
sqsClient.deleteQueue(deleteQueueRequest);
```

See the complete sample.

More Info

- How Amazon SQS Queues Work in the *Amazon SQS Developer Guide*
- CreateQueue in the *Amazon SQS API Reference*
- GetQueueUrl in the *Amazon SQS API Reference*
- ListQueues in the *Amazon SQS API Reference*
- DeleteQueues in the *Amazon SQS API Reference*

Sending, Receiving, and Deleting Amazon SQS Messages

A message is a piece of data that can be sent and recieved by distributed components. Messages are always delivered using an SQS Queue (p. 30).

Send a Message

Add a single message to an Amazon SQS queue by calling the SQSClient client `sendMessage` method. Provide a SendMessageRequest object that contains the queue's URL (p. 31), message body, and optional delay value (in seconds).

Imports

```
import software.amazon.awssdk.services.sqs.SQSClient;
import software.amazon.awssdk.services.sqs.model.SendMessageRequest;
```

Code

```
System.out.println("\nSend message");
sqsClient.sendMessage(SendMessageRequest.builder()
        .queueUrl(queueUrl)
        .messageBody("Hello world!")
        .delaySeconds(10)
        .build());
```

Send Multiple Messages in a Request

Send more than one message in a single request by using the SQSClient`sendMessageBatch` method. This method takes a SendMessageBatchRequest that contains the queue URL and a list of messages to send. (Each message is a SendMessageBatchRequestEntry.) You can also delay sending a specific message by setting a delay value on the message.

Imports

```
import software.amazon.awssdk.services.sqs.SQSClient;
import software.amazon.awssdk.services.sqs.model.SendMessageBatchRequest;
import software.amazon.awssdk.services.sqs.model.SendMessageBatchRequestEntry;
```

Code

```
System.out.println("\nSend multiple messages");
SendMessageBatchRequest sendMessageBatchRequest = SendMessageBatchRequest.builder()
        .queueUrl(queueUrl)
        .entries(SendMessageBatchRequestEntry.builder().id("id1").messageBody("Hello from
 msg 1").build(),
                SendMessageBatchRequestEntry.builder().id("id2").messageBody("msg
 2").delaySeconds(10).build())
        .build();
sqsClient.sendMessageBatch(sendMessageBatchRequest);
```

See the complete sample.

Retrieve Messages

Retrieve any messages that are currently in the queue by calling the SQSClient`receiveMessage` method. This method takes a ReceiveMessageRequest that contains the queue URL. You can also specify the maximum number of messages to return. Messages are returned as a list of Message objects.

Imports

```
import software.amazon.awssdk.services.sqs.SQSClient;
import software.amazon.awssdk.services.sqs.model.ReceiveMessageRequest;
```

Code

```
System.out.println("\nReceive messages");
ReceiveMessageRequest receiveMessageRequest = ReceiveMessageRequest.builder()
        .queueUrl(queueUrl)
        .maxNumberOfMessages(5)
        .build();
List<Message> messages= sqsClient.receiveMessage(receiveMessageRequest).messages();
```

Delete a Message After Receipt

After receiving a message and processing its contents, delete the message from the queue by sending the message's receipt handle and queue URL to the SQSClient`deleteMessage` method.

Imports

```
import software.amazon.awssdk.services.sqs.SQSClient;
import software.amazon.awssdk.services.sqs.model.DeleteQueueRequest;
```

Code

```
System.out.println("\nDelete Messages");
for (Message message : messages) {
    DeleteMessageRequest deleteMessageRequest = DeleteMessageRequest.builder()
            .queueUrl(queueUrl)
            .receiptHandle(message.receiptHandle())
            .build();
```

```
    sqsClient.deleteMessage(deleteMessageRequest);
}
```

See the complete sample.

More Info

- How Amazon SQS Queues Work in the *Amazon SQS Developer Guide*
- SendMessage in the *Amazon SQS API Reference*
- SendMessageBatch in the *Amazon SQS API Reference*
- ReceiveMessage in the *Amazon SQS API Reference*
- DeleteMessage in the *Amazon SQS API Reference*

Document History

This topic describes important changes to the *AWS SDK for Java Developer Guide* over the course of its history.

This documentation was built on: Sep 19, 2017

Jun 28, 2017

New SDK version, 2.0 released.