
AWS SDK for Java

Developer Guide

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

AWS SDK for Java 2.x	1
Get started with the AWS SDK for Java 2.x	1
Developing AWS applications for Android	1
Maintenance and support for SDK major versions	1
Additional resources	1
Features not yet in the AWS SDK for Java 2.x	2
Contributing to the SDK	2
Quick start	3
Step 1: Set up for this tutorial	3
Create an AWS account	3
Create an IAM user	3
Install Java and Apache Maven	4
Configure credentials	4
Step 2: Create the project	5
Step 3: Write the code	6
Step 4: Build and run the application	8
Success!	8
Cleanup	8
Next steps	9
Setting up	10
Overview	10
Create an AWS account	10
Create an IAM user and programmatic access key	10
Set default credentials and Region	11
Setting the default credentials	11
Setting the default AWS Region	12
Install Java and a build tool	13
Next steps	13
Setting up an Apache Maven project	15
Prerequisites	15
Create a Maven project	15
Configure the Java compiler for Maven	15
Declare the SDK as a dependency	16
Set dependencies for SDK modules	17
Build your project	18
Setting up a Gradle project	18
Using the SDK	20
Creating service clients	20
Obtaining a client builder	20
Using the default client	21
Client lifecycle	21
Supplying and retrieving AWS credentials	21
Use the default credential provider chain	21
Use a specific credential provider or provider chain	24
Supply credentials explicitly	24
Configuring IAM roles for Amazon EC2	24
AWS region selection	26
Choosing a region	26
Choosing a specific endpoint	27
Automatically determine the AWS region from the environment	27
Checking for service availability in an AWS region	28
Asynchronous programming	28
Non-streaming operations	28
Streaming operations	30

Advanced operations	33
Enabling SDK metrics	34
Prerequisites	15
How to enable metrics collection	35
What information is collected?	35
Service client metrics	36
Retrieving paginated results	38
Synchronous pagination	38
Asynchronous pagination	40
Using waiters	44
Prerequisites	15
Using waiters	42
Configuring waiters	45
Code examples	46
Optimizing cold start performance for AWS Lambda	46
Example client configuration	46
HTTP configuration	47
Setting maximum connections	47
Timeouts and error handling	47
Local address	48
Configuring the AWS CRT-based HTTP client	48
Configuring the Netty-based HTTP client	51
HTTP/2 programming	52
Exception handling	52
Why unchecked exceptions?	52
SdkServiceException (and subclasses)	52
SdkClientException	53
Logging AWS SDK for Java calls	53
Add the Log4J JAR	53
Log4j configuration file	53
Setting the classpath	54
Service-specific errors and warnings	54
Request/response summary logging	54
Verbose wire logging	55
Setting the JVM TTL for DNS name lookups	56
How to set the JVM TTL	56
Working with Amazon Web Services	57
Amazon S3	58
Bucket operations	58
Object operations	62
Amazon S3 presigned examples	68
Amazon DynamoDB	71
Work with tables in DynamoDB	71
Work with items in DynamoDB	77
Map items in DynamoDB tables	82
Amazon EC2	89
Manage Amazon EC2 instances	89
Use elastic IP addresses in Amazon EC2	94
Use regions and availability zones	96
Work with Amazon EC2 key pairs	98
Work with security groups in Amazon EC2	100
Amazon IAM	103
Manage IAM access keys	103
Managing IAM Users	107
Use IAM account aliases	110
Work with IAM policies	112
Work with IAM server certificates	117

Amazon Athena	120
Amazon CloudWatch	120
Get metrics from CloudWatch	120
Publish custom metric data	122
Work with CloudWatch alarms	123
Use alarm actions in CloudWatch	126
Send events to CloudWatch	127
Amazon CloudTrail	130
Amazon Cognito	130
Create a user pool	130
List users from a user pool	131
Create an identity pool	131
Add an app client	132
Add a third-party identity provider	133
Get credentials for an ID	134
Amazon Comprehend	134
Amazon EventBridge	135
Amazon Firehose	135
Amazon Forecast	135
Amazon S3 Glacier	135
AWS Glue	135
Amazon Kinesis	136
Subscribing to Amazon Kinesis Data Streams	136
Amazon KMS	141
AWS Lambda	142
Invoke a Lambda function	142
List Lambda functions	143
Delete a Lambda function	143
Amazon MediaConvert	144
Amazon MediaStore	144
Amazon MigrationHub	144
Amazon Personalize	144
Amazon Pinpoint	145
Create a project	145
Create a dynamic segment	145
Import a static segment	147
List segments for your project	148
Create a campaign	149
Send a message	150
Amazon Polly	152
Amazon RDS	152
Amazon Redshift	152
Amazon Rekognition	152
Amazon SageMaker	153
Amazon Secrets Manager	153
Amazon Simple Email Service	153
Amazon Simple Notification Service	153
Create a topic	153
List your SNS topics	154
Subscribe an endpoint to a topic	155
Publish a message to a topic	155
Unsubscribe an endpoint from a topic	156
Delete a topic	157
Amazon SQS	157
Queue Operations	158
Message Operations	160
AWS Systems Manager	162

Amazon Simple Workflow Service	162
Amazon Textract	162
Amazon Transcribe	162
Working with Amazon Transcribe	163
Amazon Translate	166
Amazon WorkDocs	166
Security	167
Data protection	167
Enforcing TLS 1.2	168
TLS support in Java	168
How to check the TLS version	168
How to set the TLS version	168
Identity and access management	169
Compliance validation	169
Resilience	170
Infrastructure security	170
Document history	171

Developer guide - AWS SDK for Java 2.x

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.

The AWS SDK for Java 2.x 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](#). For guidance on migrating your application from 1.11.x to 2.x, see the [migration guide](#).

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](#).

Get started with the AWS SDK for Java 2.x

If you're ready to get hands-on with the SDK, follow the [Quick Start \(p. 3\)](#) tutorial.

To set up your development environment, see [Setting up \(p. 10\)](#).

For information on making requests to Amazon S3, DynamoDB, Amazon EC2 and other Amazon Web Services, see [Code examples for the AWS SDK for Java 2.x \(p. 57\)](#).

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.

Maintenance and support for SDK major versions

For information about maintenance and support for SDK major versions and their underlying dependencies, see the following in the [AWS SDKs and Tools Shared Configuration and Credentials Reference Guide](#)

- [AWS SDKs and Tools Maintenance Policy](#)
- [AWS SDKs and Tools Version Support Matrix](#)

Additional resources

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

- [AWS SDK for Java 2.x Reference](#)
- [Java developer blog](#)

- [Java developer forums](#)
- GitHub:
 - [Documentation source](#)
 - [SDK source](#)
- The [AWS Code Sample Catalog](#)
- [@awsforjava](#) (Twitter)

Features not yet in the AWS SDK for Java 2.x

See the following Github issues for details about additional features not yet in 2.x. Comments and feedback are also welcome.

- High-level libraries
 - [S3 Transfer manager](#)
 - [S3 Encryption Client](#)
 - [DynamoDB document APIs](#)
 - [DynamoDB Encryption Client](#)
 - [SQS Client-side Buffering](#)
- [Progress Listeners](#)

Contributing to the SDK

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.x [gitter 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.

Get started with the AWS SDK for Java 2.x

The AWS SDK for Java 2.x provides Java APIs for Amazon Web Services (AWS). Using the SDK, you can build Java applications that work with Amazon S3, Amazon EC2, DynamoDB, and more.

This tutorial shows you how you can use [Apache Maven](#) to define dependencies for the AWS SDK for Java and then write code that connects to Amazon S3 to upload a file.

Follow these steps to complete this tutorial:

- [Step 1: Set up for this tutorial \(p. 3\)](#)
- [Step 2: Create the project \(p. 5\)](#)
- [Step 3: Write the code \(p. 6\)](#)
- [Step 4: Build and run the application \(p. 8\)](#)

Step 1: Set up for this tutorial

Before you begin this tutorial, you need an active AWS account, an AWS Identity and Access Management (IAM) user with a programmatic access key and permissions to Amazon S3, and a Java development environment configured to use that access key as credentials for AWS.

Follow these steps to set up for this tutorial:

- [Create an AWS account \(p. 3\)](#)
- [Create an IAM user \(p. 3\)](#)
- [Install Java and Apache Maven \(p. 4\)](#)
- [Configure credentials \(p. 4\)](#)

Create an AWS account

If you do not have an AWS account, visit [the Amazon Web Services signup page](#) and follow the on-screen prompts to create and activate a new account. For detailed instructions, see [How do I create and activate a new AWS account?](#).

After you activate your new AWS account, follow the instructions in [Creating your first IAM admin user and group](#) in the IAM User Guide. Use this account instead of the root account when accessing the AWS Console. For more information, see [Security best practices in IAM](#) in the IAM User Guide.

Create an IAM user

To complete this tutorial, you need to use credentials for an IAM user that has read and write access to Amazon S3. To make requests to Amazon Web Services using the AWS SDK for Java, create an access key to use as credentials.

To create an IAM user with a programmatic access key and the required permissions for this tutorial

1. Sign in to [the IAM console](#)
2. In the navigation pane on the left, choose **Users**. Then choose **Add user**.
3. Enter *TestSDK* as the **User name** and select the **Programmatic access** checkbox. Choose **Next: Permissions**.
4. Under **Set permissions**, select **Attach existing policies directly**.
5. In the list of policies, select the checkbox for the **AmazonS3FullAccess** policy. Choose **Next: Tags**.
6. Choose **Next: Review**. Then choose **Create user**.
7. On the *Success* screen, choose **Download .csv**.

The downloaded file contains the Access Key ID and the Secret Access Key for this tutorial. Treat your Secret Access Key as a password; save in a trusted location and do not share it.

Note

You will **not** have another opportunity to download or copy the Secret Access Key.

Install Java and Apache Maven

Your development environment needs to have Java 8 or later and Apache Maven installed.

- For Java, use [Oracle Java SE Development Kit](#), [Amazon Corretto](#), [Red Hat OpenJDK](#), or [AdoptOpenJDK](#).
- For Maven, go to <https://maven.apache.org/>.

Configure credentials

Configure your development environment with your Access Key ID and the Secret Access Key. The AWS SDK for Java uses this access key as credentials when your application makes requests to Amazon Web Services.

To configure credentials

1. In a text editor, create a new file with the following code:

```
[default]
aws_access_key_id = YOUR_AWS_ACCESS_KEY_ID
aws_secret_access_key = YOUR_AWS_SECRET_ACCESS_KEY
```

2. In the text file you just created, replace *YOUR_AWS_ACCESS_KEY* with your unique AWS access key ID, and replace *YOUR_AWS_SECRET_ACCESS_KEY* with your unique AWS secret access key.
3. Save the file without a file extension. Refer to the following table for the correct location and file name based on your operating system.

Operating system	File name
Windows	C:\Users\<yourUserName>\.aws\credentials
Linux, macOS, Unix	~/.aws/credentials

Step 2: Create the project

To create the project for this tutorial, you first create a Maven project. Next, you configure your project with a dependency on AWS SDK for Java and for any AWS service you use, for example Amazon S3. Then you configure the Maven compiler to use Java 1.8.

To create the Maven project

1. Open a terminal or command prompt window and navigate to a directory of your choice, for example, your Desktop or Home folder.
2. Use the following command to create a new directory called `myapp` with a project configuration file (`pom.xml`) and a basic Java class.

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

To configure your project with dependencies for the AWS SDK for Java and Amazon S3, and to use Java 1.8

- In the folder `myapp` that you created in the previous procedure, open the `pom.xml` file. Replace its contents with the following code, and then save your changes.

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/
maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>
  <groupId>com.example.myapp</groupId>
  <artifactId>myapp</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>myapp</name>
  <dependencyManagement>
    <dependencies>
      <dependency>
        <groupId>software.amazon.awssdk</groupId>
        <artifactId>bom</artifactId>
        <version>2.15.15</version>
        <type>pom</type>
        <scope>import</scope>
      </dependency>
    </dependencies>
  </dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
    <dependency>
      <groupId>software.amazon.awssdk</groupId>
      <artifactId>s3</artifactId>
    </dependency>
  </dependencies>
```

```
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>3.8.1</version>
      <configuration>
        <source>8</source>
        <target>8</target>
      </configuration>
    </plugin>
  </plugins>
</build>
</project>
```

The `dependencyManagement` section contains a dependency to the AWS SDK for Java and the `dependencies` section has a dependency for Amazon S3. The Apache Maven Compiler Plugin is configured in the `build` section to use Java 1.8.

Step 3: Write the code

After the project has been created and configured, edit the project's default class `App` to use the example code below.

The example class below creates a service client for Amazon S3 and then uses it to upload a text file. To create a service client for Amazon S3, instantiate an [S3Client](#) object using the static factory method `builder`. To upload a file to Amazon S3, first build a [PutObjectRequest](#) object, supplying a bucket name and a key name. Then, call the `S3Client`'s `putObject` method, with a [RequestBody](#) that contains the object content and the `PutObjectRequest` object.

To create the Java class for this tutorial

1. In your project folder `myapp`, navigate to the directory `src/main/java/com/example/myapp`. Open the `App.java` file.
2. Replace its contents with the following code and save the file.

```
package com.example.myapp;

import java.io.IOException;
import software.amazon.awssdk.core.sync.RequestBody;
import software.amazon.awssdk.regions.Region;
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.HeadBucketRequest;
import software.amazon.awssdk.services.s3.model.PutObjectRequest;
import software.amazon.awssdk.services.s3.model.S3Exception;
import software.amazon.awssdk.services.s3.S3Client;

public class App {

    public static void main(String[] args) throws IOException {

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

```
String bucket = "bucket" + System.currentTimeMillis();
String key = "key";

tutorialSetup(s3, bucket, region);

System.out.println("Uploading object...");

s3.putObject(PutObjectRequest.builder().bucket(bucket).key(key)
    .build(),
    RequestBody.fromString("Testing with the AWS SDK for Java"));

System.out.println("Upload complete");
System.out.printf("%n");

cleanUp(s3, bucket, key);

System.out.println("Closing the connection to Amazon S3");
s3.close();
System.out.println("Connection closed");
System.out.println("Exiting...");
}

public static void tutorialSetup(S3Client s3Client, String bucketName, Region
region) {
    try {
        s3Client.createBucket(CreateBucketRequest
            .builder()
            .bucket(bucketName)
            .createBucketConfiguration(
                CreateBucketConfiguration.builder()
                    .locationConstraint(region.id())
                    .build())
            .build());
        System.out.println("Creating bucket: " + bucketName);
        s3Client.waitFor().waitUntilBucketExists(HeadBucketRequest.builder()
            .bucket(bucketName)
            .build());
        System.out.println(bucketName + " is ready.");
        System.out.printf("%n");
    } catch (S3Exception e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}

public static void cleanUp(S3Client s3Client, String bucketName, String keyName) {
    System.out.println("Cleaning up...");
    try {
        System.out.println("Deleting object: " + keyName);
        DeleteObjectRequest deleteObjectRequest =
DeleteObjectRequest.builder().bucket(bucketName).key(keyName).build();
        s3Client.deleteObject(deleteObjectRequest);
        System.out.println(keyName + " has been deleted.");
        System.out.println("Deleting bucket: " + bucketName);
        DeleteBucketRequest deleteBucketRequest =
DeleteBucketRequest.builder().bucket(bucketName).build();
        s3Client.deleteBucket(deleteBucketRequest);
        System.out.println(bucketName + " has been deleted.");
        System.out.printf("%n");
    } catch (S3Exception e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
    System.out.println("Cleanup complete");
    System.out.printf("%n");
}
```

```
}
```

Step 4: Build and run the application

After the project is created and contains the example class, build and run the application. To view the uploaded file in the Amazon S3 console, edit the code to remove the cleanup steps and then rebuild the project.

To build the project using Maven

1. Open a terminal or command prompt window and navigate to your project directory `myapp`.
2. Use the following command to build your project:

```
mvn package
```

To run the application

1. Open a terminal or command prompt window and navigate to your project directory `myapp`.
2. Use the following command to run the application.

```
mvn exec:java -Dexec.mainClass="com.example.myapp.App"
```

When you run the application, it uploads a new a text file to a new bucket in Amazon S3. Afterward, it will also delete the file and bucket.

To see the file in the Amazon S3 console after it uploads

1. In `App.java`, comment out the line `cleanup(s3, bucket, key);` and save the file.
2. Rebuild the project by running `mvn package`.
3. Upload the file by running `mvn exec:java -Dexec.mainClass="com.example.myapp.App"` again.
4. Sign in to [the S3 console](#) to view the new file in the newly-created bucket.

After you view the file, clean up test resources by deleting the object and then deleting the bucket.

Success!

If your Maven project built and ran without error, then congratulations! You have successfully built your first Java application using the AWS SDK for Java.

Cleanup

To clean up the resources you created during this tutorial:

- In [the S3 console](#), delete any objects and any buckets created when you ran the application.
- In [the IAM console](#), delete the `TestSDK` user.

If you delete this user, also remove the contents of the `credentials` file you created during setup.

- Delete the project folder (`myapp`).

Next steps

Now that you have the basics down, you can learn about:

- [Working with Amazon S3 \(p. 58\)](#)
- [Working with other Amazon Web Services \(p. 57\)](#), such as [DynamoDB \(p. 71\)](#), [Amazon EC2 \(p. 89\)](#), and [IAM \(p. 103\)](#)
- [Using the SDK \(p. 20\)](#)
- [Security for the AWS SDK for Java \(p. 167\)](#)

Setting up the AWS SDK for Java 2.x

The AWS SDK for Java 2.x provides Java APIs for Amazon Web Services (AWS). Using the SDK, you can build Java applications that work with Amazon S3, Amazon EC2, DynamoDB, and more.

This section provides information about how to set up your development environment and projects to use the latest version (2.x) of the AWS SDK for Java.

Topics

- [Overview \(p. 10\)](#)
- [Create an AWS account \(p. 10\)](#)
- [Create an IAM user and programmatic access key \(p. 10\)](#)
- [Set default credentials and Region \(p. 11\)](#)
- [Install Java and a build tool \(p. 13\)](#)
- [Setting up an Apache Maven project \(p. 15\)](#)
- [Setting up a Gradle project \(p. 18\)](#)

Overview

To make requests to AWS using the AWS SDK for Java, you need the following:

- An active AWS account
- An AWS Identity and Access Management (IAM) user with:
 - A programmatic access key
 - Permissions to the AWS resources you'll access using your application
- A development environment with:
 - Your access key configured as credentials for AWS
 - Java 8 or later
 - A build automation tool

Create an AWS account

If you do not have an AWS account, visit [the Amazon Web Services signup page](#) and follow the on-screen prompts to create and activate a new account.

For more detailed instructions, see [How do I create and activate a new AWS account?](#).

After you activate your new AWS account, follow the instructions in [Creating your first IAM admin user and group](#) in the IAM User Guide. Use this account instead of the root account when accessing the AWS Console. For more information, see [Security best practices in IAM](#) in the IAM User Guide.

Create an IAM user and programmatic access key

To use the AWS SDK for Java to access AWS, you need an AWS account and AWS credentials. To increase the security of your AWS account, for access credentials, we recommend that you use an IAM user instead of your AWS account credentials.

Note

For an overview of IAM users and why they are important for the security of your account, see [AWS security credentials](#) in the Amazon Web Services General Reference.

For instructions on creating an access key for an existing IAM user, see [Programmatic access](#) in the [IAM User Guide](#).

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.

Set default credentials and Region

To make requests to AWS using the AWS SDK for Java, you must use cryptographically-signed credentials issued by AWS. With AWS SDKs and Tools like the AWS SDK for Java, you use a programmatic access key, consisting of an Access Key ID and a Secret Access Key, as credentials. You should set your credentials as the default credentials for accessing AWS with your application.

If you already have an IAM account created, see [Create an IAM user and programmatic access key \(p. 10\)](#) for instructions on creating a programmatic access key.

You should also set a default AWS Region for accessing AWS with your application. Some operations require a Region to be set. For the best network performance, you can select a Region that is geographically near to you or your customers.

The most common way to set the default credentials and AWS Region is to use the shared `config` and `credentials` files. You can also set the default credentials and Region using environment variables, using Java system properties or, for your applications running on Amazon EC2, using [ContainerCredentialsProvider](#) or [InstanceProfileCredentialsProvider](#).

Setting the default credentials

Select one of these options to set the default credentials:

- 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.
- Set the `aws.accessKeyId` and `aws.secretAccessKey` Java system properties.

```
java app.jar -Daws.accessKeyId=\
"your_access_key_id" \
-Daws.secretAccessKey=\
"your_secret_access_key"
```

Setting the default AWS Region

Select one of these options to set the default 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 additional information about setting credentials and Region, see [The .aws/credentials and .aws/config files](#), [AWS Region](#), and [Using environment variables](#) in the [AWS SDKs and Tools Reference Guide](#).

Install Java and a build tool

Your development environment needs the following:

- Java 8 or later. The AWS SDK for Java works with the [Oracle Java SE Development Kit](#) and with distributions of Open Java Development Kit (OpenJDK) such as [Amazon Corretto](#), [Red Hat OpenJDK](#), and [AdoptOpenJDK](#).
- A build tool or IDE that supports Maven Central such as Apache Maven, Gradle, or IntelliJ.
 - For information about how to install and use Maven, see <http://maven.apache.org/>.
 - For information about how to install and use Gradle, see <https://gradle.org/>.
 - For information about how to install and use IntelliJ IDEA, see <https://www.jetbrains.com/idea/>.

Next steps

Once you have your AWS account and development environment set up, create a Java project using your preferred build tool. Import [the Maven bill of materials \(BOM\) for the AWS SDK for Java 2.x from Maven Central](#), `software.amazon.awssdk`. Then add dependencies for the services you'll use in your application.

Example Maven `pom.xml` file:

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/
maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>
  <groupId>com.example.myapp</groupId>
  <artifactId>myapp</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>myapp</name>
  <dependencyManagement>
    <dependencies>
      <dependency>
        <groupId>software.amazon.awssdk</groupId>
        <artifactId>bom</artifactId>
        <version>2.15.0</version>
        <type>pom</type>
        <scope>import</scope>
      </dependency>
    </dependencies>
  </dependencyManagement>
  <dependencies>
```

```
<dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
  <version>3.8.1</version>
  <scope>test</scope>
</dependency>
<dependency>
  <groupId>software.amazon.awssdk</groupId>
  <artifactId>dynamodb</artifactId>
</dependency>
<dependency>
  <groupId>software.amazon.awssdk</groupId>
  <artifactId>iam</artifactId>
</dependency>
<dependency>
  <groupId>software.amazon.awssdk</groupId>
  <artifactId>kinesis</artifactId>
</dependency>
<dependency>
  <groupId>software.amazon.awssdk</groupId>
  <artifactId>s3</artifactId>
</dependency>
</dependencies>
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>3.8.1</version>
      <configuration>
        <source>8</source>
        <target>8</target>
      </configuration>
    </plugin>
  </plugins>
</build>
</project>
```

Example build.gradle file:

```
group 'com.example.myapp'
version '1.0'

apply plugin: 'java'

sourceCompatibility = 1.8

repositories {
  mavenCentral()
}

dependencies {
  implementation platform('software.amazon.awssdk:bom:2.15.0')
  implementation 'software.amazon.awssdk:dynamodb'
  implementation 'software.amazon.awssdk:iam'
  implementation 'software.amazon.awssdk:kinesis'
  implementation 'software.amazon.awssdk:s3'
  testImplementation group: 'junit', name: 'junit', version: '4.11'
}
```

For more information, see [Setting up an Apache Maven project \(p. 15\)](#) or [Setting up a Gradle project \(p. 18\)](#).

Setting up an Apache Maven project

You can use [Apache Maven](#) to set up and build AWS SDK for Java projects, or to build the SDK itself.

Prerequisites

To use the AWS SDK for Java with Maven, you need the following:

- **Java 8.0 or later.** You can download the latest Java SE Development Kit software from <http://www.oracle.com/technetwork/java/javase/downloads/>. The AWS SDK for Java also works with [OpenJDK](#) and Amazon Corretto, a distribution of the Open Java Development Kit (OpenJDK). Download the latest OpenJDK version from <https://openjdk.java.net/install/index.html>. Download the latest Amazon Corretto 8 or Amazon Corretto 11 version from <https://aws.amazon.com/corretto/>.
- **Apache Maven.** If you need to install Maven, go to <http://maven.apache.org/> to download and install it.

Create a Maven project

To create a Maven project from the command line, open a terminal or command prompt window, enter or paste the following command, and then press Enter or Return.

```
mvn -B archetype:generate \
  -DarchetypeGroupId=software.amazon.awssdk \
  -DarchetypeArtifactId=archetype-lambda -Dservice=s3 -Dregion=US_WEST_2 \
  -DgroupId=com.example.myapp \
  -DartifactId=myapp
```

Note

Replace *com.example.myapp* with the full package namespace of your application. Also replace *myapp* with your project name. This becomes the name of the directory for your project.

This command creates a Maven project using the AWS Lambda project archetype. This project archetype is preconfigured to compile with Java SE 8 and includes a dependency to the AWS SDK for Java.

For more information about creating and configuring Maven projects, see the [Maven Getting Started Guide](#).

Configure the Java compiler for Maven

If you created your project using the AWS Lambda project archetype as described earlier, this is already done for you.

To verify that this configuration is present, start by opening the `pom.xml` file from the project folder you created (for example, `myapp`) when you executed the previous command. Look on lines 11 and 12 to see the Java compiler version setting for this Maven project, and the required inclusion of the Maven compiler plugin on lines 71-75.

```
<project>
  <properties>
    <maven.compiler.source>1.8</maven.compiler.source>
    <maven.compiler.target>1.8</maven.compiler.target>
  </properties>
  <build>
    <plugins>
      <plugin>
```

```
<groupId>org.apache.maven.plugins</groupId>
<artifactId>maven-compiler-plugin</artifactId>
<version>${maven.compiler.plugin.version}</version>
</plugin>
</plugins>
</build>
</project>
```

If you create your project with a different archetype or by using another method, you must ensure that the Maven compiler plugin is part of the build and that its source and target properties are both set to **1.8** in the `pom.xml` file.

See the previous snippet for one way to configure these required settings.

Alternatively, you can configure the compiler configuration inline with the plugin declaration, as follows.

```
<project>
  <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>
</project>
```

Declare the SDK as a 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.

If you created your project using the project archetype as described earlier, the SDK is already configured as a dependency in your project. We recommend that you update this configuration to reference the latest version of the AWS SDK for Java. To do so, open the `pom.xml` file and change the `aws.java.sdk.version` property (on line 16) to the latest version. The following is an example.

```
<project>
  <properties>
    <aws.java.sdk.version>2.13.7</aws.java.sdk.version>
  </properties>
</project>
```

Find the latest version of the AWS SDK for Java in the [AWS SDK for Java API Reference version 2.x](#).

If you created your Maven project in a different way, configure the latest version of the SDK for your project by ensuring that the `pom.xml` file contains the following.

```
<project>
  <dependencyManagement>
    <dependencies>
      <dependency>
        <groupId>software.amazon.awssdk</groupId>
        <artifactId>bom</artifactId>
        <version>2.X.X</version>
```

```
        <type>pom</type>
        <scope>import</scope>
    </dependency>
</dependencies>
</dependencyManagement>
</project>
```

Note

Replace 2.X.X in the `pom.xml` file with a valid version of the AWS SDK for Java version 2.

Set dependencies for SDK modules

Now that you have configured the SDK, you can add dependencies for one or more of the AWS SDK for Java modules to use in your project.

Although you can specify the version number for each component, you don't need to because you already declared the SDK version in the `dependencyManagement` section. To load a custom version of a given module, specify a version number for its dependency.

If you created your project using the project archetype as described earlier, your project is already configured with multiple dependencies. These include dependences for Lambda and Amazon DynamoDB, as follows.

```
<project>
  <dependencies>
    <dependency>
      <groupId>software.amazon.awssdk</groupId>
      <artifactId>dynamodb</artifactId>
    </dependency>
    <dependency>
      <groupId>com.amazonaws</groupId>
      <artifactId>aws-lambda-java-core</artifactId>
      <version>1.2.0</version>
    </dependency>
  </dependencies>
</project>
```

Add the modules to your project for the AWS service and features you need for your project. The modules (dependencies) that are managed by the AWS SDK for Java BOM are listed on the Maven central repository (<https://mvnrepository.com/artifact/software.amazon.awssdk/bom/latest>).

Note

You can look at the `pom.xml` file from a code example to determine which dependencies you need for your project. For example, if you're interested in the dependencies for the Amazon S3 service, see [this example](#) from the [AWS Code Examples Repository](#) on GitHub. (Look for the `pom.xml` file file under `/java2/example_code/s3`.)

Build the entire SDK into your project

To optimize your application, we strongly recommend that you pull in only the components you need instead of the entire SDK. However, to build the entire AWS SDK for Java into your project, declare it in your `pom.xml` file, as follows.

```
<project>
  <dependencies>
    <dependency>
      <groupId>software.amazon.awssdk</groupId>
      <artifactId>aws-sdk-java</artifactId>
      <version>2.X.X</version>
```

```
</dependency>
</dependencies>
</project>
```

Build your project

After you configure the `pom.xml` file, you can use Maven to build your project.

To build your Maven project from the command line, open a terminal or command prompt window, navigate to your project directory (for example, `myapp`), enter or paste the following command, then press Enter or Return.

```
mvn package
```

This creates a single `.jar` file (JAR) in the `target` directory (for example, `myapp/target`). This JAR contains all of the SDK modules you specified as dependencies in your `pom.xml` file.

Setting up a Gradle project

You can use [Gradle](#) to set up and build AWS SDK for Java projects.

To manage SDK dependencies for your Gradle project, import the Maven bill of materials (BOM) for the AWS SDK for Java into the `build.gradle` file.

Note

In the following examples, replace `2.15.0` in the `build.gradle` file with the latest version of the AWS SDK for Java v2. Find the latest version in the [AWS SDK for Java API Reference version 2.x](#).

To configure the AWS SDK for Java for Gradle version 5.0 or later

1. Add the BOM to the *dependencies* section of the file.

```
...
dependencies {
    implementation platform('software.amazon.awssdk:bom:2.15.0')

    // Declare individual SDK dependencies without version
    ...
}
```

2. Specify the SDK modules to use in the *dependencies* section. For example, the following includes a dependency for Amazon Kinesis.

```
...
dependencies {
    ...
    implementation 'software.amazon.awssdk:kinesis'
    ...
}
```

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

The following is an example of a complete `build.gradle` file that includes a dependency for Kinesis.


```
group 'aws.test'
version '1.0'

apply plugin: 'java'

sourceCompatibility = 1.8

repositories {
    mavenCentral()
}

dependencies {
    implementation platform('software.amazon.awssdk:bom:2.15.0')
    implementation 'software.amazon.awssdk:kinesis'
    testImplementation group: 'junit', name: 'junit', version: '4.11'
}
```

Note

In the previous example, replace the dependency for Kinesis with the dependencies of the AWS services you will use in your project. The modules (dependencies) that are managed by the AWS SDK for Java BOM are listed on Maven central repository (<https://mvnrepository.com/artifact/software.amazon.awssdk/bom/latest>).

For more information about specifying SDK dependencies by using the BOM, see [Setting up an Apache Maven project \(p. 15\)](#).

Using the AWS SDK for Java 2.x

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 \(p. 20\)](#)
- [Supplying and retrieving AWS credentials \(p. 21\)](#)
- [AWS region selection \(p. 26\)](#)
- [Asynchronous programming \(p. 28\)](#)
- [Enabling SDK metrics \(p. 34\)](#)
- [Retrieving paginated results \(p. 38\)](#)
- [Using waiters \(p. 44\)](#)
- [Optimizing cold start performance for AWS Lambda \(p. 46\)](#)
- [HTTP configuration \(p. 47\)](#)
- [HTTP/2 programming \(p. 52\)](#)
- [Exception handling \(p. 52\)](#)
- [Logging AWS SDK for Java calls \(p. 53\)](#)
- [Setting the JVM TTL for DNS name lookups \(p. 56\)](#)

Creating service clients

To make requests to Amazon Web Services, you first create a service client object. In version 2.x 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 the default client

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 [Supplying and retrieving AWS credentials \(p. 21\)](#) and `java-dg-region-selection` 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 `AutoCloseable` interface. For best practices, explicitly close a client by calling the `close` method.

To close a client

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

Supplying and retrieving AWS credentials

To make requests to Amazon Web Services (AWS), you must supply AWS credentials to the AWS SDK for Java. You can do this by using the following methods:

- Use the default credential provider chain (*recommended*).
- Use a specific credential provider or provider chain.
- Supply credentials explicitly.

Each of these methods is discussed in the following sections.

Use 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. It uses the *default credential provider chain* implemented by the `DefaultCredentialsProvider` class.

The following example creates a new service client that uses the default credential provider chain:

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

Credential retrieval order

You can use a supported credential retrieval technique to retrieve credentials required to perform AWS operations. For example, the following Java code shows how to create a **DynamoDbClient** object by using an **EnvironmentVariableCredentialsProvider** object.

```
Region region = Region.US_WEST_2;
DynamoDbClient ddb = DynamoDbClient.builder()
    .region(region)
    .credentialsProvider(EnvironmentVariableCredentialsProvider.create())
    .build();
```

The following list shows the supported credential retrieval techniques:

1. **Java system properties**—`aws.accessKeyId` and `aws.secretAccessKey`. 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**— The specific location of this file can vary per platform, but is typically located at `~/.aws/credentials`. This file is 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. You can also create it by editing the file with a text editor. For information about the credentials file format, see [AWS credentials file format \(p. 23\)](#).

4. **Amazon ECS container credentials**— This is loaded from Amazon ECS if the environment variable `AWS_CONTAINER_CREDENTIALS_RELATIVE_URI` is set. The AWS SDK for Java uses the [ContainerCredentialsProvider](#) to load these credentials.
5. **Instance profile credentials**— This is 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 use AWS credentials, supply them in at least one of the preceding locations. For information about setting credentials, see the following topics:

- To supply credentials in the *environment* or in the default *credential profiles file*, see [Set default credentials and Region \(p. 11\)](#).
- 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 `java-dg-roles`.

Setting an alternate credentials profile

The AWS SDK for Java uses the default profile, 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, in Linux, macOS, or Unix, you run the following command to change the profile to *myProfile*.

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

In Windows, run the following command.

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

Setting the `AWS_PROFILE` environment variable affects credential loading for all officially supported AWS SDKs and tools, for example the AWS CLI and the AWS Tools for PowerShell. To change only the profile for a Java application, 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, set this variable with the command line. You can also set the environment variable in your user or system environment to change it for the user specifically or across the system.

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 command 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. You can add or edit them 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.

Use a specific credential provider or provider chain

You can use 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 creates a new service client that uses the *environment* credentials provided, called *EnvironmentVariableCredentialsProvider*:

```
S3Client s3 = S3Client.builder()
    .credentialsProvider(EnvironmentVariableCredentialsProvider.create())
    .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 supply credential providers or provider chains that you create by using your own credential provider that implements the `AwsCredentialsProvider` interface.

Supply credentials explicitly

If the default credential chain or a specific or custom provider or provider chain doesn't work for your code, you can supply the credentials that you want. These can be AWS account credentials, IAM credentials, or temporary credentials retrieved from AWS Security Token Service (AWS STS). If you've retrieved temporary credentials using AWS STS, use this method to specify the credentials for AWS access.

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.

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 to use for the connection.
2. Create an [StaticCredentialsProvider](#) with the `AwsCredentials` object.
3. Configure the client builder with the `StaticCredentialsProvider` and build the client.

The following example creates a new service client that uses credentials that you supplied:

```
AwsSessionCredentials awsCreds = AwsSessionCredentials.create(
    "your_access_key_id_here",
    "your_secret_key_id_here",
    "your_session_token_here");

S3Client s32 = S3Client.builder()
    .credentialsProvider(StaticCredentialsProvider.create(awsCreds))
    .build();
```

Configuring 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.secretAccessKey`.
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.

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.

Choosing a specific endpoint

Each AWS client can be configured to use a *specific endpoint* within a region by calling the `endpointOverride` method.

For example, to configure the Amazon EC2 client to use the Europe (Ireland) Region, use the following code.

```
Ec2Client ec2 = Ec2Client.builder()
    .region(Region.EU_WEST_1)
    .endpointOverride(URI.create("https://ec2.eu-
west-1.amazonaws.com"))
    .build();
```

See [Regions and Endpoints](#) for the current list of regions and their corresponding endpoints for all AWS services.

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 [Use the default credential provider chain \(p. 21\)](#)) 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

The AWS SDK for Java 2.0 features truly nonblocking asynchronous clients that implement high concurrency across a few threads. The AWS SDK for Java 1.11.x has asynchronous clients that are wrappers around a thread pool and blocking synchronous clients that don't provide the full benefit of nonblocking 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 allow you to access the response when it's ready.

Non-streaming operations

For non-streaming operations, asynchronous method calls are similar to synchronous methods. However, 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

```
<problematic>:code-java: `CompletableFuture` </problematic>  
<problematic>:methodname: `whenComplete()` </problematic>  
method with an action to complete when the result is available.  
<problematic>:code-java: `CompletableFuture` </problematic>  
implements the  
<problematic>:code-java: `Future` </problematic>  
interface so you can also get the response object by calling the  
<problematic>:methodname: `get()` </problematic>  
method as well.
```

The following is an example of an asynchronous operation that calls a Amazon DynamoDB function to get a list of tables, receiving a

```
<problematic>:code-java: `CompletableFuture` </problematic>  
that can hold a ListTablesResponse object. The action defined in the call to  
<problematic>:methodname: `whenComplete()` </problematic>  
is done only when the asynchronous call is complete.
```

Imports

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbAsyncClient;
import software.amazon.awssdk.services.dynamodb.model.ListTablesRequest;
import software.amazon.awssdk.services.dynamodb.model.ListTablesResponse;
import java.util.List;
import java.util.concurrent.CompletableFuture;
```

Code

```
public class DynamoDBAsyncListTables {

    public static void main(String[] args) throws InterruptedException {

        // Create the DynamoDbAsyncClient object
        Region region = Region.US_EAST_1;
        DynamoDbAsyncClient client = DynamoDbAsyncClient.builder()
            .region(region)
            .build();

        listTables(client);
    }

    public static void listTables(DynamoDbAsyncClient client) {

        CompletableFuture<ListTablesResponse> response =
            client.listTables(ListTablesRequest.builder()
                .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) -> {
            try {
                if (tables != null) {
                    tables.forEach(System.out::println);
                } else {
                    // Handle error
                    err.printStackTrace();
                }
            } finally {
                // Lets the application shut down. Only close the client when you are
                // completely done with it.
                client.close();
            }
        });
        tableNames.join();
    }
}
```

The following code example shows you how to retrieve an Item from a table by using the Asynchronous client. Invoke the

`<problematic>:methodname: `getItem` </problematic>`

method of the `DynamoDbAsyncClient` and pass it a `GetItemRequest` object with the table name and primary key value of the item you want. This is typically how you pass data that the operation requires. In this example, notice that a String value is passed.

Imports

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.model.GetItemRequest;
```

```
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.DynamoDbAsyncClient;
import java.util.HashMap;
import java.util.Map;
import java.util.Set;
import java.util.stream.Collectors;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
```

Code

```
ic static void getItem(DynamoDbAsyncClient client, String tableName, String key, String
keyVal) {

    HashMap<String, AttributeValue> keyToGet =
        new HashMap<String, AttributeValue>();

    keyToGet.put(key, AttributeValue.builder()
        .s(keyVal).build());

    try {

        // Create a GetItemRequest instance
        GetItemRequest request = GetItemRequest.builder()
            .key(keyToGet)
            .tableName(tableName)
            .build();

        // Invoke the DynamoDbAsyncClient object's getItem
        java.util.Collection<software.amazon.awssdk.services.dynamodb.model.AttributeValue>
returnedItem = client.getItem(request).join().item().values();

        // Convert Set to Map
        Map<String, AttributeValue> map =
returnedItem.stream().collect(Collectors.toMap(AttributeValue::s, s->s));
        Set<String> keys = map.keySet();
        for (String sinKey : keys) {
            System.out.format("%s: %s\n", sinKey, map.get(sinKey).toString());
        }

    } catch (DynamoDbException e) {
        System.err.println(e.getMessage());
        System.exit(1);
    }
}
```

See the [complete example](#) on GitHub.

Streaming operations

For streaming operations, you must provide an [AsyncRequestBody](#) to provide the content incrementally, or an [AsyncResponseTransformer](#) to receive and process the response.

The following example uploads a file to Amazon S3 asynchronously by using the `<problematic>:methodName: `PutObject` </problematic>` operation.

Imports

```
import software.amazon.awssdk.core.async.AsyncRequestBody;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.s3.S3AsyncClient;
```

```
import software.amazon.awssdk.services.s3.model.PutObjectRequest;
import software.amazon.awssdk.services.s3.model.PutObjectResponse;
import java.nio.file.Paths;
import java.util.concurrent.CompletableFuture;
```

Code

```
public class S3AsyncOps {

    public static void main(String[] args) {

        final String USAGE = "\n" +
            "Usage:\n" +
            "    S3AsyncOps <bucketName> <key> <path>\n\n" +
            "Where:\n" +
            "    bucketName - the name of the Amazon S3 bucket (for example, bucket1).
\n\n" +
            "    key - the name of the object (for example, book.pdf). \n" +
            "    path - the local path to the file (for example, C:/AWS/book.pdf).
\n" ;

        if (args.length != 3) {
            System.out.println(USAGE);
            System.exit(1);
        }

        String bucketName = args[0];
        String key = args[1];
        String path = args[2];

        Region region = Region.US_WEST_2;
        S3AsyncClient client = S3AsyncClient.builder()
            .region(region)
            .build();

        PutObjectRequest objectRequest = PutObjectRequest.builder()
            .bucket(bucketName)
            .key(key)
            .build();

        // Put the object into the bucket
        CompletableFuture<PutObjectResponse> future = client.putObject(objectRequest,
            AsyncRequestBody.fromFile(Paths.get(path))
        );
        future.whenComplete((resp, err) -> {
            try {
                if (resp != null) {
                    System.out.println("Object uploaded. Details: " + resp);
                } else {
                    // Handle error
                    err.printStackTrace();
                }
            } finally {
                // Only close the client when you are completely done with it
                client.close();
            }
        });

        future.join();
    }
}
```

The following example gets a file from Amazon S3 asynchronously by using the

<problematic>:methodname: `GetObject` </problematic>
operation.

Imports

```
import software.amazon.awssdk.core.async.AsyncResponseTransformer;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.s3.S3AsyncClient;
import software.amazon.awssdk.services.s3.model.GetObjectRequest;
import software.amazon.awssdk.services.s3.model.GetObjectResponse;
import java.nio.file.Paths;
import java.util.concurrent.CompletableFuture;
```

Code

```
public class S3AsyncStreamOps {

    public static void main(String[] args) {

        final String USAGE = "\n" +
            "Usage:\n" +
            "    S3AsyncStreamOps <bucketName> <objectKey> <path>\n\n" +
            "Where:\n" +
            "    bucketName - the name of the Amazon S3 bucket (for example, bucket1).
\n\n" +
            "    objectKey - the name of the object (for example, book.pdf). \n" +
            "    path - the local path to the file (for example, C:/AWS/book.pdf).
\n" ;

        if (args.length != 3) {
            System.out.println(USAGE);
            System.exit(1);
        }

        String bucketName = args[0];
        String objectKey = args[1];
        String path = args[2];

        Region region = Region.US_WEST_2;
        S3AsyncClient client = S3AsyncClient.builder()
            .region(region)
            .build();

        GetObjectRequest objectRequest = GetObjectRequest.builder()
            .bucket(bucketName)
            .key(objectKey)
            .build();

        CompletableFuture<GetObjectResponse> futureGet = client.getObject(objectRequest,
            AsyncResponseTransformer.toFile(Paths.get(path)));

        futureGet.whenComplete((resp, err) -> {
            try {
                if (resp != null) {
                    System.out.println("Object downloaded. Details: "+resp);
                } else {
                    err.printStackTrace();
                }
            } finally {
                // Only close the client when you are completely done with it
                client.close();
            }
        });
        futureGet.join();
    }
}
```

```
}  
}
```

Advanced operations

The AWS SDK for Java 2.0 uses [Netty](#) an asynchronous event-driven network application framework, to handle I/O threads. The AWS SDK for Java 2.0 creates an `ExecutorService` behind Netty, to complete the futures returned from the HTTP client request through to the Netty client. This abstraction reduces the risk of an application breaking the async process if developers choose to stop or sleep threads. By default, 50 Threads are generated for each asynchronous client, and managed in a queue within the

`ExecutorService`.

Advanced users can specify their thread pool size when creating an asynchronous client using the following option when building.

Code

```
S3AsyncClient clientThread = S3AsyncClient.builder()  
    .asyncConfiguration(  
        b -> b.advancedOption(SdkAdvancedAsyncClientOption  
            .FUTURE_COMPLETION_EXECUTOR,  
            Executors.newFixedThreadPool(10)  
        )  
    )  
    .build();
```

To optimize performance, you can manage your own thread pool executor, and include it when configuring your client.

```
ThreadPoolExecutor executor = new ThreadPoolExecutor(50, 50,  
    10, TimeUnit.SECONDS,  
    new LinkedBlockingQueue<>(10_000),  
    new ThreadFactoryBuilder()  
        .threadNamePrefix("sdk-async-response").build());  
  
// Allow idle core threads to time out  
executor.allowCoreThreadTimeOut(true);  
  
S3AsyncClient clientThread = S3AsyncClient.builder()  
    .asyncConfiguration(  
        b -> b.advancedOption(SdkAdvancedAsyncClientOption  
            .FUTURE_COMPLETION_EXECUTOR,  
            executor  
        )  
    )  
    .build();
```

If you prefer to not use a thread pool, at all, use `Runnable::run` instead of using a thread pool executor.

```
S3AsyncClient clientThread = S3AsyncClient.builder()  
    .asyncConfiguration(  
        b -> b.advancedOption(SdkAdvancedAsyncClientOption  
            .FUTURE_COMPLETION_EXECUTOR,  
            Runnable::run  
        )  
    )
```

```
.build();
```

Enabling SDK metrics

With the AWS SDK for Java 2.x, you can collect metrics about the service clients in your application, analyze the output in Amazon CloudWatch, and then act on it.

By default, metrics collection is disabled in the SDK. This topic helps you to enable and configure it.

Topics

- [Prerequisites \(p. 15\)](#)
- [How to enable metrics collection \(p. 35\)](#)
- [What information is collected? \(p. 35\)](#)
- [Service client metrics \(p. 36\)](#)

Prerequisites

Before you can enable and use metrics, you must complete the following steps:

- [Sign up for AWS and create an IAM user](#)
- [Set up AWS credentials and region for development](#)
- Configure your project dependencies (for example, in your `pom.xml` or `build.gradle` file) to use version 2.14.0 or later of the AWS SDK for Java.

To enabling publishing of metrics to CloudWatch, also include the artifactId `cloudwatch-metric-publisher` with the version number 2.14.0 or later in your project's dependencies.

For example:

```
<project>
  <dependencyManagement>
    <dependencies>
      <dependency>
        <groupId>software.amazon.awssdk</groupId>
        <artifactId>bom</artifactId>
        <version>2.14.0</version>
        <type>pom</type>
        <scope>import</scope>
      </dependency>
    </dependencies>
  </dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>software.amazon.awssdk</groupId>
      <artifactId>cloudwatch-metric-publisher</artifactId>
      <version>2.14.0</version>
    </dependency>
  </dependencies>
</project>
```

Note

To enhance the security of your application, you can use dedicated set of credentials for publishing metrics to CloudWatch. Create a separate IAM user with [cloudwatch:PutMetricData](#)

permissions and then use that user's access key as credentials in the `MetricPublisher` configuration for your application. For more information, see the [Amazon CloudWatch Permissions Reference](#) in the [Amazon CloudWatch User Guide](#) and [Adding and Removing IAM Identity Permissions](#) in the [AWS IAM User Guide](#).

How to enable metrics collection

You can enable metrics in your application for a service client or on individual requests.

Enable metrics for a specific request

The following code snippet shows how to enable the CloudWatch metrics publisher for a request to Amazon DynamoDB. It uses the default metrics publisher configuration.

```
MetricPublisher metricsPub = CloudWatchMetricPublisher.create();
DynamoDbClient ddb = DynamoDbClient.create();
ddb.listTables(ListTablesRequest.builder()
    .overrideConfiguration(c -> c.addMetricPublisher(metricsPub))
    .build());
```

Enable metrics for a specific service client

The following code snippet shows how to enable the CloudWatch metrics publisher for a service client.

```
MetricPublisher metricsPub = CloudWatchMetricPublisher.create();

DynamoDbClient ddb = DynamoDbClient.builder()
    .overrideConfiguration(c -> c.addMetricPublisher(metricsPub))
    .build();
```

The following snippet demonstrates how to use a custom configuration for the metrics publisher for a specific service client. The customizations include loading a separate credentials profile, specifying a different region than the service client, and customizing how often the publisher sends metrics to CloudWatch.

```
MetricPublisher metricsPub = CloudWatchMetricPublisher.builder()

    .credentialsProvider(EnvironmentVariableCredentialsProvider.create("cloudwatch"))
    .region(Region.US_WEST_2)
    .publishFrequency(5, TimeUnit.MINUTES)
    .build();

Region region = Region.US_EAST_1;
DynamoDbClient ddb = DynamoDbClient.builder()
    .region(region)
    .overrideConfiguration(c -> c.addMetricPublisher(metricsPub))
    .build();
```

What information is collected?

Metrics collection includes the following:

- Number of API requests, including whether they succeed or fail
- Information about the AWS services you call in your API requests, including exceptions returned

- The duration for various operations such as Marshalling, Signing, and HTTP requests
- HTTP client metrics, such as the number of open connections, the number of pending requests, and the name of the HTTP client used

Note

The metrics available vary by HTTP client.

For a complete list, see [configuration-metrics-list](#).

How can I use this information?

You can use the metrics the SDK collects to monitor the service clients in your application. You can look at overall usage trends, identify anomalies, review service client exceptions returned, or to dig in to understand a particular issue. Using Amazon CloudWatch, you can also create alarms to notify you as soon as your application reaches a condition that you define.

For more information, see [Using Amazon CloudWatch Metrics](#) and [Using Amazon CloudWatch Alarms](#) in the [Amazon CloudWatch User Guide](#).

Service client metrics

With the AWS SDK for Java version 2 (v2), you can collect metrics about the service clients in your application and then publish (output) those metrics to CloudWatch.

This topic contains the list and descriptions for the metrics that are collected.

For more information about enabling and configuring metrics for the SDK, see [configuration-metrics](#).

Metrics collected with each request

Metric name	Description	Type	Collected by default?
ServiceId	Service ID of the AWS service that the API request is made against	String	Yes
OperationName	The name of the AWS API the request is made to	String	Yes
ApiCallSuccessful	True if the API call was successful; false if not	Boolean	Yes
RetryCount	Number of times the SDK retried the API call	Integer	Yes
ApiCallDuration	The total time taken to finish a request (inclusive of all retries)	Duration	Yes
MarshallingDuration	The time taken to marshall the request	Duration	Yes
CredentialsFetchDuration	The time taken to fetch signing credentials for the request	Duration	Yes

Metrics collected for each request attempt

Each API call that your application makes may take multiple attempts before responded with a success or failure. These metrics are collected for each attempt.

Metric name	Description	Type	Collected by default?
BackoffDelayDuration	The duration of time the SDK waited before this API call attempt	Duration	Yes
MarshallingDuration	The time it takes to marshal an SDK request to an HTTP request	Duration	Yes
SigningDuration	The time it takes to sign the HTTP request	Duration	Yes
ServiceCallDuration	The time it takes to connect to the service, send the request, and receive the HTTP status code and header from the response	Duration	Yes
UnmarshallingDuration	The time it takes to unmarshal an HTTP response to an SDK response	Duration	Yes
AwsRequestId	The request ID of the service request	String	Yes
AwsExtendedRequestId	The extended request ID of the service request	String	Yes
HttpClientName	The name of the HTTP being use for the request	String	Yes
MaxConcurrency	The max number of concurrent requests supported by the HTTP client	Integer	Yes
AvailableConcurrency	The number of remaining concurrent requests that can be supported by the HTTP client without needing to establish another connection	Integer	Yes
LeasedConcurrency	The number of request currently being executed by the HTTP client	Integer	Yes

Metric name	Description	Type	Collected by default?
PendingConcurrencyAcquired	The number of requests that are blocked, waiting for another TCP connection or a new stream to be available from the connection pool	Integer	Yes
HttpStatusCode	The status code returned with the HTTP response	Integer	Yes
LocalStreamWindowSize	The local HTTP/2 window size in bytes this request's stream	Integer	Yes
RemoteStreamWindowSize	The remote HTTP/2 window size in bytes this request's stream	Integer	Yes

Retrieving paginated results

Many AWS operations return paginated results when the response object is too large to return in a single response. In the AWS SDK for Java 1.0, the response contained a token you had to use to retrieve the next page of results. New in the AWS SDK for Java 2.0 are autopagination methods that make multiple service calls to get the next page of results for you automatically. You only have to write code that processes the results. Additionally both types of methods have synchronous and asynchronous versions. See [examples-asynchronous](#) for more detail about asynchronous clients.

The following examples use Amazon S3 and Amazon DynamoDB operations to demonstrate the various methods of retrieving your data from paginated responses.

Note

These code snippets assume that you understand the material in basics, and have configured default AWS credentials using the information in [Set up AWS credentials and region for development](#) (p. 11).

Synchronous pagination

These examples use the synchronous pagination methods for listing objects in an Amazon S3 bucket.

Iterate over pages

Build a [ListObjectsV2Request](#) and provide a bucket name. Optionally you can provide the maximum number of keys to retrieve at one time. Pass it to the `S3Client`'s `listObjectsV2Paginator` method. This method returns a [ListObjectsV2Iterable](#) object, which is an `Iterable` of the [ListObjectsV2Response](#) class.

The first example demonstrates using the paginator object to iterate through all the response pages with the `stream` method. You can directly stream over the response pages, convert the response stream to a stream of [S3Object](#) content, and then process the content of the Amazon S3 object.

Imports

```
import java.io.IOException;
import java.nio.ByteBuffer;
import java.util.Random;
import software.amazon.awssdk.core.waiters.WaiterResponse;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.s3.S3Client;
import software.amazon.awssdk.services.s3.paginators.ListObjectsV2Iterable;
import software.amazon.awssdk.core.sync.RequestBody;
import software.amazon.awssdk.services.s3.model.S3Exception;
import software.amazon.awssdk.services.s3.model.PutObjectRequest;
import software.amazon.awssdk.services.s3.model.ListObjectsV2Request;
import software.amazon.awssdk.services.s3.model.ListObjectsV2Response;
import software.amazon.awssdk.services.s3.model.S3Object;
import software.amazon.awssdk.services.s3.model.GetObjectRequest;
import software.amazon.awssdk.services.s3.model.DeleteObjectRequest;
import software.amazon.awssdk.services.s3.model.DeleteBucketRequest;
import software.amazon.awssdk.services.s3.model.CreateMultipartUploadRequest;
import software.amazon.awssdk.services.s3.model.CreateMultipartUploadResponse;
import software.amazon.awssdk.services.s3.model.CompletedMultipartUpload;
import software.amazon.awssdk.services.s3.model.CreateBucketRequest;
import software.amazon.awssdk.services.s3.model.CompletedPart;
import software.amazon.awssdk.services.s3.model.CreateBucketConfiguration;
import software.amazon.awssdk.services.s3.model.UploadPartRequest;
import software.amazon.awssdk.services.s3.model.CompleteMultipartUploadRequest;
import software.amazon.awssdk.services.s3.waiters.S3Waiter;
import software.amazon.awssdk.services.s3.model.HeadBucketRequest;
import software.amazon.awssdk.services.s3.model.HeadBucketResponse;
```

Code

```
ListObjectsV2Request listReq = ListObjectsV2Request.builder()
    .bucket(bucketName)
    .maxKeys(1)
    .build();

ListObjectsV2Iterable listRes = s3.listObjectsV2Paginator(listReq);
// Process response pages
listRes.stream()
    .flatMap(r -> r.contents().stream())
    .forEach(content -> System.out.println(" Key: " + content.key() + " size = " +
        content.size()));
```

See the [complete example](#) on GitHub.

Iterate over objects

The following examples show ways to iterate over the objects returned in the response instead of the pages of the response.

Use a stream

Use the `stream` method on the response content to iterate over the paginated item collection.

Code

```
// Helper method to work with paginated collection of items directly
listRes.contents().stream()
    .forEach(content -> System.out.println(" Key: " + content.key() + " size = " +
        content.size()));
```

See the [complete example](#) on GitHub.

Use a for loop

Use a standard `for` loop to iterate through the contents of the response.

Code

```
for (S3Object content : listRes.contents()) {
    System.out.println(" Key: " + content.key() + " size = " + content.size());
}
```

See the [complete example](#) on GitHub.

Manual pagination

If your use case requires it, manual pagination is still available. Use the next token in the response object for the subsequent requests. Here's an example using a `while` loop.

Code

```
ListObjectsV2Request listObjectsReqManual = ListObjectsV2Request.builder()
    .bucket(bucketName)
    .maxKeys(1)
    .build();

boolean done = false;
while (!done) {
    ListObjectsV2Response listObjResponse = s3.listObjectsV2(listObjectsReqManual);
    for (S3Object content : listObjResponse.contents()) {
        System.out.println(content.key());
    }

    if (listObjResponse.nextContinuationToken() == null) {
        done = true;
    }

    listObjectsReqManual = listObjectsReqManual.toBuilder()
        .continuationToken(listObjResponse.nextContinuationToken())
        .build();
}
```

See the [complete example](#) on GitHub.

Asynchronous pagination

These examples use the asynchronous pagination methods for listing tables in DynamoDB. A manual pagination example is available in the [basics-async](#) topic.

Iterate over pages of table names

First, create an asynchronous DynamoDB client. Then, call the `listTablesPaginator` method to get a [ListTablesPublisher](#). This is an implementation of the reactive streams Publisher interface. To learn more about the reactive streams model, see the [Reactive Streams Github repo](#).

Call the `subscribe` method on the [ListTablesPublisher](#) and pass a subscriber implementation. In this example, the subscriber has an `onNext` method that requests one item at a time from the publisher. This

is the method that is called repeatedly until all pages are retrieved. The `onSubscribe` method calls the `Subscription.request` method to initiate requests for data from the publisher. This method must be called to start getting data from the publisher. The `onError` method is triggered if an error occurs while retrieving data. Finally, the `onComplete` method is called when all pages have been requested.

Use a subscriber

Imports

```
import java.util.List;
import java.util.concurrent.CompletableFuture;
import java.util.concurrent.ExecutionException;

import org.reactivestreams.Subscriber;
import org.reactivestreams.Subscription;

import software.amazon.awssdk.core.async.SdkPublisher;
import software.amazon.awssdk.services.dynamodb.DynamoDbAsyncClient;
import software.amazon.awssdk.services.dynamodb.model.ListTablesRequest;
import software.amazon.awssdk.services.dynamodb.model.ListTablesResponse;
import software.amazon.awssdk.services.dynamodb.paginators.ListTablesPublisher;
import io.reactivex.Flowable;
import reactor.core.publisher.Flux;
```

Code

First create a async client

```
// Creates a default client with credentials and regions loaded from the environment
final DynamoDbAsyncClient asyncClient = DynamoDbAsyncClient.create();

ListTablesRequest listTablesRequest = ListTablesRequest.builder().limit(3).build();
```

Then use `Subscriber` to get results.

```
// Or subscribe method should be called to create a new Subscription.
// A Subscription represents a one-to-one life-cycle of a Subscriber subscribing to a
// Publisher.
publisher.subscribe(new Subscriber<ListTablesResponse>() {
    // Maintain a reference to the subscription object, which is required to request data
    // from the publisher
    private Subscription subscription;

    @Override
    public void onSubscribe(Subscription s) {
        subscription = s;
        // Request method should be called to demand data. Here we request a single page
        subscription.request(1);
    }

    @Override
    public void onNext(ListTablesResponse response) {
        response.tableNames().forEach(System.out::println);
        // Once you process the current page, call the request method to signal that you
        // are ready for next page
        subscription.request(1);
    }

    @Override
    public void onError(Throwable t) {
        // Called when an error has occurred while processing the requests
    }
});
```

```
}

@Override
public void onComplete() {
    // This indicates all the results are delivered and there are no more pages left
}
```

See the [complete example](#) on GitHub.

Use a for loop

Use a for loop to iterate through the pages for simple use cases when creating a new subscriber might be too much overhead. The response publisher object has a `forEach` helper method for this purpose.

Code

```
ListTablesPublisher publisher = asyncClient.listTablesPaginator(listTablesRequest);

// Use a for-loop for simple use cases
CompletableFuture<Void> future = publisher.subscribe(response -> response.tableNames()
    .forEach(System.out::println));
```

See the [complete example](#) on GitHub.

Iterate over table names

The following examples show ways to iterate over the objects returned in the response instead of the pages of the response. Similar to the synchronous result, the asynchronous result class has a method to interact with the underlying item collection. The return type of the convenience method is a publisher that can be used to request items across all pages.

Use a subscriber

Code

First create a asyc client

```
System.out.println("running AutoPagination - iterating on item collection...\n");

// Creates a default client with credentials and regions loaded from the environment
final DynamoDbAsyncClient asyncClient = DynamoDbAsyncClient.create();

ListTablesRequest listTablesRequest = ListTablesRequest.builder().limit(3).build();
```

Then use Subscriber to get results.

```
// Use subscriber
publisher.subscribe(new Subscriber<String>() {
    private Subscription subscription;

    @Override
    public void onSubscribe(Subscription s) {
        subscription = s;
        subscription.request(1);
    }

    @Override
    public void onNext(String tableName) {
```



```
        System.out.println(tableName);
        subscription.request(1);
    }

    @Override
    public void onError(Throwable t) { }

    @Override
    public void onComplete() { }
```

See the [complete example](#) on GitHub.

Use a for loop

Use the `forEach` convenience method to iterate through the results.

Code

```
// Use forEach
CompletableFuture<Void> future = publisher.subscribe(System.out::println);
future.get();
```

See the [complete example](#) on GitHub.

Use third-party library

You can use other third party libraries instead of implementing a custom subscriber. This example demonstrates using the RxJava implementation but any library that implements the reactive stream interfaces can be used. See the [RxJava wiki page on Github](#) for more information on that library.

To use the library, add it as a dependency. If using Maven, the example shows the POM snippet to use.

POM Entry

```
        <version>2.11.4-PREVIEW</version>
    </dependency>
    <dependency>
        <groupId>org.junit.jupiter</groupId>
        <artifactId>junit-jupiter-api</artifactId>
```

Imports

```
import java.util.List;
import java.util.concurrent.CompletableFuture;
import java.util.concurrent.ExecutionException;

import org.reactivestreams.Subscriber;
import org.reactivestreams.Subscription;

import software.amazon.awssdk.core.async.SdkPublisher;
import software.amazon.awssdk.services.dynamodb.DynamoDbAsyncClient;
import software.amazon.awssdk.services.dynamodb.model.ListTablesRequest;
import software.amazon.awssdk.services.dynamodb.model.ListTablesResponse;
import software.amazon.awssdk.services.dynamodb.paginators.ListTablesPublisher;
import io.reactivex.Flowable;
import reactor.core.publisher.Flux;
```

Code

```
System.out.println("running AutoPagination - using third party subscriber...\n");

DynamoDbAsyncClient asyncClient = DynamoDbAsyncClient.create();
ListTablesPublisher publisher = asyncClient.listTablesPaginator(ListTablesRequest.builder()
                                                                .build());

// The Flowable class has many helper methods that work with any reactive streams
// compatible publisher implementation
List<String> tables = Flowable.fromPublisher(publisher)
                             .flatMapIterable(ListTablesResponse::tableNames)
                             .toList()
                             .blockingGet();

System.out.println(tables);
```

Using waiters

The waiters utility of the AWS SDK for Java 2.x enables you to validate that AWS resources are in a specified state before performing operations on those resources.

A *waiter* is an abstraction used to poll AWS resources, such as DynamoDB tables or Amazon S3 buckets, until a desired state is reached (or until a determination is made that the resource won't ever reach the desired state). Instead of writing logic to continuously poll your AWS resources, which can be cumbersome and error-prone, you can use waiters to poll a resource and have your code continue to run after the resource is ready.

Prerequisites

Before you can use waiters in a project with the AWS SDK for Java, you must complete the steps in [Setting up the AWS SDK for Java 2.x \(p. 10\)](#).

You must also configure your project dependencies (for example, in your `pom.xml` or `build.gradle` file) to use version 2.15.0 or later of the AWS SDK for Java.

For example:

```
<project>
  <dependencyManagement>
    <dependencies>
      <dependency>
        <groupId>software.amazon.awssdk</groupId>
        <artifactId>bom</artifactId>
        <version>2.15.0</version>
        <type>pom</type>
        <scope>import</scope>
      </dependency>
    </dependencies>
  </dependencyManagement>
</project>
```

Using waiters

To instantiate a waiters object, first create a service client. Set the service client's `waiter()` method as the value of the waiter object. Once the waiter instance exists, set its response options to execute the appropriate code.

Synchronous programming

The following code snippet shows how to wait for a DynamoDB table to exist and be in an **ACTIVE** state.

```
DynamoDbClient dynamo = DynamoDbClient.create();
DynamoDbWaiter waiter = dynamo.waiter();

WaiterResponse<DescribeTableResponse> waiterResponse =
    waiter.waitUntilTableExists(r -> r.tableName("myTable"));

// print out the matched response with a tableStatus of ACTIVE
waiterResponse.matched().response().ifPresent(System.out::println);
```

Asynchronous programming

The following code snippet shows how to wait for a DynamoDB table to no longer exist.

```
DynamoDbAsyncClient asyncDynamo = DynamoDbAsyncClient.create();
DynamoDbAsyncWaiter asyncWaiter = asyncDynamo.waiter();

CompletableFuture<WaiterResponse<DescribeTableResponse>> waiterResponse =
    asyncWaiter.waitUntilTableNotExists(r -> r.tableName("myTable"));

waiterResponse.whenComplete((r, t) -> {
    if (t == null) {
        // print out the matched ResourceNotFoundException
        r.matched().exception().ifPresent(System.out::println);
    }
}).join();
```

Configuring waiters

You can customize the configuration for a waiter by using the `overrideConfiguration()` on its builder. For some operations, you can apply a custom configuration when you make the request.

Configure a waiter

The following code snippet shows how to override the configuration on a waiter.

```
// sync
DynamoDbWaiter waiter =
    DynamoDbWaiter.builder()
        .overrideConfiguration(b -> b.maxAttempts(10))
        .client(dynamoDbClient)
        .build();

// async
DynamoDbAsyncWaiter asyncWaiter =
    DynamoDbAsyncWaiter.builder()
        .client(dynamoDbAsyncClient)
        .overrideConfiguration(o -> o.backoffStrategy(
            FixedDelayBackoffStrategy.create(Duration.ofSeconds(2))))
        .scheduledExecutorService(Executors.newScheduledThreadPool(3))
        .build();
```

Override configuration for a specific request

The following code snippet shows how to override the configuration for a waiter on a per-request basis. Note that only some operations have customizable configurations.

```
waiter.waitForTableNotExists(b -> b.tableName("myTable"),
                             o -> o.maxAttempts(10));

asyncWaiter.waitForTableExists(b -> b.tableName("myTable"),
                               o -> o.waitFor(Duration.ofMinutes(1)));
```

Code examples

For a complete example using waiters with DynamoDB, see [CreateTable.java](#) in the AWS Code Examples Repository.

For a complete example using waiters with Amazon S3, see [S3BucketOps.java](#) in the AWS Code Examples Repository.

Optimizing cold start performance for AWS Lambda

Among the improvements in the AWS SDK for Java 2.0 is the SDK cold startup time for Java functions in Lambda. This is the time it takes for a Java Lambda function to start up and respond to its first request.

Version 2.x includes three primary changes that contribute to this improvement:

- Use of [Jackson-JR](#), which is a serialization library that improves initialization time.
- Use of the [java.time](#) libraries for date and time objects.
- Use of [Slf4j](#) for a logging facade.

You can gain additional SDK startup time improvement by setting specific configuration values on the client builder. They each save some time at startup by reducing the amount of information your application needs to find for initialization.

In your client builder, specify a region, use Environment Variable credentials provider, and specify `URLConnectionClient` as the `httpClient`. See the code snippet below for an example.

- [The region lookup process](#) for the SDK takes time. By specifying a region, you can save up to 80ms of initialization time.

Note

By specifying an AWS region, the code will not run in other regions without modification.

- The process the SDK uses to look for credentials can take up to 90ms. By using the [EnvironmentVariableCredentialsProvider](#)

Note

Using this credentials provider enables the code to be used in Lambda functions, but may not work on Amazon EC2 or other systems.

- Instantiation time for JDK's [URLConnection](#) library is much lower than Apache HTTP Client or Netty. You can save up to 1 second by using this HTTP client.

Example client configuration

```
S3Client client = S3Client.builder()
```

```
.region(Region.US_WEST_2)
.credentialsProvider(EnvironmentVariableCredentialsProvider.create())
.httpClient(URLConnectionHttpClient.builder().build())
.build();
```

HTTP configuration

You can change the default configuration for HTTP clients in applications you build with the AWS SDK for Java. This section discusses how to configure HTTP clients and settings for the AWS SDK for Java 2.0. Some of the available settings including specifying which HTTP client to use, as well as setting max concurrency, connection timeout, and maximum retries.

You can use the [NettyNioAsyncHttpClient](#) or [AwsCrtAsyncHttpClient](#) for asynchronous clients. For more information, see [configuration-http-netty](#) or [configuration-http-crt](#).

For synchronous clients, you can use [ApacheHttpClient](#). For more information about Apache HTTPClient, see [HttpClient Overview](#).

For a full list of options you can set with these clients, see the [AWS SDK for Java 2.x API Reference](#).

Topics

- [Setting maximum connections \(p. 47\)](#)
- [Timeouts and error handling \(p. 47\)](#)
- [Local address \(p. 48\)](#)
- [Configuring the AWS CRT-based HTTP client \(p. 48\)](#)
- [Configuring the Netty-based HTTP client \(p. 51\)](#)

Setting maximum connections

You can set the maximum allowed number of open HTTP connections by using the `maxConcurrency` method. The `maxPendingConnectionAcquires` method enables you to set the maximum requests allowed to queue up once max concurrency is reached.

- Default for `maxConcurrency`: 50
- Default for `maxPendingConnectionAcquires`: 10_000

Note

Set the maximum connections to the number of concurrent transactions to avoid connection contentions and poor performance.

Timeouts and error handling

You can set options related to timeouts and handling errors with HTTP connections.

• Connection Timeout

The connection timeout is the amount of time (in milliseconds) that the HTTP connection will wait to establish a connection before giving up. The default is 10,000 ms.

To set this value yourself, use the [ClientConfiguration.setConnectionTimeout](#) method.

• Connection Time to Live (TTL)

By default, the SDK will attempt to reuse HTTP connections as long as possible. In failure situations where a connection is established to a server that has been brought out of service, having a finite TTL can help with application recovery. For example, setting a 15 minute TTL will ensure that even if you have a connection established to a server that is experiencing issues, you'll reestablish a connection to a new server within 15 minutes.

To set the HTTP connection TTL, use the [ClientConfiguration.setConnectionTTL](#) method.

- **Maximum Error Retries**

The default maximum retry count for retrievable errors is 3. You can set a different value by using the [ClientConfiguration.setMaxErrorRetry](#) method.

Local address

To set the local address that the HTTP client will bind to, use [ClientConfiguration.setLocalAddress](#).

Configuring the AWS CRT-based HTTP client

The AWS Common Runtime (CRT) HTTP client is a new HTTP client you can use with the AWS SDK for Java 2.0. The CRT-based HTTP client is an asynchronous, non-blocking HTTP client built on top of the Java bindings of the [AWS Common Runtime](#). You can use the CRT-based HTTP client to benefit from features such as improved performance, connection health checks, and post-quantum TLS support.

For asynchronous operations in the AWS SDK for Java 2.0, you can use Netty ([NettyNioAsyncHttpClient](#)) as the HTTP client or you can use the new AWS Common Runtime (CRT) HTTP client [AwsCrtAsyncHttpClient](#). This topics shows you how to configure the AWS CRT-based HTTP client.

Prerequisites

Before you can use the AWS CRT client, you need to configure your project dependencies in your `pom.xml` or `build.gradle` file to do the following:

- Use version 2.14.13 or later of the AWS SDK for Java.
- Include version 2.14.13-PREVIEW of the artifactId `aws-crt-client`.

The following code example shows how to configure your project dependencies.

```
<project>
  <dependencyManagement>
    <dependencies>
      <dependency>
        <groupId>software.amazon.awssdk</groupId>
        <artifactId>bom</artifactId>
        <version>2.14.13</version>
        <type>pom</type>
        <scope>import</scope>
      </dependency>
    </dependencies>
  </dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>software.amazon.awssdk</groupId>
      <artifactId>aws-crt-client</artifactId>
      <version>2.14.13-PREVIEW</version>
    </dependency>
  </dependencies>
</project>
```

```
</dependencies>  
</project>
```

Using the CRT-based HTTP client

You can use the CRT-based HTTP client for a specific service client, or you can create a single HTTP client to share across multiple service clients. These options are recommended for most use cases. Alternatively, you can set the CRT-based client as the default HTTP client for all asynchronous service clients and requests in your application.

The following code example shows how to use the CRT-based HTTP client for a specific service client.

```
S3AsyncClient.builder()  
    .httpClientBuilder(AwsCrtAsyncHttpClient.builder()  
        .maxConcurrency(50))  
    .build();
```

The following code example shows how to use the CRT-based HTTP client as a shared HTTP client.

```
SdkAsyncHttpClient crtClient = AwsCrtAsyncHttpClient.create()  
S3AsyncClient.builder()  
    .httpClient(crtClient)  
    .build();
```

Note

Your application must manage the lifecycle of an HTTP client instantiated outside of a service client builder. (A `builder` is static factory method used by the AWS SDK for Java to connect to Amazon Web Services such as Amazon S3 and AWS KMS. For more information, see [Creating service clients \(p. 20\)](#).)

Setting the CRT-based HTTP client as the default

For asynchronous operations in the AWS SDK for Java 2.0, you can use Netty ([NettyNioAsyncHttpClient](#)) or the new AWS CRT-based HTTP client ([AwsCrtAsyncHttpClient](#)) as the default asynchronous HTTP client in the AWS SDK for Java 2.0.

Instead of using Netty as the asynchronous HTTP client, you can set the CRT-based HTTP client to be the default for your application. You can set this in your project's dependencies (for example, Maven `pom.xml` file) by explicitly excluding Netty. Alternatively, you can set the default HTTP client via Java system property when you run your app or in your application code.

Remove Netty from the project dependencies

Refer to the following snippet of a Maven `pom.xml` file.

```
<project>  
  <dependencies>  
    <dependency>  
      <groupId>software.amazon.awssdk</groupId>  
      <artifactId>s3</artifactId>  
      <version>2.14.13</version>  
      <exclusions>  
        <exclusion>  
          <groupId>software.amazon.awssdk</groupId>  
          <artifactId>netty-nio-client</artifactId>  
        </exclusion>  
      </exclusions>  
    </dependency>  
  </dependencies>  
</project>
```

```
</dependency>
<dependency>
  <groupId>software.amazon.awssdk</groupId>
  <artifactId>aws-crt-client</artifactId>
  <version>2.14.13-PREVIEW</version>
</dependency>
</dependencies>
</project>
```

Setting via Java system property

To use the CRT-based HTTP client as the default HTTP for your application, you can set the Java system property `software.amazon.awssdk.http.async.service.impl` to a value of `software.amazon.awssdk.http.crt.AwsCrtSdkHttpClientService`.

To set during application startup, run a command similar to the following.

```
java app.jar -Dsoftware.amazon.awssdk.http.async.service.impl=\
software.amazon.awssdk.http.crt.AwsCrtSdkHttpClientService
```

Use the following code snippet to set in your application code.

```
System.setProperty("software.amazon.awssdk.http.async.service.impl",
"software.amazon.awssdk.http.crt.AwsCrtSdkHttpClientService");
```

Configuring the CRT-based HTTP client

With the CRT-based HTTP client within the AWS SDK for Java, you can configure various settings including connection health checks and maximum idle time. You can also configure post-quantum TLS support when you make requests to AWS Key Management Service (AWS KMS).

Connection health checks

You can configure connection health checks for the CRT-based HTTP client using the `connectionHealthChecks` method on the HTTP client builder. Refer to the following example code snippet and the [API documentation](#).

```
AwsCrtAsyncHttpClient.builder()
    .connectionHealthChecksConfiguration(
        b -> b.minThroughputInBytesPerSecond(32000L)
    )
    .allowableThroughputFailureInterval(Duration.ofSeconds(3))
    .build();
```

Post-quantum TLS support

You can configure the CRT-based HTTP client to use post-quantum TLS when your application makes requests to AWS KMS. Use the `tlsCipherPreference` method on the HTTP client builder. Refer to the following example code snippet and the [API documentation](#).

```
SdkAsyncHttpClient awsCrtHttpClient = AwsCrtAsyncHttpClient.builder()
    .tlsCipherPreference(TlsCipherPreference.TLS_CIPHER_KMS_PQ_TLSv1_0_2019_06)
    .build();
KmsAsyncClient kms = KmsAsyncClient.builder()
    .httpClient(awsCrtHttpClient)
```



```
.build();
```

Configuring the Netty-based HTTP client

For asynchronous operations in the AWS SDK for Java 2.0, you can use Netty ([NettyNioAsyncHttpClient](#)) as the HTTP client or you can use the new AWS Common Runtime (CRT) HTTP client [AwsCrtAsyncHttpClient](#). This topics shows you how to configure the Netty-based HTTP client.

For a full list of options you can set with these clients, see the [AWS SDK for Java API Reference version 2.x](#).

Prerequisite

Before you can use the Netty client, you need to configure your project dependencies in your `pom.xml` or `build.gradle` file to include version 2.0.0 or later of the artifactId `netty-nio-client`.

The following code example shows how to configure your project dependencies.

```
<dependency>
  <artifactId>netty-nio-client</artifactId>
  <groupId>software.amazon.awssdk</groupId>
  <version>2.0.0</version>
</dependency>
```

Configuring service clients

Use the HTTP client builder to have the SDK manage its lifecycle. The HTTP client will be closed for you when the service client is shut down.

Imports

```
import software.amazon.awssdk.http.async.SdkAsyncHttpClient;
import software.amazon.awssdk.http.nio.netty.NettyNioAsyncHttpClient;
import software.amazon.awssdk.services.kinesis.KinesisAsyncClient;
```

Code

```
KinesisAsyncClient client = KinesisAsyncClient.builder()
    .httpClientBuilder(NettyNioAsyncHttpClient.builder()
        .maxConcurrency(100)
        .maxPendingConnectionAcquires(10_000))
    .build();
```

You can also pass the HTTP client directly to the service client if you want to manage the lifecycle yourself.

Code

```
SdkAsyncHttpClient httpClient = NettyNioAsyncHttpClient.builder()
    .maxConcurrency(100)
    .maxPendingConnectionAcquires(10_000)
    .build();
```

```
KinesisAsyncClient kinesisClient = KinesisAsyncClient.builder()  
    .httpClient(httpClient)  
    .build();  
  
httpClient.close();
```

HTTP/2 programming

HTTP/2 is a major revision of the HTTP protocol. This new version has several enhancements to improve performance:

- Binary data encoding provides more efficient data transfer.
- Header compression reduces the overhead bytes downloaded by the client, helping get the content to the client sooner. This is especially useful for mobile clients that are already constrained on bandwidth.
- Bidirectional asynchronous communication (multiplexing) allows multiple requests and response messages between the client and AWS to be in flight at the same time over a single connection, instead of over multiple connections, which improves performance.

Developers upgrading to the latest SDKs will automatically use HTTP/2 when it's supported by the service they're working with. New programming interfaces seamlessly take advantage of HTTP/2 features and provide new ways to build applications.

The AWS SDK for Java 2.0 features new APIs for event streaming that implement the HTTP/2 protocol. For examples of how to use these new APIs, see [Working with Kinesis \(p. 136\)](#).

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.

SdkServiceException (and subclasses)

[SdkServiceException](#) 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 `SdkServiceException` that's thrown. For some cases, a subclass of `SdkServiceException` is thrown to allow developers fine-grained control over handling error cases through catch blocks.

When you encounter an `SdkServiceException`, 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.

`SdkServiceException` 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

SdkClientException

`SdkClientException` 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 `SdkClientException` is generally more severe than an `SdkServiceException`, 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 `SdkClientException` 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 [Slf4j](#), 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](#).

Add the Log4J JAR

To use Log4j with the SDK, you need to download the Log4j JAR from the [Log4j website](#) or use Maven by adding a dependency on Log4j in your pom.xml file. The SDK doesn't include the JAR.

Log4j configuration file

Log4j uses a configuration file, log4j2.xml. Example configuration files are shown below. To learn more about the values used in the configuration file, see the [manual for Log4j configuration](#).

Place your configuration file in a directory on your classpath. The Log4j JAR and the log4j2.xml file do not have to be in the same directory.

The log4j2.xml 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:

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

Setting the classpath

Both the Log4j JAR and the log4j2.xml file must be located on your classpath. To configure the log4j binding for SL4j in Maven you can add the following to your pom.xml:

```
<dependency>
  <groupId>org.apache.logging.log4j</groupId>
  <artifactId>log4j-core</artifactId>
</dependency>
<dependency>
  <groupId>org.apache.logging.log4j</groupId>
  <artifactId>log4j-api</artifactId>
</dependency>
<dependency>
  <groupId>org.apache.logging.log4j</groupId>
  <artifactId>log4j-slf4j-impl</artifactId>
</dependency>
```

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 log4j2.xml 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.

```
<Configuration status="WARN">
  <Appenders>
    <Console name="ConsoleAppender" target="SYSTEM_OUT">
      <PatternLayout pattern="%d{YYYY-MM-dd HH:mm:ss} [%t] %-5p %c:%L - %m%n" />
    </Console>
  </Appenders>

  <Loggers>
    <Root level="WARN">
      <AppenderRef ref="ConsoleAppender"/>
    </Root>
    <Logger name="software.amazon.awssdk" level="WARN" />
  </Loggers>
</Configuration>
```

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 log4j2.xml file enables a summary of requests and responses.

```
<Configuration status="WARN">
```

```
<Appenders>
  <Console name="ConsoleAppender" target="SYSTEM_OUT">
    <PatternLayout pattern="%d{YYYY-MM-dd HH:mm:ss} [%t] %-5p %c:%L - %m%n" />
  </Console>
</Appenders>

<Loggers>
  <Root level="WARN">
    <AppenderRef ref="ConsoleAppender"/>
  </Root>
  <Logger name="software.amazon.awssdk" level="WARN" />
  <Logger name="software.amazon.awssdk.request" level="DEBUG" />
</Loggers>
</Configuration>
```

Here is an example of the log output:

```
2018-01-28 19:31:56 [main] DEBUG software.amazon.awssdk.request:Logger.java:78 - Sending
Request: software.amazon.awssdk.http.DefaultSdkHttpFullRequest@3a80515c
```

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 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 `log4j2.xml` file turns on full wire logging in Apache HttpClient.

```
<Configuration status="WARN">
  <Appenders>
    <Console name="ConsoleAppender" target="SYSTEM_OUT">
      <PatternLayout pattern="%d{YYYY-MM-dd HH:mm:ss} [%t] %-5p %c:%L - %m%n" />
    </Console>
  </Appenders>

  <Loggers>
    <Root level="WARN">
      <AppenderRef ref="ConsoleAppender"/>
    </Root>
    <Logger name="software.amazon.awssdk" level="WARN" />
    <Logger name="software.amazon.awssdk.request" level="DEBUG" />
    <Logger name="org.apache.http.wire" level="DEBUG" />
  </Loggers>
</Configuration>
```

Additional Maven dependency on `log4j-1.2-api` is required for wire-logging with Apache as it uses 1.2 under the hood. Add the following to the `pom.xml` file if you enable wire logging.

```
<dependency>
  <groupId>org.apache.logging.log4j</groupId>
  <artifactId>log4j-1.2-api</artifactId>
</dependency>
```

Setting the JVM TTL for DNS name lookups

The Java virtual machine (JVM) caches DNS name lookups. When the JVM resolves a hostname to an IP address, it caches the IP address for a specified period of time, known as the *time-to-live* (TTL).

Because AWS resources use DNS name entries that occasionally change, we recommend that you configure your JVM with a TTL value of no more than 60 seconds. This ensures that when a resource's IP address changes, your application will be able to receive and use the resource's new IP address by requering the DNS.

On some Java configurations, the JVM default TTL is set so that it will *never* refresh DNS entries until the JVM is restarted. Thus, if the IP address for an AWS resource changes while your application is still running, it won't be able to use that resource until you *manually restart* the JVM and the cached IP information is refreshed. In this case, it's crucial to set the JVM's TTL so that it will periodically refresh its cached IP information.

Note

The default TTL can vary according to the version of your JVM and whether a [security manager](#) is installed. Many JVMs provide a default TTL less than 60 seconds. If you're using such a JVM and not using a security manager, you can ignore the remainder of this topic.

How to set the JVM TTL

To modify the JVM's TTL, set the `networkaddress.cache.ttl` property value. Use one of the following methods, depending on your needs:

- **globally, for all applications that use the JVM.** Set `networkaddress.cache.ttl` in the `$JAVA_HOME/jre/lib/security/java.security` file:

```
networkaddress.cache.ttl=60
```

- **for your application only,** set `networkaddress.cache.ttl` in your application's initialization code:

```
java.security.Security.setProperty("networkaddress.cache.ttl" , "60");
```

Code examples for the AWS SDK for Java 2.x

This section provides programming examples you can use with the AWS SDK for Java 2.x for specific features, use cases, and Amazon Web Services.

Find the source code for these examples and others in the AWS documentation [code examples repository on GitHub](#).

To propose a new code example for the AWS documentation team to consider producing, create a new request. The team is looking to produce code examples that cover broader scenarios and use cases, versus simple code snippets that cover only individual API calls. For instructions, see the “Proposing new code examples” section in the [Readme on GitHub](#).

Topics

- [Working with Amazon S3 \(p. 58\)](#)
- [Working with DynamoDB \(p. 71\)](#)
- [Working with Amazon EC2 \(p. 89\)](#)
- [Working with IAM \(p. 103\)](#)
- [Amazon Athena \(p. 120\)](#)
- [Working with CloudWatch \(p. 120\)](#)
- [AWS CloudTrail \(p. 130\)](#)
- [Working with Amazon Cognito \(p. 130\)](#)
- [Amazon Comprehend \(p. 134\)](#)
- [Amazon EventBridge \(p. 135\)](#)
- [Amazon Kinesis Data Firehose \(p. 135\)](#)
- [Amazon Forecast examples \(p. 135\)](#)
- [Amazon S3 Glacier examples \(p. 135\)](#)
- [AWS Glue examples \(p. 135\)](#)
- [Working with Kinesis \(p. 136\)](#)
- [AWS KMS \(p. 141\)](#)
- [Invoke, list, and delete AWS Lambda functions \(p. 142\)](#)
- [WS Elemental MediaConvert \(p. 144\)](#)
- [AWS Elemental MediaStore examples \(p. 144\)](#)
- [AWS Elemental MediaStore examples \(p. 144\)](#)
- [Amazon Personalize examples \(p. 144\)](#)
- [Working with Amazon Pinpoint \(p. 145\)](#)
- [Amazon Polly examples \(p. 152\)](#)
- [Amazon RDS \(p. 152\)](#)
- [Amazon Redshift \(p. 152\)](#)
- [Amazon Rekognition \(p. 152\)](#)
- [Amazon SageMaker \(p. 153\)](#)
- [AWS Secrets Manager \(p. 153\)](#)
- [Amazon Simple Email Service \(p. 153\)](#)
- [Amazon Simple Notification Service examples \(p. 153\)](#)

- [Working with Amazon SQS](#) (p. 157)
- [AWS Systems Manager Agent](#) (p. 162)
- [Amazon SWF](#) (p. 162)
- [Amazon Textract](#) (p. 162)
- [Working with Amazon Transcribe](#) (p. 162)
- [Amazon Translate](#) (p. 166)
- [Amazon WorkDocs](#) (p. 166)

Working with Amazon S3

This section provides examples of programming with [Amazon S3](#) using the AWS SDK for Java 2.0.

The following 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](#) (p. 58)
- [Work with Amazon S3 objects](#) (p. 62)
- [Working with Amazon S3 Presigned URLs](#) (p. 68)

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.

Note

Best Practice

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.

Note

These code snippets assume that you understand the material in basics, and have configured default AWS credentials using the information in [Set up AWS credentials and region for development](#) (p. 11).

Topics

- [Create a bucket](#) (p. 58)
- [List the buckets](#) (p. 59)
- [Delete a bucket](#) (p. 60)

Create a bucket

Build a [CreateBucketRequest](#) and provide a bucket name. Pass it to the S3Client's `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.core.waiters.WaiterResponse;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.s3.S3Client;
import software.amazon.awssdk.services.s3.model.*;
import software.amazon.awssdk.services.s3.waiters.S3Waiter;
```

Code

First create an S3Client.

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

Make a Create Bucket Request.

```
// Create a bucket by using a S3Waiter object
public static void createBucket( S3Client s3Client, String bucketName, Region region) {

    try {
        S3Waiter s3Waiter = s3Client.waiter();
        CreateBucketRequest bucketRequest = CreateBucketRequest.builder()
            .bucket(bucketName)
            .createBucketConfiguration(
                CreateBucketConfiguration.builder()
                    .locationConstraint(region.id())
                    .build()
            )
            .build();

        s3Client.createBucket(bucketRequest);
        HeadBucketRequest bucketRequestWait = HeadBucketRequest.builder()
            .bucket(bucketName)
            .build();

        // Wait until the bucket is created and print out the response
        WaiterResponse<HeadBucketResponse> waiterResponse =
        s3Waiter.waitUntilBucketExists(bucketRequestWait);
        waiterResponse.matched().response().ifPresent(System.out::println);
        System.out.println(bucketName + " is ready");

    } catch (S3Exception e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
```

See the [complete example](#) on GitHub.

List the buckets

Build a [ListBucketsRequest](#). Use the S3Client's `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.core.waiters.WaiterResponse;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.s3.S3Client;
import software.amazon.awssdk.services.s3.model.*;
import software.amazon.awssdk.services.s3.waiters.S3Waiter;
```

Code

First create an S3Client.

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

Make a List Buckets Request.

```
// List buckets
ListBucketsRequest listBucketsRequest = ListBucketsRequest.builder().build();
ListBucketsResponse listBucketsResponse = s3.listBuckets(listBucketsRequest);
listBucketsResponse.buckets().stream().forEach(x -> System.out.println(x.name()));
```

See the [complete example](#) on GitHub.

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 \(p. 60\)](#)
- [Delete an empty bucket \(p. 61\)](#)

Delete objects in a bucket

Build a [ListObjectsV2Request](#) and use the S3Client's `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.core.waiters.WaiterResponse;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.s3.S3Client;
import software.amazon.awssdk.services.s3.model.*;
import software.amazon.awssdk.services.s3.waiters.S3Waiter;
```

Code

First create an S3Client.

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

```
public static void listAllObjects(S3Client s3, String bucket) {

    try {
        // To delete a bucket, all the objects in the bucket must be deleted first
        ListObjectsV2Request listObjectsV2Request =
        ListObjectsV2Request.builder().bucket(bucket).build();
        ListObjectsV2Response listObjectsV2Response;

        do {
            listObjectsV2Response = s3.listObjectsV2(listObjectsV2Request);
            for (S3Object s3Object : listObjectsV2Response.contents()) {
                s3.deleteObject(DeleteObjectRequest.builder()
                    .bucket(bucket)
                    .key(s3Object.key())
                    .build());
            }

            listObjectsV2Request = ListObjectsV2Request.builder().bucket(bucket)
                .continuationToken(listObjectsV2Response.nextContinuationToken())
                .build();

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

See the [complete example](#) on GitHub.

Delete an empty bucket

Build a [DeleteBucketRequest](#) with a bucket name and pass it to the S3Client's deleteBucket method.

Imports

```
import software.amazon.awssdk.core.waiters.WaiterResponse;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.s3.S3Client;
import software.amazon.awssdk.services.s3.model.*;
import software.amazon.awssdk.services.s3.waiters.S3Waiter;
```

Code

First create an S3Client.

```
public static void listAllObjects(S3Client s3, String bucket) {

    try {
        // To delete a bucket, all the objects in the bucket must be deleted first
        ListObjectsV2Request listObjectsV2Request =
        ListObjectsV2Request.builder().bucket(bucket).build();
        ListObjectsV2Response listObjectsV2Response;

        do {
            listObjectsV2Response = s3.listObjectsV2(listObjectsV2Request);
            for (S3Object s3Object : listObjectsV2Response.contents()) {
                s3.deleteObject(DeleteObjectRequest.builder()
                    .bucket(bucket)
                    .key(s3Object.key())
                    .build());
            }

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

```
        .continuationToken(listObjectsV2Response.nextContinuationToken())  
        .build();  
  
    } while(listObjectsV2Response.isTruncated());
```

Delete all objects in the bucket.

```
public static void listAllObjects(S3Client s3, String bucket) {  
  
    try {  
        // To delete a bucket, all the objects in the bucket must be deleted first  
        ListObjectsV2Request listObjectsV2Request =  
        ListObjectsV2Request.builder().bucket(bucket).build();  
        ListObjectsV2Response listObjectsV2Response;  
  
        do {  
            listObjectsV2Response = s3.listObjectsV2(listObjectsV2Request);  
            for (S3Object s3Object : listObjectsV2Response.contents()) {  
                s3.deleteObject(DeleteObjectRequest.builder()  
                    .bucket(bucket)  
                    .key(s3Object.key())  
                    .build());  
            }  
  
            listObjectsV2Request = ListObjectsV2Request.builder().bucket(bucket)  
                .continuationToken(listObjectsV2Response.nextContinuationToken())  
                .build();  
  
        } while(listObjectsV2Response.isTruncated());  
    }  
}
```

See the [complete example](#) on GitHub.

Work with Amazon S3 objects

An Amazon S3 object represents a file or collection of data. Every object must be contained in a [bucket](#) (p. 58).

Note

Best Practice

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.

Note

These code snippets assume that you understand the material in basics, and have configured default AWS credentials using the information in [Set up AWS credentials and region for development](#) (p. 11).

Topics

- [Upload an object](#) (p. 63)
- [Upload objects in multiple parts](#) (p. 63)
- [Download an object](#) (p. 65)
- [Delete an object](#) (p. 66)
- [Copy an object](#) (p. 66)
- [List objects](#) (p. 67)

Upload an object

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

Imports

```
import java.io.IOException;
import java.nio.ByteBuffer;
import java.util.Random;
import software.amazon.awssdk.core.waiters.WaiterResponse;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.s3.S3Client;
import software.amazon.awssdk.services.s3.paginators.ListObjectsV2Iterable;
import software.amazon.awssdk.core.sync.RequestBody;
import software.amazon.awssdk.services.s3.model.S3Exception;
import software.amazon.awssdk.services.s3.model.PutObjectRequest;
import software.amazon.awssdk.services.s3.model.ListObjectsV2Request;
import software.amazon.awssdk.services.s3.model.ListObjectsV2Response;
import software.amazon.awssdk.services.s3.model.S3Object;
import software.amazon.awssdk.services.s3.model.GetObjectRequest;
import software.amazon.awssdk.services.s3.model.DeleteObjectRequest;
import software.amazon.awssdk.services.s3.model.DeleteBucketRequest;
import software.amazon.awssdk.services.s3.model.CreateMultipartUploadRequest;
import software.amazon.awssdk.services.s3.model.CreateMultipartUploadResponse;
import software.amazon.awssdk.services.s3.model.CompletedMultipartUpload;
import software.amazon.awssdk.services.s3.model.CreateBucketRequest;
import software.amazon.awssdk.services.s3.model.CompletedPart;
import software.amazon.awssdk.services.s3.model.CreateBucketConfiguration;
import software.amazon.awssdk.services.s3.model.UploadPartRequest;
import software.amazon.awssdk.services.s3.model.CompleteMultipartUploadRequest;
import software.amazon.awssdk.services.s3.waiters.S3Waiter;
import software.amazon.awssdk.services.s3.model.HeadBucketRequest;
import software.amazon.awssdk.services.s3.model.HeadBucketResponse;
```

Code

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

createBucket(s3, bucketName, region);

PutObjectRequest objectRequest = PutObjectRequest.builder()
    .bucket(bucketName)
    .key(key)
    .build();

s3.putObject(objectRequest, RequestBody.fromByteBuffer(getRandomByteBuffer(10_000)));
```

See the [complete example](#) on GitHub.

Upload objects in multiple parts

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

Imports

```
import java.io.IOException;
import java.nio.ByteBuffer;
import java.util.Random;
import software.amazon.awssdk.core.waiters.WaiterResponse;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.s3.S3Client;
import software.amazon.awssdk.services.s3.paginators.ListObjectsV2Iterable;
import software.amazon.awssdk.core.sync.RequestBody;
import software.amazon.awssdk.services.s3.model.S3Exception;
import software.amazon.awssdk.services.s3.model.PutObjectRequest;
import software.amazon.awssdk.services.s3.model.ListObjectsV2Request;
import software.amazon.awssdk.services.s3.model.ListObjectsV2Response;
import software.amazon.awssdk.services.s3.model.S3Object;
import software.amazon.awssdk.services.s3.model.GetObjectRequest;
import software.amazon.awssdk.services.s3.model.DeleteObjectRequest;
import software.amazon.awssdk.services.s3.model.DeleteBucketRequest;
import software.amazon.awssdk.services.s3.model.CreateMultipartUploadRequest;
import software.amazon.awssdk.services.s3.model.CreateMultipartUploadResponse;
import software.amazon.awssdk.services.s3.model.CompletedMultipartUpload;
import software.amazon.awssdk.services.s3.model.CreateBucketRequest;
import software.amazon.awssdk.services.s3.model.CompletedPart;
import software.amazon.awssdk.services.s3.model.CreateBucketConfiguration;
import software.amazon.awssdk.services.s3.model.UploadPartRequest;
import software.amazon.awssdk.services.s3.model.CompleteMultipartUploadRequest;
import software.amazon.awssdk.services.s3.waiters.S3Waiter;
import software.amazon.awssdk.services.s3.model.HeadBucketRequest;
import software.amazon.awssdk.services.s3.model.HeadBucketResponse;
```

Code

```
// First create a multipart upload and get the 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.fromByteBuffer(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.fromByteBuffer(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](#) on GitHub.

Download an object

Build a [GetObjectRequest](#) and supply a bucket name and key name. Use the `S3Client`'s `getObject` method, passing it the `GetObjectRequest` object and a `ResponseTransformer` object. The `ResponseTransformer` 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.util.Random;
import software.amazon.awssdk.core.waiters.WaiterResponse;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.s3.S3Client;
import software.amazon.awssdk.services.s3.paginators.ListObjectsV2Iterable;
import software.amazon.awssdk.core.sync.RequestBody;
import software.amazon.awssdk.services.s3.model.S3Exception;
import software.amazon.awssdk.services.s3.model.PutObjectRequest;
import software.amazon.awssdk.services.s3.model.ListObjectsV2Request;
import software.amazon.awssdk.services.s3.model.ListObjectsV2Response;
import software.amazon.awssdk.services.s3.model.S3Object;
import software.amazon.awssdk.services.s3.model.GetObjectRequest;
import software.amazon.awssdk.services.s3.model.DeleteObjectRequest;
import software.amazon.awssdk.services.s3.model.DeleteBucketRequest;
import software.amazon.awssdk.services.s3.model.CreateMultipartUploadRequest;
import software.amazon.awssdk.services.s3.model.CreateMultipartUploadResponse;
import software.amazon.awssdk.services.s3.model.CompletedMultipartUpload;
import software.amazon.awssdk.services.s3.model.CreateBucketRequest;
import software.amazon.awssdk.services.s3.model.CompletedPart;
import software.amazon.awssdk.services.s3.model.CreateBucketConfiguration;
import software.amazon.awssdk.services.s3.model.UploadPartRequest;
import software.amazon.awssdk.services.s3.model.CompleteMultipartUploadRequest;
import software.amazon.awssdk.services.s3.waiters.S3Waiter;
import software.amazon.awssdk.services.s3.model.HeadBucketRequest;
import software.amazon.awssdk.services.s3.model.HeadBucketResponse;
```

Code

```
GetObjectRequest getObjectRequest = GetObjectRequest.builder()
    .bucket(bucketName)
    .key(key)
    .build();
```

```
s3.getObject(getObjectRequest);
```

See the [complete example](#) on GitHub.

Delete an object

Build a [DeleteObjectRequest](#) and supply a bucket name and key name. Use the `S3Client`'s `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 java.io.IOException;
import java.nio.ByteBuffer;
import java.util.Random;
import software.amazon.awssdk.core.waiters.WaiterResponse;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.s3.S3Client;
import software.amazon.awssdk.services.s3.paginators.ListObjectsV2Iterable;
import software.amazon.awssdk.core.sync.RequestBody;
import software.amazon.awssdk.services.s3.model.S3Exception;
import software.amazon.awssdk.services.s3.model.PutObjectRequest;
import software.amazon.awssdk.services.s3.model.ListObjectsV2Request;
import software.amazon.awssdk.services.s3.model.ListObjectsV2Response;
import software.amazon.awssdk.services.s3.model.S3Object;
import software.amazon.awssdk.services.s3.model.GetObjectRequest;
import software.amazon.awssdk.services.s3.model.DeleteObjectRequest;
import software.amazon.awssdk.services.s3.model.DeleteBucketRequest;
import software.amazon.awssdk.services.s3.model.CreateMultipartUploadRequest;
import software.amazon.awssdk.services.s3.model.CreateMultipartUploadResponse;
import software.amazon.awssdk.services.s3.model.CompletedMultipartUpload;
import software.amazon.awssdk.services.s3.model.CreateBucketRequest;
import software.amazon.awssdk.services.s3.model.CompletedPart;
import software.amazon.awssdk.services.s3.model.CreateBucketConfiguration;
import software.amazon.awssdk.services.s3.model.UploadPartRequest;
import software.amazon.awssdk.services.s3.model.CompleteMultipartUploadRequest;
import software.amazon.awssdk.services.s3.waiters.S3Waiter;
import software.amazon.awssdk.services.s3.model.HeadBucketRequest;
import software.amazon.awssdk.services.s3.model.HeadBucketResponse;
```

Code

```
DeleteObjectRequest deleteObjectRequest = DeleteObjectRequest.builder()
    .bucket(bucketName)
    .key(key)
    .build();

s3.deleteObject(deleteObjectRequest);
```

See the [complete example](#) on GitHub.

Copy an object

Build a [CopyObjectRequest](#) and supply a bucket name that the object is copied into, a URL encoded string value (see the `URLEncoder.encode` method), and the key name of the object. Use the `S3Client`'s `copyObject` method, and pass the [CopyObjectRequest](#) object. *The specified bucket and object key must exist, or the service will return an error.*

Imports


```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.s3.S3Client;
import software.amazon.awssdk.services.s3.model.CopyObjectRequest;
import software.amazon.awssdk.services.s3.model.CopyObjectResponse;
import software.amazon.awssdk.services.s3.model.S3Exception;
import java.io.UnsupportedEncodingException;
import java.net.URLEncoder;
import java.nio.charset.StandardCharsets;
```

Code

```
public static String copyBucketObject (S3Client s3, String fromBucket, String objectKey,
String toBucket) {

    String encodedUrl = null;
    try {
        encodedUrl = URLEncoder.encode(fromBucket + "/" + objectKey,
StandardCharsets.UTF_8.toString());
    } catch (UnsupportedEncodingException e) {
        System.out.println("URL could not be encoded: " + e.getMessage());
    }
    CopyObjectRequest copyReq = CopyObjectRequest.builder()
        .copySource(encodedUrl)
        .destinationBucket(toBucket)
        .destinationKey(objectKey)
        .build();

    try {
        CopyObjectResponse copyRes = s3.copyObject(copyReq);
        return copyRes.copyObjectResult().toString();
    } catch (S3Exception e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
    return "";
}
```

See the [complete example](#) on GitHub.

List objects

Build a [ListObjectsRequest](#) and supply the bucket name. Then invoke the `S3Client`'s `listObjects` method and pass the [ListObjectsRequest](#) object. This method returns a [ListObjectsResponse](#) that contains all of the objects in the bucket. You can invoke this object's `contents` method to get a list of objects. You can iterate through this list to display the objects, as shown in the following code example.

Imports

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.s3.S3Client;
import software.amazon.awssdk.services.s3.model.ListObjectsRequest;
import software.amazon.awssdk.services.s3.model.ListObjectsResponse;
import software.amazon.awssdk.services.s3.model.S3Exception;
import software.amazon.awssdk.services.s3.model.S3Object;
import java.util.List;
import java.util.ListIterator;
```

Code

```
public static void listBucketObjects(S3Client s3, String bucketName ) {
```

```
try {
    ListObjectsRequest listObjects = ListObjectsRequest
        .builder()
        .bucket(bucketName)
        .build();

    ListObjectsResponse res = s3.listObjects(listObjects);
    List<S3Object> objects = res.contents();

    for (ListIterator iterVals = objects.listIterator(); iterVals.hasNext(); ) {
        S3Object myValue = (S3Object) iterVals.next();
        System.out.print("\n The name of the key is " + myValue.key());
        System.out.print("\n The object is " + calKb(myValue.size()) + " KBs");
        System.out.print("\n The owner is " + myValue.owner());
    }

} catch (S3Exception e) {
    System.err.println(e.awsErrorDetails().errorMessage());
    System.exit(1);
}

//convert bytes to kbs
private static long calKb(Long val) {
    return val/1024;
}
```

See the [complete example](#) on GitHub.

Working with Amazon S3 Presigned URLs

You can use a [S3Presigner](#) object to sign an `AmazonS3SdkRequest` so that it's executed without requiring authentication on the part of the caller. For example, assume Alice has access to an S3 object, and she wants to temporarily share access to that object with Bob. Alice can generate a pre-signed [GetObjectRequest](#) object to secure share with Bob so that he can download the object without requiring access to Alice's credentials.

Topics

- [Generate a Presigned URL and Upload an Object \(p. 68\)](#)
- [Get a Presigned Object \(p. 69\)](#)

Generate a Presigned URL and Upload an Object

Build a [S3Presigner](#) object that represents the client object. Next create a [PresignedPutObjectRequest](#) object that can be executed at a later time without requiring additional signing or authentication. When you create this object, you can specify the bucket name and the key name. In addition, you can also specify the time in minutes that the bucket can be accessed without using credentials by invoking the `signatureDuration` method (as shown in the following code example).

You can use the [PresignedPutObjectRequest](#) object to obtain the URL by invoking its `url` method.

Imports

```
import java.io.IOException;
import java.io.OutputStreamWriter;
import java.net.HttpURLConnection;
import java.net.URL;
import java.time.Duration;
import software.amazon.awssdk.regions.Region;
```

```
import software.amazon.awssdk.services.s3.model.PutObjectRequest;
import software.amazon.awssdk.services.s3.model.S3Exception;
import software.amazon.awssdk.services.s3.presigner.model.PresignedPutObjectRequest;
import software.amazon.awssdk.services.s3.presigner.S3Presigner;
import software.amazon.awssdk.services.s3.presigner.model.PutObjectPresignRequest;
```

Code

The following Java code example uploads content to a presigned S3 bucket.

```
public static void signBucket(S3Presigner presigner, String bucketName, String keyName)
{
    try {
        PutObjectRequest objectRequest = PutObjectRequest.builder()
            .bucket(bucketName)
            .key(keyName)
            .contentType("text/plain")
            .build();

        PutObjectPresignRequest presignRequest = PutObjectPresignRequest.builder()
            .signatureDuration(Duration.ofMinutes(10))
            .putObjectRequest(objectRequest)
            .build();

        PresignedPutObjectRequest presignedRequest =
presigner.presignPutObject(presignRequest);

        System.out.println("Presigned URL to upload a file to: " +
            presignedRequest.url());
        System.out.println("Which HTTP method needs to be used when uploading a file: "
+
            presignedRequest.httpRequest().method());

        // Upload content to the Amazon S3 bucket by using this URL
        URL url = presignedRequest.url();

        // Create the connection and use it to upload the new object by using the
presigned URL
        HttpURLConnection connection = (HttpURLConnection) url.openConnection();
        connection.setDoOutput(true);
        connection.setRequestProperty("Content-Type", "text/plain");
        connection.setRequestMethod("PUT");
        OutputStreamWriter out = new OutputStreamWriter(connection.getOutputStream());
        out.write("This text was uploaded as an object by using a presigned URL.");
        out.close();

        connection.getResponseCode();
        System.out.println("HTTP response code is " + connection.getResponseCode());

    } catch (S3Exception e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

See the [complete example](#) on GitHub.

Get a Presigned Object

Build a [S3Presigner](#) object that represents the client object. Next, create a [GetObjectRequest](#) object and specify the bucket name and key name. In addition, create a [GetObjectPresignRequest](#) object that can be executed at a later time without requiring additional signing or authentication. When you create this

object, you can specify the time in minutes that the bucket can be accessed without using credentials by invoking the `signatureDuration` method (as shown in the following code example).

Invoke the `presignGetObject` method that belongs to the [S3Presigner](#) object to create a [PresignedGetObjectRequest](#) object. You can invoke this object's `url` method to obtain the URL to use. Once you have the URL, you can use standard HTTP Java logic to read the contents of the bucket, as shown in the following Java code example.

Imports

```
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.net.HttpURLConnection;
import java.time.Duration;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.s3.model.GetObjectRequest;
import software.amazon.awssdk.services.s3.model.S3Exception;
import software.amazon.awssdk.services.s3.presigner.model.GetObjectPresignRequest;
import software.amazon.awssdk.services.s3.presigner.model.PresignedGetObjectRequest;
import software.amazon.awssdk.services.s3.presigner.S3Presigner;
import software.amazon.awssdk.utils.IoUtils;
```

Code

The following Java code example reads content from a presigned S3 bucket.

```
public static void getPresignedUrl(S3Presigner presigner, String bucketName, String
keyName ) {
    try {
        GetObjectRequest getObjectRequest =
            GetObjectRequest.builder()
                .bucket(bucketName)
                .key(keyName)
                .build();

        GetObjectPresignRequest getObjectPresignRequest =
            GetObjectPresignRequest.builder()
                .signatureDuration(Duration.ofMinutes(10))
                .getObjectRequest(getObjectRequest)
                .build();

        // Generate the presigned request
        PresignedGetObjectRequest presignedGetObjectRequest =
            presigner.presignGetObject(getObjectPresignRequest);

        // Log the presigned URL
        System.out.println("Presigned URL: " + presignedGetObjectRequest.url());

        HttpURLConnection connection = (HttpURLConnection)
presignedGetObjectRequest.url().openConnection();
        presignedGetObjectRequest.httpRequest().headers().forEach((header, values) -> {
            values.forEach(value -> {
                connection.addRequestProperty(header, value);
            });
        });

        // Send any request payload that the service needs (not needed when
isBrowserExecutable is true)
        if (presignedGetObjectRequest.signedPayload().isPresent()) {
            connection.setDoOutput(true);
            try (InputStream signedPayload =
presignedGetObjectRequest.signedPayload().get().asInputStream();
```

```
        OutputStream httpOutputStream = connection.getOutputStream()) {
            IoUtils.copy(signedPayload, httpOutputStream);
        }
    }

    // Download the result of executing the request
    try (InputStream content = connection.getInputStream()) {
        System.out.println("Service returned response: ");
        IoUtils.copy(content, System.out);
    }

} catch (S3Exception e) {
    e.printStackTrace();
} catch (IOException e) {
    e.printStackTrace();
}
```

See the [complete example](#) on GitHub.

Working with DynamoDB

This section provides examples that show you how to program [DynamoDB](#) by using the AWS SDK for Java 2.0.

The following 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

- [Work with tables in DynamoDB \(p. 71\)](#)
- [Work with items in DynamoDB \(p. 77\)](#)
- [Map items in DynamoDB tables \(p. 82\)](#)

Work with tables in DynamoDB

Tables are the containers for all items in a DynamoDB database. Before you can add or remove data from DynamoDB, you must create a table.

For each table, you must define:

- A table *name* that is unique for your account and region.
- A *primary key* for which every value must be unique; no two items in your table can have the same primary key value.

A primary key can be *simple*, consisting of a single partition (HASH) key, or *composite*, consisting of a partition and a sort (RANGE) key.

Each key value has an associated *data type*, enumerated by the [ScalarAttributeType](#) class. The key value can be binary (B), numeric (N), or a string (S). For more information, see [Naming Rules and Data Types](#) in the Amazon DynamoDB Developer Guide.

- *Provisioned throughput* are values that define the number of reserved read/write capacity units for the table.

Note

[Amazon DynamoDB pricing](#) is based on the provisioned throughput values that you set on your tables, so reserve only as much capacity as you think you'll need for your table.

Provisioned throughput for a table can be modified at any time, so you can adjust capacity as your needs change.

Create a table

Use the `DynamoDbClient`'s `createTable` method to create a new DynamoDB table. You need to construct table attributes and a table schema, both of which are used to identify the primary key of your table. You must also supply initial provisioned throughput values and a table name.

Note

If a table with the name you chose already exists, an [DynamoDbException](#) is thrown.

Imports

```
import software.amazon.awssdk.core.waiters.WaiterResponse;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.model.CreateTableRequest;
import software.amazon.awssdk.services.dynamodb.model.AttributeDefinition;
import software.amazon.awssdk.services.dynamodb.model.ScalarAttributeType;
import software.amazon.awssdk.services.dynamodb.model.ScalarAttributeType;
import software.amazon.awssdk.services.dynamodb.model.KeySchemaElement;
import software.amazon.awssdk.services.dynamodb.model.ProvisionedThroughput;
import software.amazon.awssdk.services.dynamodb.model.KeyType;
import software.amazon.awssdk.services.dynamodb.model.CreateTableResponse;
import software.amazon.awssdk.services.dynamodb.model.DescribeTableRequest;
import software.amazon.awssdk.services.dynamodb.model.DescribeTableResponse;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.waiters.DynamoDbWaiter;
```

Create a table with a simple primary key

This code creates a table with a simple primary key ("Name").

Code

```
public static String createTable(DynamoDbClient ddb, String tableName, String key) {

    DynamoDbWaiter dbWaiter = ddb.waiter();
    CreateTableRequest request = CreateTableRequest.builder()
        .attributeDefinitions(AttributeDefinition.builder()
            .attributeName(key)
            .attributeType(ScalarAttributeType.S)
            .build())
        .keySchema(KeySchemaElement.builder()
            .attributeName(key)
            .keyType(KeyType.HASH)
            .build())
        .provisionedThroughput(ProvisionedThroughput.builder()
            .readCapacityUnits(new Long(10))
            .writeCapacityUnits(new Long(10))
            .build())
        .tableName(tableName)
        .build();

    String newTable = "";
    try {
        CreateTableResponse response = ddb.createTable(request);
        DescribeTableRequest tableRequest = DescribeTableRequest.builder()
            .tableName(tableName)
            .build();
```

```
// Wait until the Amazon DynamoDB table is created
WaiterResponse<DescribeTableResponse> waiterResponse =
dbWaiter.waitUntilTableExists(tableRequest);
waiterResponse.matched().response().ifPresent(System.out::println);

newTable = response.tableDescription().tableName();
return newTable;

} catch (DynamoDbException e) {
    System.err.println(e.getMessage());
    System.exit(1);
}
return "";
}
```

See the [complete example](#) on GitHub.

Create a table with a composite primary key

Add another [AttributeDefinition](#) and [KeySchemaElement](#) to [CreateTableRequest](#).

Imports

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.AttributeDefinition;
import software.amazon.awssdk.services.dynamodb.model.CreateTableRequest;
import software.amazon.awssdk.services.dynamodb.model.CreateTableResponse;
import software.amazon.awssdk.services.dynamodb.model.KeySchemaElement;
import software.amazon.awssdk.services.dynamodb.model.KeyType;
import software.amazon.awssdk.services.dynamodb.model.ProvisionedThroughput;
import software.amazon.awssdk.services.dynamodb.model.ScalarAttributeType;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
```

Code

```
public static String createTableComKey(DynamoDbClient ddb, String tableName) {
    CreateTableRequest request = CreateTableRequest.builder()
        .attributeDefinitions(
            AttributeDefinition.builder()
                .attributeName("Language")
                .attributeType(ScalarAttributeType.S)
                .build(),
            AttributeDefinition.builder()
                .attributeName("Greeting")
                .attributeType(ScalarAttributeType.S)
                .build()
        )
        .keySchema(
            KeySchemaElement.builder()
                .attributeName("Language")
                .keyType(KeyType.HASH)
                .build(),
            KeySchemaElement.builder()
                .attributeName("Greeting")
                .keyType(KeyType.RANGE)
                .build()
        )
        .provisionedThroughput(
            ProvisionedThroughput.builder()
                .readCapacityUnits(new Long(10))
                .writeCapacityUnits(new Long(10)).build()
        )
        .tableName(tableName)
        .build();
}
```

```
String tableId = "";

try {
    CreateTableResponse result = ddb.createTable(request);
    tableId = result.tableDescription().tableId();
    return tableId;
} catch (DynamoDbException e) {
    System.err.println(e.getMessage());
    System.exit(1);
}
```

See the [complete example](#) on GitHub.

List tables

You can list the tables in a particular region by calling the `DynamoDbClient`'s `listTables` method.

Note

If the named table doesn't exist for your account and region, a [ResourceNotFoundException](#) is thrown.

Imports

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.ListTablesResponse;
import software.amazon.awssdk.services.dynamodb.model.ListTablesRequest;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import java.util.List;
```

Code

```
public static void listAllTables(DynamoDbClient ddb){

    boolean moreTables = true;
    String lastName = null;

    while(moreTables) {
        try {
            ListTablesResponse response = null;
            if (lastName == null) {
                ListTablesRequest request = ListTablesRequest.builder().build();
                response = ddb.listTables(request);
            } else {
                ListTablesRequest request = ListTablesRequest.builder()
                    .exclusiveStartTableName(lastName).build();
                response = ddb.listTables(request);
            }

            List<String> tableNames = response.tableNames();

            if (tableNames.size() > 0) {
                for (String curName : tableNames) {
                    System.out.format("%s\n", curName);
                }
            } else {
                System.out.println("No tables found!");
                System.exit(0);
            }

            lastName = response.lastEvaluatedTableName();
            if (lastName == null) {
```



```
        moreTables = false;
    }
} catch (DynamoDbException e) {
    System.err.println(e.getMessage());
    System.exit(1);
}
}
```

By default, up to 100 tables are returned per call—use `lastEvaluatedTableName` on the returned [ListTablesResponse](#) object to get the last table that was evaluated. You can use this value to start the listing after the last returned value of the previous listing.

See the [complete example](#) on GitHub.

Describe (get information about) a table

Call the `DynamoDbClient`'s `describeTable` method.

Note

If the named table doesn't exist for your account and region, a [ResourceNotFoundException](#) is thrown.

Imports

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeDefinition;
import software.amazon.awssdk.services.dynamodb.model.DescribeTableRequest;
import software.amazon.awssdk.services.dynamodb.model.ProvisionedThroughputDescription;
import software.amazon.awssdk.services.dynamodb.model.TableDescription;
import java.util.List;
```

Code

```
public static void describeDynamoDBTable(DynamoDbClient ddb, String tableName) {

    DescribeTableRequest request = DescribeTableRequest.builder()
        .tableName(tableName)
        .build();

    try {
        TableDescription tableInfo =
            ddb.describeTable(request).table();

        if (tableInfo != null) {
            System.out.format("Table name   : %s\n",
                tableInfo.tableName());
            System.out.format("Table ARN   : %s\n",
                tableInfo.tableArn());
            System.out.format("Status      : %s\n",
                tableInfo.tableStatus());
            System.out.format("Item count   : %d\n",
                tableInfo.itemCount().longValue());
            System.out.format("Size (bytes): %d\n",
                tableInfo.tableSizeBytes().longValue());

            ProvisionedThroughputDescription throughputInfo =
                tableInfo.provisionedThroughput();
            System.out.println("Throughput");
            System.out.format("  Read Capacity : %d\n",
                throughputInfo.readCapacityUnits().longValue());
        }
    }
}
```

```
        System.out.format("  Write Capacity: %d\n",
                           throughputInfo.writeCapacityUnits().longValue());

        List<AttributeDefinition> attributes =
            tableInfo.attributeDefinitions();
        System.out.println("Attributes");

        for (AttributeDefinition a : attributes) {
            System.out.format("  %s (%s)\n",
                              a.attributeName(), a.attributeType());
        }
    }
} catch (DynamoDbException e) {
    System.err.println(e.getMessage());
    System.exit(1);
}
```

See the [complete example](#) on GitHub.

Modify (update) a table

You can modify your table's provisioned throughput values at any time by calling the `DynamoDbClient`'s `updateTable` method.

Note

If the named table doesn't exist for your account and region, a [ResourceNotFoundException](#) is thrown.

Imports

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.model.ProvisionedThroughput;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.UpdateTableRequest;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
```

Code

```
public static void updateDynamoDBTable(DynamoDbClient ddb,
                                       String tableName,
                                       Long readCapacity,
                                       Long writeCapacity) {

    System.out.format(
        "Updating %s with new provisioned throughput values\n",
        tableName);
    System.out.format("Read capacity : %d\n", readCapacity);
    System.out.format("Write capacity : %d\n", writeCapacity);

    ProvisionedThroughput tableThroughput = ProvisionedThroughput.builder()
        .readCapacityUnits(readCapacity)
        .writeCapacityUnits(writeCapacity)
        .build();

    UpdateTableRequest request = UpdateTableRequest.builder()
        .provisionedThroughput(tableThroughput)
        .tableName(tableName)
        .build();

    try {
        ddb.updateTable(request);
    } catch (DynamoDbException e) {
```

```
        System.err.println(e.getMessage());  
        System.exit(1);  
    }
```

See the [complete example](#) on GitHub.

Delete a table

Call the `DynamoDbClient`'s `deleteTable` method and pass it the table's name.

Note

If the named table doesn't exist for your account and region, a [ResourceNotFoundException](#) is thrown.

Imports

```
import software.amazon.awssdk.regions.Region;  
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;  
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;  
import software.amazon.awssdk.services.dynamodb.model.DeleteTableRequest;
```

Code

```
public static void deleteDynamoDBTable(DynamoDbClient ddb, String tableName) {  
  
    DeleteTableRequest request = DeleteTableRequest.builder()  
        .tableName(tableName)  
        .build();  
  
    try {  
        ddb.deleteTable(request);  
    } catch (DynamoDbException e) {  
        System.err.println(e.getMessage());  
        System.exit(1);  
    }  
}
```

See the [complete example](#) on GitHub.

More information

- [Guidelines for Working with Tables](#) in the Amazon DynamoDB Developer Guide
- [Working with Tables in DynamoDB](#) in the Amazon DynamoDB Developer Guide

Work with items in DynamoDB

In DynamoDB, an item is a collection of *attributes*, each of which has a *name* and a *value*. An attribute value can be a scalar, set, or document type. For more information, see [Naming Rules and Data Types](#) in the Amazon DynamoDB Developer Guide.

Retrieve (get) an item from a table

Call the `DynamoDbClient`'s `getItem` method and pass it a [GetItemRequest](#) object with the table name and primary key value of the item you want. It returns a [GetItemResponse](#) object with all of the attributes for that item. You can specify one or more [projection expressions](#) in the `GetItemRequest` to retrieve specific attributes.

You can use the returned `GetItemResponse` object's `item()` method to retrieve a [Map](#) of key (String) and value ([AttributeValue](#)) pairs that are associated with the item.

Imports

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.GetItemRequest;
import java.util.HashMap;
import java.util.Map;
import java.util.Set;
```

Code

```
public static void getDynamoDBItem(DynamoDbClient ddb,String tableName,String key,String
keyVal ) {

    HashMap<String,AttributeValue> keyToGet = new HashMap<String,AttributeValue>();

    keyToGet.put(key, AttributeValue.builder()
        .s(keyVal).build());

    GetItemRequest request = GetItemRequest.builder()
        .key(keyToGet)
        .tableName(tableName)
        .build();

    try {
        Map<String,AttributeValue> returnedItem = ddb.getItem(request).item();

        if (returnedItem != null) {
            Set<String> keys = returnedItem.keySet();
            System.out.println("Amazon DynamoDB table attributes: \n");

            for (String key1 : keys) {
                System.out.format("%s: %s\n", key1, returnedItem.get(key1).toString());
            }
        } else {
            System.out.format("No item found with the key %s!\n", key);
        }
    } catch (DynamoDbException e) {
        System.err.println(e.getMessage());
        System.exit(1);
    }
}
```

See the [complete example](#) on GitHub.

Retrieve (get) an item from a table using the asynchronous client

Invoke the `getItem` method of the `DynamoDbAsyncClient` and pass it a [GetItemRequest](#) object with the table name and primary key value of the item you want.

You can return a [Collection](#) instance with all of the attributes for that item (refer to the following example).

Imports

```
import software.amazon.awssdk.regions.Region;
```

```
import software.amazon.awssdk.services.dynamodb.model.GetItemRequest;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.DynamoDbAsyncClient;
import java.util.HashMap;
import java.util.Map;
import java.util.Set;
import java.util.stream.Collectors;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
```

Code

```
ic static void getItem(DynamoDbAsyncClient client, String tableName, String key, String
    keyVal) {

    HashMap<String, AttributeValue> keyToGet =
        new HashMap<String, AttributeValue>();

    keyToGet.put(key, AttributeValue.builder()
        .s(keyVal).build());

    try {

        // Create a GetItemRequest instance
        GetItemRequest request = GetItemRequest.builder()
            .key(keyToGet)
            .tableName(tableName)
            .build();

        // Invoke the DynamoDbAsyncClient object's getItem
        java.util.Collection<software.amazon.awssdk.services.dynamodb.model.AttributeValue>
            returnedItem = client.getItem(request).join().item().values();

        // Convert Set to Map
        Map<String, AttributeValue> map =
            returnedItem.stream().collect(Collectors.toMap(AttributeValue::s, s->s));
        Set<String> keys = map.keySet();
        for (String sinKey : keys) {
            System.out.format("%s: %s\n", sinKey, map.get(sinKey).toString());
        }

    } catch (DynamoDbException e) {
        System.err.println(e.getMessage());
        System.exit(1);
    }
}
```

See the [complete example](#) on GitHub.

Add a new item to a table

Create a [Map](#) of key-value pairs that represent the item's attributes. These must include values for the table's primary key fields. If the item identified by the primary key already exists, its fields are *updated* by the request.

Note

If the named table doesn't exist for your account and region, a [ResourceNotFoundException](#) is thrown.

Imports

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
```

```
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.PutItemRequest;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;
import java.util.HashMap;
```

Code

```
public static void putItemInTable(DynamoDbClient ddb,
                                String tableName,
                                String key,
                                String keyVal,
                                String albumTitle,
                                String albumTitleValue,
                                String awards,
                                String awardVal,
                                String songTitle,
                                String songTitleVal){

    HashMap<String,AttributeValue> itemValues = new HashMap<String,AttributeValue>();

    // Add all content to the table
    itemValues.put(key, AttributeValue.builder().s(keyVal).build());
    itemValues.put(songTitle, AttributeValue.builder().s(songTitleVal).build());
    itemValues.put(albumTitle, AttributeValue.builder().s(albumTitleValue).build());
    itemValues.put(awards, AttributeValue.builder().s(awardVal).build());

    PutItemRequest request = PutItemRequest.builder()
        .tableName(tableName)
        .item(itemValues)
        .build();

    try {
        ddb.putItem(request);
        System.out.println(tableName + " was successfully updated");

    } catch (ResourceNotFoundException e) {
        System.err.format("Error: The Amazon DynamoDB table \"%s\" can't be found.\n",
            tableName);
        System.err.println("Be sure that it exists and that you've typed its name correctly!");
        System.exit(1);
    } catch (DynamoDbException e) {
        System.err.println(e.getMessage());
        System.exit(1);
    }
}
```

See the [complete example](#) on GitHub.

Update an existing item in a table

You can update an attribute for an item that already exists in a table by using the `DynamoDbClient`'s `updateItem` method, providing a table name, primary key value, and a map of fields to update.

Note

If the named table doesn't exist for your account and region, or if the item identified by the primary key you passed in doesn't exist, a [ResourceNotFoundException](#) is thrown.

Imports

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.AttributeAction;
```

```
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.AttributeValueUpdate;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemRequest;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import java.util.HashMap;
```

Code

```
public static void updateTableItem(DynamoDbClient ddb,
                                   String tableName,
                                   String key,
                                   String keyVal,
                                   String name,
                                   String updateVal){

    HashMap<String,AttributeValue> itemKey = new HashMap<String,AttributeValue>();

    itemKey.put(key, AttributeValue.builder().s(keyVal).build());

    HashMap<String,AttributeValueUpdate> updatedValues =
        new HashMap<String,AttributeValueUpdate>();

    // Update the column specified by name with updatedVal
    updatedValues.put(name, AttributeValueUpdate.builder()
        .value(AttributeValue.builder().s(updateVal).build())
        .action(AttributeAction.PUT)
        .build());

    UpdateItemRequest request = UpdateItemRequest.builder()
        .tableName(tableName)
        .key(itemKey)
        .attributeUpdates(updatedValues)
        .build();

    try {
        ddb.updateItem(request);
    } catch (ResourceNotFoundException e) {
        System.err.println(e.getMessage());
        System.exit(1);
    } catch (DynamoDbException e) {
        System.err.println(e.getMessage());
        System.exit(1);
    }
}
```

See the [complete example](#) on GitHub.

Delete an existing item in a table

You can delete an item that exists in a table by using the `DynamoDbClient`'s `deleteItem` method and providing a table name as well as the primary key value.

Note

If the named table doesn't exist for your account and region, or if the item identified by the primary key you passed in doesn't exist, a [ResourceNotFoundException](#) is thrown.

Imports

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DeleteItemRequest;
```

```
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import java.util.HashMap;
```

Code

```
public static void deleteDynamoDBItem(DynamoDbClient ddb, String tableName, String key,
String keyVal) {

    HashMap<String,AttributeValue> keyToGet =
        new HashMap<String,AttributeValue>();

    keyToGet.put(key, AttributeValue.builder()
        .s(keyVal)
        .build());

    DeleteItemRequest deleteReq = DeleteItemRequest.builder()
        .tableName(tableName)
        .key(keyToGet)
        .build();

    try {
        ddb.deleteItem(deleteReq);
    } catch (DynamoDbException e) {
        System.err.println(e.getMessage());
        System.exit(1);
    }
}
```

See the [complete example](#) on GitHub.

More information

- [Guidelines for Working with Items](#) in the Amazon DynamoDB Developer Guide
- [Working with Items in DynamoDB](#) in the Amazon DynamoDB Developer Guide

Map items in DynamoDB tables

The Amazon DynamoDB enhanced client is a high-level library that is part of the AWS SDK for Java version 2 (v2). It offers a straightforward way to map client-side classes to DynamoDB tables. You define the relationships between tables and their corresponding model classes in your code. Then you can intuitively perform various create, read, update, or delete (CRUD) operations on tables or items in DynamoDB.

The AWS SDK for Java v2 includes a set of annotations that you can use with a Java bean to quickly generate a [TableSchema](#) for mapping your classes to tables. Alternatively, if you declare each [TableSchema](#) explicitly, you don't need to include annotations in your classes.

To work with items in a DynamoDB table using the enhanced client, first create a [DynamoDbEnhancedClient](#) from an existing [DynamoDbClient](#) object.

```
Region region = Region.US_EAST_1;
DynamoDbClient ddb = DynamoDbClient.builder()
    .region(region)
    .build();

DynamoDbEnhancedClient enhancedClient = DynamoDbEnhancedClient.builder()
    .dynamoDbClient(ddb)
    .build();
```



```
createDynamoDBTable(enhancedClient);
```

Create a table using the enhanced client

To easily create a [TableSchema](#) using the enhanced client, start by creating a Java data class that includes a default public constructor and standardized names of getters and setters for each property in the class. Include a class-level annotation to indicate it is a `DynamoDbBean` and, at a minimum, include a `DynamoDbPartitionKey` annotation on the getter or setter for the primary key of the table record.

Once this data class has been defined, call `TableSchema`'s `fromBean()` with that data class to create the table schema.

See the code snippet below for an example of how to do this.

Imports

```
import software.amazon.awssdk.enhanced.dynamodb.DynamoDbEnhancedClient;
import software.amazon.awssdk.enhanced.dynamodb.DynamoDbTable;
import software.amazon.awssdk.enhanced.dynamodb.TableSchema;
import software.amazon.awssdk.enhanced.dynamodb.mapper.annotations.DynamoDbSortKey;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.enhanced.dynamodb.mapper.annotations.DynamoDbBean;
import software.amazon.awssdk.enhanced.dynamodb.mapper.annotations.DynamoDbPartitionKey;
import java.time.Instant;
import java.time.LocalDate;
import java.time.LocalDateTime;
import java.time.ZoneOffset;
```

Code

```
// Puts an item into a DynamoDB table
public static void putRecord(DynamoDbEnhancedClient enhancedClient) {

    try {
        DynamoDbTable<Customer> custTable = enhancedClient.table("Customer",
            TableSchema.fromBean(Customer.class));

        // Create an Instant
        LocalDate localDate = LocalDate.parse("2020-04-07");
        LocalDateTime localDateTime = localDate.atStartOfDay();
        Instant instant = localDateTime.toInstant(ZoneOffset.UTC);

        // Populate the Table
        Customer custRecord = new Customer();
        custRecord.setCustName("Susan Blue");
        custRecord.setId("id103");
        custRecord.setEmail("sblue@noserver.com");
        custRecord.setRegistrationDate(instant) ;

        // Put the customer data into a DynamoDB table
        custTable.putItem(custRecord);

    } catch (DynamoDbException e) {
        System.err.println(e.getMessage());
        System.exit(1);
    }
    System.out.println("done");
}
```

```
@DynamoDbBean
public static class Customer {

    private String id;
    private String name;
    private String email;
    private Instant regDate;

    @DynamoDbPartitionKey
    public String getId() {
        return this.id;
    };

    public void setId(String id) {

        this.id = id;
    }

    @DynamoDbSortKey
    public String getCustName() {
        return this.name;
    }

    public void setCustName(String name) {

        this.name = name;
    }

    public String getEmail() {
        return this.email;
    }

    public void setEmail(String email) {

        this.email = email;
    }

    public Instant getRegistrationDate() {
        return regDate;
    }
    public void setRegistrationDate(Instant registrationDate) {

        this.regDate = registrationDate;
    }
}
```

See the [complete example](#) on GitHub.

Retrieve (get) an item from a table

To get an item from a DynamoDB table, create a [DynamoDbTable](#) object and call `getItem()` with a [GetItemEnhancedRequest](#) object to get the actual item.

For example, the following code snippet demonstrates one way to use the enhanced client to get information from an item in a DynamoDB table.

Imports

```
import software.amazon.awssdk.enhanced.dynamodb.DynamoDbEnhancedClient;
import software.amazon.awssdk.enhanced.dynamodb.DynamoDbTable;
import software.amazon.awssdk.enhanced.dynamodb.Key;
import software.amazon.awssdk.enhanced.dynamodb.TableSchema;
```

```
import software.amazon.awssdk.enhanced.dynamodb.mapper.annotations.DynamoDbBean;
import software.amazon.awssdk.enhanced.dynamodb.mapper.annotations.DynamoDbPartitionKey;
import software.amazon.awssdk.enhanced.dynamodb.mapper.annotations.DynamoDbSortKey;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import java.time.Instant;
```

Code

```
public static String getItem(DynamoDbEnhancedClient enhancedClient) {
    try {
        //Create a DynamoDbTable object
        DynamoDbTable<Customer> mappedTable = enhancedClient.table("Customer",
            TableSchema.fromBean(Customer.class));

        //Create a KEY object
        Key key = Key.builder()
            .partitionValue("id110")
            .sortValue("Fred Pink")
            .build();

        // Get the item by using the key
        Customer result = mappedTable.getItem(r->r.key(key));
        return "The record id is "+result.getId();

    } catch (DynamoDbException e) {
        System.err.println(e.getMessage());
        System.exit(1);
    }
}
```

See the [complete example](#) on GitHub.

Batch create (put) and delete items

You can batch a series of put requests ([PutItemEnhancedRequest](#)) and delete requests ([DeleteItemEnhancedRequest](#)) to one or more tables, and then send all of the changes in a single request.

In the following code snippet, a [DynamoDbTable](#) object is created, two items are queued up to be added to the table, and then the items are written to the table in a single call. Include multiple entries of `addDeleteItem()` and `addPutItem()` ([part of WriteBatch.Builder](#)) in each batch, as needed. To queue up changes to a different table, add another instance of `WriteBatch.builder()` and provide a corresponding [DynamoDbTable](#) object in `mappedTableResource()`.

Imports

```
import software.amazon.awssdk.enhanced.dynamodb.DynamoDbEnhancedClient;
import software.amazon.awssdk.enhanced.dynamodb.DynamoDbTable;
import software.amazon.awssdk.enhanced.dynamodb.TableSchema;
import software.amazon.awssdk.enhanced.dynamodb.mapper.annotations.DynamoDbBean;
import software.amazon.awssdk.enhanced.dynamodb.mapper.annotations.DynamoDbPartitionKey;
import software.amazon.awssdk.enhanced.dynamodb.mapper.annotations.DynamoDbSortKey;
import software.amazon.awssdk.enhanced.dynamodb.model.BatchWriteItemEnhancedRequest;
import software.amazon.awssdk.enhanced.dynamodb.model.WriteBatch;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import java.time.Instant;
import java.time.LocalDate;
import java.time.LocalDateTime;
```

```
import java.time.ZoneOffset;
```

Code

```
public static void putBatchRecords(DynamoDbEnhancedClient enhancedClient) {
    try {

        DynamoDbTable<Customer> mappedTable = enhancedClient.table("Customer",
            TableSchema.fromBean(Customer.class));

        LocalDate localDate = LocalDate.parse("2020-04-07");
        LocalDateTime localDateTime = localDate.atStartOfDay();
        Instant instant = localDateTime.toInstant(ZoneOffset.UTC);

        Customer record2 = new Customer();
        record2.setCustName("Fred Pink");
        record2.setId("id110");
        record2.setEmail("fredp@noserver.com");
        record2.setRegistrationDate(instant) ;

        Customer record3 = new Customer();
        record3.setCustName("Susan Pink");
        record3.setId("id120");
        record3.setEmail("spink@noserver.com");
        record3.setRegistrationDate(instant) ;

        // Create a BatchWriteItemEnhancedRequest object
        BatchWriteItemEnhancedRequest batchWriteItemEnhancedRequest =
            BatchWriteItemEnhancedRequest.builder()
                .writeBatches(
                    WriteBatch.builder(Customer.class)
                        .mappedTableResource(mappedTable)
                        .addPutItem(r -> r.item(record2))
                        .addPutItem(r -> r.item(record3))
                        .build()
                )
                .build();

        // Add these two items to the table
        enhancedClient.batchWriteItem(batchWriteItemEnhancedRequest);
        System.out.println("done");

    } catch (DynamoDbException e) {
        System.err.println(e.getMessage());
        System.exit(1);
    }
}

@DynamoDbBean
public static class Customer {

    private String id;
    private String name;
    private String email;
    private Instant regDate;

    @DynamoDbPartitionKey
    public String getId() {
        return this.id;
    };

    public void setId(String id) {

        this.id = id;
    }
}
```

```
@DynamoDbSortKey
public String getCustName() {
    return this.name;
}

public void setCustName(String name) {
    this.name = name;
}

public String getEmail() {
    return this.email;
}

public void setEmail(String email) {
    this.email = email;
}

public Instant getRegistrationDate() {
    return regDate;
}
public void setRegistrationDate(Instant registrationDate) {

    this.regDate = registrationDate;
}
}
```

See the [complete example](#) on GitHub.

Use a filtered query to get items from a table

You can get items from a table based on filterable queries, and then perform operations (for example, return item values) on one or more of the items in the query results.

In the following code snippet, you build a filter by first defining the value or values you're searching for as an [AttributeValue](#) object. Then you put this into a `HashMap` and build an [Expression](#) from the `classname:HashMap`. Build a [QueryConditional](#) object to specify the primary key to match against in the query, and then execute the query on your [DynamoDbTable](#) object.

Note

The [QueryConditional](#) interface has several methods you can use to build your queries, including common conditional statements like greater than, less than, and in between.

Imports

```
import java.time.Instant;
import java.util.Map;
import java.util.Iterator;
import java.util.HashMap;
import software.amazon.awssdk.enhanced.dynamodb.DynamoDbEnhancedClient;
import software.amazon.awssdk.enhanced.dynamodb.DynamoDbTable;
import software.amazon.awssdk.enhanced.dynamodb.Expression;
import software.amazon.awssdk.enhanced.dynamodb.Key;
import software.amazon.awssdk.enhanced.dynamodb.TableSchema;
import software.amazon.awssdk.enhanced.dynamodb.mapper.annotations.DynamoDbBean;
import software.amazon.awssdk.enhanced.dynamodb.mapper.annotations.DynamoDbPartitionKey;
import software.amazon.awssdk.enhanced.dynamodb.mapper.annotations.DynamoDbSortKey;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.enhanced.dynamodb.model.QueryConditional;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
```

Code

```
public static void queryTableFilter(DynamoDbEnhancedClient enhancedClient) {  
    try{  
        DynamoDbTable<EnhancedQueryRecords.Customer> mappedTable =  
            enhancedClient.table("Customer",  
                TableSchema.fromBean(EnhancedQueryRecords.Customer.class));  
  
        AttributeValue att = AttributeValue.builder()  
            .s("sblue@noserver.com")  
            .build();  
  
        Map<String, AttributeValue> expressionValues = new HashMap<>();  
        expressionValues.put(":value", att);  
  
        Expression expression = Expression.builder()  
            .expression("email = :value")  
            .expressionValues(expressionValues)  
            .build();  
  
        // Create a QueryConditional object that is used in the query operation  
        QueryConditional queryConditional = QueryConditional  
            .keyEqualTo(Key.builder().partitionValue("id103")  
                .build());  
  
        // Get items in the Customer table and write out the ID value  
        Iterator<EnhancedQueryRecords.Customer> results = mappedTable.query(r ->  
            r.queryConditional(queryConditional).filterExpression(expression)).items().iterator();  
  
        while (results.hasNext()) {  
            EnhancedQueryRecords.Customer rec = results.next();  
            System.out.println("The record id is "+rec.getId());  
        }  
  
    } catch (DynamoDbException e) {  
        System.err.println(e.getMessage());  
        System.exit(1);  
    }  
    System.out.println("Done");  
}
```

See the [complete example](#) on GitHub.

Retrieve (get) all items from a table

When you want to get all of the records in a given DynamoDB table, use the `scan()` method of your [DynamoDbTable](#) object and the `items()` method to create a set of results against which you can execute various item operations. For example, the following code snippet prints out the ID value of each item in the **Record** table.

Imports

```
import java.time.Instant;  
import java.util.Iterator;  
import software.amazon.awssdk.enhanced.dynamodb.DynamoDbEnhancedClient;  
import software.amazon.awssdk.enhanced.dynamodb.DynamoDbTable;  
import software.amazon.awssdk.enhanced.dynamodb.TableSchema;  
import software.amazon.awssdk.enhanced.dynamodb.mapper.annotations.DynamoDbBean;  
import software.amazon.awssdk.enhanced.dynamodb.mapper.annotations.DynamoDbPartitionKey;  
import software.amazon.awssdk.enhanced.dynamodb.mapper.annotations.DynamoDbSortKey;  
import software.amazon.awssdk.regions.Region;  
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;  
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
```

Code

```
public static void scan( DynamoDbEnhancedClient enhancedClient) {  
    try{  
        // Create a DynamoDbTable object  
        DynamoDbTable<Customer> custTable = enhancedClient.table("Customer",  
            TableSchema.fromBean(Customer.class));  
        Iterator<Customer> results = custTable.scan().items().iterator();  
        while (results.hasNext()) {  
            Customer rec = results.next();  
            System.out.println("The record id is "+rec.getId());  
        }  
    } catch (DynamoDbException e) {  
        System.err.println(e.getMessage());  
        System.exit(1);  
    }  
    System.out.println("Done");  
}
```

See the [complete example](#) on GitHub.

For more information, see [Working with items in DynamoDB](#) in the Amazon DynamoDB Developer Guide.

Working with Amazon EC2

This section provides examples of programming [Amazon EC2](#) that use the AWS SDK for Java 2.0.

Topics

- [Manage Amazon EC2 instances](#) (p. 89)
- [Use elastic IP addresses in Amazon EC2](#) (p. 94)
- [Use regions and availability zones](#) (p. 96)
- [Work with Amazon EC2 key pairs](#) (p. 98)
- [Work with security groups in Amazon EC2](#) (p. 100)

Manage Amazon EC2 instances

Create an instance

Create a new Amazon EC2 instance by calling the `Ec2Client`'s `runInstances` method, providing it with a [RunInstancesRequest](#) containing the [Amazon Machine Image \(AMI\)](#) to use and an [instance type](#).

Imports

```
import software.amazon.awssdk.regions.Region;  
import software.amazon.awssdk.services.ec2.Ec2Client;  
import software.amazon.awssdk.services.ec2.model.InstanceType;  
import software.amazon.awssdk.services.ec2.model.RunInstancesRequest;  
import software.amazon.awssdk.services.ec2.model.RunInstancesResponse;  
import software.amazon.awssdk.services.ec2.model.Tag;  
import software.amazon.awssdk.services.ec2.model.CreateTagsRequest;  
import software.amazon.awssdk.services.ec2.model.Ec2Exception;
```

Code

```
public static String createEC2Instance(Ec2Client ec2, String name, String amiId ) {

    RunInstancesRequest runRequest = RunInstancesRequest.builder()
        .imageId(amiId)
        .instanceType(InstanceType.T1_MICRO)
        .maxCount(1)
        .minCount(1)
        .build();

    RunInstancesResponse response = ec2.runInstances(runRequest);
    String instanceId = response.instances().get(0).instanceId();

    Tag tag = Tag.builder()
        .key("Name")
        .value(name)
        .build();

    CreateTagsRequest tagRequest = CreateTagsRequest.builder()
        .resources(instanceId)
        .tags(tag)
        .build();

    try {
        ec2.createTags(tagRequest);
        System.out.printf(
            "Successfully started EC2 Instance %s based on AMI %s",
            instanceId, amiId);

        return instanceId;
    } catch (Ec2Exception e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
```

See the [complete example](#) on GitHub.

Start an instance

To start an Amazon EC2 instance, call the `Ec2Client`'s `startInstances` method, providing it with a [StartInstancesRequest](#) containing the ID of the instance to start.

Imports

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.ec2.Ec2Client;
import software.amazon.awssdk.services.ec2.model.StartInstancesRequest;
import software.amazon.awssdk.services.ec2.model.StopInstancesRequest;
```

Code

```
public static void startInstance(Ec2Client ec2, String instanceId) {

    StartInstancesRequest request = StartInstancesRequest.builder()
        .instanceIds(instanceId)
        .build();

    ec2.startInstances(request);
}
```


See the [complete example](#) on GitHub.

Stop an instance

To stop an Amazon EC2 instance, call the `Ec2Client`'s `stopInstances` method, providing it with a [StopInstancesRequest](#) containing the ID of the instance to stop.

Imports

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.ec2.Ec2Client;
import software.amazon.awssdk.services.ec2.model.StartInstancesRequest;
import software.amazon.awssdk.services.ec2.model.StopInstancesRequest;
```

Code

```
public static void stopInstance(Ec2Client ec2, String instanceId) {

    StopInstancesRequest request = StopInstancesRequest.builder()
        .instanceIds(instanceId)
        .build();

    ec2.stopInstances(request);
}
```

See the [complete example](#) on GitHub.

Reboot an instance

To reboot an Amazon EC2 instance, call the `Ec2Client`'s `rebootInstances` method, providing it with a [RebootInstancesRequest](#) containing the ID of the instance to reboot.

Imports

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.ec2.Ec2Client;
import software.amazon.awssdk.services.ec2.model.Ec2Exception;
import software.amazon.awssdk.services.ec2.model.RebootInstancesRequest;
```

Code

```
public static void rebootEC2Instance(Ec2Client ec2, String instanceId) {

    try {
        RebootInstancesRequest request = RebootInstancesRequest.builder()
            .instanceIds(instanceId)
            .build();

        ec2.rebootInstances(request);
        System.out.printf(
            "Successfully rebooted instance %s", instanceId);
    } catch (Ec2Exception e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
```

See the [complete example](#) on GitHub.

Describe instances

To list your instances, create a [DescribeInstancesRequest](#) and call the `Ec2Client`'s `describeInstances` method. It will return a [DescribeInstancesResponse](#) object that you can use to list the Amazon EC2 instances for your account and region.

Instances are grouped by *reservation*. Each reservation corresponds to the call to `startInstances` that launched the instance. To list your instances, you must first call the `DescribeInstancesResponse` class' `reservations` method, and then call `instances` on each returned [Reservation](#) object.

Imports

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.ec2.Ec2Client;
import software.amazon.awssdk.services.ec2.model.DescribeInstancesRequest;
import software.amazon.awssdk.services.ec2.model.DescribeInstancesResponse;
import software.amazon.awssdk.services.ec2.model.Instance;
import software.amazon.awssdk.services.ec2.model.Reservation;
import software.amazon.awssdk.services.ec2.model.Ec2Exception;
```

Code

```
public static void describeEC2Instances( Ec2Client ec2){

    boolean done = false;
    String nextToken = null;

    try {
        do {
            DescribeInstancesRequest request =
                DescribeInstancesRequest.builder().maxResults(6).nextToken(nextToken).build();
            DescribeInstancesResponse response = ec2.describeInstances(request);

            for (Reservation reservation : response.reservations()) {
                for (Instance instance : reservation.instances()) {
                    System.out.printf(
                        "Found Reservation with id %s, " +
                        "AMI %s, " +
                        "type %s, " +
                        "state %s " +
                        "and monitoring state %s",
                        instance.instanceId(),
                        instance.imageId(),
                        instance.instanceType(),
                        instance.state().name(),
                        instance.monitoring().state());
                    System.out.println("");
                }
            }
            nextToken = response.nextToken();
        } while (nextToken != null);

    } catch (Ec2Exception e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
```

Results are paged; you can get further results by passing the value returned from the result object's `nextToken` method to a new request object's `nextToken` method, then using the new request object in your next call to `describeInstances`.

See the [complete example](#) on GitHub.

Monitor an instance

You can monitor various aspects of your Amazon EC2 instances, such as CPU and network utilization, available memory, and disk space remaining. To learn more about instance monitoring, see [Monitoring Amazon EC2](#) in the Amazon EC2 User Guide for Linux Instances.

To start monitoring an instance, you must create a [MonitorInstancesRequest](#) with the ID of the instance to monitor, and pass it to the `Ec2Client`'s `monitorInstances` method.

Imports

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.ec2.Ec2Client;
import software.amazon.awssdk.services.ec2.model.MonitorInstancesRequest;
import software.amazon.awssdk.services.ec2.model.UnmonitorInstancesRequest;
```

Code

```
MonitorInstancesRequest request = MonitorInstancesRequest.builder()
    .instanceIds(instanceId).build();

ec2.monitorInstances(request);
```

See the [complete example](#) on GitHub.

Stop instance monitoring

To stop monitoring an instance, create an [UnmonitorInstancesRequest](#) with the ID of the instance to stop monitoring, and pass it to the `Ec2Client`'s `unmonitorInstances` method.

Imports

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.ec2.Ec2Client;
import software.amazon.awssdk.services.ec2.model.MonitorInstancesRequest;
import software.amazon.awssdk.services.ec2.model.UnmonitorInstancesRequest;
```

Code

```
UnmonitorInstancesRequest request = UnmonitorInstancesRequest.builder()
    .instanceIds(instanceId).build();

ec2.unmonitorInstances(request);
```

See the [complete example](#) on GitHub.

More information

- [RunInstances](#) in the Amazon EC2 API Reference
- [DescribeInstances](#) in the Amazon EC2 API Reference
- [StartInstances](#) in the Amazon EC2 API Reference
- [StopInstances](#) in the Amazon EC2 API Reference
- [RebootInstances](#) in the Amazon EC2 API Reference

- [DescribeInstances](#) in the Amazon EC2 API Reference
- [MonitorInstances](#) in the Amazon EC2 API Reference
- [UnmonitorInstances](#) in the Amazon EC2 API Reference

Use elastic IP addresses in Amazon EC2

Allocate an elastic IP address

To use an Elastic IP address, you first allocate one to your account, and then associate it with your instance or a network interface.

To allocate an Elastic IP address, call the `Ec2Client`'s `allocateAddress` method with an [AllocateAddressRequest](#) object containing the network type (classic EC2 or VPC).

The returned [AllocateAddressResponse](#) contains an allocation ID that you can use to associate the address with an instance, by passing the allocation ID and instance ID in a [AssociateAddressRequest](#) to the `Ec2Client`'s `associateAddress` method.

Imports

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.ec2.Ec2Client;
import software.amazon.awssdk.services.ec2.model.AllocateAddressRequest;
import software.amazon.awssdk.services.ec2.model.DomainType;
import software.amazon.awssdk.services.ec2.model.AllocateAddressResponse;
import software.amazon.awssdk.services.ec2.model.AssociateAddressRequest;
import software.amazon.awssdk.services.ec2.model.AssociateAddressResponse;
import software.amazon.awssdk.services.ec2.model.Ec2Exception;
```

Code

```
public static String getAllocateAddress( Ec2Client ec2, String instanceId) {
    try {
        AllocateAddressRequest allocateRequest = AllocateAddressRequest.builder()
            .domain(DomainType.VPC)
            .build();

        AllocateAddressResponse allocateResponse =
            ec2.allocateAddress(allocateRequest);

        String allocationId = allocateResponse.allocationId();

        AssociateAddressRequest associateRequest =
            AssociateAddressRequest.builder()
                .instanceId(instanceId)
                .allocationId(allocationId)
                .build();

        AssociateAddressResponse associateResponse =
            ec2.associateAddress(associateRequest);
        return associateResponse.associationId();

    } catch (Ec2Exception e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
    return "";
}
```

See the [complete example](#) on GitHub.

Describe elastic IP addresses

To list the Elastic IP addresses assigned to your account, call the `Ec2Client`'s `describeAddresses` method. It returns a [DescribeAddressesResponse](#) which you can use to get a list of [Address](#) objects that represent the Elastic IP addresses on your account.

Imports

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.ec2.Ec2Client;
import software.amazon.awssdk.services.ec2.model.Address;
import software.amazon.awssdk.services.ec2.model.DescribeAddressesResponse;
import software.amazon.awssdk.services.ec2.model.Ec2Exception;
```

Code

```
public static void describeEC2Address(Ec2Client ec2 ) {

    try {
        DescribeAddressesResponse response = ec2.describeAddresses();

        for(Address address : response.addresses()) {
            System.out.printf(
                "Found address with public IP %s, " +
                "domain %s, " +
                "allocation id %s " +
                "and NIC id %s",
                address.publicIp(),
                address.domain(),
                address.allocationId(),
                address.networkInterfaceId());
        }
    } catch (Ec2Exception e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
```

See the [complete example](#) on GitHub.

Release an elastic IP address

To release an Elastic IP address, call the `Ec2Client`'s `releaseAddress` method, passing it a [ReleaseAddressRequest](#) containing the allocation ID of the Elastic IP address you want to release.

Imports

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.ec2.Ec2Client;
import software.amazon.awssdk.services.ec2.model.Ec2Exception;
import software.amazon.awssdk.services.ec2.model.ReleaseAddressRequest;
import software.amazon.awssdk.services.ec2.model.ReleaseAddressResponse;
```

Code

```
public static void releaseEC2Address(Ec2Client ec2,String allocId) {

    try {
        ReleaseAddressRequest request = ReleaseAddressRequest.builder()
```

```
        .allocationId(allocId).build();

        ReleaseAddressResponse response = ec2.releaseAddress(request);

        System.out.printf(
            "Successfully released elastic IP address %s", allocId);
    } catch (Ec2Exception e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
```

After you release an Elastic IP address, it is released to the AWS IP address pool and might be unavailable to you afterward. Be sure to update your DNS records and any servers or devices that communicate with the address.

If you are using *EC2-Classic* or a *default VPC*, then releasing an Elastic IP address automatically disassociates it from any instance that it's associated with. To disassociate an Elastic IP address without releasing it, use the `Ec2Client`'s `disassociateAddress` method.

If you are using a non-default VPC, you *must* use `disassociateAddress` to disassociate the Elastic IP address before you try to release it. Otherwise, Amazon EC2 returns an error (*InvalidIPAddress.InUse*).

See the [complete example](#) on GitHub.

More information

- [Elastic IP Addresses](#) in the Amazon EC2 User Guide for Linux Instances
- [AllocateAddress](#) in the Amazon EC2 API Reference
- [DescribeAddresses](#) in the Amazon EC2 API Reference
- [ReleaseAddress](#) in the Amazon EC2 API Reference

Use regions and availability zones

Describe regions

To list the Regions available to your account, call the `Ec2Client`'s `describeRegions` method. It returns a [DescribeRegionsResponse](#). Call the returned object's `regions` method to get a list of [Region](#) objects that represent each Region.

Imports

```
import software.amazon.awssdk.services.ec2.Ec2Client;
import software.amazon.awssdk.services.ec2.model.DescribeRegionsResponse;
import software.amazon.awssdk.services.ec2.model.Region;
import software.amazon.awssdk.services.ec2.model.AvailabilityZone;
import software.amazon.awssdk.services.ec2.model.Ec2Exception;
import software.amazon.awssdk.services.ec2.model.DescribeAvailabilityZonesResponse;
```

Code

```
try {

    DescribeRegionsResponse regionsResponse = ec2.describeRegions();

    for(Region region : regionsResponse.regions()) {
        System.out.printf(
```

```
        "Found Region %s " +  
            "with endpoint %s",  
        region.regionName(),  
        region.endpoint());  
System.out.println();
```

See the [complete example](#) on GitHub.

Describe availability zones

To list each Availability Zone available to your account, call the `Ec2Client`'s `describeAvailabilityZones` method. It returns a [DescribeAvailabilityZonesResponse](#). Call its `availabilityZones` method to get a list of [AvailabilityZone](#) objects that represent each Availability Zone.

Imports

```
import software.amazon.awssdk.services.ec2.Ec2Client;  
import software.amazon.awssdk.services.ec2.model.DescribeRegionsResponse;  
import software.amazon.awssdk.services.ec2.model.Region;  
import software.amazon.awssdk.services.ec2.model.AvailabilityZone;  
import software.amazon.awssdk.services.ec2.model.Ec2Exception;  
import software.amazon.awssdk.services.ec2.model.DescribeAvailabilityZonesResponse;
```

Code

Create the `Ec2Client`.

```
Ec2Client ec2 = Ec2Client.create();
```

Then call `describeAvailabilityZones()` and retrieve results.

```
DescribeAvailabilityZonesResponse zonesResponse =  
    ec2.describeAvailabilityZones();  
  
for(AvailabilityZone zone : zonesResponse.availabilityZones()) {  
    System.out.printf(  
        "Found Availability Zone %s " +  
            "with status %s " +  
            "in region %s",  
        zone.zoneName(),  
        zone.state(),  
        zone.regionName());  
    System.out.println();  
}
```

See the [complete example](#) on GitHub.

Describe accounts

To describe your account, call the `Ec2Client`'s `describeAccountAttributes` method. This method returns a [DescribeAccountAttributesResponse](#) object. Invoke this object's `accountAttributes` method to get a list of [AccountAttribute](#) objects. You can iterate through the list to retrieve an [AccountAttribute](#) object.

You can get your account's attribute values by invoking the [AccountAttribute](#) object's `attributeValues` method. This method returns a list of [AccountAttributeValue](#) objects. You can iterate through this second list to display the value of attributes (see the following code example).

Imports

```
import software.amazon.awssdk.services.ec2.Ec2Client;
import software.amazon.awssdk.services.ec2.model.DescribeRegionsResponse;
import software.amazon.awssdk.services.ec2.model.Region;
import software.amazon.awssdk.services.ec2.model.AvailabilityZone;
import software.amazon.awssdk.services.ec2.model.Ec2Exception;
import software.amazon.awssdk.services.ec2.model.DescribeAvailabilityZonesResponse;
```

Code

```
try {

    DescribeRegionsResponse regionsResponse = ec2.describeRegions();

    for(Region region : regionsResponse.regions()) {
        System.out.printf(
            "Found Region %s " +
            "with endpoint %s",
            region.regionName(),
            region.endpoint());
        System.out.println();
    }
}
```

See the [complete example](#) on GitHub.

More information

- [Regions and Availability Zones](#) in the Amazon EC2 User Guide for Linux Instances
- [DescribeRegions](#) in the Amazon EC2 API Reference
- [DescribeAvailabilityZones](#) in the Amazon EC2 API Reference

Work with Amazon EC2 key pairs

Create a key pair

To create a key pair, call the `Ec2Client`'s `createKeyPair` method with a [CreateKeyPairRequest](#) that contains the key's name.

Imports

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.ec2.Ec2Client;
import software.amazon.awssdk.services.ec2.model.CreateKeyPairRequest;
import software.amazon.awssdk.services.ec2.model.CreateKeyPairResponse;
import software.amazon.awssdk.services.ec2.model.Ec2Exception;
```

Code

```
public static void createEC2KeyPair(Ec2Client ec2,String keyName ) {

    try {
        CreateKeyPairRequest request = CreateKeyPairRequest.builder()
            .keyName(keyName).build();

        CreateKeyPairResponse response = ec2.createKeyPair(request);
        System.out.printf(
            "Successfully created key pair named %s",
            keyName);
    }
}
```



```
    } catch (Ec2Exception e) {  
        System.err.println(e.awsErrorDetails().errorMessage());  
        System.exit(1);  
    }
```

See the [complete example](#) on GitHub.

Describe key pairs

To list your key pairs or to get information about them, call the `Ec2Client`'s `describeKeyPairs` method. It returns a [DescribeKeyPairsResponse](#) that you can use to access the list of key pairs by calling its `keyPairs` method, which returns a list of [KeyPairInfo](#) objects.

Imports

```
import software.amazon.awssdk.regions.Region;  
import software.amazon.awssdk.services.ec2.Ec2Client;  
import software.amazon.awssdk.services.ec2.model.DescribeKeyPairsResponse;  
import software.amazon.awssdk.services.ec2.model.KeyPairInfo;  
import software.amazon.awssdk.services.ec2.model.Ec2Exception;
```

Code

```
public static void describeEC2Keys( Ec2Client ec2){  
  
    try {  
        DescribeKeyPairsResponse response = ec2.describeKeyPairs();  
  
        for(KeyPairInfo keyPair : response.keyPairs()) {  
            System.out.printf(  
                "Found key pair with name %s " +  
                "and fingerprint %s",  
                keyPair.keyName(),  
                keyPair.keyFingerprint());  
        }  
    } catch (Ec2Exception e) {  
        System.err.println(e.awsErrorDetails().errorMessage());  
        System.exit(1);  
    }  
}
```

See the [complete example](#) on GitHub.

Delete a key pair

To delete a key pair, call the `Ec2Client`'s `deleteKeyPair` method, passing it a [DeleteKeyPairRequest](#) that contains the name of the key pair to delete.

Imports

```
import software.amazon.awssdk.regions.Region;  
import software.amazon.awssdk.services.ec2.Ec2Client;  
import software.amazon.awssdk.services.ec2.model.DeleteKeyPairRequest;  
import software.amazon.awssdk.services.ec2.model.DeleteKeyPairResponse;  
import software.amazon.awssdk.services.ec2.model.Ec2Exception;
```

Code

```
public static void deleteKeys(Ec2Client ec2, String keyPair) {
```

```
try {  
  
    DeleteKeyPairRequest request = DeleteKeyPairRequest.builder()  
        .keyName(keyPair)  
        .build();  
  
    DeleteKeyPairResponse response = ec2.deleteKeyPair(request);  
  
}
```

See the [complete example](#) on GitHub.

More information

- [Amazon EC2 Key Pairs](#) in the Amazon EC2 User Guide for Linux Instances
- [CreateKeyPair](#) in the Amazon EC2 API Reference
- [DescribeKeyPairs](#) in the Amazon EC2 API Reference
- [DeleteKeyPair](#) in the Amazon EC2 API Reference

Work with security groups in Amazon EC2

Create a security group

To create a security group, call the `Ec2Client`'s `createSecurityGroup` method with a [CreateSecurityGroupRequest](#) that contains the key's name.

Imports

```
import software.amazon.awssdk.regions.Region;  
import software.amazon.awssdk.services.ec2.Ec2Client;  
import software.amazon.awssdk.services.ec2.model.CreateSecurityGroupRequest;  
import software.amazon.awssdk.services.ec2.model.AuthorizeSecurityGroupIngressRequest;  
import software.amazon.awssdk.services.ec2.model.AuthorizeSecurityGroupIngressResponse;  
import software.amazon.awssdk.services.ec2.model.Ec2Exception;  
import software.amazon.awssdk.services.ec2.model.IpPermission;  
import software.amazon.awssdk.services.ec2.model.CreateSecurityGroupResponse;  
import software.amazon.awssdk.services.ec2.model.IpRange;
```

Code

```
CreateSecurityGroupRequest createRequest = CreateSecurityGroupRequest.builder()  
    .groupName(groupName)  
    .description(groupDesc)  
    .vpcId(vpcId)  
    .build();  
  
CreateSecurityGroupResponse resp= ec2.createSecurityGroup(createRequest);
```

See the [complete example](#) on GitHub.

Configure a security group

A security group can control both inbound (ingress) and outbound (egress) traffic to your Amazon EC2 instances.

To add ingress rules to your security group, use the `Ec2Client`'s `authorizeSecurityGroupIngress` method, providing the name of the security group and the access rules ([IpPermission](#)) you want to assign

to it within an [AuthorizeSecurityGroupIngressRequest](#) object. The following example shows how to add IP permissions to a security group.

Imports

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.ec2.Ec2Client;
import software.amazon.awssdk.services.ec2.model.CreateSecurityGroupRequest;
import software.amazon.awssdk.services.ec2.model.AuthorizeSecurityGroupIngressRequest;
import software.amazon.awssdk.services.ec2.model.AuthorizeSecurityGroupIngressResponse;
import software.amazon.awssdk.services.ec2.model.Ec2Exception;
import software.amazon.awssdk.services.ec2.model.IpPermission;
import software.amazon.awssdk.services.ec2.model.CreateSecurityGroupResponse;
import software.amazon.awssdk.services.ec2.model.IpRange;
```

Code

First, create an `Ec2Client`

```
Region region = Region.US_WEST_2;
Ec2Client ec2 = Ec2Client.builder()
    .region(region)
    .build();
```

Then use the `Ec2Client`'s `authorizeSecurityGroupIngress` method,

```
IpRange ipRange = IpRange.builder()
    .cidrIp("0.0.0.0/0").build();

IpPermission ipPerm = IpPermission.builder()
    .ipProtocol("tcp")
    .toPort(80)
    .fromPort(80)
    .ipRanges(ipRange)
    .build();

IpPermission ipPerm2 = IpPermission.builder()
    .ipProtocol("tcp")
    .toPort(22)
    .fromPort(22)
    .ipRanges(ipRange)
    .build();

AuthorizeSecurityGroupIngressRequest authRequest =
    AuthorizeSecurityGroupIngressRequest.builder()
        .groupName(groupName)
        .ipPermissions(ipPerm, ipPerm2)
        .build();

AuthorizeSecurityGroupIngressResponse authResponse =
    ec2.authorizeSecurityGroupIngress(authRequest);
```

To add an egress rule to the security group, provide similar data in an [AuthorizeSecurityGroupEgressRequest](#) to the `Ec2Client`'s `authorizeSecurityGroupEgress` method.

See the [complete example](#) on GitHub.

Describe security groups

To describe your security groups or get information about them, call the `Ec2Client`'s `describeSecurityGroups` method. It returns a [DescribeSecurityGroupsResponse](#) that you can use

to access the list of security groups by calling its `securityGroups` method, which returns a list of [SecurityGroup](#) objects.

Imports

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.ec2.Ec2Client;
import software.amazon.awssdk.services.ec2.model.DescribeSecurityGroupsRequest;
import software.amazon.awssdk.services.ec2.model.DescribeSecurityGroupsResponse;
import software.amazon.awssdk.services.ec2.model.SecurityGroup;
import software.amazon.awssdk.services.ec2.model.Ec2Exception;
```

Code

```
public static void describeEC2SecurityGroups(Ec2Client ec2, String groupId) {

    try {
        DescribeSecurityGroupsRequest request =
            DescribeSecurityGroupsRequest.builder()
                .groupIds(groupId).build();

        DescribeSecurityGroupsResponse response =
            ec2.describeSecurityGroups(request);

        for(SecurityGroup group : response.securityGroups()) {
            System.out.printf(
                "Found Security Group with id %s, " +
                "vpc id %s " +
                "and description %s",
                group.groupId(),
                group.vpcId(),
                group.description());
        }
    } catch (Ec2Exception e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
```

See the [complete example](#) on GitHub.

Delete a security group

To delete a security group, call the `Ec2Client`'s `deleteSecurityGroup` method, passing it a [DeleteSecurityGroupRequest](#) that contains the ID of the security group to delete.

Imports

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.ec2.Ec2Client;
import software.amazon.awssdk.services.ec2.model.DeleteSecurityGroupRequest;
import software.amazon.awssdk.services.ec2.model.DeleteSecurityGroupResponse;
import software.amazon.awssdk.services.ec2.model.Ec2Exception;
```

Code

```
public static void deleteEC2SecGroup(Ec2Client ec2, String groupId) {

    try {
        DeleteSecurityGroupRequest request = DeleteSecurityGroupRequest.builder()
            .groupId(groupId)
            .build();
    }
```

```
ec2.deleteSecurityGroup(request);
System.out.printf(
    "Successfully deleted Security Group with id %s", groupId);

} catch (Ec2Exception e) {
    System.err.println(e.awsErrorDetails().errorMessage());
    System.exit(1);
}
```

See the [complete example](#) on GitHub.

More information

- [Amazon EC2 Security Groups](#) in the Amazon EC2 User Guide for Linux Instances
- [Authorizing Inbound Traffic for Your Linux Instances](#) in the Amazon EC2 User Guide for Linux Instances
- [CreateSecurityGroup](#) in the Amazon EC2 API Reference
- [DescribeSecurityGroups](#) in the Amazon EC2 API Reference
- [DeleteSecurityGroup](#) in the Amazon EC2 API Reference
- [AuthorizeSecurityGroupIngress](#) in the Amazon EC2 API Reference

Working with IAM

This section provides examples of programming [IAM](#) by using the AWS SDK for Java 2.0.

AWS Identity and Access Management (IAM) enables you to securely control access to AWS services and resources for your users. Using IAM, you can create and manage AWS users and groups, and use permissions to allow and deny their access to AWS resources. For a complete guide to IAM, visit the [IAM User Guide](#).

The following 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

- [Manage IAM access keys \(p. 103\)](#)
- [Managing IAM Users \(p. 107\)](#)
- [Use IAM account aliases \(p. 110\)](#)
- [Work with IAM policies \(p. 112\)](#)
- [Work with IAM server certificates \(p. 117\)](#)

Manage IAM access keys

Create an access key

To create an IAM access key, call the `IamClient`'s `createAccessKey` method with a [CreateAccessKeyRequest](#) object.

Note

You must set the region to `AWS_GLOBAL` for `IamClient` calls to work because IAM is a global service.

Imports

```
import software.amazon.awssdk.services.iam.model.CreateAccessKeyRequest;
import software.amazon.awssdk.services.iam.model.CreateAccessKeyResponse;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.iam.IamClient;
import software.amazon.awssdk.services.iam.model.IamException;
```

Code

```
public static String createIAMAccessKey(IamClient iam,String user) {

    try {
        CreateAccessKeyRequest request = CreateAccessKeyRequest.builder()
            .userName(user).build();

        CreateAccessKeyResponse response = iam.createAccessKey(request);
        String keyId = response.accessKey().accessKeyId();
        return keyId;

    } catch (IamException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
    return "";
}
```

See the [complete example](#) on GitHub.

List access keys

To list the access keys for a given user, create a [ListAccessKeysRequest](#) object that contains the user name to list keys for, and pass it to the `IamClient`'s `listAccessKeys` method.

Note

If you do not supply a user name to `listAccessKeys`, it will attempt to list access keys associated with the AWS account that signed the request.

Imports

```
import software.amazon.awssdk.services.iam.model.AccessKeyMetadata;
import software.amazon.awssdk.services.iam.model.IamException;
import software.amazon.awssdk.services.iam.model.ListAccessKeysRequest;
import software.amazon.awssdk.services.iam.model.ListAccessKeysResponse;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.iam.IamClient;
```

Code

```
public static void listKeys( IamClient iam,String userName ){

    try {
        boolean done = false;
        String newMarker = null;

        while (!done) {
            ListAccessKeysResponse response;

            if(newMarker == null) {
                ListAccessKeysRequest request = ListAccessKeysRequest.builder()
                    .userName(userName).build();
                response = iam.listAccessKeys(request);
            }
        }
    }
}
```

```
    } else {
        ListAccessKeysRequest request = ListAccessKeysRequest.builder()
            .userName(userName)
            .marker(newMarker).build();
        response = iam.listAccessKeys(request);
    }

    for (AccessKeyMetadata metadata :
        response.accessKeyMetadata()) {
        System.out.format("Retrieved access key %s",
            metadata.accessKeyId());
    }

    if (!response.isTruncated()) {
        done = true;
    } else {
        newMarker = response.marker();
    }
}

} catch (IamException e) {
    System.err.println(e.awsErrorDetails().errorMessage());
    System.exit(1);
}
```

The results of `listAccessKeys` are paged (with a default maximum of 100 records per call). You can call `isTruncated` on the returned [ListAccessKeysResponse](#) object to see if the query returned fewer results than are available. If so, then call `marker` on the `ListAccessKeysResponse` and use it when creating a new request. Use that new request in the next invocation of `listAccessKeys`.

See the [complete example](#) on GitHub.

Retrieve an access key's last used time

To get the time an access key was last used, call the `IamClient`'s `getAccessKeyLastUsed` method with the access key's ID (which can be passed in using a [GetAccessKeyLastUsedRequest](#) object).

You can then use the returned [GetAccessKeyLastUsedResponse](#) object to retrieve the key's last used time.

Imports

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.iam.IamClient;
import software.amazon.awssdk.services.iam.model.GetAccessKeyLastUsedRequest;
import software.amazon.awssdk.services.iam.model.GetAccessKeyLastUsedResponse;
import software.amazon.awssdk.services.iam.model.IamException;
```

Code

```
public static void getAccessKeyLastUsed(IamClient iam, String accessId ){

    try {
        GetAccessKeyLastUsedRequest request = GetAccessKeyLastUsedRequest.builder()
            .accessKeyId(accessId).build();

        GetAccessKeyLastUsedResponse response = iam.getAccessKeyLastUsed(request);

        System.out.println("Access key was last used at: " +
            response.accessKeyLastUsed().lastUsedDate());

    } catch (IamException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
    }
}
```

```
        System.exit(1);
    }
    System.out.println("Done");
```

See the [complete example](#) on GitHub.

Activate or deactivate access keys

You can activate or deactivate an access key by creating an [UpdateAccessKeyRequest](#) object, providing the access key ID, optionally the user name, and the desired [status](#), then passing the request object to the `IamClient`'s `updateAccessKey` method.

Imports

```
import software.amazon.awssdk.services.iam.model.IamException;
import software.amazon.awssdk.services.iam.model.StatusType;
import software.amazon.awssdk.services.iam.model.UpdateAccessKeyRequest;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.iam.IamClient;
```

Code

```
public static void updateKey(IamClient iam, String username, String accessId, String
    status ) {

    try {
        if (status.toLowerCase().equalsIgnoreCase("active")) {
            statusType = StatusType.ACTIVE;
        } else if (status.toLowerCase().equalsIgnoreCase("inactive")) {
            statusType = StatusType.INACTIVE;
        } else {
            statusType = StatusType.UNKNOWN_TO_SDK_VERSION;
        }
        UpdateAccessKeyRequest request = UpdateAccessKeyRequest.builder()
            .accessKeyId(accessId)
            .userName(username)
            .status(statusType)
            .build();

        iam.updateAccessKey(request);

        System.out.printf(
            "Successfully updated the status of access key %s to" +
            "status %s for user %s", accessId, status, username);

    } catch (IamException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
```

See the [complete example](#) on GitHub.

Delete an access key

To permanently delete an access key, call the `IamClient`'s `deleteKey` method, providing it with a [DeleteAccessKeyRequest](#) containing the access key's ID and username.

Note

Once deleted, a key can no longer be retrieved or used. To temporarily deactivate a key so that it can be activated again later, use [updateAccessKey](#) (p. 106) method instead.

Imports

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.iam.IamClient;
import software.amazon.awssdk.services.iam.model.DeleteAccessKeyRequest;
import software.amazon.awssdk.services.iam.model.IamException;
```

Code

```
public static void deleteKey(IamClient iam ,String username, String accessKey ) {

    try {
        DeleteAccessKeyRequest request = DeleteAccessKeyRequest.builder()
            .accessKeyId(accessKey)
            .userName(username)
            .build();

        iam.deleteAccessKey(request);
        System.out.println("Successfully deleted access key " + accessKey +
            " from user " + username);

    } catch (IamException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
```

See the [complete example](#) on GitHub.

More information

- [CreateAccessKey](#) in the IAM API Reference
- [ListAccessKeys](#) in the IAM API Reference
- [GetAccessKeyLastUsed](#) in the IAM API Reference
- [UpdateAccessKey](#) in the IAM API Reference
- [DeleteAccessKey](#) in the IAM API Reference

Managing IAM Users

Creating a User

Create a new IAM user by providing the user name to the `IamClient`'s `createUser` method using a [CreateUserRequest](#) object containing the user name.

Imports

```
import software.amazon.awssdk.core.waiters.WaiterResponse;
import software.amazon.awssdk.services.iam.model.CreateUserRequest;
import software.amazon.awssdk.services.iam.model.CreateUserResponse;
import software.amazon.awssdk.services.iam.model.IamException;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.iam.IamClient;
import software.amazon.awssdk.services.iam.waiters.IamWaiter;
import software.amazon.awssdk.services.iam.model.GetUserRequest;
import software.amazon.awssdk.services.iam.model.GetUserResponse;
```

Code

```
public static String createIAMUser(IamClient iam, String username ) {

    try {
        // Create an IamWaiter object
        IamWaiter iamWaiter = iam.waiter();

        CreateUserRequest request = CreateUserRequest.builder()
            .userName(username)
            .build();

        CreateUserResponse response = iam.createUser(request);

        // Wait until the user is created
        GetUserRequest userRequest = GetUserRequest.builder()
            .userName(response.user().userName())
            .build();

        WaiterResponse<GetUserResponse> waitUntilUserExists =
            iamWaiter.waitUntilUserExists(userRequest);
        waitUntilUserExists.matched().response().ifPresent(System.out::println);
        return response.user().userName();

    } catch (IamException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
    return "";
}
```

See the [complete example](#) on GitHub.

Listing Users

To list the IAM users for your account, create a new [ListUsersRequest](#) and pass it to the `IamClient`'s `listUsers` method. You can retrieve the list of users by calling `users` on the returned [ListUsersResponse](#) object.

The list of users returned by `listUsers` is paged. You can check to see there are more results to retrieve by calling the response object's `isTruncated` method. If it returns `true`, then call the response object's `marker()` method. Use the marker value to create a new request object. Then call the `listUsers` method again with the new request.

Imports

```
import software.amazon.awssdk.services.iam.model.IamException;
import software.amazon.awssdk.services.iam.model.ListUsersRequest;
import software.amazon.awssdk.services.iam.model.ListUsersResponse;
import software.amazon.awssdk.services.iam.model.User;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.iam.IamClient;
```

Code

```
public static void listAllUsers(IamClient iam ) {

    try {

        boolean done = false;
        String newMarker = null;
```

```
while(!done) {
    ListUsersResponse response;

    if (newMarker == null) {
        ListUsersRequest request = ListUsersRequest.builder().build();
        response = iam.listUsers(request);
    } else {
        ListUsersRequest request = ListUsersRequest.builder()
            .marker(newMarker).build();
        response = iam.listUsers(request);
    }

    for(User user : response.users()) {
        System.out.format("\n Retrieved user %s", user.userName());
    }

    if(!response.isTruncated()) {
        done = true;
    } else {
        newMarker = response.marker();
    }
}
} catch (IamException e) {
    System.err.println(e.awsErrorDetails().errorMessage());
    System.exit(1);
}
```

See the [complete example](#) on GitHub.

Updating a User

To update a user, call the `IamClient` object's `updateUser` method, which takes a [UpdateUserRequest](#) object that you can use to change the user's *name* or *path*.

Imports

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.iam.IamClient;
import software.amazon.awssdk.services.iam.model.IamException;
import software.amazon.awssdk.services.iam.model.UpdateUserRequest;
```

Code

```
public static void updateIAMUser(IamClient iam, String curName,String newName ) {

    try {
        UpdateUserRequest request = UpdateUserRequest.builder()
            .userName(curName)
            .newUserName(newName)
            .build();

        iam.updateUser(request);
        System.out.printf("Successfully updated user to username %s",
            newName);
    } catch (IamException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
```

See the [complete example](#) on GitHub.

Deleting a User

To delete a user, call the `IamClient`'s `deleteUser` request with a [UpdateUserRequest](#) object set with the user name to delete.

Imports

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.iam.IamClient;
import software.amazon.awssdk.services.iam.model.DeleteUserRequest;
import software.amazon.awssdk.services.iam.model.IamException;
```

Code

```
public static void deleteIAMUser(IamClient iam, String userName) {
    try {
        DeleteUserRequest request = DeleteUserRequest.builder()
            .userName(userName)
            .build();

        iam.deleteUser(request);
        System.out.println("Successfully deleted IAM user " + userName);
    } catch (IamException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
```

See the [complete example](#) on GitHub.

More Information

- [IAM Users](#) in the IAM User Guide
- [Managing IAM Users](#) in the IAM User Guide
- [CreateUser](#) in the IAM API Reference
- [ListUsers](#) in the IAM API Reference
- [UpdateUser](#) in the IAM API Reference
- [DeleteUser](#) in the IAM API Reference

Use IAM account aliases

If you want the URL for your sign-in page to contain your company name or other friendly identifier instead of your AWS account ID, you can create an alias for your AWS account.

Note

AWS supports exactly one account alias per account.

Create an account alias

To create an account alias, call the `IamClient`'s `createAccountAlias` method with a [CreateAccountAliasRequest](#) object that contains the alias name.

Imports

```
import software.amazon.awssdk.services.iam.model.CreateAccountAliasRequest;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.iam.IamClient;
import software.amazon.awssdk.services.iam.model.IamException;
```

Code

```
public static void createIAMAccountAlias(IamClient iam, String alias) {

    try {
        CreateAccountAliasRequest request = CreateAccountAliasRequest.builder()
            .accountAlias(alias)
            .build();

        iam.createAccountAlias(request);
        System.out.println("Successfully created account alias: " + alias);

    } catch (IamException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
```

See the [complete example](#) on GitHub.

List account aliases

To list your account's alias, if any, call the `IamClient`'s `listAccountAliases` method.

Note

The returned [ListAccountAliasesResponse](#) supports the same `isTruncated` and `marker` methods as other AWS SDK for Java *list* methods, but an AWS account can have only *one* account alias.

Imports

```
import software.amazon.awssdk.services.iam.model.IamException;
import software.amazon.awssdk.services.iam.model.ListAccountAliasesResponse;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.iam.IamClient;
```

Code

```
public static void listAliases(IamClient iam) {

    try {
        ListAccountAliasesResponse response = iam.listAccountAliases();

        for (String alias : response.accountAliases()) {
            System.out.printf("Retrieved account alias %s", alias);
        }

    } catch (IamException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
```

see the [complete example](#) on GitHub.

Delete an account alias

To delete your account's alias, call the `IamClient`'s `deleteAccountAlias` method. When deleting an account alias, you must supply its name using a [DeleteAccountAliasRequest](#) object.

Imports

```
import software.amazon.awssdk.services.iam.model.DeleteAccountAliasRequest;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.iam.IamClient;
import software.amazon.awssdk.services.iam.model.IamException;
```

Code

```
public static void deleteIAMAccountAlias(IamClient iam, String alias ) {

    try {
        DeleteAccountAliasRequest request = DeleteAccountAliasRequest.builder()
            .accountAlias(alias)
            .build();

        iam.deleteAccountAlias(request);
        System.out.println("Successfully deleted account alias " + alias);

    } catch (IamException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
    System.out.println("Done");
}
```

See the [complete example](#) on GitHub.

More information

- [Your AWS Account ID and Its Alias](#) in the IAM User Guide
- [CreateAccountAlias](#) in the IAM API Reference
- [ListAccountAliases](#) in the IAM API Reference
- [DeleteAccountAlias](#) in the IAM API Reference

Work with IAM policies

Create a policy

To create a new policy, provide the policy's name and a JSON-formatted policy document in a [CreatePolicyRequest](#) to the `IamClient`'s `createPolicy` method.

Imports

```
import software.amazon.awssdk.core.waiters.WaiterResponse;
import software.amazon.awssdk.services.iam.model.CreatePolicyRequest;
import software.amazon.awssdk.services.iam.model.CreatePolicyResponse;
import software.amazon.awssdk.services.iam.model.GetPolicyRequest;
import software.amazon.awssdk.services.iam.model.GetPolicyResponse;
import software.amazon.awssdk.services.iam.model.IamException;
```

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.iam.IamClient;
import software.amazon.awssdk.services.iam.waiters.IamWaiter;
```

Code

```
public static String createIAMPolicy(IamClient iam, String policyName ) {

    try {
        // Create an IamWaiter object
        IamWaiter iamWaiter = iam.waiter();

        CreatePolicyRequest request = CreatePolicyRequest.builder()
            .policyName(policyName)
            .policyDocument(PolicyDocument).build();

        CreatePolicyResponse response = iam.createPolicy(request);

        // Wait until the policy is created
        GetPolicyRequest polRequest = GetPolicyRequest.builder()
            .policyArn(response.policy().arn())
            .build();

        WaiterResponse<GetPolicyResponse> waitUntilPolicyExists =
            iamWaiter.waitUntilPolicyExists(polRequest);
        waitUntilPolicyExists.matched().response().ifPresent(System.out::println);
        return response.policy().arn();

    } catch (IamException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }

    return "" ;
}
```

See the [complete example](#) on GitHub.

Get a policy

To retrieve an existing policy, call the `IamClient`'s `getPolicy` method, providing the policy's ARN within a [GetPolicyRequest](#) object.

Imports

```
import software.amazon.awssdk.services.iam.model.GetPolicyRequest;
import software.amazon.awssdk.services.iam.model.GetPolicyResponse;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.iam.IamClient;
import software.amazon.awssdk.services.iam.model.IamException;
```

Code

```
public static void getIAMPolicy(IamClient iam, String policyArn) {

    try {

        GetPolicyRequest request = GetPolicyRequest.builder()
            .policyArn(policyArn).build();

        GetPolicyResponse response = iam.getPolicy(request);
        System.out.format("Successfully retrieved policy %s",
```

```
        response.policy().policyName());  
    } catch (IamException e) {  
        System.err.println(e.awsErrorDetails().errorMessage());  
        System.exit(1);  
    }  
}
```

See the [complete example](#) on GitHub.

Attach a role policy

You can attach a policy to an IAM [role](#) by calling the `IamClient`'s `attachRolePolicy` method, providing it with the role name and policy ARN in an [AttachRolePolicyRequest](#).

Imports

```
import software.amazon.awssdk.regions.Region;  
import software.amazon.awssdk.services.iam.IamClient;  
import software.amazon.awssdk.services.iam.model.IamException;  
import software.amazon.awssdk.services.iam.model.AttachRolePolicyRequest;  
import software.amazon.awssdk.services.iam.model.AttachedPolicy;  
import software.amazon.awssdk.services.iam.model.ListAttachedRolePoliciesRequest;  
import software.amazon.awssdk.services.iam.model.ListAttachedRolePoliciesResponse;  
import java.util.ArrayList;  
import java.util.List;
```

Code

```
public static void attachIAMRolePolicy(IamClient iam, String roleName, String policyArn) {  
    try {  
        List<AttachedPolicy> matchingPolicies = new ArrayList<>();  
        ListAttachedRolePoliciesRequest request = ListAttachedRolePoliciesRequest.builder()  
            .roleName(roleName)  
            .build();  
  
        ListAttachedRolePoliciesResponse response = iam.listAttachedRolePolicies(request);  
        List<AttachedPolicy> attachedPolicies = response.attachedPolicies();  
  
        // Ensure that the policy is not attached to this role  
        String polArn = "";  
        for (AttachedPolicy policy: attachedPolicies) {  
            polArn = policy.policyArn();  
            if (polArn.compareTo(policyArn)==0) {  
                System.out.println(roleName +  
                    " policy is already attached to this role.");  
                return;  
            }  
        }  
  
        AttachRolePolicyRequest attachRequest =  
            AttachRolePolicyRequest.builder()  
                .roleName(roleName)  
                .policyArn(policyArn)  
                .build();  
  
        iam.attachRolePolicy(attachRequest);  
  
        System.out.println("Successfully attached policy " + policyArn +
```



```
        " to role " + roleName));  
  
    } catch (IamException e) {  
        System.err.println(e.awsErrorDetails().errorMessage());  
        System.exit(1);  
    }  
}
```

See the [complete example](#) on GitHub.

List attached role policies

List attached policies on a role by calling the `IamClient`'s `listAttachedRolePolicies` method. It takes a [ListAttachedRolePoliciesRequest](#) object that contains the role name to list the policies for.

Call `getAttachedPolicies` on the returned [ListAttachedRolePoliciesResponse](#) object to get the list of attached policies. Results may be truncated; if the `ListAttachedRolePoliciesResponse` object's `isTruncated` method returns `true`, call the `ListAttachedRolePoliciesResponse` object's `marker` method. Use the marker returned to create a new request and use it to call `listAttachedRolePolicies` again to get the next batch of results.

Imports

```
import software.amazon.awssdk.regions.Region;  
import software.amazon.awssdk.services.iam.IamClient;  
import software.amazon.awssdk.services.iam.model.IamException;  
import software.amazon.awssdk.services.iam.model.AttachRolePolicyRequest;  
import software.amazon.awssdk.services.iam.model.AttachedPolicy;  
import software.amazon.awssdk.services.iam.model.ListAttachedRolePoliciesRequest;  
import software.amazon.awssdk.services.iam.model.ListAttachedRolePoliciesResponse;  
import java.util.ArrayList;  
import java.util.List;
```

Code

```
public static void attachIAMRolePolicy(IamClient iam, String roleName, String policyArn ) {  
  
    try {  
  
        List<AttachedPolicy> matchingPolicies = new ArrayList<>();  
        ListAttachedRolePoliciesRequest request = ListAttachedRolePoliciesRequest.builder()  
            .roleName(roleName)  
            .build();  
  
        ListAttachedRolePoliciesResponse response = iam.listAttachedRolePolicies(request);  
        List<AttachedPolicy> attachedPolicies = response.attachedPolicies();  
  
        // Ensure that the policy is not attached to this role  
        String polArn = "";  
        for (AttachedPolicy policy: attachedPolicies) {  
  
            polArn = policy.policyArn();  
            if (polArn.compareTo(policyArn)==0) {  
                System.out.println(roleName +  
                    " policy is already attached to this role.");  
                return;  
            }  
        }  
  
        AttachRolePolicyRequest attachRequest =  
            AttachRolePolicyRequest.builder()  
                .roleName(roleName)
```

```
        .policyArn(policyArn)
        .build();

iam.attachRolePolicy(attachRequest);

System.out.println("Successfully attached policy " + policyArn +
    " to role " + roleName);

} catch (IamException e) {
    System.err.println(e.awsErrorDetails().errorMessage());
    System.exit(1);
}
```

See the [complete example](#) on GitHub.

Detach a role policy

To detach a policy from a role, call the `IamClient`'s `detachRolePolicy` method, providing it with the role name and policy ARN in a [DetachRolePolicyRequest](#).

Imports

```
import software.amazon.awssdk.services.iam.model.DetachRolePolicyRequest;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.iam.IamClient;
import software.amazon.awssdk.services.iam.model.IamException;
```

Code

```
public static void detachPolicy(IamClient iam, String roleName, String policyArn ) {

    try {
        DetachRolePolicyRequest request = DetachRolePolicyRequest.builder()
            .roleName(roleName)
            .policyArn(policyArn)
            .build();

        iam.detachRolePolicy(request);
        System.out.println("Successfully detached policy " + policyArn +
            " from role " + roleName);

    } catch (IamException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
```

See the [complete example](#) on GitHub.

More information

- [Overview of IAM Policies](#) in the IAM User Guide.
- [AWS IAM Policy Reference](#) in the IAM User Guide.
- [CreatePolicy](#) in the IAM API Reference
- [GetPolicy](#) in the IAM API Reference
- [AttachRolePolicy](#) in the IAM API Reference
- [ListAttachedRolePolicies](#) in the IAM API Reference
- [DetachRolePolicy](#) in the IAM API Reference

Work with IAM server certificates

To enable HTTPS connections to your website or application on AWS, you need an SSL/TLS *server certificate*. You can use a server certificate provided by AWS Certificate Manager or one that you obtained from an external provider.

We recommend that you use ACM to provision, manage, and deploy your server certificates. With ACM you can request a certificate, deploy it to your AWS resources, and let ACM handle certificate renewals for you. Certificates provided by ACM are free. For more information about ACM, see the [ACM User Guide](#).

Get a server certificate

You can retrieve a server certificate by calling the `IamClient`'s `getServerCertificate` method, passing it a [GetServerCertificateRequest](#) with the certificate's name.

Imports

```
import software.amazon.awssdk.services.iam.model.GetServerCertificateRequest;
import software.amazon.awssdk.services.iam.model.GetServerCertificateResponse;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.iam.IamClient;
import software.amazon.awssdk.services.iam.model.IamException;
```

Code

```
public static void getCertificate(IamClient iam,String certName ) {
    try {
        GetServerCertificateRequest request = GetServerCertificateRequest.builder()
            .serverCertificateName(certName)
            .build();

        GetServerCertificateResponse response = iam.getServerCertificate(request);
        System.out.format("Successfully retrieved certificate with body %s",
            response.serverCertificate().certificateBody());
    } catch (IamException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
```

See the [complete example](#) on GitHub.

List server certificates

To list your server certificates, call the `IamClient`'s `listServerCertificates` method with a [ListServerCertificatesRequest](#). It returns a [ListServerCertificatesResponse](#).

Call the returned `ListServerCertificateResponse` object's `serverCertificateMetadataList` method to get a list of [ServerCertificateMetadata](#) objects that you can use to get information about each certificate.

Results may be truncated; if the `ListServerCertificateResponse` object's `isTruncated` method returns `true`, call the `ListServerCertificatesResponse` object's `marker` method and use the marker to create a new request. Use the new request to call `listServerCertificates` again to get the next batch of results.

Imports

```
import software.amazon.awssdk.services.iam.model.IamException;
import software.amazon.awssdk.services.iam.model.ListServerCertificatesRequest;
import software.amazon.awssdk.services.iam.model.ListServerCertificatesResponse;
import software.amazon.awssdk.services.iam.model.ServerCertificateMetadata;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.iam.IamClient;
```

Code

```
public static void listCertificates(IamClient iam) {

    try {
        boolean done = false;
        String newMarker = null;

        while(!done) {
            ListServerCertificatesResponse response;

            if (newMarker == null) {
                ListServerCertificatesRequest request =
                    ListServerCertificatesRequest.builder().build();
                response = iam.listServerCertificates(request);
            } else {
                ListServerCertificatesRequest request =
                    ListServerCertificatesRequest.builder()
                        .marker(newMarker).build();
                response = iam.listServerCertificates(request);
            }

            for(ServerCertificateMetadata metadata :
                response.serverCertificateMetadataList()) {
                System.out.printf("Retrieved server certificate %s",
                    metadata.serverCertificateName());
            }

            if(!response.isTruncated()) {
                done = true;
            } else {
                newMarker = response.marker();
            }
        }

    } catch (IamException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
```

See the [complete example](#) on GitHub.

Update a server certificate

You can update a server certificate's name or path by calling the `IamClient`'s `updateServerCertificate` method. It takes a [UpdateServerCertificateRequest](#) object set with the server certificate's current name and either a new name or new path to use.

Imports

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.iam.IamClient;
import software.amazon.awssdk.services.iam.model.IamException;
```

```
import software.amazon.awssdk.services.iam.model.UpdateServerCertificateRequest;
import software.amazon.awssdk.services.iam.model.UpdateServerCertificateResponse;
```

Code

```
public static void updateCertificate(IamClient iam, String curName, String newName) {
    try {
        UpdateServerCertificateRequest request =
            UpdateServerCertificateRequest.builder()
                .serverCertificateName(curName)
                .newServerCertificateName(newName)
                .build();

        UpdateServerCertificateResponse response =
            iam.updateServerCertificate(request);

        System.out.printf("Successfully updated server certificate to name %s",
            newName);
    } catch (IamException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
```

See the [complete example](#) on GitHub.

Delete a server certificate

To delete a server certificate, call the `IamClient`'s `deleteServerCertificate` method with a [DeleteServerCertificateRequest](#) containing the certificate's name.

Imports

```
import software.amazon.awssdk.services.iam.model.DeleteServerCertificateRequest;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.iam.IamClient;
import software.amazon.awssdk.services.iam.model.IamException;
```

Code

```
public static void deleteCert(IamClient iam, String certName ) {
    try {
        DeleteServerCertificateRequest request =
            DeleteServerCertificateRequest.builder()
                .serverCertificateName(certName)
                .build();

        iam.deleteServerCertificate(request);
        System.out.println("Successfully deleted server certificate " +
            certName);
    } catch (IamException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
```

See the [complete example](#) on GitHub.

More information

- [Working with Server Certificates](#) in the IAM User Guide
- [GetServerCertificate](#) in the IAM API Reference
- [ListServerCertificates](#) in the IAM API Reference
- [UpdateServerCertificate](#) in the IAM API Reference
- [DeleteServerCertificate](#) in the IAM API Reference
- [ACM User Guide](#)

Amazon Athena

Amazon Athena is a serverless, interactive query service to query data and analyze big data in Amazon S3 by using standard SQL. See the following resources for complete code examples with instructions.

[Link to Github](#)

[Link to Code Catalog](#)

Working with CloudWatch

This section provides examples of programming [CloudWatch](#) by using the AWS SDK for Java 2.0.

Amazon CloudWatch monitors your Amazon Web Services (AWS) resources and the applications you run on AWS in real time. You can use CloudWatch to collect and track metrics, which are variables you can measure for your resources and applications. CloudWatch alarms send notifications or automatically make changes to the resources you are monitoring based on rules that you define.

For more information about CloudWatch, see the [Amazon CloudWatch User Guide](#).

The following 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

- [Get metrics from CloudWatch \(p. 120\)](#)
- [Publish custom metric data \(p. 122\)](#)
- [Work with CloudWatch alarms \(p. 123\)](#)
- [Use alarm actions in CloudWatch \(p. 126\)](#)
- [Send events to CloudWatch \(p. 127\)](#)

Get metrics from CloudWatch

Listing metrics

To list CloudWatch metrics, create a [ListMetricsRequest](#) and call the CloudWatchClient's `listMetrics` method. You can use the `ListMetricsRequest` to filter the returned metrics by namespace, metric name, or dimensions.

Note

A list of metrics and dimensions that are posted by AWS services can be found within the [Amazon CloudWatch Metrics and Dimensions Reference](#) in the Amazon CloudWatch User Guide.

Imports

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.cloudwatch.CloudWatchClient;
import software.amazon.awssdk.services.cloudwatch.model.CloudWatchException;
import software.amazon.awssdk.services.cloudwatch.model.ListMetricsRequest;
import software.amazon.awssdk.services.cloudwatch.model.ListMetricsResponse;
import software.amazon.awssdk.services.cloudwatch.model.Metric;
```

Code

```
public static void listMets( CloudWatchClient cw, String namespace) {

    boolean done = false;
    String nextToken = null;

    try {
        while(!done) {

            ListMetricsResponse response;

            if (nextToken == null) {
                ListMetricsRequest request = ListMetricsRequest.builder()
                    .namespace(namespace)
                    .build();

                response = cw.listMetrics(request);
            } else {
                ListMetricsRequest request = ListMetricsRequest.builder()
                    .namespace(namespace)
                    .nextToken(nextToken)
                    .build();

                response = cw.listMetrics(request);
            }

            for (Metric metric : response.metrics()) {
                System.out.printf(
                    "Retrieved metric %s", metric.metricName());
                System.out.println();
            }

            if(response.nextToken() == null) {
                done = true;
            } else {
                nextToken = response.nextToken();
            }
        }
    } catch (CloudWatchException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
```

The metrics are returned in a [ListMetricsResponse](#) by calling its `getMetrics` method.

The results may be *paged*. To retrieve the next batch of results, call `nextToken` on the response object and use the token value to build a new request object. Then call the `listMetrics` method again with the new request.

See the [complete example](#) on GitHub.

More information

- [ListMetrics](#) in the Amazon CloudWatch API Reference.

Publish custom metric data

A number of AWS services publish [their own metrics](#) in namespaces beginning with "AWS/" You can also publish custom metric data using your own namespace (as long as it doesn't begin with "AWS/").

Publish custom metric data

To publish your own metric data, call the CloudWatchClient's `putMetricData` method with a [PutMetricDataRequest](#). The `PutMetricDataRequest` must include the custom namespace to use for the data, and information about the data point itself in a [MetricDatum](#) object.

Note

You cannot specify a namespace that begins with "AWS/". Namespaces that begin with "AWS/" are reserved for use by Amazon Web Services products.

Imports

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.cloudwatch.CloudWatchClient;
import software.amazon.awssdk.services.cloudwatch.model.Dimension;
import software.amazon.awssdk.services.cloudwatch.model.MetricDatum;
import software.amazon.awssdk.services.cloudwatch.model.StandardUnit;
import software.amazon.awssdk.services.cloudwatch.model.PutMetricDataRequest;
import software.amazon.awssdk.services.cloudwatch.model.PutMetricDataResponse;
import software.amazon.awssdk.services.cloudwatch.model.CloudWatchException;

import java.time.Instant;
import java.time.ZoneOffset;
import java.time.ZonedDateTime;
import java.time.format.DateTimeFormatter;
```

Code

```
public static void putMetData(CloudWatchClient cw, Double dataPoint ) {

    try {
        Dimension dimension = Dimension.builder()
            .name("UNIQUE_PAGES")
            .value("URLS")
            .build();

        // Set an Instant object
        String time =
            ZonedDateTime.now( ZoneOffset.UTC ).format( DateTimeFormatter.ISO_INSTANT );
        Instant instant = Instant.parse(time);

        MetricDatum datum = MetricDatum.builder()
            .metricName("PAGES_VISITED")
            .unit(StandardUnit.NONE)
            .value(dataPoint)
            .timestamp(instant)
            .dimensions(dimension).build();
```



```
PutMetricDataRequest request = PutMetricDataRequest.builder()
    .namespace("SITE/TRAFFIC")
    .metricData(datum).build();

cw.putMetricData(request);

} catch (CloudWatchException e) {
    System.err.println(e.awsErrorDetails().errorMessage());
    System.exit(1);
}
System.out.printf("Successfully put data point %f", dataPoint);
```

See the [complete example](#) on GitHub.

More information

- [Using Amazon CloudWatch Metrics](#) in the Amazon CloudWatch User Guide.
- [AWS Namespaces](#) in the Amazon CloudWatch User Guide.
- [PutMetricData](#) in the Amazon CloudWatch API Reference.

Work with CloudWatch alarms

Create an alarm

To create an alarm based on a CloudWatch metric, call the `CloudWatchClient`'s `putMetricAlarm` method with a [PutMetricAlarmRequest](#) filled with the alarm conditions.

Imports

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.cloudwatch.CloudWatchClient;
import software.amazon.awssdk.services.cloudwatch.model.Dimension;
import software.amazon.awssdk.services.cloudwatch.model.PutMetricAlarmRequest;
import software.amazon.awssdk.services.cloudwatch.model.ComparisonOperator;
import software.amazon.awssdk.services.cloudwatch.model.Statistic;
import software.amazon.awssdk.services.cloudwatch.model.StandardUnit;
import software.amazon.awssdk.services.cloudwatch.model.CloudWatchException;
```

Code

```
public static void putMetricAlarm(CloudWatchClient cw, String alarmName, String instanceId)
{
    try {
        Dimension dimension = Dimension.builder()
            .name("InstanceId")
            .value(instanceId).build();

        PutMetricAlarmRequest request = PutMetricAlarmRequest.builder()
            .alarmName(alarmName)
            .comparisonOperator(
                ComparisonOperator.GREATER_THAN_THRESHOLD)
            .evaluationPeriods(1)
            .metricName("CPUUtilization")
            .namespace("AWS/EC2")
            .period(60)
            .statistic(Statistic.AVERAGE)
            .threshold(70.0)
```

```
.actionsEnabled(false)
.alarmDescription(
    "Alarm when server CPU utilization exceeds 70%")
.unit(StandardUnit.SECONDS)
.dimensions(dimension)
.build();

cw.putMetricAlarm(request);
System.out.printf(
    "Successfully created alarm with name %s", alarmName);

} catch (CloudWatchException e) {
    System.err.println(e.awsErrorDetails().errorMessage());
    System.exit(1);
}
```

See the [complete example](#) on GitHub.

List alarms

To list the CloudWatch alarms that you have created, call the `CloudWatchClient`'s `describeAlarms` method with a [DescribeAlarmsRequest](#) that you can use to set options for the result.

Imports

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.cloudwatch.CloudWatchClient;
import software.amazon.awssdk.services.cloudwatch.model.CloudWatchException;
import software.amazon.awssdk.services.cloudwatch.model.DescribeAlarmsRequest;
import software.amazon.awssdk.services.cloudwatch.model.DescribeAlarmsResponse;
import software.amazon.awssdk.services.cloudwatch.model.MetricAlarm;
```

Code

```
public static void deleteCWAlarms( CloudWatchClient cw) {

    try {

        boolean done = false;
        String newToken = null;

        while(!done) {
            DescribeAlarmsResponse response;

            if (newToken == null) {
                DescribeAlarmsRequest request = DescribeAlarmsRequest.builder().build();
                response = cw.describeAlarms(request);
            } else {
                DescribeAlarmsRequest request = DescribeAlarmsRequest.builder()
                    .nextToken(newToken)
                    .build();
                response = cw.describeAlarms(request);
            }

            for(MetricAlarm alarm : response.metricAlarms()) {
                System.out.printf("\n Retrieved alarm %s", alarm.alarmName());
            }

            if(response.nextToken() == null) {
                done = true;
            } else {
                newToken = response.nextToken();
            }
        }
    }
}
```

```
    }  
}  
  
} catch (CloudWatchException e) {  
    System.err.println(e.awsErrorDetails().errorMessage());  
    System.exit(1);  
}  
System.out.printf("Done");
```

The list of alarms can be obtained by calling `MetricAlarms` on the [DescribeAlarmsResponse](#) that is returned by `describeAlarms`.

The results may be *paged*. To retrieve the next batch of results, call `nextToken` on the response object and use the token value to build a new request object. Then call the `describeAlarms` method again with the new request.

Note

You can also retrieve alarms for a specific metric by using the `CloudWatchClient`'s `describeAlarmsForMetric` method. Its use is similar to `describeAlarms`.

See the [complete example](#) on GitHub.

Delete alarms

To delete CloudWatch alarms, call the `CloudWatchClient`'s `deleteAlarms` method with a [DeleteAlarmsRequest](#) containing one or more names of alarms that you want to delete.

Imports

```
import software.amazon.awssdk.regions.Region;  
import software.amazon.awssdk.services.cloudwatch.CloudWatchClient;  
import software.amazon.awssdk.services.cloudwatch.model.CloudWatchException;  
import software.amazon.awssdk.services.cloudwatch.model.DeleteAlarmsRequest;
```

Code

```
public static void deleteCWAlarm(CloudWatchClient cw, String alarmName) {  
    try {  
        DeleteAlarmsRequest request = DeleteAlarmsRequest.builder()  
            .alarmNames(alarmName)  
            .build();  
  
        cw.deleteAlarms(request);  
        System.out.printf("Successfully deleted alarm %s", alarmName);  
  
    } catch (CloudWatchException e) {  
        System.err.println(e.awsErrorDetails().errorMessage());  
        System.exit(1);  
    }  
}
```

See the [complete example](#) on GitHub.

More information

- [Creating Amazon CloudWatch Alarms](#) in the Amazon CloudWatch User Guide
- [PutMetricAlarm](#) in the Amazon CloudWatch API Reference
- [DescribeAlarms](#) in the Amazon CloudWatch API Reference

- [DeleteAlarms](#) in the Amazon CloudWatch API Reference

Use alarm actions in CloudWatch

Using CloudWatch alarm actions, you can create alarms that perform actions such as automatically stopping, terminating, rebooting, or recovering Amazon EC2 instances.

Note

Alarm actions can be added to an alarm by using the [PutMetricAlarmRequest](#)'s `alarmActions` method when [creating an alarm](#) (p. 123).

Enable alarm actions

To enable alarm actions for a CloudWatch alarm, call the `CloudWatchClient`'s `enableAlarmActions` with a [EnableAlarmActionsRequest](#) containing one or more names of alarms whose actions you want to enable.

Imports

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.cloudwatch.CloudWatchClient;
import software.amazon.awssdk.services.cloudwatch.model.CloudWatchException;
import software.amazon.awssdk.services.cloudwatch.model.EnableAlarmActionsRequest;
import software.amazon.awssdk.services.cloudwatch.model.EnableAlarmActionsResponse;
```

Code

```
public static void enableActions(CloudWatchClient cw, String alarm) {
    try {
        EnableAlarmActionsRequest request = EnableAlarmActionsRequest.builder()
            .alarmNames(alarm).build();

        cw.enableAlarmActions(request);
        System.out.printf(
            "Successfully enabled actions on alarm %s", alarm);
    } catch (CloudWatchException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
```

See the [complete example](#) on GitHub.

Disable alarm actions

To disable alarm actions for a CloudWatch alarm, call the `CloudWatchClient`'s `disableAlarmActions` with a [DisableAlarmActionsRequest](#) containing one or more names of alarms whose actions you want to disable.

Imports

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.cloudwatch.CloudWatchClient;
import software.amazon.awssdk.services.cloudwatch.model.CloudWatchException;
import software.amazon.awssdk.services.cloudwatch.model.DisableAlarmActionsRequest;
```

Code

```
public static void disableActions(CloudWatchClient cw, String alarmName) {  
    try {  
        DisableAlarmActionsRequest request = DisableAlarmActionsRequest.builder()  
            .alarmNames(alarmName)  
            .build();  
  
        cw.disableAlarmActions(request);  
        System.out.printf(  
            "Successfully disabled actions on alarm %s", alarmName);  
    } catch (CloudWatchException e) {  
        System.err.println(e.awsErrorDetails().errorMessage());  
        System.exit(1);  
    }  
}
```

See the [complete example](#) on GitHub.

More information

- [Create Alarms to Stop, Terminate, Reboot, or Recover an Instance](#) in the Amazon CloudWatch User Guide
- [PutMetricAlarm](#) in the Amazon CloudWatch API Reference
- [EnableAlarmActions](#) in the Amazon CloudWatch API Reference
- [DisableAlarmActions](#) in the Amazon CloudWatch API Reference

Send events to CloudWatch

CloudWatch Events delivers a near real-time stream of system events that describe changes in AWS resources to Amazon EC2 instances, Lambda functions, Kinesis streams, Amazon ECS tasks, Step Functions state machines, Amazon SNS topics, Amazon SQS queues, or built-in targets. You can match events and route them to one or more target functions or streams by using simple rules.

Add events

To add custom CloudWatch events, call the CloudWatchEventsClient's `putEvents` method with a [PutEventsRequest](#) object that contains one or more [PutEventsRequestEntry](#) objects that provide details about each event. You can specify several parameters for the entry such as the source and type of the event, resources associated with the event, and so on.

Note

You can specify a maximum of 10 events per call to `putEvents`.

Imports

```
import software.amazon.awssdk.services.cloudwatch.model.CloudWatchException;  
import software.amazon.awssdk.services.cloudwatchevents.CloudWatchEventsClient;  
import software.amazon.awssdk.services.cloudwatchevents.model.PutEventsRequest;  
import software.amazon.awssdk.services.cloudwatchevents.model.PutEventsRequestEntry;  
import software.amazon.awssdk.services.cloudwatchevents.model.PutEventsResponse;
```

Code

```
public static void putCWEvents(CloudWatchEventsClient cwe, String resourceArn ) {
```

```
try {

    final String EVENT_DETAILS =
        "{ \"key1\": \"value1\", \"key2\": \"value2\" }";

    PutEventsRequestEntry requestEntry = PutEventsRequestEntry.builder()
        .detail(EVENT_DETAILS)
        .detailType("sampleSubmitted")
        .resources(resourceArn)
        .source("aws-sdk-java-cloudwatch-example")
        .build();

    PutEventsRequest request = PutEventsRequest.builder()
        .entries(requestEntry)
        .build();

    cwe.putEvents(request);
    System.out.println("Successfully put CloudWatch event");

} catch (CloudWatchException e) {
    System.err.println(e.awsErrorDetails().errorMessage());
    System.exit(1);
}
```

See the [complete example](#) on GitHub.

Add rules

To create or update a rule, call the `CloudWatchEventsClient`'s `putRule` method with a [PutRuleRequest](#) with the name of the rule and optional parameters such as the [event pattern](#), IAM role to associate with the rule, and a [scheduling expression](#) that describes how often the rule is run.

Imports

```
import software.amazon.awssdk.services.cloudwatch.model.CloudWatchException;
import software.amazon.awssdk.services.cloudwatchevents.CloudWatchEventsClient;
import software.amazon.awssdk.services.cloudwatchevents.model.PutRuleRequest;
import software.amazon.awssdk.services.cloudwatchevents.model.PutRuleResponse;
import software.amazon.awssdk.services.cloudwatchevents.model.RuleState;
```

Code

```
public static void putCWRule(CloudWatchEventsClient cwe, String ruleName, String roleArn) {

    try {
        PutRuleRequest request = PutRuleRequest.builder()
            .name(ruleName)
            .roleArn(roleArn)
            .scheduleExpression("rate(5 minutes)")
            .state(RuleState.ENABLED)
            .build();

        PutRuleResponse response = cwe.putRule(request);
        System.out.printf(
            "Successfully created CloudWatch events rule %s with arn %s",
            ruleArn, response.ruleArn());
    } catch (
        CloudWatchException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
```

```
}
```

See the [complete example](#) on GitHub.

Add targets

Targets are the resources that are invoked when a rule is triggered. Example targets include Amazon EC2 instances, Lambda functions, Kinesis streams, Amazon ECS tasks, Step Functions state machines, and built-in targets.

To add a target to a rule, call the CloudWatchEventsClient's `putTargets` method with a [PutTargetsRequest](#) containing the rule to update and a list of targets to add to the rule.

Imports

```
import software.amazon.awssdk.services.cloudwatch.model.CloudWatchException;
import software.amazon.awssdk.services.cloudwatchevents.CloudWatchEventsClient;
import software.amazon.awssdk.services.cloudwatchevents.model.PutTargetsRequest;
import software.amazon.awssdk.services.cloudwatchevents.model.PutTargetsResponse;
import software.amazon.awssdk.services.cloudwatchevents.model.Target;
```

Code

```
public static void putCWTargets(CloudWatchEventsClient cwe, String ruleName, String
    functionArn, String targetId ) {

    try {
        Target target = Target.builder()
            .arn(functionArn)
            .id(targetId)
            .build();

        PutTargetsRequest request = PutTargetsRequest.builder()
            .targets(target)
            .rule(ruleName)
            .build();

        PutTargetsResponse response = cwe.putTargets(request);
        System.out.printf(
            "Successfully created CloudWatch events target for rule %s",
            ruleName);
    } catch (CloudWatchException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
```

See the [complete example](#) on GitHub.

More information

- [Adding Events with PutEvents](#) in the Amazon CloudWatch Events User Guide
- [Schedule Expressions for Rules](#) in the Amazon CloudWatch Events User Guide
- [Event Types for CloudWatch Events](#) in the Amazon CloudWatch Events User Guide
- [Events and Event Patterns](#) in the Amazon CloudWatch Events User Guide
- [PutEvents](#) in the Amazon CloudWatch Events API Reference
- [PutTargets](#) in the Amazon CloudWatch Events API Reference

- [PutRule](#) in the Amazon CloudWatch Events API Reference

AWS CloudTrail

AWS CloudTrail is an AWS service that helps you enable governance, compliance, and operational and risk auditing of your AWS account. See the following resources for complete code examples with instructions.

[Link to Github](#)

[Link to Code Catalog](#)

Working with Amazon Cognito

With Amazon Cognito, you can quickly add user sign-up or sign-in capability to your web or mobile app. The examples here demonstrate some of the basic functionality of Cognito.

Create a user pool

A user pool is a directory of users that you can configure for your web or mobile app.

To create a user pool, start by building a [CreateUserPoolRequest](#) object, with the name of the user pool as the value of its `poolName()`. Call the `createUserPool()` method of your [CreateUserPoolRequest](#), passing in the `CreateUserPoolRequest` object. You can capture the result of this request as a [CreateUserPoolResponse](#) object, as demonstrated in the following code snippet.

Imports

```
import software.amazon.awssdk.regions.Region;
import
    software.amazon.awssdk.services.cognitoidentityprovider.CognitoIdentityProviderClient;
import
    software.amazon.awssdk.services.cognitoidentityprovider.model.CognitoIdentityProviderException;
import software.amazon.awssdk.services.cognitoidentityprovider.model.CreateUserPoolRequest;
import
    software.amazon.awssdk.services.cognitoidentityprovider.model.CreateUserPoolResponse;
```

Code

```
public static String createPool(CognitoIdentityProviderClient cognitoclient,String
    userPoolName ) {

    try {
        CreateUserPoolResponse repsonse = cognitoclient.createUserPool(
            CreateUserPoolRequest.builder()
                .poolName(userPoolName)
                .build()
        );
        return repsonse.userPool().id();

    } catch (CognitoIdentityProviderException e){
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
    return "";
```


See the [complete example](#) on GitHub.

List users from a user pool

To list users from your user pools, start by building a [ListUserPoolsRequest](#) object, with the number of maximum results as the value of its `maxResults()`. Call the `listUserPools()` method of your `CognitoIdentityProviderClient`, passing in the `ListUserPoolsRequest` object. You can capture the result of this request as a [ListUserPoolsResponse](#) object, as demonstrated in the following code snippet. Create a [UserPoolDescriptionType](#) object to easily iterate over the results and pull out the attributes of each user.

Imports

```
import software.amazon.awssdk.regions.Region;
import
    software.amazon.awssdk.services.cognitoidentityprovider.CognitoIdentityProviderClient;
import
    software.amazon.awssdk.services.cognitoidentityprovider.model.CognitoIdentityProviderException;
import software.amazon.awssdk.services.cognitoidentityprovider.model.ListUserPoolsResponse;
import software.amazon.awssdk.services.cognitoidentityprovider.model.ListUserPoolsRequest;
import
    software.amazon.awssdk.services.cognitoidentityprovider.model.UserPoolDescriptionType;
```

Code

```
public static void listAllUserPools(CognitoIdentityProviderClient cognitoclient ) {

    try {
        ListUserPoolsResponse response = cognitoclient
            .listUserPools(
                ListUserPoolsRequest.builder()
                    .maxResults(10)
                    .build()
            );

        for (UserPoolDescriptionType userpool : response.userPools()) {
            System.out.println("User pool " + userpool.name() + ", User ID " +
                userpool.id() + ", Status " + userpool.status());
        }
    } catch (CognitoIdentityProviderException e){
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
```

See the [complete example](#) on GitHub.

Create an identity pool

An identity pool is a container that organizes the IDs from your external identity provider, keeping a unique identifier for each user. To create an identity pool, start by building a [CreateIdentityPoolRequest](#) with the name of the user pool as the value of its `identityPoolName()`. Set `allowUnauthenticatedIdentities()` to true or false. Call the `createIdentityPool()` method of your `CognitoIdentityClient` object, passing in the `CreateIdentityPoolRequest` object. You can capture the result of this request as a [CreateIdentityPoolResponse](#) object, as demonstrated in the following code snippet.

Imports

```
import software.amazon.awssdk.regions.Region;
```

```
import software.amazon.awssdk.services.cognitoidentity.CognitoIdentityClient;
import software.amazon.awssdk.services.cognitoidentity.model.CreateIdentityPoolRequest;
import software.amazon.awssdk.services.cognitoidentity.model.CreateIdentityPoolResponse;
import
software.amazon.awssdk.services.cognitoidentityprovider.model.CognitoIdentityProviderException;
```

Code

```
public static String createIdPool(CognitoIdentityClient cognitoclient, String
identityPoolName ) {

    try {
        CreateIdentityPoolResponse response = cognitoclient.createIdentityPool(
            CreateIdentityPoolRequest.builder()
                .allowUnauthenticatedIdentities(false)
                .identityPoolName(identityPoolName)
                .build()
        );

        return response.identityPoolId();

    } catch (CognitoIdentityProviderException e){
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
    return "";
}
```

See the [complete example](#) on GitHub.

Add an app client

To enable the hosted web sign-up or sign-in UI for your app, create an app client. To create an app client, start by building a [CreateUserPoolClientRequest](#) object, with the name of the client as the value of its `clientName()`. Set `userPoolId()` to the ID of the user pool to which you want to attach this app client. Call the `createUserPoolClient()` method of your `CognitoIdentityProviderClient`, passing in the `CreateUserPoolClientRequest` object. You can capture the result of this request as a [CreateUserPoolClientResponse](#) object, as demonstrated in the following code snippet.

Imports

```
import software.amazon.awssdk.regions.Region;
import
software.amazon.awssdk.services.cognitoidentityprovider.CognitoIdentityProviderClient;
import
software.amazon.awssdk.services.cognitoidentityprovider.model.CognitoIdentityProviderException;
import
software.amazon.awssdk.services.cognitoidentityprovider.model.CreateUserPoolClientRequest;
import
software.amazon.awssdk.services.cognitoidentityprovider.model.CreateUserPoolClientResponse;
```

Code

```
public static void createPoolClient ( CognitoIdentityProviderClient cognitoclient,
String clientName,
String userPoolId ) {

    try {

        CreateUserPoolClientResponse repsonse = cognitoclient.createUserPoolClient(
            CreateUserPoolClientRequest.builder()
```

```
        .clientName(clientName)
        .userPoolId(userPoolId)
        .build()

    );

    System.out.println("User pool " + repsonse.userPoolClient().clientName() + "
created. ID: " + repsonse.userPoolClient().clientId());

    } catch (CognitoIdentityProviderException e){
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
```

See the [complete example](#) on GitHub.

Add a third-party identity provider

Adding an external identity provider (IdP) enables your users to log into your app using that service's login mechanism. To add a third-party IdP, start by building an [UpdateIdentityPoolRequest](#) object, with the name of the identity pool as the value of its `identityPoolName()`. Set `allowUnauthenticatedIdentities()` to true or false, specify the `identityPoolId()`, and define which login providers will be supported with `supportedLoginProviders()`. Call the `updateIdentityPool()` method of your `CognitoIdentityClient`, passing in the `UpdateIdentityPoolRequest` object. You can capture the result of this request as an [UpdateIdentityPoolResponse](#) object, as demonstrated in the following code snippet.

Imports

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.cognitoidentity.CognitoIdentityClient;
import software.amazon.awssdk.services.cognitoidentity.model.CognitoIdentityProvider;
import software.amazon.awssdk.services.cognitoidentity.model.UpdateIdentityPoolRequest;
import software.amazon.awssdk.services.cognitoidentity.model.UpdateIdentityPoolResponse;
import
    software.amazon.awssdk.services.cognitoidentityprovider.model.CognitoIdentityProviderException;
import java.util.ArrayList;
import java.util.List;
```

Code

```
public static void createAdmin(CognitoIdentityProviderClient cognitoclient,
                              String userPoolId,
                              String name,
                              String email){

    try{
        AdminCreateUserResponse response = cognitoclient.adminCreateUser(
            AdminCreateUserRequest.builder()
                .userPoolId(userPoolId)
                .username(name)
                .userAttributes(AttributeType.builder()
                    .name("email")
                    .value(email)
                    .build())
                .messageAction("SUPPRESS")
                .build()
            );

        System.out.println("User " + response.user().username() + "is created. Status: " +
            response.user().userStatus());
    }
```

```
} catch (CognitoIdentityProviderException e){
    System.err.println(e.awsErrorDetails().errorMessage());
    System.exit(1);
}
```

See the [complete example](#) on GitHub.

Get credentials for an ID

To get the credentials for an identity in an identity pool, first build a [GetCredentialsForIdentityRequest](#) with the identity ID as the value of its `identityId()`. Call the `getCredentialsForIdentity()` method of your `CognitoIdentityClient`, passing in the `GetCredentialsForIdentityRequest`. You can capture the result of this request as a [GetCredentialsForIdentityResponse](#) object, as demonstrated in the following code snippet.

Imports

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.cognitoidentity.CognitoIdentityClient;
import
    software.amazon.awssdk.services.cognitoidentity.model.GetCredentialsForIdentityRequest;
import
    software.amazon.awssdk.services.cognitoidentity.model.GetCredentialsForIdentityResponse;
import
    software.amazon.awssdk.services.cognitoidentityprovider.model.CognitoIdentityProviderException;
```

Code

```
public static void getCredsForIdentity(CognitoIdentityClient cognitoclient, String
    identityId) {

    try {
        GetCredentialsForIdentityRequest getCredentialsForIdentityRequest =
        GetCredentialsForIdentityRequest.builder()
            .identityId(identityId)
            .build();

        GetCredentialsForIdentityResponse response =
        cognitoclient.getCredentialsForIdentity(getCredentialsForIdentityRequest);
        System.out.println("Identity ID " + response.identityId() + ", Access key ID " +
        response.credentials().accessKeyId());

    } catch (CognitoIdentityProviderException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
```

See the [complete example](#) on GitHub.

For more information, see the [Amazon Cognito Developer Guide](#).

Amazon Comprehend

Amazon Comprehend is a natural language processing (NLP) service that uses machine learning to find insights and relationships in text. See the following resources for complete code examples with instructions.

[Link to Github](#)

[Link to Code Catalog](#)

Amazon EventBridge

Amazon EventBridge delivers a stream of real-time data from event sources, such as Zendesk, Datadog, or Pagerduty, and routes that data to targets like AWS Lambda. See the following resources for complete code examples with instructions.

[Link to Github](#)

[Link to Code Catalog](#)

Amazon Kinesis Data Firehose

Amazon Kinesis Data Firehose provides a simple way to capture, transform, and load streaming data. See the following resources for complete code examples with instructions.

[Link to Github](#)

[Link to Code Catalog](#)

Amazon Forecast examples

Amazon Forecast is a fully managed service for time-series forecasting. See the following resources for complete code examples with instructions.

[Link to Github](#)

[Link to Code Catalog](#)

Amazon S3 Glacier examples

Amazon S3 Glacier is an extremely low-cost storage service that provides secure, durable, and flexible storage for data backup and archival. See the following resources for complete code examples with instructions.

[Link to Github](#)

[Link to Code Catalog](#)

AWS Glue examples

With AWS Glue, you can fully manage, extract, transform, and load (ETL) your data for analytics. See the following resources for complete code examples with instructions.

[Link to Github](#)

[Link to Code Catalog](#)

Working with Kinesis

This section provides examples of programming [Amazon Kinesis](#) using the AWS SDK for Java 2.0.

For more information about Kinesis, see the [Amazon Kinesis Developer Guide](#).

The following 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

- [Subscribing to Amazon Kinesis Data Streams \(p. 136\)](#)

Subscribing to Amazon Kinesis Data Streams

The following examples show you how to retrieve and process data from Amazon Kinesis Data Streams using the `subscribeToShard` method. Kinesis Data Streams now employs the enhanced fanout feature and a low-latency HTTP/2 data retrieval API, making it easier for developers to run multiple low-latency, high-performance applications on the same Kinesis Data Stream.

Set Up

First, create an asynchronous Kinesis client and a [SubscribeToShardRequest](#) object. These objects are used in each of the following examples to subscribe to Kinesis events.

Imports

```
import java.util.concurrent.CompletableFuture;
import java.util.concurrent.atomic.AtomicInteger;
import org.reactivestreams.Subscriber;
import org.reactivestreams.Subscription;
import software.amazon.awssdk.core.async.SdkPublisher;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.kinesis.KinesisAsyncClient;
import software.amazon.awssdk.services.kinesis.model.ShardIteratorType;
import software.amazon.awssdk.services.kinesis.model.SubscribeToShardEvent;
import software.amazon.awssdk.services.kinesis.model.SubscribeToShardEventStream;
import software.amazon.awssdk.services.kinesis.model.SubscribeToShardRequest;
import software.amazon.awssdk.services.kinesis.model.SubscribeToShardResponse;
import software.amazon.awssdk.services.kinesis.model.SubscribeToShardResponseHandler;
```

Code

```
Region region = Region.US_EAST_1;
KinesisAsyncClient client = KinesisAsyncClient.builder()
    .region(region)
    .build();

SubscribeToShardRequest request = SubscribeToShardRequest.builder()
    .consumerARN(CONSUMER_ARN)
    .shardId("arn:aws:kinesis:us-east-1:111122223333:stream/StockTradeStream")
    .startingPosition(s -> s.type(ShardIteratorType.LATEST)).build();
```

Use the Builder Interface

You can use the builder method to simplify the creation of the [SubscribeToShardResponseHandler](#).

Using the builder, you can set each lifecycle callback with a method call instead of implementing the full interface.

Code

```
private static CompletableFuture<Void> responseHandlerBuilder(KinesisAsyncClient client,
    SubscribeToShardRequest request) {
    SubscribeToShardResponseHandler responseHandler = SubscribeToShardResponseHandler
        .builder()
        .onError(t -> System.err.println("Error during stream - " + t.getMessage()))
        .onComplete(() -> System.out.println("All records stream successfully"))
        // Must supply some type of subscriber
        .subscriber(e -> System.out.println("Received event - " + e))
        .build();
    return client.subscribeToShard(request, responseHandler);
}
```

For more control of the publisher, you can use the `publisherTransformer` method to customize the publisher.

Code

```
private static CompletableFuture<Void>
    responseHandlerBuilderPublisherTransformer(KinesisAsyncClient client,
        SubscribeToShardRequest request) {
    SubscribeToShardResponseHandler responseHandler = SubscribeToShardResponseHandler
        .builder()
        .onError(t -> System.err.println("Error during stream - " + t.getMessage()))
        .publisherTransformer(p -> p.filter(e -> e instanceof
        SubscribeToShardEvent).limit(100))
        .subscriber(e -> System.out.println("Received event - " + e))
        .build();
    return client.subscribeToShard(request, responseHandler);
}
```

See the [complete example](#) on GitHub.

Use a Custom Response Handler

For full control of the subscriber and publisher, implement the [SubscribeToShardResponseHandler](#) interface.

In this example, you implement the `onEventStream` method, which allows you full access to the publisher. This demonstrates how to transform the publisher to event records for printing by the subscriber.

Code

```
private static CompletableFuture<Void> responseHandlerBuilderClassic(KinesisAsyncClient
    client, SubscribeToShardRequest request) {
    SubscribeToShardResponseHandler responseHandler = new SubscribeToShardResponseHandler()
    {
        @Override
```

```
public void responseReceived(SubscribeToShardResponse response) {
    System.out.println("Receieved initial response");
}

@Override
public void onEventStream(SdkPublisher<SubscribeToShardEventStream> publisher) {
    publisher
        // Filter to only SubscribeToShardEvents
        .filter(SubscribeToShardEvent.class)
        // Flat map into a publisher of just records
        .flatMapIterable(SubscribeToShardEvent::records)
        // Limit to 1000 total records
        .limit(1000)
        // Batch records into lists of 25
        .buffer(25)
        // Print out each record batch
        .subscribe(batch -> System.out.println("Record Batch - " + batch));
}

@Override
public void complete() {
    System.out.println("All records stream successfully");
}

@Override
public void exceptionOccurred(Throwable throwable) {
    System.err.println("Error during stream - " + throwable.getMessage());
}
};
return client.subscribeToShard(request, responseHandler);
}
```

See the [complete example](#) on GitHub.

Use the Visitor Interface

You can use a [Visitor](#) object to subscribe to specific events you're interested in watching.

Code

```
private static CompletableFuture<Void>
responseHandlerBuilderVisitorBuilder(KinesisAsyncClient client, SubscribeToShardRequest
request) {
    SubscribeToShardResponseHandler.Visitor visitor =
    SubscribeToShardResponseHandler.Visitor
        .builder()
        .onSubscribeToShardEvent(e -> System.out.println("Received subscribe to shard
event " + e))
        .build();
    SubscribeToShardResponseHandler responseHandler = SubscribeToShardResponseHandler
        .builder()
        .onError(t -> System.err.println("Error during stream - " + t.getMessage()))
        .subscriber(visitor)
        .build();
    return client.subscribeToShard(request, responseHandler);
}
```

See the [complete example](#) on GitHub.

Use a Custom Subscriber

You can also implement your own custom subscriber to subscribe to the stream.

This code snippet shows an example subscriber.

Code

```
private static class MySubscriber implements Subscriber<SubscribeToShardEventStream> {

    private Subscription subscription;
    private AtomicInteger eventCount = new AtomicInteger(0);

    @Override
    public void onSubscribe(Subscription subscription) {
        this.subscription = subscription;
        this.subscription.request(1);
    }

    @Override
    public void onNext(SubscribeToShardEventStream shardSubscriptionEventStream) {
        System.out.println("Received event " + shardSubscriptionEventStream);
        if (eventCount.incrementAndGet() >= 100) {
            // You can cancel the subscription at any time if you wish to stop receiving
            events.
            subscription.cancel();
        }
        subscription.request(1);
    }

    @Override
    public void onError(Throwable throwable) {
        System.err.println("Error occurred while stream - " + throwable.getMessage());
    }

    @Override
    public void onComplete() {
        System.out.println("Finished streaming all events");
    }
}
```

You can pass that custom subscriber to the subscribe method, similarly to preview examples. The following code snippet shows this example.

Code

```
private static CompletableFuture<Void> responseHandlerBuilderSubscriber(KinesisAsyncClient
    client, SubscribeToShardRequest request) {
    SubscribeToShardResponseHandler responseHandler = SubscribeToShardResponseHandler
        .builder()
        .onError(t -> System.err.println("Error during stream - " + t.getMessage()))
        .subscriber(MySubscriber::new)
        .build();
    return client.subscribeToShard(request, responseHandler);
}
```

See the [complete example](#) on GitHub.

Write data records into a Kinesis data stream

You can use the [KinesisClient](#) object to write data records into a Kinesis data stream by using the `putRecords` method. To successfully invoke this method, create a [PutRecordsRequest](#) object. You pass the name of the data stream to the `streamName` method. Also you must pass the data by using the `putRecords` method (as shown in the following code example).

Imports

```
import java.net.URI;
import java.util.concurrent.CompletableFuture;

import io.reactivex.Flowable;
import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
import software.amazon.awssdk.core.async.SdkPublisher;
import software.amazon.awssdk.http.Protocol;
import software.amazon.awssdk.http.SdkHttpConfigurationOption;
import software.amazon.awssdk.http.nio.netty.NettyNioAsyncHttpClient;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.kinesis.KinesisAsyncClient;
import software.amazon.awssdk.services.kinesis.model.ShardIteratorType;
import software.amazon.awssdk.services.kinesis.model.StartingPosition;
import software.amazon.awssdk.services.kinesis.model.SubscribeToShardEvent;
import software.amazon.awssdk.services.kinesis.model.SubscribeToShardRequest;
import software.amazon.awssdk.services.kinesis.model.SubscribeToShardResponseHandler;
import software.amazon.awssdk.utils.AttributeMap;
```

In the following Java code example, notice that **StockTrade** object is used as the data to write to the Kinesis data stream. Before running this example, ensure that you have created the data stream.

Code

```
private static CompletableFuture<Void> responseHandlerBuilderSubscriber(KinesisAsyncClient
    client, SubscribeToShardRequest request) {
    SubscribeToShardResponseHandler responseHandler = SubscribeToShardResponseHandler
        .builder()
        .onError(t -> System.err.println("Error during stream - " + t.getMessage()))
        .subscriber(MySubscriber::new)
        .build();
    return client.subscribeToShard(request, responseHandler);
}
```

See the [complete example](#) on GitHub.

Use a Third-Party Library

You can use other third-party libraries instead of implementing a custom subscriber. This example demonstrates using the RxJava implementation, but you can use any library that implements the Reactive Streams interfaces. See the [RxJava wiki page on Github](#) for more information on that library.

To use the library, add it as a dependency. If you're using Maven, the example shows the POM snippet to use.

POM Entry

```
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
```

Imports

```
import java.net.URI;
import java.util.concurrent.CompletableFuture;

import io.reactivex.Flowable;
```

```
import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
import software.amazon.awssdk.core.async.SdkPublisher;
import software.amazon.awssdk.http.Protocol;
import software.amazon.awssdk.http.SdkHttpConfigurationOption;
import software.amazon.awssdk.http.nio.netty.NettyNioAsyncHttpClient;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.kinesis.KinesisAsyncClient;
import software.amazon.awssdk.services.kinesis.model.ShardIteratorType;
import software.amazon.awssdk.services.kinesis.model.StartingPosition;
import software.amazon.awssdk.services.kinesis.model.SubscribeToShardEvent;
import software.amazon.awssdk.services.kinesis.model.SubscribeToShardRequest;
import software.amazon.awssdk.services.kinesis.model.SubscribeToShardResponseHandler;
import software.amazon.awssdk.utils.AttributeMap;
```

This example uses RxJava in the `onEventStream` lifecycle method. This gives you full access to the publisher, which can be used to create an Rx Flowable.

Code

```
SubscribeToShardResponseHandler responseHandler = SubscribeToShardResponseHandler
    .builder()
    .onError(t -> System.err.println("Error during stream - " + t.getMessage()))
    .onEventStream(p -> Flowable.fromPublisher(p)
        .ofType(SubscribeToShardEvent.class)
        .flatMapIterable(SubscribeToShardEvent::records)
        .limit(1000)
        .buffer(25)
        .subscribe(e -> System.out.println("Record batch = " + e)))
    .build();
```

You can also use the `publisherTransformer` method with the Flowable publisher. You must adapt the Flowable publisher to an *SdkPublisher*, as shown in the following example.

Code

```
SubscribeToShardResponseHandler responseHandler = SubscribeToShardResponseHandler
    .builder()
    .onError(t -> System.err.println("Error during stream - " + t.getMessage()))
    .publisherTransformer(p -> SdkPublisher.adapt(Flowable.fromPublisher(p).limit(100)))
    .build();
```

See the [complete example](#) on GitHub.

More Information

- [SubscribeToShardEvent](#) in the Amazon Kinesis API Reference
- [SubscribeToShard](#) in the Amazon Kinesis API Reference

AWS KMS

AWS KMS is a secure and resilient service that uses hardware security modules that have been validated under FIPS 140-2, or are in the process of being validated, to protect your keys. See the following resources for complete code examples with instructions.

[Link to Github](#)

[Link to Code Catalog](#)

Invoke, list, and delete AWS Lambda functions

This section provides examples of programming with the Lambda service client by using the AWS SDK for Java 2.0.

Topics

- [Invoke a Lambda function \(p. 142\)](#)
- [List Lambda functions \(p. 143\)](#)
- [Delete a Lambda function \(p. 143\)](#)

Invoke a Lambda function

You can invoke a Lambda function by creating a [LambdaClient](#) object and invoking its `invoke` method. Create an [InvokeRequest](#) object to specify additional information such as the function name and the payload to pass to the Lambda function. Function names appear as `arn:aws:lambda:us-west-2:555556330391:function:HelloFunction`. You can retrieve the value by looking at the function in the AWS Console.

To pass payload data to a function, create a [SdkBytes](#) object that contains information. For example, in the following code example, notice the JSON data passed to the Lambda function.

Imports

```
import software.amazon.awssdk.services.lambda.LambdaClient;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.lambda.model.InvokeRequest;
import software.amazon.awssdk.core.SdkBytes;
import software.amazon.awssdk.services.lambda.model.InvokeResponse;
import software.amazon.awssdk.services.lambda.model.LambdaException;
```

Code

The following code example demonstrates how to invoke a Lambda function.

```
public static void invokeFunction(LambdaClient awsLambda, String functionName) {

    InvokeResponse res = null ;
    try {
        //Need a SdkBytes instance for the payload
        SdkBytes payload = SdkBytes.fromUtf8String("{\n" +
            "  \"Hello \": \"Paris\",\n" +
            "  \"countryCode\": \"FR\"\n" +
            "}" );

        //Setup an InvokeRequest
        InvokeRequest request = InvokeRequest.builder()
            .functionName(functionName)
            .payload(payload)
            .build();

        //Invoke the Lambda function
        res = awsLambda.invoke(request);
        String value = res.payload().asUtf8String() ;
        System.out.println(value);

    } catch(LambdaException e) {
        System.err.println(e.getMessage());
        System.exit(1);
    }
}
```

```
}
```

See the [complete example](#) on GitHub.

List Lambda functions

Build a [LambdaClient](#) object and invoke its `listFunctions` method. This method returns a [ListFunctionsResponse](#) object. You can invoke this object's `functions` method to return a list of [FunctionConfiguration](#) objects. You can iterate through the list to retrieve information about the functions. For example, the following Java code example shows how to get each function name.

Imports

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.lambda.LambdaClient;
import software.amazon.awssdk.services.lambda.model.LambdaException;
import software.amazon.awssdk.services.lambda.model.ListFunctionsResponse;
import software.amazon.awssdk.services.lambda.model.FunctionConfiguration;
import java.util.List;
```

Code

The following Java code example demonstrates how to retrieve a list of function names.

```
public static void listFunctions(LambdaClient awsLambda) {
    try {
        ListFunctionsResponse functionResult = awsLambda.listFunctions();
        List<FunctionConfiguration> list = functionResult.functions();

        for (FunctionConfiguration config: list) {
            System.out.println("The function name is "+config.functionName());
        }
    } catch (LambdaException e) {
        System.err.println(e.getMessage());
        System.exit(1);
    }
}
```

See the [complete example](#) on GitHub.

Delete a Lambda function

Build a [LambdaClient](#) object and invoke its `deleteFunction` method. Create a [DeleteFunctionRequest](#) object and pass it to the `deleteFunction` method. This object contains information such as the name of the function to delete. Function names appear as *arn:aws:lambda:us-west-2:555556330391:function:HelloFunction*. You can retrieve the value by looking at the function in the AWS Console.

Imports

```
import software.amazon.awssdk.services.lambda.LambdaClient;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.lambda.model.DeleteFunctionRequest;
import software.amazon.awssdk.services.lambda.model.LambdaException;
```

Code

The following Java code demonstrates how to delete a Lambda function.

```
public static void deleteLambdaFunction(LambdaClient awsLambda, String functionName ) {
    try {
        //Setup an DeleteFunctionRequest
        DeleteFunctionRequest request = DeleteFunctionRequest.builder()
            .functionName(functionName)
            .build();

        awsLambda.deleteFunction(request);
        System.out.println("The "+functionName +" function was deleted");

    } catch(LambdaException e) {
        System.err.println(e.getMessage());
        System.exit(1);
    }
}
```

See the [complete example](#) on GitHub.

WS Elemental MediaConvert

AWS Elemental MediaConvert is a file-based video processing service that allows video providers to transcode content for broadcast and multiscreen delivery. See the following resources for complete code examples with instructions.

[Link to Github](#)

[Link to Code Catalog](#)

AWS Elemental MediaStore examples

AWS Elemental MediaStore is an AWS storage service optimized for media. See the following resources for complete code examples with instructions.

[Link to Github](#)

[Link to Code Catalog](#)

AWS Elemental MediaStore examples

AWS Migration Hub provides a single place to monitor migrations in any AWS region where your migration tools are available. See the following resources for complete code examples with instructions.

[Link to Github](#)

[Link to Code Catalog](#)

Amazon Personalize examples

Amazon Personalize is a machine learning service that makes it easy for developers to create individualized recommendations for customers. See the following resources for complete code examples with instructions.

[Link to Github](#)

[Link to Code Catalog](#)

Working with Amazon Pinpoint

You can use Amazon Pinpoint to send relevant, personalized messages to your customers via multiple communication channels, such as push notifications, SMS, and email.

Create a project

A project (or application) in Amazon Pinpoint is a collection of settings, customer data, segments, and campaigns.

To create a project, start by building a [CreateApplicationRequest](#) object with the name of the project as the value of its `name()`. Then build a [CreateAppRequest](#) object, passing in the [CreateApplicationRequest](#) object as the value of its `createApplicationRequest()` method. Call the `createApp()` method of your [PinpointClient](#), passing in the [CreateAppRequest](#) object. Capture the result of this request as a [CreateAppResponse](#) object, as demonstrated in the following code snippet.

Imports

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.pinpoint.PinpointClient;
import software.amazon.awssdk.services.pinpoint.model.CreateAppRequest;
import software.amazon.awssdk.services.pinpoint.model.CreateAppResponse;
import software.amazon.awssdk.services.pinpoint.model.CreateApplicationRequest;
import software.amazon.awssdk.services.pinpoint.model.PinpointException;
```

Code

```
public static String createApplication(PinpointClient pinpoint, String appName) {
    try {
        CreateApplicationRequest appRequest = CreateApplicationRequest.builder()
            .name(appName)
            .build();

        CreateAppRequest request = CreateAppRequest.builder()
            .createApplicationRequest(appRequest)
            .build();

        CreateAppResponse result = pinpoint.createApp(request);

        return result.applicationResponse().id();
    } catch (PinpointException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
    return "";
}
```

See the [complete example](#) on GitHub.

Create a dynamic segment

A segment is a set of customers who share specific attributes, such as the city they live in or how frequently they visit your website. A dynamic segment is one that's based on attributes that you define, and can change over time.

To create a dynamic segment, first build all of the dimensions you want for this segment. For example, the following code snippet is set to include customers who were active on the site in the last 30 days. You can do this by first building a [RecencyDimension](#) object with the `duration()` and `recencyType()` you want (that is, `ACTIVE` or `INACTIVE`), and then passing this object to a [SegmentBehaviors](#) builder object as the value of `recency()`.

When you have defined your segment attributes, build them into a [SegmentDimensions](#) object. Then build a [WriteSegmentRequest](#) object, passing in the `SegmentDimensions` object as the value of its `dimensions()`. Next, build a [CreateSegmentRequest](#) object, passing in the `WriteSegmentRequest` object as the value of its `writeSegmentRequest()`. Finally, call the `createSegment()` method of your `PinpointClient`, passing in the `CreateSegmentRequest` object. Capture the result of this request as a [CreateSegmentResponse](#) object.

Imports

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.pinpoint.PinpointClient;
import software.amazon.awssdk.services.pinpoint.model.AttributeDimension;
import software.amazon.awssdk.services.pinpoint.model.SegmentResponse;
import software.amazon.awssdk.services.pinpoint.model.AttributeType;
import software.amazon.awssdk.services.pinpoint.model.RecencyDimension;
import software.amazon.awssdk.services.pinpoint.model.SegmentBehaviors;
import software.amazon.awssdk.services.pinpoint.model.SegmentDemographics;
import software.amazon.awssdk.services.pinpoint.model.SegmentLocation;
import software.amazon.awssdk.services.pinpoint.model.SegmentDimensions;
import software.amazon.awssdk.services.pinpoint.model.WriteSegmentRequest;
import software.amazon.awssdk.services.pinpoint.model.CreateSegmentRequest;
import software.amazon.awssdk.services.pinpoint.model.CreateSegmentResponse;
import software.amazon.awssdk.services.pinpoint.model.PinpointException;
import java.util.HashMap;
import java.util.Map;
```

Code

```
public static SegmentResponse createSegment(PinpointClient client, String appId) {
    try {
        Map<String, AttributeDimension> segmentAttributes = new HashMap<>();
        segmentAttributes.put("Team", AttributeDimension.builder()
            .attributeType(AttributeType.INCLUSIVE)
            .values("Lakers")
            .build());

        RecencyDimension recencyDimension = RecencyDimension.builder()
            .duration("DAY_30")
            .recencyType("ACTIVE")
            .build();

        SegmentBehaviors segmentBehaviors = SegmentBehaviors.builder()
            .recency(recencyDimension)
            .build();

        SegmentDemographics segmentDemographics = SegmentDemographics
            .builder()
            .build();

        SegmentLocation segmentLocation = SegmentLocation
            .builder()
            .build();

        SegmentDimensions dimensions = SegmentDimensions
            .builder()
```



```
        .attributes(segmentAttributes)
        .behavior(segmentBehaviors)
        .demographic(segmentDemographics)
        .location(segmentLocation)
        .build();

WriteSegmentRequest writeSegmentRequest = WriteSegmentRequest.builder()
    .name("MySegment")
    .dimensions(dimensions)
    .build();

CreateSegmentRequest createSegmentRequest = CreateSegmentRequest.builder()
    .applicationId(appId)
    .writeSegmentRequest(writeSegmentRequest)
    .build();

CreateSegmentResponse createSegmentResult =
client.createSegment(createSegmentRequest);
System.out.println("Segment ID: " + createSegmentResult.segmentResponse().id());
System.out.println("Done");
return createSegmentResult.segmentResponse();

} catch (PinpointException e) {
    System.err.println(e.awsErrorDetails().errorMessage());
    System.exit(1);
}
return null;
}
```

See the [complete example](#) on GitHub.

Import a static segment

A static segment is one you create and import from outside of Amazon Pinpoint. The following example code shows how to create a static segment by importing it from Amazon S3.

Prerequisite

Before you can complete this example, you need to create an IAM role that grants Amazon Pinpoint access to Amazon S3. For more information, see [IAM role for importing endpoints or segments](#) in the Amazon Pinpoint Developer Guide.

To import a static segment, start by building an [ImportJobRequest](#) object. In the builder, specify the `s3Url()`, `roleArn()`, and `format()`.

Note

For more information about the properties of an `ImportJobRequest`, see [the ImportJobRequest section of Import Jobs](#) in the Amazon Pinpoint API Reference.

Then build a [CreateImportJobRequest](#) object, passing in the `ImportJobRequest` object as the value of its `importJobRequest()`, and the ID of your project as the `applicationId()`. Call the `createImportJob()` method of your `PinpointClient`, passing in the `CreateImportJobRequest` object. Capture the result of this request as a `CreateImportJobResponse` object, as demonstrated in the following code snippet.

Imports

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.pinpoint.PinpointClient;
import software.amazon.awssdk.services.pinpoint.model.CreateImportJobRequest;
import software.amazon.awssdk.services.pinpoint.model.ImportJobResponse;
```

```
import software.amazon.awssdk.services.pinpoint.model.ImportJobRequest;
import software.amazon.awssdk.services.pinpoint.model.Format;
import software.amazon.awssdk.services.pinpoint.model.CreateImportJobResponse;
import software.amazon.awssdk.services.pinpoint.model.PinpointException;
```

Code

```
public static ImportJobResponse createImportSegment(PinpointClient client,
                                                    String appId,
                                                    String bucket,
                                                    String key,
                                                    String roleArn) {

    try {
        ImportJobRequest importRequest = ImportJobRequest.builder()
            .defineSegment(true)
            .registerEndpoints(true)
            .roleArn(roleArn)
            .format(Format.JSON)
            .s3Url("s3://" + bucket + "/" + key)
            .build();

        CreateImportJobRequest jobRequest = CreateImportJobRequest.builder()
            .importJobRequest(importRequest)
            .applicationId(appId)
            .build();

        CreateImportJobResponse jobResponse = client.createImportJob(jobRequest);

        return jobResponse.importJobResponse();

    } catch (PinpointException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }

    return null;
}
```

See the [complete example](#) on GitHub.

List segments for your project

To list the segments associated with a particular project, start by building a [GetSegmentsRequest](#) object, with the ID of the project as the value of its `applicationId()`. Next, call the `getSegments()` method of your `PinpointClient`, passing in the `GetSegmentsRequest` object. Capture the result of this request as a [GetSegmentsResponse](#) object. Finally, instantiate a [List](#) object upcasted to the [SegmentResponse](#) class. Then call the `segmentsResponse().item()` of `GetSegmentsResponse`, as demonstrated in the following code snippet. From there, you can iterate through the results.

Imports

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.pinpoint.PinpointClient;
import software.amazon.awssdk.services.pinpoint.model.GetSegmentsRequest;
import software.amazon.awssdk.services.pinpoint.model.GetSegmentsResponse;
import software.amazon.awssdk.services.pinpoint.model.PinpointException;
import software.amazon.awssdk.services.pinpoint.model.SegmentResponse;
import java.util.List;
```

Code

```
public static void listSegs( PinpointClient pinpoint, String appId) {

    try {
        GetSegmentsRequest request = GetSegmentsRequest.builder()
            .applicationId(appId)
            .build();

        GetSegmentsResponse response = pinpoint.getSegments(request);
        List<SegmentResponse> segments = response.segmentsResponse().item();

        for(SegmentResponse segment: segments) {
            System.out.println("Segment " + segment.id() + " " + segment.name() + " " +
            segment.lastModifiedDate());
        }
    } catch ( PinpointException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
```

See the [complete example](#) on GitHub.

Create a campaign

A campaign is an initiative meant to engage a particular audience segment by sending messages to those customers.

To create a campaign, first build all of the settings you want for this campaign. In the following code snippet, for example, the campaign will start immediately because the `startTime()` of the [Schedule](#) is set to `IMMEDIATE`. To set it to start at a specific time instead, specify a time in ISO 8601 format.

Note

For more information about the settings available for campaigns, see the **Schedule** section of [Campaigns](#) in the Amazon Pinpoint API Reference.

After you define your campaign configuration, build it into a [WriteCampaignRequest](#) object. None of the methods of the `builder()` of the `WriteCampaignRequest` are required. But you do need to include any of the configuration settings ([MessageConfiguration](#)) that you set for the campaign. We also recommend that you include a name and a description for your campaign so you can easily distinguish it from other campaigns. Call the `createCampaign()` method of your `PinpointClient`, passing in the `WriteCampaignRequest` object. Capture the result of this request as a [CreateCampaignResponse](#) object.

Imports

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.pinpoint.PinpointClient;
import software.amazon.awssdk.services.pinpoint.model.CampaignResponse;
import software.amazon.awssdk.services.pinpoint.model.Message;
import software.amazon.awssdk.services.pinpoint.model.Schedule;
import software.amazon.awssdk.services.pinpoint.model.Action;
import software.amazon.awssdk.services.pinpoint.model.MessageConfiguration;
import software.amazon.awssdk.services.pinpoint.model.WriteCampaignRequest;
import software.amazon.awssdk.services.pinpoint.model.CreateCampaignResponse;
import software.amazon.awssdk.services.pinpoint.model.CreateCampaignRequest;
import software.amazon.awssdk.services.pinpoint.model.PinpointException;
```

Code

```
public static void createPinCampaign(PinpointClient pinpoint, String appId, String
segmentId) {
```

```

CampaignResponse result = createCampaign(pinpoint, appId, segmentId);
System.out.println("Campaign " + result.name() + " created.");
System.out.println(result.description());
}

public static CampaignResponse createCampaign(PinpointClient client, String appId, String
segmentID) {
    try {
        Schedule schedule = Schedule.builder()
            .startTime("IMMEDIATE")
            .build();

        Message defaultMessage = Message.builder()
            .action(Action.OPEN_APP)
            .body("My message body.")
            .title("My message title.")
            .build();

        MessageConfiguration messageConfiguration = MessageConfiguration.builder()
            .defaultMessage(defaultMessage)
            .build();

        WriteCampaignRequest request = WriteCampaignRequest.builder()
            .description("My description")
            .schedule(schedule)
            .name("MyCampaign")
            .segmentId(segmentID)
            .messageConfiguration(messageConfiguration)
            .build();

        CreateCampaignResponse result = client.createCampaign(
            CreateCampaignRequest.builder()
                .applicationId(appId)
                .writeCampaignRequest(request).build()
        );

        System.out.println("Campaign ID: " + result.campaignResponse().id());

        return result.campaignResponse();
    } catch (PinpointException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }

    return null;
}

```

See the [complete example](#) on GitHub.

Send a message

To send an SMS text message through Amazon Pinpoint, first build an [AddressConfiguration](#) object to specify the `channelType()`. (In the following example, it's set to `ChannelType.SMS` to indicate the message will be sent via SMS.) Initialize a [HashMap](#) to store the destination phone number and the [AddressConfiguration](#) object. Next, build an [SMSMessage](#) object containing the relevant values. These include the `originationNumber`, the type of message (`messageType`), and the body of the message itself.

When you have created the message, build the `SMSMessage` object into a `DirectMessageConfiguration` object. Build your `Map` object and `DirectMessageConfiguration` object into a `MessageRequest` object. Build a `SendMessagesRequest` object, including your project ID (`applicationId`) and your `MessageRequest` object. Call the `sendMessages()` method of your `PinpointClient`, passing in the `SendMessagesRequest` object. Capture the result of this request as a `SendMessagesResponse` object.

Imports

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.pinpoint.PinpointClient;
import software.amazon.awssdk.services.pinpoint.model.DirectMessageConfiguration;
import software.amazon.awssdk.services.pinpoint.model.SMSMessage;
import software.amazon.awssdk.services.pinpoint.model.AddressConfiguration;
import software.amazon.awssdk.services.pinpoint.model.ChannelType;
import software.amazon.awssdk.services.pinpoint.model.MessageRequest;
import software.amazon.awssdk.services.pinpoint.model.SendMessagesRequest;
import software.amazon.awssdk.services.pinpoint.model.SendMessagesResponse;
import software.amazon.awssdk.services.pinpoint.model.MessageResponse;
import software.amazon.awssdk.services.pinpoint.model.PinpointException;
import java.util.HashMap;
import java.util.Map;
```

Code

```
public static void sendSMSMessage(PinpointClient pinpoint, String message, String appId,
    String originationNumber, String destinationNumber) {

    try {

        Map<String, AddressConfiguration> addressMap =
            new HashMap<String, AddressConfiguration>();

        AddressConfiguration addConfig = AddressConfiguration.builder()
            .channelType(ChannelType.SMS)
            .build();

        addressMap.put(destinationNumber, addConfig);

        SMSMessage smsMessage = SMSMessage.builder()
            .body(message)
            .messageType(messageType)
            .originationNumber(originationNumber)
            .senderId(senderId)
            .keyword(registeredKeyword)
            .build();

        // Create a DirectMessageConfiguration object
        DirectMessageConfiguration direct = DirectMessageConfiguration.builder()
            .smsMessage(smsMessage)
            .build();

        MessageRequest msgReq = MessageRequest.builder()
            .addresses(addressMap)
            .messageConfiguration(direct)
            .build();

        // create a SendMessagesRequest object
        SendMessagesRequest request = SendMessagesRequest.builder()
            .applicationId(appId)
            .messageRequest(msgReq)
            .build();

        SendMessagesResponse response= pinpoint.sendMessages(request);
```

```
MessageResponse msg1 = response.messageResponse();
Map map1 = msg1.result();

//Write out the result of sendMessage
map1.forEach((k, v) -> System.out.println((k + ":" + v)));

} catch (PinpointException e) {
    System.err.println(e.awsErrorDetails().errorMessage());
    System.exit(1);
}
```

See the [complete example](#) on GitHub.

For more information, see the [Amazon Pinpoint Developer Guide](#).

Amazon Polly examples

Amazon Polly is a service that turns text into lifelike speech, allowing you to create applications that talk, and build entirely new categories of speech-enabled functionality. See the following resources for complete code examples with instructions.

[Link to Github](#)

[Link to Code Catalog](#)

Amazon RDS

Amazon Relational Database Service (Amazon RDS) makes it easy to set up, operate, and scale a relational database in the cloud. See the following resources for complete code examples with instructions.

[Link to Github](#)

[Link to Code Catalog](#)

Amazon Redshift

Amazon Redshift is a fully managed, petabyte-scale data warehouse service in the cloud. See the following resources for complete code examples with instructions.

[Link to Github](#)

[Link to Code Catalog](#)

Amazon Rekognition

With Amazon Rekognition, you can perform fast and accurate face searches, allowing you to identify a person in a photo or video using your private repository of face images. You can also verify identity by analyzing a face image against images you have stored for comparison. See the following resources for complete code examples with instructions.

[Link to Github](#)

[Link to Code Catalog](#)

Amazon SageMaker

Amazon SageMaker is a fully managed service that provides every developer and data scientist with the ability to build, train, and deploy machine learning (ML) models. See the following resources for complete code examples with instructions.

[Link to Github](#)

[Link to Code Catalog](#)

AWS Secrets Manager

AWS Secrets Manager helps you protect secrets needed to access your applications, services, and IT resources. See the following resources for complete code examples with instructions.

[Link to Github](#)

[Link to Code Catalog](#)

Amazon Simple Email Service

Amazon Simple Email Service (SES) is a cost-effective, flexible, and scalable email service that enables developers to send mail from within any application. See the following resources for complete code examples with instructions.

[Link to Github](#)

[Link to Code Catalog](#)

Amazon Simple Notification Service examples

With Amazon Simple Notification Service, you can easily push real-time notification messages from your applications to subscribers over multiple communication channels. This topic describes how to perform some of the basic functions of SNS.

Create a topic

A **topic** is a logical grouping of communication channels that defines which systems to send a message to, for example, fanning out a message to AWS Lambda and an HTTP webhook. You send messages to Amazon SNS, then they're distributed to the channels defined in the topic. This makes the messages available to subscribers.

To create a topic, first build a [CreateTopicRequest](#) object, with the name of the topic set using the `name()` method in the builder. Then, send the request object to Amazon SNS by using the `createTopic()` method of the [SnsClient](#). You can capture the result of this request as a [CreateTopicResponse](#) object, as demonstrated in the following code snippet.

Imports

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sns.SnsClient;
import software.amazon.awssdk.services.sns.model.CreateTopicRequest;
import software.amazon.awssdk.services.sns.model.CreateTopicResponse;
import software.amazon.awssdk.services.sns.model.SnsException;
```

Code

```
public static String createSNSTopic(SnsClient snsClient, String topicName ) {

    CreateTopicResponse result = null;
    try {
        CreateTopicRequest request = CreateTopicRequest.builder()
            .name(topicName)
            .build();

        result = snsClient.createTopic(request);
        return result.topicArn();
    } catch (SnsException e) {

        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
    return "";
}
```

See the [complete example](#) on GitHub.

List your SNS topics

To retrieve a list of your existing Amazon SNS topics, build a [ListTopicsRequest](#) object. Then, send the request object to Amazon SNS by using the `listTopics()` method of the `SnsClient`. You can capture the result of this request as a [ListTopicsResponse](#) object.

The following code snippet prints out the HTTP status code of the request and a list of Amazon Resource Names (ARNs) for your Amazon SNS topics.

Imports

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sns.SnsClient;
import software.amazon.awssdk.services.sns.model.ListTopicsRequest;
import software.amazon.awssdk.services.sns.model.ListTopicsResponse;
import software.amazon.awssdk.services.sns.model.SnsException;
```

Code

```
public static void listSNSTopics(SnsClient snsClient) {

    try {
        ListTopicsRequest request = ListTopicsRequest.builder()
            .build();

        ListTopicsResponse result = snsClient.listTopics(request);
        System.out.println("Status was " + result.sdkHttpResponse().statusCode() + "\n\n"
            + result.topics());
    } catch (SnsException e) {
```



```
System.err.println(e.awsErrorDetails().errorMessage());
System.exit(1);
}
```

See the [complete example](#) on GitHub.

Subscribe an endpoint to a topic

After you create a topic, you can configure which communication channels will be endpoints for that topic. Messages are distributed to these endpoints after Amazon SNS receives them.

To configure a communication channel as an endpoint for a topic, subscribe that endpoint to the topic. To start, build a [SubscribeRequest](#) object. Specify the communication channel (for example, `lambda` or `email`) as the `protocol()`. Set the `endpoint()` to the relevant output location (for example, the ARN of a Lambda function or an email address), and then set the ARN of the topic to which you want to subscribe as the `topicArn()`. Send the request object to SNS by using the `subscribe()` method of the `SnsClient`. You can capture the result of this request as a [SubscribeResponse](#) object.

The following code snippet shows how to subscribe an email address to a topic.

Imports

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sns.SnsClient;
import software.amazon.awssdk.services.sns.model.SnsException;
import software.amazon.awssdk.services.sns.model.SubscribeRequest;
import software.amazon.awssdk.services.sns.model.SubscribeResponse;
```

Code

```
public static void subEmail(SnsClient snsClient, String topicArn, String email) {
    try {
        SubscribeRequest request = SubscribeRequest.builder()
            .protocol("email")
            .endpoint(email)
            .returnSubscriptionArn(true)
            .topicArn(topicArn)
            .build();

        SubscribeResponse result = snsClient.subscribe(request);
        System.out.println("Subscription ARN: " + result.subscriptionArn() + "\n\n Status
was " + result.sdkHttpResponse().statusCode());

    } catch (SnsException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
```

See the [complete example](#) on GitHub.

Publish a message to a topic

After you have a topic and one or more endpoints configured for it, you can publish a message to it. To start, build a [PublishRequest](#) object. Specify the `message()` to send, and the ARN of the topic (`topicArn()`) to send it to. Then, send the request object to Amazon SNS by using the `publish()` method of the `SnsClient`. You can capture the result of this request as a [PublishResponse](#) object.

Imports

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sns.SnsClient;
import software.amazon.awssdk.services.sns.model.PublishRequest;
import software.amazon.awssdk.services.sns.model.PublishResponse;
import software.amazon.awssdk.services.sns.model.SnsException;
```

Code

```
public static void pubTopic(SnsClient snsClient, String message, String topicArn) {

    try {
        PublishRequest request = PublishRequest.builder()
            .message(message)
            .topicArn(topicArn)
            .build();

        PublishResponse result = snsClient.publish(request);
        System.out.println(result.messageId() + " Message sent. Status was " +
            result.sdkHttpResponse().statusCode());

    } catch (SnsException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
```

See the [complete example](#) on GitHub.

Unsubscribe an endpoint from a topic

You can remove the communication channels configured as endpoints for a topic. After doing that, the topic itself continues to exist and distribute messages to any other endpoints configured for that topic.

To remove a communication channel as an endpoint for a topic, unsubscribe that endpoint from the topic. To start, build an [UnsubscribeRequest](#) object and set the ARN of the topic you want to unsubscribe from as the `subscriptionArn()`. Then send the request object to SNS by using the `unsubscribe()` method of the `SnsClient`. You can capture the result of this request as an [UnsubscribeResponse](#) object.

Imports

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sns.SnsClient;
import software.amazon.awssdk.services.sns.model.SnsException;
import software.amazon.awssdk.services.sns.model.UnsubscribeRequest;
import software.amazon.awssdk.services.sns.model.UnsubscribeResponse;
```

Code

```
public static void unSub(SnsClient snsClient, String subscriptionArn) {

    try {
        UnsubscribeRequest request = UnsubscribeRequest.builder()
            .subscriptionArn(subscriptionArn)
            .build();

        UnsubscribeResponse result = snsClient.unsubscribe(request);

        System.out.println("\n\nStatus was " + result.sdkHttpResponse().statusCode())
    }
}
```

```
        + "\n\nSubscription was removed for " + request.subscriptionArn());  
    } catch (SnsException e) {  
        System.err.println(e.awsErrorDetails().errorMessage());  
        System.exit(1);  
    }
```

See the [complete example](#) on GitHub.

Delete a topic

To delete an Amazon SNS topic, first build a [DeleteTopicRequest](#) object with the ARN of the topic set as the `topicArn()` method in the builder. Then send the request object to Amazon SNS by using the `deleteTopic()` method of the `SnsClient`. You can capture the result of this request as a [DeleteTopicResponse](#) object, as demonstrated in the following code snippet.

Imports

```
import software.amazon.awssdk.regions.Region;  
import software.amazon.awssdk.services.sns.SnsClient;  
import software.amazon.awssdk.services.sns.model.DeleteTopicRequest;  
import software.amazon.awssdk.services.sns.model.DeleteTopicResponse;  
import software.amazon.awssdk.services.sns.model.SnsException;
```

Code

```
public static void deleteSNSTopic(SnsClient snsClient, String topicArn ) {  
    try {  
        DeleteTopicRequest request = DeleteTopicRequest.builder()  
            .topicArn(topicArn)  
            .build();  
  
        DeleteTopicResponse result = snsClient.deleteTopic(request);  
        System.out.println("\n\nStatus was " + result.sdkHttpResponse().statusCode());  
    } catch (SnsException e) {  
        System.err.println(e.awsErrorDetails().errorMessage());  
        System.exit(1);  
    }  
}
```

See the [complete example](#) on GitHub.

For more information, see the [Amazon SNS Developer Guide](#).

Working with Amazon SQS

This section provides examples of programming [Amazon SQS](#) using the AWS SDK for Java 2.0.

The following 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

- [Work with Amazon SQS message queues \(p. 158\)](#)
- [Sending, Receiving, and Deleting Amazon SQS Messages \(p. 160\)](#)

Work 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`'s `createQueue` method, and provide a [CreateQueueRequest](#) object that describes the queue parameters.

Imports

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sqs.SqsClient;
import software.amazon.awssdk.services.sqs.model.*;
```

Code

```
CreateQueueRequest createQueueRequest = CreateQueueRequest.builder()
    .queueName(queueName)
    .build();

sqsClient.createQueue(createQueueRequest);
```

See the [complete sample](#) on GitHub.

List queues

To list the Amazon SQS queues for your account, call the `SqsClient`'s `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.regions.Region;
import software.amazon.awssdk.services.sqs.SqsClient;
import software.amazon.awssdk.services.sqs.model.*;
```

Code

```
String prefix = "que";

try {
    ListQueuesRequest listQueuesRequest =
        ListQueuesRequest.builder().queueNamePrefix(prefix).build();
    ListQueuesResponse listQueuesResponse = sqsClient.listQueues(listQueuesRequest);

    for (String url : listQueuesResponse.queueUrls()) {
        System.out.println(url);
    }
}
```

```
    }  
  } catch (SqsException e) {  
    System.err.println(e.awsErrorDetails().errorMessage());  
    System.exit(1);  
  }  
}
```

See the [complete sample](#) on GitHub.

Get the URL for a queue

Call the `SqsClient`'s `getQueueUrl` method, with a [GetQueueUrlRequest](#) object.

Imports

```
import software.amazon.awssdk.regions.Region;  
import software.amazon.awssdk.services.sqs.SqsClient;  
import software.amazon.awssdk.services.sqs.model.*;
```

Code

```
GetQueueUrlResponse getQueueUrlResponse =  
    sqsClient.getQueueUrl(GetQueueUrlRequest.builder().queueName(queueName).build());  
String queueUrl = getQueueUrlResponse.queueUrl();  
return queueUrl;  
  
} catch (SqsException e) {  
    System.err.println(e.awsErrorDetails().errorMessage());  
    System.exit(1);  
}  
return "";
```

See the [complete sample](#) on GitHub.

Delete a queue

Provide the queue's [URL \(p. 159\)](#) to the [DeleteQueueRequest](#) object. Then call the `SqsClient`'s `deleteQueue` method.

Imports

```
import software.amazon.awssdk.regions.Region;  
import software.amazon.awssdk.services.sqs.SqsClient;  
import software.amazon.awssdk.services.sqs.model.*;
```

Code

```
public static void deleteSQSQueue(SqsClient sqsClient, String queueName) {  
    try {  
        GetQueueUrlRequest getQueueRequest = GetQueueUrlRequest.builder()  
            .queueName(queueName)  
            .build();  
  
        String queueUrl = sqsClient.getQueueUrl(getQueueRequest).queueUrl();  
  
        DeleteQueueRequest deleteQueueRequest = DeleteQueueRequest.builder()  
            .queueUrl(queueUrl)
```

```
        .build();

        sqsClient.deleteQueue(deleteQueueRequest);

    } catch (SqsException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
```

See the [complete sample](#) on GitHub.

More information

- [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 received by distributed components. Messages are always delivered using an [SQS Queue](#) (p. 158).

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. 159), message body, and optional delay value (in seconds).

Imports

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sqs.SqsClient;
import software.amazon.awssdk.services.sqs.model.*;
```

Code

```
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`'s `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.regions.Region;
import software.amazon.awssdk.services.sqs.SqsClient;
import software.amazon.awssdk.services.sqs.model.*;
```

Code

```
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](#) on GitHub.

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.regions.Region;
import software.amazon.awssdk.services.sqs.SqsClient;
import software.amazon.awssdk.services.sqs.model.*;
```

Code

```
ReceiveMessageRequest receiveMessageRequest = ReceiveMessageRequest.builder()
    .queueUrl(queueUrl)
    .maxNumberOfMessages(5)
    .build();
List<Message> messages = sqsClient.receiveMessage(receiveMessageRequest).messages();
return messages;
} catch (SqsException e) {
    System.err.println(e.awsErrorDetails().errorMessage());
    System.exit(1);
}
return null;
```

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.regions.Region;
import software.amazon.awssdk.services.sqs.SqsClient;
import software.amazon.awssdk.services.sqs.model.*;
```

Code

```
try {
    for (Message message : messages) {
        DeleteMessageRequest deleteMessageRequest = DeleteMessageRequest.builder()
            .queueUrl(queueUrl)
            .receiptHandle(message.receiptHandle())
            .build();
        sqsClient.deleteMessage(deleteMessageRequest);
    }
}
```

See the [complete sample](#) on GitHub.

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

AWS Systems Manager Agent

AWS Systems Manager Agent (SSM Agent) is Amazon software that can be installed and configured on an EC2 instance, an on-premises server, or a virtual machine (VM). See the following resources for complete code examples with instructions.

[Link to Github](#)

[Link to Code Catalog](#)

Amazon SWF

The Amazon Simple Workflow Service (Amazon SWF) makes it easy to build applications that coordinate work across distributed components. See the following resources for complete code examples with instructions.

[Link to Github](#)

[Link to Code Catalog](#)

Amazon Textract

Amazon Textract is a fully managed machine learning service that automatically extracts text and data from scanned documents. See the following resources for complete code examples with instructions.

[Link to Github](#)

[Link to Code Catalog](#)

Working with Amazon Transcribe

This section provides examples of programming [Amazon Transcribe](#) using the AWS SDK for Java 2.0.

The following 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 Transcribe \(p. 163\)](#)

Working with Amazon Transcribe

The following example shows how bidirectional streaming works using Amazon Transcribe. Bidirectional streaming implies that there's both a stream of data going to the service and being received back in real time. The example uses Amazon Transcribe streaming transcription to send an audio stream and receive a stream of transcribed text back in real time.

See [Streaming Transcription](#) in the Amazon Transcribe Developer Guide to learn more about this feature.

See [Getting Started](#) in the Amazon Transcribe Developer Guide to get started using Amazon Transcribe.

Set up the Microphone

This code uses the `javax.sound.sampled` package to stream audio from an input device.

Code

```
import javax.sound.sampled.AudioFormat;
import javax.sound.sampled.AudioSystem;
import javax.sound.sampled.DataLine;
import javax.sound.sampled.TargetDataLine;

public class Microphone {

    public static TargetDataLine get() throws Exception {
        AudioFormat format = new AudioFormat(16000, 16, 1, true, false);
        DataLine.Info dataLineInfo = new DataLine.Info(TargetDataLine.class, format);

        TargetDataLine dataLine = (TargetDataLine) AudioSystem.getLine(dataLineInfo);
        dataLine.open(format);

        return dataLine;
    }
}
```

See the [complete example](#) on GitHub.

Create a Publisher

This code implements a publisher that publishes audio data from the Amazon Transcribe audio stream.

Code

```
public class AudioStreamPublisher implements Publisher<AudioStream> {
    private final InputStream inputStream;

    public AudioStreamPublisher(InputStream inputStream) {
        this.inputStream = inputStream;
    }
}
```

```
@Override
public void subscribe(Subscriber<? super AudioStream> s) {
    s.onSubscribe(new SubscriptionImpl(s, inputStream));
}

private class SubscriptionImpl implements Subscription {
    private static final int CHUNK_SIZE_IN_BYTES = 1024 * 1;
    private ExecutorService executor = Executors.newFixedThreadPool(1);
    private AtomicLong demand = new AtomicLong(0);

    private final Subscriber<? super AudioStream> subscriber;
    private final InputStream inputStream;

    private SubscriptionImpl(Subscriber<? super AudioStream> s, InputStream
inputStream) {
        this.subscriber = s;
        this.inputStream = inputStream;
    }

    @Override
    public void request(long n) {
        if (n <= 0) {
            subscriber.onError(new IllegalArgumentException("Demand must be
positive"));
        }

        demand.getAndAdd(n);

        executor.submit(() -> {
            try {
                do {
                    ByteBuffer audioBuffer = getNextEvent();
                    if (audioBuffer.remaining() > 0) {
                        AudioEvent audioEvent = audioEventFromBuffer(audioBuffer);
                        subscriber.onNext(audioEvent);
                    } else {
                        subscriber.onComplete();
                        break;
                    }
                } while (demand.decrementAndGet() > 0);
            } catch (TranscribeStreamingException e) {
                subscriber.onError(e);
            }
        });
    }

    @Override
    public void cancel() {
    }

    private ByteBuffer getNextEvent() {
        ByteBuffer audioBuffer;
        byte[] audioBytes = new byte[CHUNK_SIZE_IN_BYTES];

        int len = 0;
        try {
            len = inputStream.read(audioBytes);

            if (len <= 0) {
                audioBuffer = ByteBuffer.allocate(0);
            } else {
                audioBuffer = ByteBuffer.wrap(audioBytes, 0, len);
            }
        } catch (IOException e) {
            throw new UncheckedIOException(e);
        }
    }
}
```

```
        }

        return audioBuffer;
    }

    private AudioEvent audioEventFromBuffer(ByteBuffer bb) {
        return AudioEvent.builder()
            .audioChunk(SdkBytes.fromByteBuffer(bb))
            .build();
    }
}
```

See the [complete example](#) on GitHub.

Create the Client and Start the Stream

In the main method, create a request object, start the audio input stream and instantiate the publisher with the audio input.

You must also create a [StartStreamTranscriptionResponseHandler](#) to specify how to handle the response from Amazon Transcribe.

Then, use the `TranscribeStreamingAsyncClient`'s `startStreamTranscription` method to start the bidirectional streaming.

Imports

```
import javax.sound.sampled.AudioFormat;
import javax.sound.sampled.AudioSystem;
import javax.sound.sampled.DataLine;
import javax.sound.sampled.TargetDataLine;
import javax.sound.sampled.AudioInputStream;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.transcribestreaming.TranscribeStreamingAsyncClient;
import software.amazon.awssdk.services.transcribestreaming.model.TranscribeStreamingException;
import software.amazon.awssdk.services.transcribestreaming.model.StartStreamTranscriptionRequest;
import software.amazon.awssdk.services.transcribestreaming.model.MediaEncoding;
import software.amazon.awssdk.services.transcribestreaming.model.LanguageCode;
import software.amazon.awssdk.services.transcribestreaming.model.StartStreamTranscriptionResponseHandler;
import software.amazon.awssdk.services.transcribestreaming.model.TranscriptEvent;
```

Code

```
public static void convertAudio(TranscribeStreamingAsyncClient client) throws Exception {

    try {

        StartStreamTranscriptionRequest request =
            StartStreamTranscriptionRequest.builder()
                .mediaEncoding(MediaEncoding.PCM)
                .languageCode(LanguageCode.EN_US)
                .mediaSampleRateHertz(16_000).build();

        TargetDataLine mic = Microphone.get();
        mic.start();

        AudioStreamPublisher publisher = new AudioStreamPublisher(new
            AudioInputStream(mic));
```

```
        StartStreamTranscriptionResponseHandler response =
            StartStreamTranscriptionResponseHandler.builder().subscriber(e -> {
                TranscriptEvent event = (TranscriptEvent) e;
                event.transcript().results().forEach(r -> r.alternatives().forEach(a
-> System.out.println(a.transcript())));
            }).build();

        // Keeps Streaming until you end the Java program
        client.startStreamTranscription(request, publisher, response);

    } catch (TranscribeStreamingException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
```

See the [complete example](#) on GitHub.

More Info

- [How It Works](#) in the Amazon Transcribe Developer Guide.
- [Getting Started With Streaming Audio](#) in the Amazon Transcribe Developer Guide.
- [Guidelines and Limits](#) in the Amazon Transcribe Developer Guide.

Amazon Translate

Amazon Translate removes the complexity of building real-time and batch translation capabilities into your applications. See the following resources for complete code examples with instructions.

[Link to Github](#)

[Link to Code Catalog](#)

Amazon WorkDocs

Amazon WorkDocs is a fully managed, secure content creation, storage, and collaboration service. See the following resources for complete code examples with instructions.

[Link to Github](#)

[Link to Code Catalog](#)

Security for the AWS SDK for Java

Cloud security at Amazon Web Services (AWS) is the highest priority. As an AWS customer, you benefit from a data center and network architecture that is built to meet the requirements of the most security-sensitive organizations. Security is a shared responsibility between AWS and you. The [Shared Responsibility Model](#) describes this as Security of the Cloud and Security in the Cloud.

Security of the Cloud– AWS is responsible for protecting the infrastructure that runs all of the services offered in the AWS Cloud and providing you with services that you can use securely. Our security responsibility is the highest priority at AWS, and the effectiveness of our security is regularly tested and verified by third-party auditors as part of the [AWS Compliance Programs](#).

Security in the Cloud– Your responsibility is determined by the AWS service you are using, and other factors including the sensitivity of your data, your organization's requirements, and applicable laws and regulations.

Topics

- [Data protection in AWS SDK for Java 2.x \(p. 167\)](#)
- [AWS SDK for Java support for TLS 1.2 \(p. 168\)](#)
- [Identity and Access Management for this AWS Product or Service \(p. 169\)](#)
- [Compliance Validation for this AWS Product or Service \(p. 169\)](#)
- [Resilience for this AWS Product or Service \(p. 170\)](#)
- [Infrastructure Security for this AWS Product or Service \(p. 170\)](#)

Data protection in AWS SDK for Java 2.x

The [shared responsibility model](#) applies to data protection in this AWS product or service. As described in this model, AWS is responsible for protecting the global infrastructure that runs all of the AWS Cloud. You are responsible for maintaining control over your content that is hosted on this infrastructure. This content includes the security configuration and management tasks for the AWS services that you use. For more information about data privacy, see the [Data Privacy FAQ](#). For information about data protection in Europe, see the [AWS Shared Responsibility Model and GDPR](#) blog post on the AWS Security Blog.

For data protection purposes, we recommend that you protect AWS account credentials and set up individual user accounts with AWS Identity and Access Management (IAM). That way each user is given only the permissions necessary to fulfill their job duties. We also recommend that you secure your data in the following ways:

- Use multi-factor authentication (MFA) with each account.
- Use SSL/TLS to communicate with AWS resources. We recommend TLS 1.2 or later.
- Set up API and user activity logging with AWS CloudTrail.
- Use AWS encryption solutions, with all default security controls within AWS services.
- Use advanced managed security services such as Amazon Macie, which assists in discovering and securing personal data that is stored in Amazon S3.
- If you require FIPS 140-2 validated cryptographic modules when accessing AWS through a command line interface or an API, use a FIPS endpoint. For more information about the available FIPS endpoints, see [Federal Information Processing Standard \(FIPS\) 140-2](#).

We strongly recommend that you never put sensitive identifying information, such as your customers' account numbers, into free-form fields such as a **Name** field. This includes when you work with this AWS product or service or other AWS services using the console, API, AWS CLI, or AWS SDKs. Any data that you enter into this AWS product or service or other services might get picked up for inclusion in diagnostic logs. When you provide a URL to an external server, don't include credentials information in the URL to validate your request to that server.

AWS SDK for Java support for TLS 1.2

The following information applies only to Java SSL implementation (the default SSL implementation in the AWS SDK for Java). If you're using a different SSL implementation, see your specific SSL implementation to learn how to enforce TLS versions.

TLS support in Java

TLS 1.2 is supported starting in Java 7.

How to check the TLS version

To check what TLS version is supported in your Java virtual machine (JVM), you can use the following code.

```
System*.out.println(*Arrays*.toString(*SSLContext*.getDefault().getSupportedSSLParameters().getProtocol
```

To see the SSL handshake in action and what version of TLS is used, you can use the system property **javax.net.debug**.

```
java app.jar -Djavax.net.debug=ssl
```

How to set the TLS version

AWS SDK for Java 1.x

- Apache HTTP client: The SDK always prefers TLS 1.2 (if it's supported in the platform).

AWS SDK for Java 2.x

- ApacheHttpClient: The SDK always prefers TLS 1.2 (if it's supported in the platform).
- UrlHttpClientConnectionClient: To enforce only TLS 1.2, you can use this Java command.

```
java app.jar -Djdk.tls.client.protocols=TLSv1.2
```

Or use this code.

```
System.setProperty("jdk.tls.client.protocols", "TLSv1.2");
```

- NettyNioHttpClient: The SDK dependency for Netty is TLS 1.2 (if it's supported in the platform).

Identity and Access Management for this AWS Product or Service

AWS Identity and Access Management (IAM) is an Amazon Web Services (AWS) service that helps an administrator securely control access to AWS resources. IAM administrators control who can be *authenticated* (signed in) and *authorized* (have permissions) to use resources in AWS services. IAM is an AWS service that you can use with no additional charge.

To use this AWS product or service to access 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.

For details about working with IAM, see [AWS Identity and Access Management](#).

For an overview of IAM users and why they are important for the security of your account, see [AWS Security Credentials](#) in the [Amazon Web Services General Reference](#).

This AWS product or service follows the [shared responsibility model](#) through the specific Amazon Web Services (AWS) services it supports. For AWS service security information, see the [AWS service security documentation page](#) and [AWS services that are in scope of AWS compliance efforts by compliance program](#).

Compliance Validation for this AWS Product or Service

This AWS product or service follows the [shared responsibility model](#) through the specific Amazon Web Services (AWS) services it supports. For AWS service security information, see the [AWS service security documentation page](#) and [AWS services that are in scope of AWS compliance efforts by compliance program](#).

The security and compliance of AWS services is assessed by third-party auditors as part of multiple AWS compliance programs. These include SOC, PCI, FedRAMP, HIPAA, and others. AWS provides a frequently updated list of AWS services in scope of specific compliance programs at [AWS Services in Scope by Compliance Program](#).

Third-party audit reports are available for you to download using AWS Artifact. For more information, see [Downloading Reports in AWS Artifact](#).

For more information about AWS compliance programs, see [AWS Compliance Programs](#).

Your compliance responsibility when using this AWS product or service to access an AWS service is determined by the sensitivity of your data, your organization's compliance objectives, and applicable laws and regulations. If your use of an AWS service is subject to compliance with standards such as HIPAA, PCI, or FedRAMP, AWS provides resources to help:

- [Security and Compliance Quick Start Guides](#) – Deployment guides that discuss architectural considerations and provide steps for deploying security-focused and compliance-focused baseline environments on AWS.
- [Architecting for HIPAA Security and Compliance Whitepaper](#) – A whitepaper that describes how companies can use AWS to create HIPAA-compliant applications.
- [AWS Compliance Resources](#) – A collection of workbooks and guides that might apply to your industry and location.

- [AWS Config](#) – A service that assesses how well your resource configurations comply with internal practices, industry guidelines, and regulations.
- [AWS Security Hub](#) – A comprehensive view of your security state within AWS that helps you check your compliance with security industry standards and best practices.

Resilience for this AWS Product or Service

The Amazon Web Services (AWS) global infrastructure is built around AWS Regions and Availability Zones.

AWS Regions provide multiple physically separated and isolated Availability Zones, which are connected with low-latency, high-throughput, and highly redundant networking.

With Availability Zones, you can design and operate applications and databases that automatically fail over between Availability Zones without interruption. Availability Zones are more highly available, fault tolerant, and scalable than traditional single or multiple data center infrastructures.

For more information about AWS Regions and Availability Zones, see [AWS Global Infrastructure](#).

This AWS product or service follows the [shared responsibility model](#) through the specific Amazon Web Services (AWS) services it supports. For AWS service security information, see the [AWS service security documentation page](#) and [AWS services that are in scope of AWS compliance efforts by compliance program](#).

Infrastructure Security for this AWS Product or Service

This AWS product or service follows the [shared responsibility model](#) through the specific Amazon Web Services (AWS) services it supports. For AWS service security information, see the [AWS service security documentation page](#) and [AWS services that are in scope of AWS compliance efforts by compliance program](#).

Document history

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

This documentation was last built on: Jan 16, 2021

30 September 2020

Added topic for Waiters

30 May 2020

Added example topics for Amazon Pinpoint, Amazon Cognito, and Amazon SNS

29 May 2020

Added AWS Lambda function performance topic

27 April 2020

Added JVM TTL DNS caching topic

21 April 2020

New Maven and Gradle set up topics

20 April 2020

Added DynamoDB enhanced client topic

19 March 2020

Added TLS 1.2 to security section

2 August 2018

Added Kinesis stream examples

5 April 2018

Added auto pagination topic

29 December 2017

Added example topics for IAM, Amazon EC2, CloudWatch and DynamoDB

7 August 2017

Added getObject example for Amazon S3

4 August 2017

Added async topic

28 June 2017

AWS SDK for Java version 2 (v2) released