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 2.0 Developer Guide (Developer Preview) ................................................................. 1
What's New in Version 2.0 .......................................................................................................................... 1
Support for 1.0 ........................................................................................................................................... 1
Additional Resources ................................................................................................................................. 1
Contributing to the Developer Preview .................................................................................................... 1
Eclipse IDE Support .................................................................................................................................. 2
Developing AWS Applications for Android ............................................................................................. 2
Getting Started ......................................................................................................................................... 3
Sign up for AWS and Create an IAM User ............................................................................................... 3
Set up the SDK .......................................................................................................................................... 4
Prerequisites ............................................................................................................................................ 4
Including the SDK in Your Project ......................................................................................................... 4
Compiling the SDK .................................................................................................................................. 4
Installing a Java Development Environment .......................................................................................... 5
Choosing a JVM ........................................................................................................................................ 5
Set up AWS Credentials and Region ....................................................................................................... 5
Setting AWS Credentials ...................................................................................................................... 5
Setting the AWS Region .......................................................................................................................... 6
Using the SDK with Apache Maven ....................................................................................................... 7
Create a New Maven Package ................................................................................................................. 7
Configure the SDK as a Maven Dependency .......................................................................................... 8
Build Your Project ................................................................................................................................. 9
Using the SDK with Gradle .................................................................................................................... 9
Using the SDK ........................................................................................................................................ 11
Creating Service Clients ....................................................................................................................... 11
Obtaining a Client Builder ..................................................................................................................... 11
Using DefaultClient .............................................................................................................................. 11
Client Lifecycle ..................................................................................................................................... 12
Working with AWS Credentials ......................................................................................................... 12
Using the Default Credential Provider Chain ..................................................................................... 12
Specifying a Credential Provider or Provider Chain ........................................................................... 14
Explicitly Specifying Credentials ......................................................................................................... 15
More Info ............................................................................................................................................... 15
AWS Region Selection .......................................................................................................................... 15
Choosing a Region .................................................................................................................................. 15
Automatically Determine the AWS Region from the Environment ................................................... 16
Checking for Service Availability in an AWS Region ........................................................................... 17
Asynchronous Programming ............................................................................................................. 17
Non-streaming Operations ................................................................................................................... 17
Streaming Operations ........................................................................................................................... 18
Exception Handling .............................................................................................................................. 19
Why Unchecked Exceptions? .............................................................................................................. 19
AmazonServiceException (and Subclasses) ..................................................................................... 19
AmazonClientException ...................................................................................................................... 20
Logging AWS SDK for Java Calls ........................................................................................................ 20
Add the Log4J JAR ................................................................................................................................. 20
Log4j Configuration file ....................................................................................................................... 20
Setting the Classpath ............................................................................................................................ 20
Service-Specific Errors and Warnings ................................................................................................. 21
Request/Response Summary Logging ................................................................................................. 21
Verbose Wire Logging ......................................................................................................................... 22
Configure IAM Roles for Amazon EC2 (Advanced) ............................................................................ 22
Default Provider Chain and Amazon EC2 Instance Profiles ................................................................. 23
Walkthrough: Using IAM roles for Amazon EC2 Instances .................................................................... 23
The AWS SDK for Java provides a Java API for Amazon Web Services. Using the SDK, you can easily build Java applications that work with Amazon S3, Amazon EC2, DynamoDB, and more. We regularly add support for new services to the AWS SDK for Java. For a list of changes and features in a particular version, view the change log.

**What's New in Version 2.0**

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

**Important**

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

**Support for 1.0**

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

**Additional Resources**

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

- AWS SDK for Java 2.0 Reference
- Java developer blog
- Java developer forums
- GitHub:
  - Documentation source
  - SDK source
  - @awsforjava (Twitter)

**Contributing to the Developer Preview**

Developers can also contribute feedback through the following channels:

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

Eclipse IDE Support

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

Developing AWS Applications for Android

If you're an Android developer, Amazon Web Services publishes an SDK made specifically for Android development: the [AWS Mobile SDK for Android](https://aws.amazon.com/sdk-for- android/). See the [AWS Mobile SDK for Android Developer Guide](https://docs.aws.amazon.com/sdk-for-android/developer-guide/) for the complete documentation.
Getting Started with AWS SDK for Java 2.0 Developer Preview

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

Topics
- Sign up for AWS and Create an IAM User (p. 3)
- Set up the AWS SDK for Java 2.0 Developer Preview (p. 4)
- Set Up AWS Credentials and Region for Development (p. 5)
- Using the SDK with Apache Maven (p. 7)
- Using the SDK with Gradle (p. 9)

Sign up for AWS and Create an IAM User

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

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

To sign up for AWS
2. Follow the on-screen instructions. Part of the sign-up procedure involves receiving a phone call and entering a PIN using your phone keypad.

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

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

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

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

Important
There is no way to obtain the secret access key once you close the dialog box. You can, however, delete its associated access key ID and create a new one.
Next, set your credentials (p. 5) in the AWS shared credentials file or in the environment.

Set up the AWS SDK for Java 2.0 Developer Preview

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

Prerequisites

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

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

Including the SDK in Your Project

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

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

Note

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

Compiling the SDK

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

To compile the SDK

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

   **Note**
   Version 1.0 of the SDK is also available in GitHub at AWS SDK for Java 1.x (GitHub).

2. Click the Clone or download button to choose your download option.

3. In a terminal window, navigate to the directory where you downloaded the SDK source.

4. Build and install the SDK by using the following command (Maven required).

   `mvn clean install`
The resulting .jar file is built into the target directory.

5. (Optional) Build the API Reference documentation using the following command.

```
mvn javadoc:javadoc
```

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

## Installing a Java Development Environment

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

### Choosing a JVM

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

## Set Up AWS Credentials and Region for Development

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

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

- When using an EC2 instance, create an IAM role and then give your EC2 instance access to that role as shown in Configure IAM Roles for Amazon EC2 (Advanced) (p. 22).
- Set up AWS credentials within Eclipse using the AWS Toolkit for Eclipse. See Set up AWS Credentials in the AWS Toolkit for Eclipse User Guide.

### Setting AWS Credentials

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

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

This file should contain lines in the following format:

```
[default]
aws_access_key_id = your_access_key_id
aws_secret_access_key = your_secret_access_key
```
Substitute your own AWS credentials values for the values `your_access_key_id` and `your_secret_access_key`.

- Set the `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` environment variables.

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

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

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

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

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

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

### Setting the AWS Region

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

**Note**

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

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

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

  This file should contain lines in the following format:

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

  Substitute your desired AWS Region (for example, "us-west-2") for `your_aws_region`.

- Set the `AWS_REGION` environment variable.

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

  ```
  export AWS_REGION=your_aws_region
  ```

  On Windows, use `set`:

  ```
  set AWS_REGION=your_aws_region
  ```

  Where `your_aws_region` is the desired AWS Region name.
Using the SDK with Apache Maven

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

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

### Create a New Maven Package

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

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

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

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

To update to Java 1.8, add the following to your `pom.xml` file.

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

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

```
mvn archetype:generate -B  \
   -DarchetypeGroupId=pl.org.miki  \
   -DarchetypeArtifactId=java8-quickstart-archetype  \
   -DarchetypeVersion=1.0.0  \
   -DgroupId=com.example  \
   -DartifactId=sdk-sandbox  \
   -Dversion=1.0  \
   -Dpackage=com.example
```

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

**Note**
For much more information about creating and configuring Maven projects, see the Maven Getting Started Guide.
Configure the SDK as a Maven Dependency

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

Specifying Individual SDK Modules (Recommended)

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

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

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

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

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

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

Importing All SDK Modules (Not Recommended)

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

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

**Build Your Project**

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

```sh
grep 'mvn package'
```

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

**Using the SDK with Gradle**

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

**To configure the SDK for Gradle**

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

```gradle
buildscript {
    repositories {
        mavenCentral()
    }
    dependencies {
        classpath "io.spring.gradle:dependency-management-plugin:1.0.3.RELEASE"
    }
}
apply plugin: "io.spring.dependency-management"
```

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

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

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

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

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

Here's the complete `build.gradle` file:

```gradle

group 'aws.test'
version '1.0-SNAPSHOT'
apply plugin: 'java'
```
sourceCompatibility = 1.8
repositories {
  mavenCentral()
}

buildscript {
  repositories {
    mavenCentral()
  }
  dependencies {
    classpath "io.spring.gradle:dependency-management-plugin:1.0.3.RELEASE"
  }
}

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

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

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

**Note**
For more detail about specifying SDK dependencies using the BOM, see Using the SDK with Apache Maven (p. 7).
Using the AWS SDK for Java 2.0 Developer Preview

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

Topics
- Creating Service Clients (p. 11)
- Working with AWS Credentials (p. 12)
- AWS Region Selection (p. 15)
- Asynchronous Programming (p. 17)
- Exception Handling (p. 19)
- Logging AWS SDK for Java Calls (p. 20)
- Configure IAM Roles for Amazon EC2 (Advanced) (p. 22)

Creating Service Clients

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

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

Obtaining a Client Builder

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

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

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

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

Using DefaultClient

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

To create a default client

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

Client Lifecycle

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

To close a client

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

Working with AWS Credentials

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

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

**Important**

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

Using the Default Credential Provider Chain

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

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

You can create a credentials file by using the `aws configure` command provided by the AWS CLI. Or you can create it by editing the file with a text editor. For information about the credentials file format, see AWS Credentials File Format (p. 14).
4. **Amazon ECS container credentials**—loaded from the Amazon ECS if the environment variable `AWS_CONTAINER_CREDENTIALS_RELATIVE_URI` is set. The AWS SDK for Java uses the `ElasticContainerCredentialsProvider` to load these credentials.

5. **Instance profile credentials**—used on Amazon EC2 instances, and delivered through the Amazon EC2 metadata service. The AWS SDK for Java uses the `InstanceProfileCredentialsProvider` to load these credentials.

### Setting Credentials

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

- To specify credentials in the environment or in the default credential profiles file, see Set Up AWS Credentials and Region for Development (p. 5).
- To set Java system properties, see the System Properties tutorial on the official Java Tutorials website.
- To set up and use instance profile credentials with your EC2 instances, see Configure IAM Roles for Amazon EC2 (Advanced) (p. 22).

### Setting an Alternate Credentials Profile

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

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

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

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

On Windows, you use the following.

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

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

### Setting an Alternate Credentials File Location

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

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

To override the default credentials file location

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

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

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

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

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

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

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

To explicitly supply credentials to an AWS client

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

The following is an example.

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

More Info

- Sign up for AWS and Create an IAM User (p. 3)
- Set Up AWS Credentials and Region for Development (p. 5)
- Configure IAM Roles for Amazon EC2 (Advanced) (p. 22)

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.

```java
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.

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

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

### Automatically Determine the AWS Region from the Environment

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

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

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

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

### Default Region Provider Chain

**The following is the region lookup process:**

1. Any explicit region set by using `region` on the builder itself takes precedence over anything else.
2. The `AWS_REGION` environment variable is checked. If it's set, that region is used to configure the client.
   
   **Note**
   
   This environment variable is set by the Lambda container.
3. The SDK checks the AWS shared configuration file (usually located at `~/.aws/config`). If the `region` property is present, the SDK uses it.
   - The `AWS_CONFIG_FILE` environment variable can be used to customize the location of the shared config file.
   - The `AWS_PROFILE` environment variable or the `aws.profile` system property can be used to customize the profile that the SDK loads.
4. The SDK attempts to use the Amazon EC2 instance metadata service to determine the region of the currently running Amazon EC2 instance.
5. If the SDK still hasn't found a region by this point, client creation fails with an exception.

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

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

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

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

Asynchronous Programming

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

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

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

Non-streaming Operations

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

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

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

```java
public class DynamoDBAsync {
    public static void main(String[] args) {
        // Creates a default async client with credentials and regions loaded from the environment
        DynamoDBAsyncClient client = DynamoDBAsyncClient.create();
        CompletableFuture<ListTablesResponse> response =
                client.listTables(ListTablesRequest.builder()
                        .limit(5)
                        .build());
        // Map the response to another CompletableFuture containing just the table names
        CompletableFuture<List<String>> tableNames = response.thenApply(ListTablesResponse::tableNames);
        // When future is complete (either successfully or in error) handle the response
        tableNames.whenComplete((tables, err) -> {
            if (tables != null) {
```

17
Streaming Operations

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

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

```java
public class S3AsyncOps {
    private static final String BUCKET = "sample-bucket";
    private static final String KEY = "testfile.in";

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

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

```java
public class S3AsyncStreamOps {
    private static final String BUCKET = "sample-bucket";
    private static final String KEY = "testfile.out";

    public static void main(String[] args) {
        S3AsyncClient client = S3AsyncClient.create();
        final CompletableFuture<GetObjectResponse> futureGet = client.getObject(
            GetObjectRequest.builder()
                .bucket(BUCKET)
                .key(KEY)
                .build(),
        );
    }
}
```
Exception Handling

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

Why Unchecked Exceptions?

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

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

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

AmazonServiceException (and Subclasses)

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

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

AmazonServiceException provides you with information such as:

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

```java
AsyncResponseHandler.toFile(Paths.get("myfile.out"));
futureGet.whenComplete((resp, err) -> {
  try {
    if (resp != null) {
      System.out.println(resp);
    } else {
      // Handle error
      err.printStackTrace();
    }
  } finally {
    // Lets the application shut down. Only close the client when you are
completely done with it
    FunctionalUtils.invokeSafely(client::close);
  }
});
```
AmazonServiceException also includes information about whether the failed request was the caller's fault (a request with illegal values) or the AWS service's fault (an internal service error).

**AmazonClientException**

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

**Logging AWS SDK for Java Calls**

The AWS SDK for Java is instrumented with 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 Slf4j in Maven you can add the following to your pom.xml:

```xml
<dependency>
  <groupId>org.apache.logging.log4j</groupId>
  <artifactId>org.apache.logging.log4j</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.

```xml
<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.

```xml
<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>
    </Loggers>
</Configuration>
```
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.

```xml
<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.

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

Configure IAM Roles for Amazon EC2 (Advanced)

All requests to AWS services must be cryptographically signed using credentials issued by AWS. You can use IAM roles to conveniently grant secure access to AWS resources from your Amazon EC2 instances.
This topic provides information about how to use IAM roles with AWS SDK for Java applications running on Amazon EC2. For more information about IAM instances, see IAM Roles for Amazon EC2 in the Amazon EC2 User Guide for Linux Instances.

Default Provider Chain and Amazon EC2 Instance Profiles

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

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

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

For example:

```java
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.
This section provides programming examples using the AWS SDK for Java 2.0 that applies to specific use cases.

Topics
- Amazon S3 Examples Using the AWS SDK for Java (p. 25)
- Amazon SQS Examples Using the AWS SDK for Java (p. 30)
- CloudWatch Examples Using the AWS SDK for Java (p. 34)
- DynamoDB Examples Using the AWS SDK for Java (p. 43)
- Amazon EC2 Examples Using the AWS SDK for Java (p. 51)
- IAM Examples Using the AWS SDK for Java (p. 63)
- Retrieving Paginated Results (p. 77)

Amazon S3 Examples Using the AWS SDK for Java

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

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

Topics
- Creating, Listing, and Deleting Amazon S3 Buckets (p. 25)
- Performing Operations on an Amazon S3 Object (p. 28)

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.

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 Using the AWS SDK for Java 2.0 Developer Preview (p. 11), and have configured default AWS credentials using the information in Set Up AWS Credentials and Region for Development (p. 5).

Topics
- Create a Bucket (p. 26)
- List the Buckets (p. 26)
- Delete a Bucket (p. 27)

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

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

Code

```java
// Create bucket
CreateBucketRequest createBucketRequest = CreateBucketRequest
     .builder()
     .bucket(bucket)
     .createBucketConfiguration(CreateBucketConfiguration.builder()
                               .locationConstraint(region.value())
                               .build())
     .build();
```

See the complete example on GitHub.

List the Buckets

Build a ListBucketRequest. 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

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

Code

```java
// List buckets
```
ListBucketsRequest listBucketsRequest = ListBucketsRequest.builder().build();
ListBucketsResponse listBucketsResponse = s3.listBuckets(listBucketsRequest);

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. 27)
- [Delete an Empty Bucket](#) (p. 27)

### 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**

```java
import software.amazon.awssdk.services.s3.S3Client;
import software.amazon.awssdk.services.s3.model.CreateBucketConfiguration;
import software.amazon.awssdk.services.s3.model.CreateBucketRequest;
import software.amazon.awssdk.services.s3.model.DeleteBucketRequest;
import software.amazon.awssdk.services.s3.model.DeleteObjectRequest;
import software.amazon.awssdk.services.s3.model.ListObjectsV2Request;
import software.amazon.awssdk.services.s3.model.ListObjectsV2Response;
```

**Code**

```java
ListObjectsV2Response listObjectsV2Response;
do {
    listObjectsV2Response = s3.listObjectsV2(listObjectsV2Request);
    for (S3Object s3Object : listObjectsV2Response.contents()) {
        s3.deleteObject(DeleteObjectRequest.builder().bucket(bucket2).key(s3Object.key()).build());
    }
    listObjectsV2Request = ListObjectsV2Request.builder().bucket(bucket2)
        .continuationToken(listObjectsV2Response.nextContinuationToken())
        .build();
} while (listObjectsV2Response.isTruncated());
```

See the complete example on GitHub.

### Delete an Empty Bucket

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

**Imports**

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

Code

// Delete empty bucket
DeleteBucketRequest deleteBucketRequest = DeleteBucketRequest.builder().bucket(bucket).build();

See the complete example on GitHub.

Performing Operations on an Amazon S3 Object

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

<admonition>
<title>Best Practice</title>
We recommend that you enable the AbortIncompleteMultipartUpload lifecycle rule on your Amazon S3 buckets.

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

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

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

Topics
• Upload an Object (p. 28)
• Upload Objects in Multiple Parts (p. 29)
• Download an Object (p. 30)
• Delete an Object (p. 30)

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 software.amazon.awssdk.services.s3.S3Client;
import software.amazon.awssdk.services.s3.model.PutObjectRequest;
import software.amazon.awssdk.core.sync.RequestBody;

Code

// Put Object
s3.putObject(PutObjectRequest.builder().bucket(bucket).key(key))
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**

```java
import software.amazon.awssdk.services.s3.S3Client;
import software.amazon.awssdk.services.s3.model.CompleteMultipartUploadRequest;
import software.amazon.awssdk.services.s3.model.CompletedMultipartUpload;
import software.amazon.awssdk.services.s3.model.CompletedPart;
import software.amazon.awssdk.services.s3.model.CreateMultipartUploadRequest;
import software.amazon.awssdk.services.s3.model.CreateMultipartUploadResponse;
import software.amazon.awssdk.services.s3.model.UploadPartRequest;
```

**Code**

```java
// First create a multipart upload and get upload id
CreateMultipartUploadRequest createMultipartUploadRequest =
    CreateMultipartUploadRequest.builder()
    .bucket(bucketName).key(key)
    .build();
CreateMultipartUploadResponse response =
    s3.createMultipartUpload(createMultipartUploadRequest);
String uploadId = response.uploadId();
System.out.println(uploadId);

// Upload all the different parts of the object
UploadPartRequest uploadPartRequest1 =
    UploadPartRequest.builder().bucket(bucketName).key(key)
    .uploadId(uploadId)
    .partNumber(1).build();
String etag1 = s3.uploadPart(uploadPartRequest1, RequestBody.of(getRandomByteBuffer(5 * MB))).eTag();
CompletedPart part1 = CompletedPart.builder().partNumber(1).eTag(etag1).build();

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

// Finally call completeMultipartUpload operation to tell S3 to merge all uploaded
// parts and finish the multipart operation.
CompletedMultipartUpload completedMultipartUpload =
    CompletedMultipartUpload.builder().parts(part1, part2).build();
CompleteMultipartUploadRequest completeMultipartUploadRequest =
    CompleteMultipartUploadRequest.builder().bucket(bucketName).key(key).uploadId(uploadId)
    .multipartUpload(completedMultipartUpload).build();
s3.completeMultipartUpload(completeMultipartUploadRequest);
```
See the complete example 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 `StreamingResponseHandler` object. The `StreamingResponseHandler` creates a response handler that writes the response content to the specified file or stream.

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

**Imports**

```java
import software.amazon.awssdk.services.s3.S3Client;
import software.amazon.awssdk.services.s3.model.GetObjectRequest;
import software.amazon.awssdk.core.sync.StreamingResponseHandler;
```

**Code**

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

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**

```java
import software.amazon.awssdk.services.s3.S3Client;
import software.amazon.awssdk.services.s3.model.DeleteObjectRequest;
```

**Code**

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

See the complete example on GitHub.

**Amazon SQS Examples Using the AWS SDK for Java**

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

**Note**

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

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

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

### Create a Queue

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

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

System.out.println("\nCreate Queue");
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 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.

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

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

**Get the URL for a Queue**

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

**Imports**

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

**Code**

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

See the complete sample on GitHub.

**Delete a Queue**

Provide the queue’s URL (p. 32) to the `DeleteMessageRequest` object. Then call the `SQSClient.deleteQueue` method.

**Imports**

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

**Code**

```java
System.out.println("\nDelete Messages\n");
```

See the complete sample on GitHub.

**More Info**

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

**Sending, Receiving, and Deleting Amazon SQS Messages**

A message is a piece of data that can be sent and received by distributed components. Messages are always delivered using an SQS Queue (p. 31).
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. 32), message body, and optional delay value (in seconds).

Imports

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

Code

```java
// List queues with filters
String name_prefix = "queue";
ListQueuesRequest filterListRequest = ListQueuesRequest.builder()
    .queueNamePrefix(name_prefix).build();
ListQueuesResponse listQueuesFilteredResponse = sqsClient.listQueues(filterListRequest);
```

Send Multiple Messages in a Request

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

Imports

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

Code

```java
for (String url : listQueuesFilteredResponse.queueUrls()) {
    System.out.println(url);
}
System.out.println("\nSend message");
sqsClient.sendMessage(SendMessageRequest.builder()
    .queueUrl(queueUrl)
    .build());
```

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

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

Code

```java
```
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

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

Code

```java
List<Message> messages = sqsClient.receiveMessage(receiveMessageRequest).messages();

System.out.println("\nChange Message Visibility");
for (Message message : messages) {
    ChangeMessageVisibilityRequest req =
        ChangeMessageVisibilityRequest.builder().queueUrl(queueUrl)
            .receiptHandle(message.receiptHandle()).visibilityTimeout(100).build();
    sqsClient.changeMessageVisibility(req);
}
```

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

CloudWatch Examples Using the AWS SDK for Java

This section provides examples of programming CloudWatch 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.

Note

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

```java
import software.amazon.awssdk.services.cloudwatch.CloudWatchClient;
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

```java
CloudWatchClient cw =
    CloudWatchClient.builder().build();

boolean done = false;
String next_token = null;

while(!done) {
    ListMetricsResponse response;
    if (next_token == null) {
        ListMetricsRequest request = ListMetricsRequest.builder()
            .namespace(namespace)
            .build();
        response = cw.listMetrics(request);
    } else {
        ListMetricsRequest request = ListMetricsRequest.builder()
            .namespace(namespace)
            .nextToken(next_token)
            .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 {
    next_token = response.nextToken();
}
}

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.

Publishing 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

```java
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.PutMetricDataRequest;
import software.amazon.awssdk.services.cloudwatch.model.PutMetricDataResponse;
import software.amazon.awssdk.services.cloudwatch.model.StandardUnit;
```

Code

```java
CloudWatchClient cw =
    CloudWatchClient.builder().build();
Dimension dimension = Dimension.builder()
    .name("UNIQUE_PAGES")
    .value("URLS").build();
MetricDatum datum = MetricDatum.builder()
    .metricName("PAGES_VISITED")
    .addDimension(dimension)
    .build();
PutMetricDataRequest putMetricDataRequest = PutMetricDataRequest.builder()
    .namespace("CUSTOM_NS")
    .metricData(datum)
    .build();
PutMetricDataResponse putMetricDataResponse = cw.putMetricData(putMetricDataRequest);
```
PutMetricDataRequest request = PutMetricDataRequest.builder()
    .namespace("SITE/TRAFFIC")
    .metricData(datum).build();

PutMetricDataResponse response = cw.putMetricData(request);
System.out.printf("Successfully put data point %f", data_point);

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.

Working 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

```java
import software.amazon.awssdk.services.cloudwatch.CloudWatchClient;
import software.amazon.awssdk.services.cloudwatch.model.ComparisonOperator;
import software.amazon.awssdk.services.cloudwatch.model.Dimension;
import software.amazon.awssdk.services.cloudwatch.model.PutMetricAlarmRequest;
import software.amazon.awssdk.services.cloudwatch.model.PutMetricAlarmResponse;
import software.amazon.awssdk.services.cloudwatch.model.StandardUnit;
import software.amazon.awssdk.services.cloudwatch.model.Statistic;
```

Code

```java
CloudWatchClient cw =
    CloudWatchClient.builder().build();

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();

PutMetricAlarmResponse response = cw.putMetricAlarm(request);

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

```java
import software.amazon.awssdk.services.cloudwatch.CloudWatchClient;
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

```java
CloudWatch cw = CloudWatchClient.builder().build();

boolean done = false;
String new_token = null;

while(!done) {
   DescribeAlarmsResponse response;
   if (new_token == null) {
      DescribeAlarmsRequest request = DescribeAlarmsRequest.builder().build();
      response = cw.describeAlarms(request);
   } else {
      DescribeAlarmsRequest request = DescribeAlarmsRequest.builder()
         .nextToken(new_token)
         .build();
      response = cw.describeAlarms(request);
   }
   for(Metric Alarm alarm : response.metricAlarms()) {
      System.out.printf("Retrieved alarm %s", alarm.alarmName());
   }
   if(response.nextToken() == null) {
      done = true;
   } else {
      new_token = response.nextToken();
   }
}
```

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](https://github.com/aws/aws-sdk-java) 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

```java
import software.amazon.awssdk.services.cloudwatch.CloudWatchClient;
import software.amazon.awssdk.services.cloudwatch.model.DeleteAlarmsRequest;
import software.amazon.awssdk.services.cloudwatch.model.DeleteAlarmsResponse;
```

#### Code

```java
CloudWatchClient cw = CloudWatchClient.builder().build();

DeleteAlarmsRequest request = DeleteAlarmsRequest.builder()
    .alarmNames(alarm_name).build();

DeleteAlarmsResponse response = cw.deleteAlarms(request);
System.out.printf("Successfully deleted alarm %s", alarm_name);
```

See the [complete example](https://github.com/aws/aws-sdk-java) 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*

### Using 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. 37).

### 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

```java
import software.amazon.awssdk.services.cloudwatch.CloudWatchClient;
import software.amazon.awssdk.services.cloudwatch.model.EnableAlarmActionsRequest;
import software.amazon.awssdk.services.cloudwatch.model.EnableAlarmActionsResponse;
```
import software.amazon.awssdk.services.cloudwatch.CloudWatchClient;
import software.amazon.awssdk.services.cloudwatch.model.EnableAlarmActionsRequest;
import software.amazon.awssdk.services.cloudwatch.model.EnableAlarmActionsResponse;

Code

CloudWatchClient cw =
    CloudWatchClient.builder().build();

EnableAlarmActionsRequest request = EnableAlarmActionsRequest.builder()
    .alarmNames(alarm).build();

EnableAlarmActionsResponse response = cw.enableAlarmActions(request);

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.services.cloudwatch.CloudWatchClient;
import software.amazon.awssdk.services.cloudwatch.model.DisableAlarmActionsRequest;
import software.amazon.awssdk.services.cloudwatch.model.DisableAlarmActionsResponse;

Code

CloudWatchClient cw = CloudWatchClient.builder().build();

DisableAlarmActionsRequest request = DisableAlarmActionsRequest.builder()
    .alarmNames(alarmName).build();

DisableAlarmActionsResponse response = cw.disableAlarmActions(request);

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

Sending 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**

```java
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**

```java
CloudWatchEventsClient cwe = CloudWatchEventsClient.builder().build();

final String EVENT_DETAILS = "{"key1": "value1", "key2": "value2" }";

PutEventsRequestEntry request_entry = PutEventsRequestEntry.builder()
    .detail(EVENT_DETAILS)
    .detailType("sampleSubmitted")
    .resources(resource_arn)
    .source("aws-sdk-java-cloudwatch-example").build();

PutEventsRequest request = PutEventsRequest.builder()
    .entries(request_entry).build();

PutEventsResponse response = cwe.putEvents(request);
```

See the [complete example on GitHub](https://github.com/aws-samples/aws-sdk-java-cloudwatch).

## 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**

```java
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**

```java
CloudWatchEventsClient cwe =
```
CloudWatchEventsClient.builder().build();

PutRuleRequest request = PutRuleRequest.builder()
    .name(rule_name)
    .roleArn(role_arn)
    .scheduleExpression("rate(5 minutes)")
    .state(RuleState.ENABLED)
    .build();

PutRuleResponse response = cwe.putRule(request);

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

```java
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

```java
CloudWatchEventsClient cwe =
    CloudWatchEventsClient.builder().build();

Target target = Target.builder()
    .arn(function_arn)
    .id(target_id)
    .build();

PutTargetsRequest request = PutTargetsRequest.builder()
    .targets(target)
    .rule(rule_name)
    .build();

PutTargetsResponse response = cwe.putTargets(request);
```

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
DynamoDB Examples Using the AWS SDK for Java

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

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

Topics
- Working with Tables in DynamoDB (p. 43)
- Working with Items in DynamoDB (p. 48)

Working 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

```java
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.CreateTableRequest;
import software.amazon.awssdk.services.dynamodb.model.CreateTableResponse;
import software.amazon.awssdk.services.dynamodb.model.KeySchemaElement;
```
Create a Table with a Simple Primary Key

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

Code

```java
import software.amazon.awssdk.services.dynamodb.model.KeyType;
import software.amazon.awssdk.services.dynamodb.model.ProvisionedThroughput;
import software.amazon.awssdk.services.dynamodb.model.ScalarAttributeType;

CreateTableRequest request = CreateTableRequest.builder()
    .attributeDefinitions(AttributeDefinition.builder()
        .attributeName("Name")
        .attributeType(ScalarAttributeType.S)
        .build())
    .keySchema(KeySchemaElement.builder()
        .attributeName("Name")
        .KeyType(KeyType.HASH)
        .build())
    .provisionedThroughput(ProvisionedThroughput.builder()
        .readCapacityUnits(new Long(10))
        .writeCapacityUnits(new Long(10))
        .build())
    .tableName(table_name)
    .build();

DynamoDBClient ddb = DynamoDBClient.create();

try {
    CreateTableResponse response = ddb.createTable(request);
    System.out.println(response.tableDescription().tableName());
} catch (DynamoDBException e) {
    System.err.println(e.getErrorMessage());
    System.exit(1);
}
System.out.println("Done!");
```

See the complete example on GitHub.

Create a Table with a Composite Primary Key

Add another AttributeDefinition and KeySchemaElement to CreateTableRequest.

Code

```java
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)
```
DynamoDBClient ddb = DynamoDBClient.create();

try {
    CreateTableResponse result = ddb.createTable(request);
} catch (DynamoDBException e) {
    System.err.println(e.getErrorMessage());
    System.exit(1);
}
System.out.println("Done!");

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

```java
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;
```

Code

```java
DynamoDBClient ddb = DynamoDBClient.create();

boolean more_tables = true;
String last_name = null;

while (more_tables) {
    try {
        ListTablesResponse response = null;
        if (last_name == null) {
            ListTablesRequest request = ListTablesRequest.builder().build();
            response = ddb.listTables(request);
        } else {
            ListTablesRequest request = ListTablesRequest.builder()
            .exclusiveStartTableName(last_name).build();
            response = ddb.listTables(request);
        }
        List<String> table_names = response.tableNames();
        if (table_names.size() > 0) {
            for (String cur_name : table_names) {
                System.out.format("* %s\n", cur_name);
            }
        }
    }
```
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](https://github.com/awsdocs/aws-sdk-for-java) 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**

```java
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;
```

**Code**

```java
DynamoDBClient ddb = DynamoDBClient.create();
DescribeTableRequest request = DescribeTableRequest.builder()
   .tableName(table_name)
   .build();

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

    if (table_info != null) {
        System.out.format("Table name : %s\n",
                        table_info.tableName());
        System.out.format("Table ARN : %s\n",
                        table_info.tableArn());
        System.out.format("Status : %s\n",
                        table_info.tableStatus());
        System.out.format("Item count : %d\n",
                        table_info.itemCount().longValue());
        System.out.format("Size (bytes) : %d\n",
                        table_info.tableSizeBytes().longValue());
```
ProvisionedThroughputDescription throughput_info =
  table_info.provisionedThroughput();
System.out.println("Throughput");
System.out.format("  Read Capacity : %d\n",
          throughput_info.readCapacityUnits().longValue());
System.out.format("  Write Capacity: %d\n",
          throughput_info.writeCapacityUnits().longValue());

List<AttributeDefinition> attributes =
  table_info.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.getErrorMessage());
    System.exit(1);
  }
System.out.println("\nDone!");

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

```java
import software.amazon.awssdk.services.dynamodb.model.ProvisionedThroughput;
import software.amazon.awssdk.services.dynamodb.DynamoDBClient;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.UpdateTableRequest;
```

Code

```java
ProvisionedThroughput table_throughput = ProvisionedThroughput.builder()
  .readCapacityUnits(read_capacity)
  .writeCapacityUnits(write_capacity)
  .build();
DynamoDBClient ddb = DynamoDBClient.create();
UpdateTableRequest request = UpdateTableRequest.builder()
  .provisionedThroughput(table_throughput)
  .tableName(table_name)
  .build();
try {
  ddb.updateTable(request);
} catch (DynamoDBException e) {
  System.err.println(e.getErrorMessage());
  System.exit(1);
}
System.out.println("Done!");
```
See the complete example on GitHub.

Delete a Table

Call the `DynamoDBClient.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

```java
import software.amazon.awssdk.services.dynamodb.model.DynamoDBException;
import software.amazon.awssdk.services.dynamodb.DynamoDBClient;
import software.amazon.awssdk.services.dynamodb.model.DeleteTableRequest;
```

Code

```java
DynamoDBClient ddb = DynamoDBClient.create();
DeleteTableRequest request = DeleteTableRequest.builder()
   .tableName(table_name)
   .build();
try {
   ddb.deleteTable(request);
} catch (DynamoDBException e) {
   System.err.println(e.getErrorMessage());
   System.exit(1);
}
System.out.println("Done!");
```

See the complete example on GitHub.

More Info

- Guidelines for Working with Tables in the Amazon DynamoDB Developer Guide
- Working with Tables in DynamoDB in the Amazon DynamoDB Developer Guide

Working 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.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.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;

Code

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

key_to_get.put("Name", AttributeValue.builder()
    .s(name).build());

GetItemRequest request = null;
if (projection_expression != null) {
    request = GetItemRequest.builder()
        .key(key_to_get)
        .tableName(table_name)
        .projectionExpression(projection_expression)
        .build();
} else {
    request = GetItemRequest.builder()
        .key(key_to_get)
        .tableName(table_name)
        .build();
}

DynamoDBClient ddb = DynamoDBClient.create();

try {
    Map<String,AttributeValue> returned_item =
        ddb.getItem(request).item();
    if (returned_item != null) {
        Set<String> keys = returned_item.keySet();
        for (String key : keys) {
            System.out.format("%s: %s\n", key, returned_item.get(key).toString());
        }
    } else {
        System.out.format("No item found with the key %s!\n", name);
    }
} catch (DynamoDBException e) {
    System.err.println(e.getErrorMessage());
    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.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;

Code

HashMap<String,AttributeValue> item_values =
   new HashMap<String,AttributeValue>() {
      item_values.put("Name", AttributeValue.builder().s(name).build());
      for (String[] field : extra_fields) {
         item_values.put(field[0], AttributeValue.builder().s(field[1]).build());
      }
   }

DynamoDBClient ddb = DynamoDBClient.create();
PutItemRequest request = PutItemRequest.builder()
   .tableName(table_name)
   .item(item_values)
   .build();

try {
   ddb.putItem(request);
} catch (ResourceNotFoundException e) {
   System.err.format("Error: The table "%s" can't be found.\n", table_name);
   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);
}
System.out.println("Done!");

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.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.UpdateItemRequest;
import software.amazon.awssdk.services.dynamodb.DynamoDBClient;

Code

HashMap<String,AttributeValue> item_key =
   new HashMap<String,AttributeValue>() {
      item_key.put("Name", AttributeValue.builder().s(name).build());
HashMap<String, AttributeValueUpdate> updated_values = new HashMap<String, AttributeValueUpdate>();

for (String[] field : extra_fields) {
    updated_values.put(field[0], AttributeValueUpdate.builder()
        .value(AttributeValue.builder().s(field[1]).build())
        .action(AttributeAction.PUT)
        .build());
}

UpdateItemRequest request = UpdateItemRequest.builder()
    .tableName(table_name)
    .key(item_key)
    .attributeUpdates(updated_values)
    .build();

DynamoDBClient ddb = DynamoDBClient.create();

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);
}

System.out.println("Done!");

See the complete example on GitHub.

More Info

- Guidelines for Working with Items in the Amazon DynamoDB Developer Guide
- Working with Items in DynamoDB in the Amazon DynamoDB Developer Guide

Amazon EC2 Examples Using the AWS SDK for Java

This section provides examples of programming Amazon EC2 with the AWS SDK for Java 2.0.

Topics

- Managing Amazon EC2 Instances (p. 51)
- Using Elastic IP Addresses in Amazon EC2 (p. 55)
- Using Regions and Availability Zones (p. 58)
- Working with Amazon EC2 Key Pairs (p. 59)
- Working with Security Groups in Amazon EC2 (p. 60)

Managing Amazon EC2 Instances

Creating 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.
Starting 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.

```java
import software.amazon.awssdk.services.ec2.EC2Client;
import software.amazon.awssdk.services.ec2.model.StartInstancesRequest;

EC2Client ec2 = EC2Client.create();
StartInstancesRequest start_request = StartInstancesRequest.builder()
    .instanceIds(ami_id)
    .build();
StartInstancesResponse response = ec2.startInstances(start_request);

System.out.println("Successfully started EC2 instance \"" + instance_id + \"\" based on AMI \"" + ami_id + \"\", instance_id, ami_id);
```

See the complete example on GitHub.
Managing Amazon EC2 Instances

```java
EC2Client ec2 = EC2Client.create();
StartInstancesRequest request = StartInstancesRequest.builder()
    .instanceIds(instance_id).build();
ec2.startInstances(request);

See the complete example on GitHub.

Stopping 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.

```java
import software.amazon.awssdk.services.ec2.EC2Client;
import software.amazon.awssdk.services.ec2.model.StopInstancesRequest;

EC2Client ec2 = EC2Client.create();
StopInstancesRequest request = StopInstancesRequest.builder()
    .instanceIds(instance_id).build();
ec2.stopInstances(request);

See the complete example on GitHub.

Rebooting 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.

```java
import software.amazon.awssdk.services.ec2.EC2Client;
import software.amazon.awssdk.services.ec2.model.RebootInstancesRequest;
import software.amazon.awssdk.services.ec2.model.RebootInstancesResponse;

EC2Client ec2 = EC2Client.create();
RebootInstancesRequest request = RebootInstancesRequest.builder()
    .instanceIds(instance_id).build();
RebootInstancesResponse response = ec2.rebootInstances(request);

See the complete example on GitHub.

Describing 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

```java
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;
```

Code

```java
DescribeInstancesRequest request = DescribeInstancesRequest.builder().build();
while(!done) {
    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("\n");
        }
        if(response.nextToken() == null) {
            done = true;
        }
    }
}
```

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.

Monitoring 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

```java
import software.amazon.awssdk.services.ec2.EC2Client;
```
import software.amazon.awssdk.services.ec2.model.MonitorInstancesRequest;

Code

EC2Client ec2 = EC2Client.create();
MonitorInstancesRequest request = MonitorInstancesRequest.builder()
    .instanceIds(instance_id).build();
ec2.monitorInstances(request);

See the complete example on GitHub.

Stopping 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.services.ec2.EC2Client;
import software.amazon.awssdk.services.ec2.model.UnmonitorInstancesRequest;

Code

EC2Client ec2 = EC2Client.create();
UnmonitorInstancesRequest request = UnmonitorInstancesRequest.builder()
    .instanceIds(instance_id).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

Using Elastic IP Addresses in Amazon EC2

Allocating 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**

```java
import software.amazon.awssdk.services.ec2.EC2Client;
import software.amazon.awssdk.services.ec2.model.AllocateAddressRequest;
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.DomainType;
```

**Code**

```java
EC2Client ec2 = EC2Client.create();
AllocateAddressRequest allocate_request = AllocateAddressRequest.builder()
    .domain(DomainType.Vpc)
    .build();
AllocateAddressResponse allocate_response =
    ec2.allocateAddress(allocate_request);
String allocation_id = allocate_response.allocationId();
AssociateAddressRequest associate_request =
    AssociateAddressRequest.builder()
    .instanceId(instance_id)
    .allocationId(allocation_id)
    .build();
AssociateAddressResponse associate_response =
    ec2.associateAddress(associate_request);
System.out.printf(
    "Successfully associated Elastic IP address %s " +
    "with instance %s",
    associate_response.associationId(),
    instance_id);
```

See the complete example on GitHub.

**Describing 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**

```java
import software.amazon.awssdk.services.ec2.EC2Client;
import software.amazon.awssdk.services.ec2.model.Address;
import software.amazon.awssdk.services.ec2.model.DescribeAddressesResponse;
```

**Code**

```java
```

56
Using Elastic IP Addresses in Amazon EC2

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

See the complete example on GitHub.

**Releasing 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**

```java
import software.amazon.awssdk.services.ec2.EC2Client;
import software.amazon.awssdk.services.ec2.model.ReleaseAddressRequest;
import software.amazon.awssdk.services.ec2.model.ReleaseAddressResponse;
```

**Code**

```java
EC2Client ec2 = EC2Client.create();
ReleaseAddressRequest request = ReleaseAddressRequest.builder()  
    .allocationId(alloc_id).build();
ReleaseAddressResponse response = ec2.releaseAddress(request);
```

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*
Using Regions and Availability Zones

Describing 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

```java
import software.amazon.awssdk.services.ec2.EC2Client;
import software.amazon.awssdk.services.ec2.model.DescribeRegionsResponse;
import software.amazon.awssdk.services.ec2.model.Region;
```

Code

```java
EC2Client ec2 = EC2Client.create();
DescribeRegionsResponse regions_response = ec2.describeRegions();
for(Region region : regions_response.regions()) {
    System.out.printf(
            "Found region %s " +
            "with endpoint %s",
            region.regionName(),
            region.endpoint());
    System.out.println();
}
```

See the complete example on GitHub.

Describing 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

```java
import software.amazon.awssdk.services.ec2.EC2Client;
import software.amazon.awssdk.services.ec2.model.AvailabilityZone;
import software.amazon.awssdk.services.ec2.model.DescribeAvailabilityZonesResponse;
```

Code

```java
EC2Client ec2 = EC2Client.create();
DescribeAvailabilityZonesResponse zones_response =
    ec2.describeAvailabilityZones();
for(AvailabilityZone zone : zones_response.availabilityZones()) {
    System.out.printf(
            "Found availability zone %s " +
            "with status %s " +
            "in region %s",
            zone.zoneName(),
            zone.state(),
            zone.region());
}
Working with Amazon EC2 Key Pairs

Creating a Key Pair

To create a key pair, call the EC2Client's createKeyPair method with a CreateKeyPairRequest that contains the key's name.

Imports

```java
import software.amazon.awssdk.services.ec2.EC2Client;
import software.amazon.awssdk.services.ec2.model.CreateKeyPairRequest;
import software.amazon.awssdk.services.ec2.model.CreateKeyPairResponse;
```

Code

```java
EC2Client ec2 = EC2Client.create();
CreateKeyPairRequest request = CreateKeyPairRequest.builder()
    .keyName(key_name).build();
CreateKeyPairResponse response = ec2.createKeyPair(request);
```

See the complete example on GitHub.

Describing 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

```java
import software.amazon.awssdk.services.ec2.EC2Client;
import software.amazon.awssdk.services.ec2.model.DescribeKeyPairsResponse;
import software.amazon.awssdk.services.ec2.model.KeyPairInfo;
```

Code

```java
EC2Client ec2 = EC2Client.create();
DescribeKeyPairsResponse response = ec2.describeKeyPairs();
for(KeyPairInfo key_pair : response.keyPairs()) {
```
System.out.printf(
    "Found key pair with name %s " +
    "and fingerprint %s",
    key_pair.keyName(),
    key_pair.keyFingerprint());
System.out.println("\n");
}

See the complete example on GitHub.

**Deleting 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**

```java
import software.amazon.awssdk.services.ec2.EC2Client;
import software.amazon.awssdk.services.ec2.model.DeleteKeyPairRequest;
import software.amazon.awssdk.services.ec2.model.DeleteKeyPairResponse;
```

**Code**

```java
EC2Client ec2 = EC2Client.create();
DeleteKeyPairRequest request = DeleteKeyPairRequest.builder()
    .keyName(key_name)
    .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*

**Working with Security Groups in Amazon EC2**

**Creating a Security Group**

To create a security group, call the EC2Client's createSecurityGroup method with a CreateSecurityGroupRequest that contains the key's name.

**Imports**

```java
import software.amazon.awssdk.services.ec2.EC2Client;
import software.amazon.awssdk.services.ec2.model.CreateSecurityGroupRequest;
import software.amazon.awssdk.services.ec2.model.CreateSecurityGroupResponse;
```

**Code**

```java
EC2Client ec2 = EC2Client.create();
CreateSecurityGroupRequest request = CreateSecurityGroupRequest.builder()
    .groupName(group_name)
    .build();
CreateSecurityGroupResponse response = ec2.createSecurityGroup(request);
```
Configuring 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

```java
import software.amazon.awssdk.services.ec2.EC2Client;
import software.amazon.awssdk.services.ec2.model.AuthorizeSecurityGroupIngressRequest;
import software.amazon.awssdk.services.ec2.model.AuthorizeSecurityGroupIngressResponse;
import software.amazon.awssdk.services.ec2.model.IpPermission;
import software.amazon.awssdk.services.ec2.model.IpRange;
```

Code

```java
EC2Client ec2 = EC2Client.create();
IpRange ip_range = IpRange.builder()
    .cidrIp("0.0.0.0/0").build();
IpPermission ip_perm = IpPermission.builder()
    .ipProtocol("tcp")
    .toPort(80)
    .fromPort(80)
    .ipv4Ranges(ip_range)
    .build();
IpPermission ip_perm2 = IpPermission.builder()
    .ipProtocol("tcp")
    .toPort(22)
    .fromPort(22)
    .ipv4Ranges(ip_range)
    .build();
AuthorizeSecurityGroupIngressRequest auth_request =
    AuthorizeSecurityGroupIngressRequest.builder()
    .groupName(group_name)
    .ipPermissions(ip_perm, ip_perm2)
    .build();
AuthorizeSecurityGroupIngressResponse auth_response =
    ec2.authorizeSecurityGroupIngress(auth_request);
To add an egress rule to the security group, provide similar data in an

See the complete example on GitHub.

Describing 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

```java
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;
```

Code

```java
EC2Client ec2 = EC2Client.create();
DescribeSecurityGroupsRequest request =
    DescribeSecurityGroupsRequest.builder()
    .groupIds(group_id).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());
}
```

See the complete example on GitHub.

Deleting 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

```java
import software.amazon.awssdk.services.ec2.EC2Client;
import software.amazon.awssdk.services.ec2.model.DeleteSecurityGroupRequest;
import software.amazon.awssdk.services.ec2.model.DeleteSecurityGroupResponse;
```

Code

```java
EC2Client ec2 = EC2Client.create();
DeleteSecurityGroupRequest request = DeleteSecurityGroupRequest.builder()
```
IAM Examples Using the AWS SDK for Java

This section provides examples of programming IAM using the AWS SDK for Java.

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.

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

Topics
• Managing IAM Access Keys (p. 63)
• Managing IAM Users (p. 67)
• Using IAM Account Aliases (p. 69)
• Working with IAM Policies (p. 71)
• Working with IAM Server Certificates (p. 75)

Managing IAM Access Keys

Creating 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

```java
import software.amazon.awssdk.services.iam.model.CreateAccessKeyRequest;
import software.amazon.awssdk.services.iam.model.CreateAccessKeyResponse;
```
import software.amazon.awssdk.core.regions.Region;
import software.amazon.awssdk.services.iam.IAMClient;

Code

Region region = Region.AWS_GLOBAL;
IAMClient iam = IAMClient.builder().region(region).build();
CreateAccessKeyRequest request = CreateAccessKeyRequest.builder()
    .userName(user).build();
CreateAccessKeyResponse response = iam.createAccessKey(request);

See the complete example on GitHub.

Listing 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.ListAccessKeysRequest;
import software.amazon.awssdk.services.iam.model.ListAccessKeysResponse;
import software.amazon.awssdk.core.regions.Region;
import software.amazon.awssdk.services.iam.IAMClient;

Code

Region region = Region.AWS_GLOBAL;
IAMClient iam = IAMClient.builder().region(region).build();

boolean done = false;
String new_marker = null;

while (!done) {
    ListAccessKeysResponse response;
    if(new_marker == null) {
        ListAccessKeysRequest request = ListAccessKeysRequest.builder()
            .userName(username).build();
        response = iam.listAccessKeys(request);
    } else {
        ListAccessKeysRequest request = ListAccessKeysRequest.builder()
            .userName(username)
            .marker(new_marker).build();
        response = iam.listAccessKeys(request);
    }
    for (AccessKeyMetadata metadata :
        response.accessKeyMetadata()) {
        System.out.format("Retrieved access key %s",}
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.

### Retrieving 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**

```java
import software.amazon.awssdk.core.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;
```

**Code**

```java
Region region = Region.AWS_GLOBAL;
IAMClient iam = IAMClient.builder().region(region).build();
GetAccessKeyLastUsedRequest request = GetAccessKeyLastUsedRequest.builder().accessKeyId(access_id).build();
GetAccessKeyLastUsedResponse response = iam.getAccessKeyLastUsed(request);
System.out.println("Access key was last used at: " + response.accessKeyLastUsed().lastUsedDate());
```

See the complete example on GitHub.

### Activating or Deactivating 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**

```java
import software.amazon.awssdk.services.iam.model.StatusType;
import software.amazon.awssdk.services.iam.model.UpdateAccessKeyRequest;
import software.amazon.awssdk.services.iam.model.UpdateAccessKeyResponse;
```
import software.amazon.awssdk.core.regions.Region;

Code

String username = args[0];
String access_id = args[1];
String status = args[2];
StatusType statusType;

if (status.toLowerCase().equalsIgnoreCase("active")) {
    statusType = StatusType.ACTIVE;
} else if (status.toLowerCase().equalsIgnoreCase("inactive")) {
    statusType = StatusType.INACTIVE;
} else {
    statusType = StatusType.UNKNOWN_TO_SDK_VERSION;
}

Region region = Region.AWS_GLOBAL;
IAMClient iam = IAMClient.builder().region(region).build();

UpdateAccessKeyRequest request = UpdateAccessKeyRequest.builder()
    .accessKeyId(access_id)
    .userName(username)
    .status(statusType)
    .build();

UpdateAccessKeyResponse response = iam.updateAccessKey(request);

See the complete example on GitHub.

Deleting 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. 65) method instead.

Imports

```java
import software.amazon.awssdk.core.regions.Region;
import software.amazon.awssdk.services.iam.IAMClient;
import software.amazon.awssdk.services.iam.model.DeleteAccessKeyRequest;
import software.amazon.awssdk.services.iam.model.DeleteAccessKeyResponse;
```

Code

```java
Region region = Region.AWS_GLOBAL;
IAMClient iam = IAMClient.builder().region(region).build();

DeleteAccessKeyRequest request = DeleteAccessKeyRequest.builder()
    .accessKeyId(access_key)
    .userName(username).build();

DeleteAccessKeyResponse response = iam.deleteAccessKey(request);
```
See the [complete example](https://github.com/aws/aws-sdk-java) on GitHub.

**More Information**

- [CreateAccessKey](https://docs.aws.amazon.com/iam/latest/api/API_CreateAccessKey.html) in the IAM API Reference
- [ListAccessKeys](https://docs.aws.amazon.com/iam/latest/api/API_ListAccessKeys.html) in the IAM API Reference
- [GetAccessKeyLastUsed](https://docs.aws.amazon.com/iam/latest/api/API_GetAccessKeyLastUsed.html) in the IAM API Reference
- [UpdateAccessKey](https://docs.aws.amazon.com/iam/latest/api/API_UpdateAccessKey.html) in the IAM API Reference
- [DeleteAccessKey](https://docs.aws.amazon.com/iam/latest/api/API_DeleteAccessKey.html) 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**

```java
import software.amazon.awssdk.services.iam.model.CreateUserRequest;
import software.amazon.awssdk.services.iam.model.CreateUserResponse;
import software.amazon.awssdk.core.regions.Region;
import software.amazon.awssdk.services.iam.IAMClient;
```

**Code**

```java
Region region = Region.AWS_GLOBAL;
IAMClient iam = IAMClient.builder().region(region).build();
CreateUserRequest request = CreateUserRequest.builder()
    .userName(username).build();
CreateUserResponse response = iam.createUser(request);
```

See the [complete example](https://github.com/aws/aws-sdk-java) 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**

```java
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.core.regions.Region;
import software.amazon.awssdk.services.iam.IAMClient;

Code

Region region = Region.AWS_GLOBAL;
IAMClient iam = IAMClient.builder().region(region).build();

boolean done = false;
String new_marker = null;
while(!done) {
    ListUsersResponse response;
    if (new_marker == null) {
        ListUsersRequest request = ListUsersRequest.builder().build();
        response = iam.listUsers(request);
    } else {
        ListUsersRequest request = ListUsersRequest.builder()
            .marker(new_marker).build();
        response = iam.listUsers(request);
    }

    for(User user : response.users()) {
        System.out.format("Retrieved user %s", user.userName());
    }

    if(!response.isTruncated()) {
        done = true;
    } else {
        new_marker = response.marker();
    }
}

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.core.regions.Region;
import software.amazon.awssdk.services.iam.IAMClient;
import software.amazon.awssdk.services.iam.model.UpdateUserRequest;
import software.amazon.awssdk.services.iam.model.UpdateUserResponse;

Code

Region region = Region.AWS_GLOBAL;
IAMClient iam = IAMClient.builder().region(region).build();

UpdateUserRequest request = UpdateUserRequest.builder()
    .userName(cur_name)
    .newUserName(new_name).build();

UpdateUserResponse response = iam.updateUser(request);
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**

```java
import software.amazon.awssdk.core.regions.Region;
import software.amazon.awssdk.services.iam.IAMClient;
import software.amazon.awssdk.services.iam.model.DeleteConflictException;
import software.amazon.awssdk.services.iam.model.DeleteUserRequest;
```

**Code**

```java
Region region = Region.AWS_GLOBAL;
IAMClient iam = IAMClient.builder().region(region).build();
DeleteUserRequest request = DeleteUserRequest.builder()
    .userName(username).build();
try {
    iam.deleteUser(request);
} catch (DeleteConflictException e) {
    System.out.println("Unable to delete user. Verify user is not" +
        " associated with any resources");
    throw e;
}
```

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

**Using 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.

**Creating an Account Alias**

To create an account alias, call the IAMClient’s createAccountAlias method with a CreateAccountAliasRequest object that contains the alias name.

**Imports**

```java
```
import software.amazon.awssdk.services.iam.model.CreateAccountAliasRequest;
import software.amazon.awssdk.services.iam.model.CreateAccountAliasResponse;

import software.amazon.awssdk.core.regions.Region;
import software.amazon.awssdk.services.iam.IAMClient;

Code

Region region = Region.AWS_GLOBAL;
IAMClient iam = IAMClient.builder().region(region).build();
CreateAccountAliasRequest request = CreateAccountAliasRequest.builder()
    .accountAlias(alias).build();
CreateAccountAliasResponse response = iam.createAccountAlias(request);

See the complete example on GitHub.

Listing 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.

import software.amazon.awssdk.services.iam.model.ListAccountAliasesResponse;
import software.amazon.awssdk.core.regions.Region;
import software.amazon.awssdk.services.iam.IAMClient;

code

Region region = Region.AWS_GLOBAL;
IAMClient iam = IAMClient.builder().region(region).build();
ListAccountAliasesResponse response = iam.listAccountAliases();
for (String alias : response.accountAliases()) {
    System.out.printf("Retrieved account alias %s", alias);
}

see the complete example on GitHub.

Deleting 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.

import software.amazon.awssdk.services.iam.model.DeleteAccountAliasRequest;
import software.amazon.awssdk.services.iam.model.DeleteAccountAliasResponse;
import software.amazon.awssdk.core.regions.Region;
import software.amazon.awssdk.services.iam.IAMClient;

Code

Region region = Region.AWS_GLOBAL;
IAMClient iam = IAMClient.builder().region(region).build();

DeleteAccountAliasRequest request = DeleteAccountAliasRequest.builder()
 .accountAlias(alias).build();

DeleteAccountAliasResponse response = iam.deleteAccountAlias(request);
System.out.println("Successfully deleted account alias " + alias);

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

Working with IAM Policies

Creating 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.services.iam.model.CreatePolicyRequest;
import software.amazon.awssdk.services.iam.model.CreatePolicyResponse;
import software.amazon.awssdk.core.regions.Region;
import software.amazon.awssdk.services.iam.IAMClient;

Code

Region region = Region.AWS_GLOBAL;
IAMClient iam = IAMClient.builder().region(region).build();

CreatePolicyRequest request = CreatePolicyRequest.builder()
 .policyName(policy_name)
 .policyDocument(POLICY_DOCUMENT).build();

CreatePolicyResponse response = iam.createPolicy(request);
System.out.println("Successfully created policy: " +
 response.policy().policyName());

IAM policy documents are JSON strings with a well-documented syntax. Here is an example that provides access to make particular requests to DynamoDB.
Getting a Policy

To retrieve an existing policy, call the IAMClient's `getPolicy` method, providing the policy's ARN within a `GetPolicyRequest` object.

Imports

```java
import software.amazon.awssdk.services.iam.model.GetPolicyRequest;
import software.amazon.awssdk.services.iam.model.GetPolicyResponse;
import software.amazon.awssdk.core.regions.Region;
import software.amazon.awssdk.services.iam.IAMClient;
```

Code

```java
Region region = Region.AWS_GLOBAL;
IAMClient iam = IAMClient.builder().region(region).build();
GetPolicyRequest request = GetPolicyRequest.builder()
    .policyArn(policy_arn).build();
GetPolicyResponse response = iam.getPolicy(request);
```

See the complete example on GitHub.

Attaching 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

```java
import software.amazon.awssdk.core.regions.Region;
import software.amazon.awssdk.services.iam.IAMClient;
```
import software.amazon.awssdk.services.iam.model.AttachRolePolicyRequest;
import software.amazon.awssdk.services.iam.model.AttachedPolicy;

Code

Region region = Region.AWS_GLOBAL;
IAMClient iam = IAMClient.builder().region(region).build();
AttachRolePolicyRequest attach_request =
    AttachRolePolicyRequest.builder()
    .roleName(role_name)
    .policyArn(POLICY_ARN).build();
iam.attachRolePolicy(attach_request);

See the complete example on GitHub.

Listing 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.core.regions.Region;
import software.amazon.awssdk.services.iam.IAMClient;
import software.amazon.awssdk.services.iam.model.ListAttachedRolePoliciesRequest;
import software.amazon.awssdk.services.iam.model.ListAttachedRolePoliciesResponse;

Code

Region region = Region.AWS_GLOBAL;
IAMClient iam = IAMClient.builder().region(region).build();
List<AttachedPolicy> matching_policies = new ArrayList<>();
boolean done = false;
String new_marker = null;
while(!done) {
    ListAttachedRolePoliciesResponse response;
    if (new_marker == null) {
        ListAttachedRolePoliciesRequest request =
            ListAttachedRolePoliciesRequest.builder()
            .roleName(role_name).build();
        response = iam.listAttachedRolePolicies(request);
    }
    else {
        ListAttachedRolePoliciesRequest request =
            ListAttachedRolePoliciesRequest.builder()
            .roleName(role_name)
            .marker(new_marker).build();
        response = iam.listAttachedRolePolicies(request);
    }
    matching_policies.addAll(response.attachedPolicies());
    done = response.isTruncated();
    new_marker = response.marker();
}
matching_policies.addAll(
    response.attachedPolicies().stream()
    .filter(p -> p.policyName().equals(role_name))
    .collect(Collectors.toList()));

if(!response.isTruncated()) {
    done = true;
} else {
    new_marker = response.marker();
}

if (matching_policies.size() > 0) {
    System.out.println(role_name + 
    " policy is already attached to this role.");
    return;
}

See the complete example on GitHub.

Detaching 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

```java
import software.amazon.awssdk.services.iam.model.DetachRolePolicyRequest;
import software.amazon.awssdk.services.iam.model.DetachRolePolicyResponse;
import software.amazon.awssdk.core.regions.Region;
import software.amazon.awssdk.services.iam.IAMClient;
```

Code

```java
Region region = Region.AWS_GLOBAL;
IAMClient iam = IAMClient.builder().region(region).build();

DetachRolePolicyRequest request = DetachRolePolicyRequest.builder()
    .roleName(role_name)
    .policyArn(policy_arn).build();

DetachRolePolicyResponse response = iam.detachRolePolicy(request);
```

See the complete example on GitHub.

More Information

- Overview of IAM Policies 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
Working 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.

Getting 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.

```
import software.amazon.awssdk.services.iam.model.GetServerCertificateRequest;
import software.amazon.awssdk.services.iam.model.GetServerCertificateResponse;
import software.amazon.awssdk.core.regions.Region;
import software.amazon.awssdk.services.iam.IAMClient;

Region region = Region.AWS_GLOBAL;
IAMClient iam = IAMClient.builder().region(region).build();
GetServerCertificateRequest request = GetServerCertificateRequest.builder()
   .serverCertificateName(cert_name).build();
GetServerCertificateResponse response = iam.getServerCertificate(request);
```

See the complete example on GitHub.

Listing 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.

```
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.core.regions.Region;
import software.amazon.awssdk.services.iam.IAMClient;

Code

Region region = Region.AWS_GLOBAL;
IAMClient iam = IAMClient.builder().region(region).build();

boolean done = false;
String new_marker = null;

while(!done) {
    ListServerCertificatesResponse response;
    if (new_marker == null) {
        ListServerCertificatesRequest request =
            ListServerCertificatesRequest.builder().build();
        response = iam.listServerCertificates(request);
    }
    else {
        ListServerCertificatesRequest request =
            ListServerCertificatesRequest.builder()
                .marker(new_marker).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 {
        new_marker = response.marker();
    }
}

See the complete example on GitHub.

**Updating 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**

```java
import software.amazon.awssdk.core.regions.Region;
import software.amazon.awssdk.services.iam.IAMClient;
import software.amazon.awssdk.services.iam.UpdateServerCertificateRequest;
import software.amazon.awssdk.services.iam.UpdateServerCertificateResponse;
```

**Code**

```java
Region region = Region.AWS_GLOBAL;
IAMClient iam = IAMClient.builder().region(region).build();```
UpdateServerCertificateRequest request =
    UpdateServerCertificateRequest.builder()
    .serverCertificateName(cur_name)
    .newServerCertificateName(new_name)
    .build();

UpdateServerCertificateResponse response =
    iam.updateServerCertificate(request);

See the complete example on GitHub.

Deleting a Server Certificate

To delete a server certificate, call the IAMClient's deleteServerCertificate method with a DeleteServerCertificateRequest containing the certificate's name.

Imports

```java
import software.amazon.awssdk.services.iam.model.DeleteServerCertificateRequest;
import software.amazon.awssdk.services.iam.model.DeleteServerCertificateResponse;
import software.amazon.awssdk.core.regions.Region;
import software.amazon.awssdk.services.iam.IAMClient;
```

Code

```java
Region region = Region.AWS_GLOBAL;
IAMClient iam = IAMClient.builder().region(region).build();

DeleteServerCertificateRequest request =
    DeleteServerCertificateRequest.builder()
    .serverCertificateName(cert_name).build();

DeleteServerCertificateResponse response =
    iam.deleteServerCertificate(request);
```

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

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. With the AWS SDK for Java 2.0, multiple service calls to get the next page of results are made for you automatically. You only have to code what to do with the results.
The examples on this page use Amazon S3 object operations to demonstrate the various methods of retrieving your data from paginated responses.

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

## 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.listObjectsV2Iterable` method. This method returns a `ListObjectsV2Paginator` 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

```java
import software.amazon.awssdk.core.regions.Region;
import software.amazon.awssdk.services.s3.S3Client;
import software.amazon.awssdk.services.s3.model.ListObjectsV2Request;
import software.amazon.awssdk.services.s3.paginators.ListObjectsV2Paginator;
```

### Code

```java
// List all objects in bucket
ListObjectsV2Request listReq = ListObjectsV2Request.builder()
    .bucket(bucket)
    .maxKeys(2)
    .build();

ListObjectsV2Paginator listRes = s3.listObjectsV2Iterable(listReq);
// Dealing with ListObjectsV2Response pages
listRes.stream() // Helper method to work with paginated collection of items directly
    .flatMap(r -> r.contents().stream().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 in the pages of the response.

### Use a Stream

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

### Code

```java
// 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

```java
// Don't want to use fancy stream. Use simple for loop
for (S3Object content : listRes.contents()) {
    System.out.println(" Key: " + content.key() + " size = " + content.size());
}
```

See the complete example on GitHub.
Document History

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

This documentation was built on: Feb 07, 2018

Jun 28, 2017

New SDK version, 2.0 released.