Developer Guide for version 1.x

AWS SDK for Java 1.x

AWS SDK for Java 1.x: Developer Guide for version 1.x

Table of Contents

The AWS SDK for Java 1.x has entered maintenance mode as of July 31, 2024, and will reach end-of-support on December 31, 2025. We recommend that you migrate to the AWS SDK for Java 2.x to continue receiving new features, availability improvements, and security updates.

Developer Guide - AWS SDK for Java 1.x

The AWS SDK for Java provides a Java API for AWS 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 the supported services and their API versions that are included with each release of the SDK, view the release notes for the version that you're working with.

Version 2 of the SDK released

Take a look at the new AWS SDK for Java 2.x at https://github.com/aws/aws-sdk-java-v2/. It includes much awaited features, such as a way to plug in an HTTP implementation. To get started, see the AWS SDK for Java 2.x Developer Guide.

Additional Documentation and Resources

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

- AWS SDK for Java API Reference
- Java developer blog
- Java developer forums
- GitHub:
  - Documentation source
  - Documentation issues
  - SDK source
  - SDK issues
  - SDK samples
  - Gitter channel
- The AWS Code Sample Catalog
- @awsforjava (Twitter)
- release notes

Eclipse IDE Support

If you develop code using the Eclipse IDE, you can use the AWS Toolkit for Eclipse to add the AWS SDK for Java to an existing Eclipse project or to create a new AWS SDK for Java project. The toolkit also supports creating and uploading Lambda functions, launching and monitoring Amazon EC2 instances, managing IAM users and security groups, a AWS CloudFormation template editor, and more.

See the AWS Toolkit for Eclipse User Guide for full documentation.

Developing Applications for Android

If you're an Android developer, Amazon Web Services publishes an SDK made specifically for Android development: the Amplify Android (AWS Mobile SDK for Android).

Viewing the SDK's Revision History

To view the release history of the AWS SDK for Java, including changes and supported services per SDK version, see the SDK's release notes.

Building Java Reference Documentation for Earlier SDK versions

The AWS SDK for Java API Reference represents the most recent build of version 1.x of the SDK. If you're using an earlier build of the 1.x version, you might want to access the SDK reference documentation that matches the version you're using.

The easiest way to build the documentation is using Apache's Maven build tool. *Download and install Maven first if you don't already have it on your system*, then use the following instructions to build the reference documentation.

1. Locate and select the SDK version that you're using on the releases page of the SDK repository on GitHub.
2. Choose either the `zip` (most platforms, including Windows) or `tar.gz` (Linux, macOS, or Unix) link to download the SDK to your computer.
3. Unpack the archive to a local directory.

4. On the command line, navigate to the directory where you unpacked the archive, and type the following.

```
mvn javadoc:javadoc
```

5. After building is complete, you'll find the generated HTML documentation in the `aws-java-sdk/target/site/apidocs/` directory.

Getting Started

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

Topics

Basic setup to work with AWS services

Overview

To successfully develop applications that access AWS services using the AWS SDK for Java, the following conditions are required:

- You must be able to sign in to the AWS access portal available in the AWS IAM Identity Center.
- The permissions of the IAM role configured for the SDK must allow access to the AWS services that your application requires. The permissions associated with the **PowerUserAccess** AWS managed policy are sufficient for most development needs.
- A development environment with the following elements:
  - Shared configuration files that are set up in the following way:
    - The `config` file contains a default profile that specifies an AWS Region.
    - The `credentials` file contains temporary credentials as part of a default profile.
  - A suitable installation of Java.
  - A build automation tool such as Maven or Gradle.
  - A text editor to work with code.
  - (Optional, but recommended) An IDE (integrated development environment) such as IntelliJ IDEA, Eclipse, or NetBeans.

    When you use an IDE, you can also integrate AWS Toolkits to more easily work with AWS services. The AWS Toolkit for IntelliJ and AWS Toolkit for Eclipse are two toolkits that you can use for Java development.

Important

The instructions in this setup section assume that you or organization uses IAM Identity Center. If your organization uses an external identity provider that works independently of IAM Identity Center, find out how you can get temporary credentials for the SDK for Java to use. Follow these instructions to add temporary credentials to the `~/.aws/credentials` file.

If your identity provider adds temporary credentials automatically to the `~/.aws/credentials` file, make sure that the profile name is `[default]` so that you do not need to provide a profile name to the SDK or AWS CLI.

Sign-in ability to the AWS access portal

The AWS access portal is the web location where you manually sign in to the IAM Identity Center. The format of the URL is `d-xxxxxxxxxx.awsapps.com/start` or *`your_subdomain`*`.awsapps.com/start`.

If you are not familiar with the AWS access portal, follow the guidance for account access in Step 1 of the IAM Identity Center authentication topic in the AWS SDKs and Tools Reference Guide. Do not follow the Step 2 because the AWS SDK for Java 1.x does not support automatic token refresh and automatic retrieval of temporary credentials for the SDK that Step 2 describes.

Set up shared configuration files

The shared configuration files reside on your development workstation and contain basic settings used by all AWS SDKs and the AWS Command Line Interface (CLI). The shared configuration files can contain a number of settings, but these instructions set up the basic elements that are required to work with the SDK.

Set up the shared `config` file

The following example shows content of a shared `config` file.

```
[default]
region=us-east-1
output=json
```

For development purposes, use the AWS Region nearest to where you plan to run your code. For a listing of region codes to use in the `config` file see the Amazon Web Services General Reference guide. The `json` setting for the output format is one of several possible values.

Follow the guidance in this section to create the `config` file.

Set up temporary credentials for the SDK

After you have access to an AWS account and IAM role through the AWS access portal, configure your development environment with temporary credentials for the SDK to access.

Steps to set up a local `credentials` file with temporary credentials

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

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

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

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

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

6. Paste the copied credentials into your local `credentials` file and remove any profile name that was pasted. Your file should resemble the following:

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

```
aws_session_token=IQoJb3JpZ2luX2IQoJb3JpZ2luX2IQoJb3JpZ2luX2IQoJb3JpZ2luX2IQoJb3JpZVERYLONG
```

7. Save the `credentials` file

The SDK for Java will access these temporary credentials when it create a service client and use them for each request. The settings for the IAM role chosen in step 5a determine how long the temporary credentials are valid. The maximum duration is twelve hours.

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

Install a Java Development Environment

The AWS SDK for Java V1 requires a Java 7 JDK or newer and all Java LTS (long-term support) JDK versions are supported. If you use version 1.12.767 or earlier of the SDK, you can use Java 7, but if you use version 1.12.768 or newer of the SDK, Java 8 is required. The Maven central repository lists the latest version of the SDK for Java.

The AWS SDK for Java works with the Oracle Java SE Development Kit and with distributions of Open Java Development Kit (OpenJDK) such as Amazon Corretto, Red Hat OpenJDK, and Adoptium.

Ways to get the AWS SDK for Java

Prerequisites

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

- You must be able to sign in to the AWS access portal available in the AWS IAM Identity Center.
- A suitable installation of Java.
- Temporary credentials set up in your local shared `credentials` file.

See the the section called “Basic setup” topic for instructions on how to get set up to use the SDK for Java.

Use a build tool to manage dependencies for the SDK for Java (recommended)

We recommend using Apache Maven or Gradle with your project to access required dependencies of the SDK for Java. This section describes how to use those tools.

Download and extract the SDK (not recommended)

We recommend that you use a build tool to access the SDK for your project, You can, however, download a prebuilt jar of latest version of the SDK .

> **Note**
>
> For information about how to download and build previous versions of the SDK, see Installing previous versions of the SDK.

1. Download the SDK from https://sdk-for-java.amazonwebservices.com/latest/aws-java-sdk.zip.
2. After downloading the SDK, extract the contents into a local directory.

The SDK contains the following directories:

- `documentation`- contains the API documentation (also available on the web: AWS SDK for Java API Reference).
- `lib`- contains the SDK `.jar` files.
- `samples`- contains working sample code that demonstrates how to use the SDK.
- `third-party/lib`- contains third-party libraries that are used by the SDK, such as Apache commons logging, AspectJ and the Spring framework.

To use the SDK, add the full path to the `lib` and `third-party` directories to the dependencies in your build file, and add them to your java `CLASSPATH` to run your code.

Build previous versions of the SDK from source (not recommended)

Only the latest version of the complete SDK is provided in pre-built form as a downloadable jar. However, you can build a previous version of the SDK using Apache Maven (open source). Maven

will download all necessary dependencies, build and install the SDK in one step. Visit http://maven.apache.org/ for installation instructions and more information.

1. Go to the SDK's GitHub page at: AWS SDK for Java (GitHub).
2. Choose the tag corresponding to the version number of the SDK that you want. For example, `1.6.10`.
3. Click the **Download ZIP** button to download the version of the SDK you selected.
4. Unzip the file to a directory on your development system. On many systems, you can use your graphical file manager to do this, or use the `unzip` utility in a terminal window.
5. In a terminal window, navigate to the directory where you unzipped the SDK source.
6. Build and install the SDK with the following command (Maven required):

   ```
   mvn clean install -Dgpg.skip=true
   ```

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

   ```
   mvn javadoc:javadoc
   ```

   The documentation is built into the `target/site/apidocs/` directory.

Use build tools

The use of build tools helps manage the development of Java projects. Several build tools are available, but we show how to get up and running with two popular build tools--Maven and Gradle. This topic shows you how to use these build tools manage the SDK for Java dependencies that you need for your projects.

Topics

Use 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:

```
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, and *myapp* with the name of your project (this will become the name of the directory for your project).

By default, creates a project template for you using the quickstart archetype, which is a good starting place for many projects. There are more archetypes available; visit the Maven archetypes page for a list of archetypes packaged with . You can choose a particular archetype to use by adding the `-DarchetypeArtifactId` argument to the `archetype:generate` command. For example:

```
mvn archetype:generate \
  -DarchetypeGroupId=org.apache.maven.archetypes \
  -DarchetypeArtifactId=maven-archetype-webapp \
  -DgroupId=org.example.webapp \
  -DartifactId=mywebapp
```

> **Note**
>
> Much more information about creating and configuring projects is provided in the Maven Getting Started Guide.

Configure the SDK as a Maven dependency

To use the AWS SDK for Java in your project, you'll need to declare it as a dependency in your project's `pom.xml` file. Beginning with version 1.9.0, you can import individual components or the entire SDK.

Specifying individual SDK modules

To select individual SDK modules, use the AWS SDK for Java bill of materials (BOM) for Maven, which will ensure 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, adding `aws-java-sdk-bom` as a dependency and specifying the version of the SDK you want to use:

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>com.amazonaws</groupId>
      <artifactId>aws-java-sdk-bom</artifactId>
      <version>1.11.1000</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, visit: https://mvnrepository.com/artifact/com.amazonaws/aws-java-sdk-bom. You can also use this page to see which modules (dependencies) 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 in 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>com.amazonaws</groupId>
```

```
    <artifactId>aws-java-sdk-s3</artifactId>
  </dependency>
  <dependency>
    <groupId>com.amazonaws</groupId>
    <artifactId>aws-java-sdk-dynamodb</artifactId>
  </dependency>
</dependencies>
```

You can also refer to the *AWS Code Sample Catalog* to learn what dependencies to use for a given AWS service. Refer to the POM file under a specific service example. For example, if you are interested in the dependencies for the AWS S3 service, see the complete example on GitHub. (Look at the pom under /java/example_code/s3).

Importing all SDK modules

If you would like to pull in the *entire* SDK as a dependency, don't use the BOM method, but simply declare it in your `pom.xml` like this:

```
<dependencies>
  <dependency>
    <groupId>com.amazonaws</groupId>
    <artifactId>aws-java-sdk</artifactId>
    <version>1.11.1000</version>
  </dependency>
</dependencies>
```

Build your project

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

```
mvn package
```

This will create your `0jar` file in the `target` directory.

Build the SDK with Maven

You can use Apache Maven to build the SDK from source. To do so, download the SDK code from GitHub, unpack it locally, and then execute the following Maven command:

```
mvn clean install
```

Use the SDK with Gradle

To manage SDK dependencies for your Gradle project, import the Maven BOM for the AWS SDK for Java into the application's `build.gradle` file.

> **Note**
>
> In the following examples, replace *1.12.529* in the build file with a valid version of the AWS SDK for Java. Find the latest version in the Maven central repository.

Project setup for Gradle 4.6 or higher

Since Gradle 4.6, you can use Gradle's improved POM support feature to import bill of materials (BOM) files by declaring a dependency on a BOM.

1. If you're using Gradle 5.0 or later, skip to step 2. Otherwise, enable the *IMPROVED_POM_SUPPORT* feature in the `settings.gradle` file.

```
enableFeaturePreview('IMPROVED_POM_SUPPORT')
```

2. Add the BOM to the *dependencies* section of the application's `build.gradle` file.

```
...
dependencies {
    implementation platform('com.amazonaws:aws-java-sdk-bom:1.12.529')

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

3. Specify the SDK modules to use in the *dependencies* section. For example, the following includes a dependency for Amazon Simple Storage Service (Amazon S3).

```
...
dependencies {
    implementation platform('com.amazonaws:aws-java-sdk-bom:1.12.529')
    implementation 'com.amazonaws:aws-java-sdk-s3'
    ...
}
```

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

The following is an example of a complete `build.gradle` file that includes a dependency for Amazon S3.

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

apply plugin: 'java'

sourceCompatibility = 1.8

repositories {
  mavenCentral()
}

dependencies {
  implementation platform('com.amazonaws:aws-java-sdk-bom:1.12.529')
  implementation 'com.amazonaws:aws-java-sdk-s3'
}
```

Note

In the previous example, replace the dependency for Amazon S3 with the dependencies of the AWS services you will use in your project. The modules (dependencies) that are managed by the AWS SDK for Java BOM are listed on Maven central repository.

Project setup for Gradle versions earlier than 4.6

Gradle versions earlier than 4.6 lack native BOM support. To manage AWS SDK for Java dependencies for your project, use Spring's dependency management plugin for Gradle to import the Maven BOM for the SDK.

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

```
buildscript {
    repositories {
        mavenCentral()
    }
```

```
    dependencies {
        classpath "io.spring.gradle:dependency-management-plugin:1.0.9.RELEASE"
    }
}

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

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

```
dependencyManagement {
    imports {
        mavenBom 'com.amazonaws:aws-java-sdk-bom:1.12.529'
    }
}
```

3. Specify the SDK modules that you'll use in the *dependencies* section. For example, the following includes a dependency for Amazon S3.

```
dependencies {
    compile 'com.amazonaws:aws-java-sdk-s3'
}
```

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

The following is an example of a complete `build.gradle` file that includes a dependency for Amazon S3.

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

apply plugin: 'java'

sourceCompatibility = 1.8

repositories {
  mavenCentral()
}

buildscript {
  repositories {
    mavenCentral()
```

```
  }
  dependencies {
    classpath "io.spring.gradle:dependency-management-plugin:1.0.9.RELEASE"
  }
}

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

dependencyManagement {
  imports {
    mavenBom 'com.amazonaws:aws-java-sdk-bom:1.12.529'
  }
}

dependencies {
  compile 'com.amazonaws:aws-java-sdk-s3'
  testCompile group: 'junit', name: 'junit', version: '4.11'
}
```

Note

In the previous example, replace the dependency for Amazon S3 with the dependencies of the AWS service you will use in your project. The modules (dependencies) that are managed by the AWS SDK for Java BOM are listed on Maven central repository.

For more information about specifying SDK dependencies by using the BOM, see Using the SDK with Apache Maven.

Set up AWS temporary credentials and AWS Region for development

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

This topic provides basic information about setting up your AWS temporary credentials for local application development using the AWS SDK for Java. If you need to set up credentials for use

within an EC2 instance or if you're using the Eclipse IDE for development, refer to 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 Using IAM Roles to Grant Access to AWS Resources on Amazon EC2.
- 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 for more information.

Configure temporary credentials

You can configure temporary credentials for the AWS SDK for Java in a number of ways, but here are the recommended approaches:

- Set temporary 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

  See the the section called "Set up temporary credentials for the SDK" in this guide for instructions on how to get your temporary credentials.
- Set the `AWS_ACCESS_KEY_ID`,`AWS_SECRET_ACCESS_KEY`, and `AWS_SESSION_TOKEN` environment variables.

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

  ```
  export AWS_ACCESS_KEY_ID=your_access_key_id
  export AWS_SECRET_ACCESS_KEY=your_secret_access_key
  export AWS_SESSION_TOKEN=your_session_token
  ```

  To set these variables on Windows, use:

  ```
  set AWS_ACCESS_KEY_ID=your_access_key_id
  set AWS_SECRET_ACCESS_KEY=your_secret_access_key
  set AWS_SESSION_TOKEN=your_session_token
  ```
- 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 have set your AWS temporary credentials using one of these methods, they will be loaded automatically by the AWS SDK for Java by using the default credential provider chain. For further information about working with AWS credentials in your Java applications, see Working with AWS Credentials.

Refreshing IMDS credentials

The AWS SDK for Java supports opt-in refreshing IMDS credentials in the background every 1 minute, regardless of the credential expiration time. This allows you to refresh credentials more frequently and reduces the chance that not reaching IMDS impacts the perceived AWS availability.

```
1. // Refresh credentials using a background thread, automatically every minute. This
will log an error if IMDS is down during
2. // a refresh, but your service calls will continue using the cached credentials
until the credentials are refreshed
3. // again one minute later.
4.
5. InstanceProfileCredentialsProvider credentials =
6.     InstanceProfileCredentialsProvider.createAsyncRefreshingProvider(true);
7.
8. AmazonS3Client.builder()
9.              .withCredentials(credentials)
10.              .build();
11.
12. // This is new: When you are done with the credentials provider, you must close it
to release the background thread.
13. credentials.close();
```

Set the AWS Region

You should set a default AWS Region that will be used 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). For a list of regions for each service, see Regions and Endpoints in the Amazon Web Services General Reference.

> **Note**
>
> If you *don't* select a region, then us-east-1 will be used by default.

You can use similar techniques to 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-east-1") 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 AWS SDK for Java

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.

For service-specific programming information and examples (for Amazon EC2, Amazon S3, Amazon SWF, etc.), see AWS SDK for Java Code Examples.

Topics

Best Practices for AWS Development with the AWS SDK for Java

The following best practices can help you avoid issues or trouble as you develop AWS applications with the AWS SDK for Java. We've organized best practices by service.

S3

Avoid ResetExceptions

When you upload objects to Amazon S3 by using streams (either through an `AmazonS3` client or `TransferManager`), you might encounter network connectivity or timeout issues. By default, the AWS SDK for Java attempts to retry failed transfers by marking the input stream before the start of a transfer and then resetting it before retrying.

If the stream doesn't support mark and reset, the SDK throws a ResetException when there are transient failures and retries are enabled.

Best Practice

We recommend that you use streams that support mark and reset operations.

The most reliable way to avoid a ResetException is to provide data by using a File or FileInputStream, which the AWS SDK for Java can handle without being constrained by mark and reset limits.

If the stream isn't a FileInputStream but does support mark and reset, you can set the mark limit by using the `setReadLimit` method of RequestClientOptions. Its default value is 128 KB. Setting the read limit value to *one byte greater than the size of stream* will reliably avoid a ResetException.

For example, if the maximum expected size of a stream is 100,000 bytes, set the read limit to 100,001 (100,000 + 1) bytes. The mark and reset will always work for 100,000 bytes or less. Be aware that this might cause some streams to buffer that number of bytes into memory.

Creating Service Clients

To make requests to Amazon Web Services, you first create a service client object. The recommended way is to use the service client builder.

Each AWS service has a service interface with methods for each action in the service API. For example, the service interface for DynamoDB is named AmazonDynamoDBClient. Each service interface has a corresponding client builder you can use to construct an implementation of the service interface. The client builder class for DynamoDB is named AmazonDynamoDBClientBuilder.

Obtaining a Client Builder

To obtain an instance of the client builder, use the static factory method `standard`, as shown in the following example.

```
AmazonDynamoDBClientBuilder builder = AmazonDynamoDBClientBuilder.standard();
```

Once you have a builder, you can customize the client's properties by using many fluent setters in the builder API. For example, you can set a custom region and a custom credentials provider, as follows.

```
AmazonDynamoDB ddb = AmazonDynamoDBClientBuilder.standard()
                         .withRegion(Regions.US_WEST_2)
                         .withCredentials(new ProfileCredentialsProvider("myProfile"))
                         .build();
```

Note

The fluent `withXXX` 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 and any calls to `setRegion` or `setEndpoint` will fail.

A builder can create multiple clients with the same configuration. When you're writing your application, be aware that the builder is mutable and not thread-safe.

The following code uses the builder as a factory for client instances.

```
public class DynamoDBClientFactory {
    private final AmazonDynamoDBClientBuilder builder =
        AmazonDynamoDBClientBuilder.standard()
            .withRegion(Regions.US_WEST_2)
            .withCredentials(new ProfileCredentialsProvider("myProfile"));

    public AmazonDynamoDB createClient() {
        return builder.build();
    }
}
```

The builder also exposes fluent setters for ClientConfiguration and RequestMetricCollector, and a custom list of RequestHandler2.

The following is a complete example that overrides all configurable properties.

```
AmazonDynamoDB ddb = AmazonDynamoDBClientBuilder.standard()
        .withRegion(Regions.US_WEST_2)
        .withCredentials(new ProfileCredentialsProvider("myProfile"))
        .withClientConfiguration(new ClientConfiguration().withRequestTimeout(5000))
        .withMetricsCollector(new MyCustomMetricsCollector())
        .withRequestHandlers(new MyCustomRequestHandler(), new
 MyOtherCustomRequestHandler)
```

```
        .build();
```

Creating Async Clients

The AWS SDK for Java has asynchronous (or async) clients for every service (except for Amazon S3), and a corresponding async client builder for every service.

To create an async DynamoDB client with the default ExecutorService

```
AmazonDynamoDBAsync ddbAsync = AmazonDynamoDBAsyncClientBuilder.standard()
        .withRegion(Regions.US_WEST_2)
        .withCredentials(new ProfileCredentialsProvider("myProfile"))
        .build();
```

In addition to the configuration options that the synchronous (or sync) client builder supports, the async client enables you to set a custom ExecutorFactory to change the `ExecutorService` that the async client uses. `ExecutorFactory` is a functional interface, so it interoperates with Java 8 lambda expressions and method references.

To create an async client with a custom executor

```
AmazonDynamoDBAsync ddbAsync = AmazonDynamoDBAsyncClientBuilder.standard()
            .withExecutorFactory(() -> Executors.newFixedThreadPool(10))
            .build();
```

Using DefaultClient

Both the sync and async client builders have another factory method named `defaultClient`. This method creates a service client with the default configuration, using 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 `defaultClient` fails. See Working with AWS Credentials and AWS Region Selection for more information about how credentials and region are determined.

To create a default service client

```
AmazonDynamoDB ddb = AmazonDynamoDBClientBuilder.defaultClient();
```

Client Lifecycle

Service clients in the SDK are thread-safe and, for best performance, you should treat them as long-lived objects. Each client has its own connection pool resource. Explicitly shut down clients when they are no longer needed to avoid resource leaks.

To explicitly shut down a client, call the `shutdown` method. After calling `shutdown`, all client resources are released and the client is unusable.

To shut down a client

```
AmazonDynamoDB ddb = AmazonDynamoDBClientBuilder.defaultClient();
ddb.shutdown();
// Client is now unusable
```

Provide temporary credentials to the AWS SDK for Java

To make requests to Amazon Web Services, you must supply AWS temporary credentials for the AWS SDK for Java to use when it calls the services. 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 temporary credentials yourself in code.

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 temporary credentials by using the *default credential provider chain* implemented by the DefaultAWSCredentialsProviderChain class. The default credential provider chain looks for credentials in this order:

1. **Environment variables**-`AWS_ACCESS_KEY_ID`, `AWS_SECRET_KEY` or `AWS_SECRET_ACCESS_KEY`, and `AWS_SESSION_TOKEN`. The AWS SDK for Java uses the EnvironmentVariableCredentialsProvider class to load these credentials.
2. **Java system properties**-`aws.accessKeyId`, `aws.secretKey` (but not `aws.secretAccessKey`), and `aws.sessionToken`. The AWS SDK for Java uses the SystemPropertiesCredentialsProvider to load these credentials.

3. **Web Identity Token credentials** from the environment or container.
4. **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.
5. **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 ContainerCredentialsProvider to load these credentials. You can specify the IP address for this value.
6. **Instance profile credentials**- used on EC2 instances, and delivered through the Amazon EC2 metadata service. The AWS SDK for Java uses the InstanceProfileCredentialsProvider to load these credentials. You can specify the IP address for this value.

> **Note**
>
> Instance profile credentials are used only if `AWS_CONTAINER_CREDENTIALS_RELATIVE_URI` is not set. See EC2ContainerCredentialsProviderWrapper for more information.

Set temporary credentials

To be able to use AWS temporary 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 the section called “Configure temporary credentials” .
- 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 Using IAM Roles to Grant Access to AWS Resources on Amazon EC2.

Set 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 would run the following command to change the profile to *myProfile*.

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

On Windows you would 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 Tools for Windows PowerShell). To change only the profile for a Java application, you can use the system property `aws.profile` instead.

> **Note**
>
> The environment variable takes precedence over the system property.

Set an alternate credentials file location

The AWS SDK for Java loads AWS temporary 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 AWS_CREDENTIAL_PROFILES_FILE=path/to/credentials_file
```

  - On Windows, use:

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

`Credentials` file format

By following the instructions in the Basic setup of this guide, your credentials file should have the following basic format.

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

[profile2]
aws_access_key_id=<value from AWS access portal>
aws_secret_access_key=<value from AWS access portal>
aws_session_token=<value from AWS access portal>
```

The profile name is specified in square brackets (for example, `[default]`), followed by the configurable fields in that profile as key-value pairs. You can have multiple profiles in your `credentials` file, which can be added or edited using `aws configure --profile` *`PROFILE_NAME`* to select the profile to configure.

You can specify additional fields, such as `metadata_service_timeout`, and `metadata_service_num_attempts`. These are not configurable with the CLI—you must edit the file by hand if you want to use them. For more information about the configuration file and its available fields, see Configuring the AWS Command Line Interface in the AWS Command Line Interface User Guide.

Load credentials

After you set temporary credentials, the SDK loads them by using the default credential provider chain.

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

```
AmazonS3 s3Client = AmazonS3ClientBuilder.standard()
                        .withRegion(Regions.US_WEST_2)
                        .build();
```

Specify 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.

```
AmazonS3 s3Client = AmazonS3ClientBuilder.standard()
                        .withCredentials(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, or by subclassing the AWSCredentialsProviderChain class.

Explicitly specify temporary 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.

1. Instantiate the BasicSessionCredentials class, and supply it with the AWS access key, AWS secret key, and AWS session token that the SDK 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.

```
BasicSessionCredentials awsCreds = new BasicSessionCredentials("access_key_id",
 "secret_key_id", "session_token");
AmazonS3 s3Client = AmazonS3ClientBuilder.standard()
                        .withCredentials(new AWSStaticCredentialsProvider(awsCreds))
                        .build();
```

More Info

- Sign Up for AWS and Create an IAM User
- Set up AWS Credentials and Region for Development
- Using IAM Roles to Grant Access to AWS Resources on Amazon EC2

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.

Checking for Service Availability in a Region

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

```
Region.getRegion(Regions.US_WEST_2)
    .isServiceSupported(AmazonDynamoDB.ENDPOINT_PREFIX);
```

See the Regions class documentation for the regions you can specify, and use the endpoint prefix of the service to query. Each service's endpoint prefix is defined in the service interface. For example, the DynamoDB endpoint prefix is defined in AmazonDynamoDB.

Choosing a Region

Beginning with version 1.4 of the AWS SDK for Java, you can specify a region name and the SDK will automatically choose an appropriate endpoint for you. To choose the endpoint yourself, see Choosing a Specific Endpoint.

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

```
AmazonEC2 ec2 = AmazonEC2ClientBuilder.standard()
                    .withRegion(Regions.US_WEST_2)
                    .build();
```

If the region you are attempting to use isn't in the `Regions` enum, you can set the region using a *string* that represents the name of the region.

```
AmazonEC2 ec2 = AmazonEC2ClientBuilder.standard()
                    .withRegion("{region_api_default}")
                    .build();
```

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

Choosing a Specific Endpoint

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

For example, to configure the Amazon S3 client to use the Europe (Ireland) Region, use the following code.

```
AmazonS3 s3 = AmazonS3ClientBuilder.standard()
     .withEndpointConfiguration(new EndpointConfiguration(
          "https://s3.eu-west-1.amazonaws.com",
          "eu-west-1"))
```

```
    .withCredentials(CREDENTIALS_PROVIDER)
    .build();
```

See Regions and Endpoints for the current list of regions and their corresponding endpoints for all AWS services.

Automatically Determine the Region from the Environment

> **Important**
>
> This section applies only when using a client builder to access AWS services. AWS clients created by using the client constructor will not automatically determine region from the environment and will, instead, use the *default* SDK region (USEast1).

When running on Amazon EC2 or 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.

You must use client builders to have the SDK automatically detect the region your code is running in.

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

```
AmazonEC2 ec2 = AmazonEC2ClientBuilder.defaultClient();
```

This is the same as using `standard` followed by `build`.

```
AmazonEC2 ec2 = AmazonEC2ClientBuilder.standard()
                    .build();
```

If you don't explicitly set a region using the `withRegion` methods, 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 `withRegion` or `setRegion` 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 is loaded by the SDK.
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) 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.

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.

For more information about the use of checked and unchecked exceptions, see:

- Unchecked Exceptions—The Controversy
- The Trouble with Checked Exceptions
- Java's checked exceptions were a mistake (and here's what I would like to do about it)

AmazonServiceException (and Subclasses)

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

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

`AmazonServiceException` provides you with information such as:

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

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

AmazonClientException

AmazonClientException indicates that a problem occurred inside the Java client code, either while trying to send a request to AWS or while trying to parse a response from AWS. An `AmazonClientException` is generally more severe than an `AmazonServiceException`, and indicates a major problem that is preventing the client from making service calls to AWS

services. For example, the AWS SDK for Java throws an `AmazonClientException` if no network connection is available when you try to call an operation on one of the clients.

Asynchronous Programming

You can use either *synchronous* or *asynchronous* methods to call operations on AWS services. 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 provides two ways: *Future objects* and *callback methods*.

Java Futures

Asynchronous methods in the AWS SDK for Java return a Future object that contains the results of the asynchronous operation *in the future*.

Call the `Future` `isDone()` method to see if the service has provided a response object yet. When the response is ready, you can get the response object by calling the `Future` `get()` method. You can use this mechanism to periodically poll for the asynchronous operation's results while your application continues to work on other things.

Here is an example of an asynchronous operation that calls a Lambda function, receiving a `Future` that can hold an InvokeResult object. The `InvokeResult` object is retrieved only after `isDone()` is `true`.

```
import com.amazonaws.services.lambda.AWSLambdaAsyncClient;
import com.amazonaws.services.lambda.model.InvokeRequest;
import com.amazonaws.services.lambda.model.InvokeResult;
import java.nio.ByteBuffer;
import java.util.concurrent.Future;
import java.util.concurrent.ExecutionException;

public class InvokeLambdaFunctionAsync
{
    public static void main(String[] args)
    {
        String function_name = "HelloFunction";
        String function_input = "{\"who\":\"SDK for Java\"}";
```

```
        AWSLambdaAsync lambda = AWSLambdaAsyncClientBuilder.defaultClient();
        InvokeRequest req = new InvokeRequest()
            .withFunctionName(function_name)
            .withPayload(ByteBuffer.wrap(function_input.getBytes()));

        Future<InvokeResult> future_res = lambda.invokeAsync(req);

        System.out.print("Waiting for future");
        while (future_res.isDone() == false) {
            System.out.print(".");
            try {
                Thread.sleep(1000);
            }
            catch (InterruptedException e) {
                System.err.println("\nThread.sleep() was interrupted!");
                System.exit(1);
            }
        }

        try {
            InvokeResult res = future_res.get();
            if (res.getStatusCode() == 200) {
                System.out.println("\nLambda function returned:");
                ByteBuffer response_payload = res.getPayload();
                System.out.println(new String(response_payload.array()));
            }
            else {
                System.out.format("Received a non-OK response from {AWS}: %d\n",
                        res.getStatusCode());
            }
        }
        catch (InterruptedException | ExecutionException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }

        System.exit(0);
    }
}
```

Asynchronous Callbacks

In addition to using the Java `Future` object to monitor the status of asynchronous requests, the SDK also enables you to implement a class that uses the AsyncHandler interface. `AsyncHandler` provides two methods that are called depending on how the request completed: `onSuccess` and `onError`.

The major advantage of the callback interface approach is that it frees you from having to poll the `Future` object to find out when the request has completed. Instead, your code can immediately start its next activity, and rely on the SDK to call your handler at the right time.

```
import com.amazonaws.services.lambda.AWSLambdaAsync;
import com.amazonaws.services.lambda.AWSLambdaAsyncClientBuilder;
import com.amazonaws.services.lambda.model.InvokeRequest;
import com.amazonaws.services.lambda.model.InvokeResult;
import com.amazonaws.handlers.AsyncHandler;
import java.nio.ByteBuffer;
import java.util.concurrent.Future;

public class InvokeLambdaFunctionCallback
{
    private class AsyncLambdaHandler implements AsyncHandler<InvokeRequest,
 InvokeResult>
    {
        public void onSuccess(InvokeRequest req, InvokeResult res) {
            System.out.println("\nLambda function returned:");
            ByteBuffer response_payload = res.getPayload();
            System.out.println(new String(response_payload.array()));
            System.exit(0);
        }

        public void onError(Exception e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }

    public static void main(String[] args)
    {
        String function_name = "HelloFunction";
        String function_input = "{\"who\":\"SDK for Java\"}";
```

```
        AWSLambdaAsync lambda = AWSLambdaAsyncClientBuilder.defaultClient();
        InvokeRequest req = new InvokeRequest()
            .withFunctionName(function_name)
            .withPayload(ByteBuffer.wrap(function_input.getBytes()));

        Future<InvokeResult> future_res = lambda.invokeAsync(req, new
 AsyncLambdaHandler());

        System.out.print("Waiting for async callback");
        while (!future_res.isDone() && !future_res.isCancelled()) {
            // perform some other tasks...
            try {
                Thread.sleep(1000);
            }
            catch (InterruptedException e) {
                System.err.println("Thread.sleep() was interrupted!");
                System.exit(0);
            }
            System.out.print(".");
        }
    }
}
```

Best Practices

Callback Execution

Your implementation of `AsyncHandler` is executed inside the thread pool owned by the asynchronous client. Short, quickly executed code is most appropriate inside your `AsyncHandler` implementation. Long-running or blocking code inside your handler methods can cause contention for the thread pool used by the asynchronous client, and can prevent the client from executing requests. If you have a long-running task that needs to begin from a callback, have the callback run its task in a new thread or in a thread pool managed by your application.

Thread Pool Configuration

The asynchronous clients in the AWS SDK for Java provide a default thread pool that should work for most applications. You can implement a custom ExecutorService and pass it to AWS SDK for Java asynchronous clients for more control over how the thread pools are managed.

For example, you could provide an `ExecutorService` implementation that uses a custom ThreadFactory to control how threads in the pool are named, or to log additional information about thread usage.

Asynchronous Access

The TransferManager class in the SDK offers asynchronous support for working with Amazon S3. `TransferManager` manages asynchronous uploads and downloads, provides detailed progress reporting on transfers, and supports callbacks into different events.

Logging AWS SDK for Java Calls

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

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

To learn more about Log4j, see the Apache website.

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

Download the Log4J JAR

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

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

The log4j.properties configuration file specifies properties such as logging level, where logging output is sent (for example, to a file or to the console), and the format of the output. The logging

level is the granularity of output that the logger generates. Log4j supports the concept of multiple logging *hierarchies*. The logging level is set independently for each hierarchy. The following two logging hierarchies are available in the AWS SDK for Java:

- log4j.logger.com.amazonaws
- log4j.logger.org.apache.http.wire

Setting the Classpath

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

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

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

Service-Specific Errors and Warnings

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

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

```
log4j.rootLogger=WARN, A1
log4j.appender.A1=org.apache.log4j.ConsoleAppender
log4j.appender.A1.layout=org.apache.log4j.PatternLayout
log4j.appender.A1.layout.ConversionPattern=%d [%t] %-5p %c -  %m%n
```

```
# Or you can explicitly enable WARN and ERROR messages for the {AWS} Java clients
log4j.logger.com.amazonaws=WARN
```

Request/Response Summary Logging

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

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

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

Here is an example of the log output.

```
2009-12-17 09:53:04,269 [main] DEBUG com.amazonaws.request - Sending
Request: POST https://rds.amazonaws.com / Parameters: (MaxRecords: 20,
Action: DescribeEngineDefaultParameters, SignatureMethod: HmacSHA256,
AWSAccessKeyId: ACCESSKEYID, Version: 2009-10-16, SignatureVersion: 2,
Engine: mysql5.1, Timestamp: 2009-12-17T17:53:04.267Z, Signature:
q963XH63Lcovl5Rr71APlzlye99rmWwT9DfuQaNznkD, ) 2009-12-17 09:53:04,464
[main] DEBUG com.amazonaws.request - Received successful response: 200, {AWS}
Request ID: 694d1242-cee0-c85e-f31f-5dab1ea18bc6 2009-12-17 09:53:04,469
[main] DEBUG com.amazonaws.request - Sending Request: POST
https://rds.amazonaws.com / Parameters: (ResetAllParameters: true, Action:
ResetDBParameterGroup, SignatureMethod: HmacSHA256, DBParameterGroupName:
java-integ-test-param-group-0000000000000, AWSAccessKeyId: ACCESSKEYID,
Version: 2009-10-16, SignatureVersion: 2, Timestamp:
2009-12-17T17:53:04.467Z, Signature:
9WcgfPwTobvLVcpyhbrdN7P7l3uH0oviYQ4yZ+TQjsQ=, )

2009-12-17 09:53:04,646 [main] DEBUG com.amazonaws.request - Received
successful response: 200, {AWS} Request ID:
```

```
694d1242-cee0-c85e-f31f-5dab1ea18bc6
```

Verbose Wire Logging

In some cases, it can be useful to see the exact requests and responses that the AWS SDK for Java sends and receives. You shouldn't enable this logging in production systems because writing out large requests (e.g., a file being uploaded to Amazon S3) or responses can significantly slow down an application. If you really need access to this information, you can temporarily enable it through the Apache HttpClient 4 logger. Enabling the DEBUG level on the `org.apache.http.wire` logger enables logging for all request and response data.

The following log4j.properties file turns on full wire logging in Apache HttpClient 4 and should only be turned on temporarily because it can have a significant performance impact on your application.

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

Latency Metrics Logging

If you are troubleshooting and want to see metrics such as which process is taking the most time or whether server or client side has the greater latency, the latency logger can be helpful. Set the `com.amazonaws.latency` logger to DEBUG to enable this logger.

> **Note**
>
> This logger is only available if SDK metrics is enabled. To learn more about the SDK metrics package, see Enabling Metrics for the AWS SDK for Java.

```
log4j.rootLogger=WARN, A1
log4j.appender.A1=org.apache.log4j.ConsoleAppender
log4j.appender.A1.layout=org.apache.log4j.PatternLayout
```

```
log4j.appender.A1.layout.ConversionPattern=%d [%t] %-5p %c -  %m%n
log4j.logger.com.amazonaws.latency=DEBUG
```

Here is an example of the log output.

```
com.amazonaws.latency - ServiceName=[{S3}], StatusCode=[200],
ServiceEndpoint=[https://list-objects-integ-test-test.s3.amazonaws.com],
RequestType=[ListObjectsV2Request], AWSRequestID=[REQUESTID],
 HttpClientPoolPendingCount=0,
RetryCapacityConsumed=0, HttpClientPoolAvailableCount=0, RequestCount=1,
HttpClientPoolLeasedCount=0, ResponseProcessingTime=[52.154],
 ClientExecuteTime=[487.041],
HttpClientSendRequestTime=[192.931], HttpRequestTime=[431.652],
 RequestSigningTime=[0.357],
CredentialsRequestTime=[0.011, 0.001], HttpClientReceiveResponseTime=[146.272]
```

Client Configuration

The AWS SDK for Java enables you to change the default client configuration, which is helpful when you want to:

- Connect to the Internet through proxy
- Change HTTP transport settings, such as connection timeout and request retries
- Specify TCP socket buffer size hints

Proxy Configuration

When constructing a client object, you can pass in an optional ClientConfiguration object to customize the client's configuration.

If you connect to the Internet through a proxy server, you'll need to configure your proxy server settings (proxy host, port, and username/password) through the `ClientConfiguration` object.

HTTP Transport Configuration

You can configure several HTTP transport options by using the ClientConfiguration object. New options are occasionally added; to see the full list of options you can retrieve or set, see the AWS SDK for Java API Reference.

Note

Each of the configurable values has a default value defined by a constant. For a list of the constant values for `ClientConfiguration`, see Constant Field Values in the AWS SDK for Java API Reference.

Maximum Connections

You can set the maximum allowed number of open HTTP connections by using the ClientConfiguration.setMaxConnections method.

Important

Set the maximum connections to the number of concurrent transactions to avoid connection contentions and poor performance. For the default maximum connections value, see Constant Field Values in the AWS SDK for Java API Reference.

Timeouts and Error Handling

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

- **Connection Timeout**

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

  To set this value yourself, use the ClientConfiguration.setConnectionTimeout method.
- **Connection Time to Live (TTL)**

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

  To set the HTTP connection TTL, use the ClientConfiguration.setConnectionTTL method.
- **Maximum Error Retries**

The default maximum retry count for retriable errors is 3. You can set a different value by using the ClientConfiguration.setMaxErrorRetry method.

Local Address

To set the local address that the HTTP client will bind to, use ClientConfiguration.setLocalAddress.

TCP Socket Buffer Size Hints

Advanced users who want to tune low-level TCP parameters can additionally set TCP buffer size hints through the ClientConfiguration object. The majority of users will never need to tweak these values, but they are provided for advanced users.

Optimal TCP buffer sizes for an application are highly dependent on network and operating system configuration and capabilities. For example, most modern operating systems provide auto-tuning logic for TCP buffer sizes.This can have a big impact on performance for TCP connections that are held open long enough for the auto-tuning to optimize buffer sizes.

Large buffer sizes (e.g., 2 MB) allow the operating system to buffer more data in memory without requiring the remote server to acknowledge receipt of that information, and so can be particularly useful when the network has high latency.

This is only a *hint*, and the operating system might not honor it. When using this option, users should always check the operating system's configured limits and defaults. Most operating systems have a maximum TCP buffer size limit configured, and won't let you go beyond that limit unless you explicitly raise the maximum TCP buffer size limit.

Many resources are available to help with configuring TCP buffer sizes and operating system-specific TCP settings, including the following:

- Host Tuning

Access Control Policies

AWS *access control policies* enable you to specify fine-grained access controls on your AWS resources. An access control policy consists of a collection of *statements*, which take the form:

Account A has permission to perform *action B* on *resource C* where *condition D* applies.

Where:

- *A* is the *principal*- The AWS account that is making a request to access or modify one of your AWS resources.
- *B* is the *action*- The way in which your AWS resource is being accessed or modified, such as sending a message to an Amazon SQS queue, or storing an object in an Amazon S3 bucket.
- *C* is the *resource*- The AWS entity that the principal wants to access, such as an Amazon SQS queue, or an object stored in Amazon S3.
- *D* is a *set of conditions*- The optional constraints that specify when to allow or deny access for the principal to access your resource. Many expressive conditions are available, some specific to each service. For example, you can use date conditions to allow access to your resources only after or before a specific time.

Amazon S3 Example

The following example demonstrates a policy that allows anyone access to read all the objects in a bucket, but restricts access to uploading objects to that bucket to two specific AWS accounts (in addition to the bucket owner's account).

```
Statement allowPublicReadStatement = new Statement(Effect.Allow)
    .withPrincipals(Principal.AllUsers)
    .withActions(S3Actions.GetObject)
    .withResources(new S3ObjectResource(myBucketName, "*"));
Statement allowRestrictedWriteStatement = new Statement(Effect.Allow)
    .withPrincipals(new Principal("123456789"), new Principal("876543210"))
    .withActions(S3Actions.PutObject)
    .withResources(new S3ObjectResource(myBucketName, "*"));

Policy policy = new Policy()
    .withStatements(allowPublicReadStatement, allowRestrictedWriteStatement);

AmazonS3 s3 = AmazonS3ClientBuilder.defaultClient();
s3.setBucketPolicy(myBucketName, policy.toJson());
```

Amazon SQS Example

One common use of policies is to authorize an Amazon SQS queue to receive messages from an Amazon SNS topic.

```
Policy policy = new Policy().withStatements(
    new Statement(Effect.Allow)
        .withPrincipals(Principal.AllUsers)
        .withActions(SQSActions.SendMessage)
        .withConditions(ConditionFactory.newSourceArnCondition(myTopicArn)));

Map queueAttributes = new HashMap();
queueAttributes.put(QueueAttributeName.Policy.toString(), policy.toJson());

AmazonSQS sqs = AmazonSQSClientBuilder.defaultClient();
sqs.setQueueAttributes(new SetQueueAttributesRequest(myQueueUrl, queueAttributes));
```

Amazon SNS Example

Some services offer additional conditions that can be used in policies. Amazon SNS provides conditions for allowing or denying subscriptions to SNS topics based on the protocol (e.g., email, HTTP, HTTPS, Amazon SQS) and endpoint (e.g., email address, URL, Amazon SQS ARN) of the request to subscribe to a topic.

```
Condition endpointCondition =
    SNSConditionFactory.newEndpointCondition("*@mycompany.com");

Policy policy = new Policy().withStatements(
    new Statement(Effect.Allow)
        .withPrincipals(Principal.AllUsers)
        .withActions(SNSActions.Subscribe)
        .withConditions(endpointCondition));

AmazonSNS sns = AmazonSNSClientBuilder.defaultClient();
sns.setTopicAttributes(
    new SetTopicAttributesRequest(myTopicArn, "Policy", policy.toJson()));
```

Set the JVM TTL for DNS name lookups

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

Because AWS resources use DNS name entries that occasionally change, we recommend that you configure your JVM with a TTL value of 5 seconds. This ensures that when a resource's IP

address changes, your application will be able to receive and use the resource's new IP address by requerying the DNS.

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

How to set the JVM TTL

To modify the JVM's TTL, set the networkaddress.cache.ttl security property value, set the `networkaddress.cache.ttl` property in the `$JAVA_HOME/jre/lib/security/java.security` file for Java 8 or `$JAVA_HOME/conf/security/java.security` file for Java 11 or higher.

The following is a snippet from a `java.security` file that shows the TTL cache set to 5 seconds.

```
#
# This is the "master security properties file".
#
# An alternate java.security properties file may be specified
...
# The Java-level namelookup cache policy for successful lookups:
#
# any negative value: caching forever
# any positive value: the number of seconds to cache an address for
# zero: do not cache
...
networkaddress.cache.ttl=5
...
```

All applications that run on the JVM represented by the`$JAVA_HOME` environment variable use this setting.

Enabling Metrics for the AWS SDK for Java

The AWS SDK for Java can generate metrics for visualization and monitoring with Amazon CloudWatch that measure:

- your application's performance when accessing AWS
- the performance of your JVMs when used with AWS
- runtime environment details such as heap memory, number of threads, and opened file descriptors

How to Enable Java SDK Metric Generation

You need to add the following Maven dependency to enable the SDK to send metrics to CloudWatch.

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>com.amazonaws</groupId>
      <artifactId>aws-java-sdk-bom</artifactId>
      <version>1.12.490*</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
<dependencies>
  <dependency>
    <groupId>com.amazonaws</groupId>
    <artifactId>aws-java-sdk-cloudwatchmetrics</artifactId>
    <scope>provided</scope>
  </dependency>
  <!-- Other SDK dependencies. -->
</dependencies>
```

*Replace the version number with the latest version of the SDK available at Maven Central.

AWS SDK for Java metrics are *disabled by default*. To enable it for your local development environment, include a system property that points to your AWS security credential file when starting up the JVM. For example:

```
-Dcom.amazonaws.sdk.enableDefaultMetrics=credentialFile=/path/aws.properties
```

You need to specify the path to your credential file so that the SDK can upload the gathered datapoints to CloudWatch for later analysis.

Note

If you are accessing AWS from an Amazon EC2 instance using the Amazon EC2 instance metadata service, you don't need to specify a credential file. In this case, you need only specify:

```
-Dcom.amazonaws.sdk.enableDefaultMetrics
```

All metrics captured by the AWS SDK for Java are under the namespace **AWSSDK/Java**, and are uploaded to the CloudWatch default region (*us-east-1*). To change the region, specify it by using the `cloudwatchRegion` attribute in the system property. For example, to set the CloudWatch region to *us-east-1*, use:

```
-Dcom.amazonaws.sdk.enableDefaultMetrics=credentialFile=/path/
aws.properties,cloudwatchRegion={region_api_default}
```

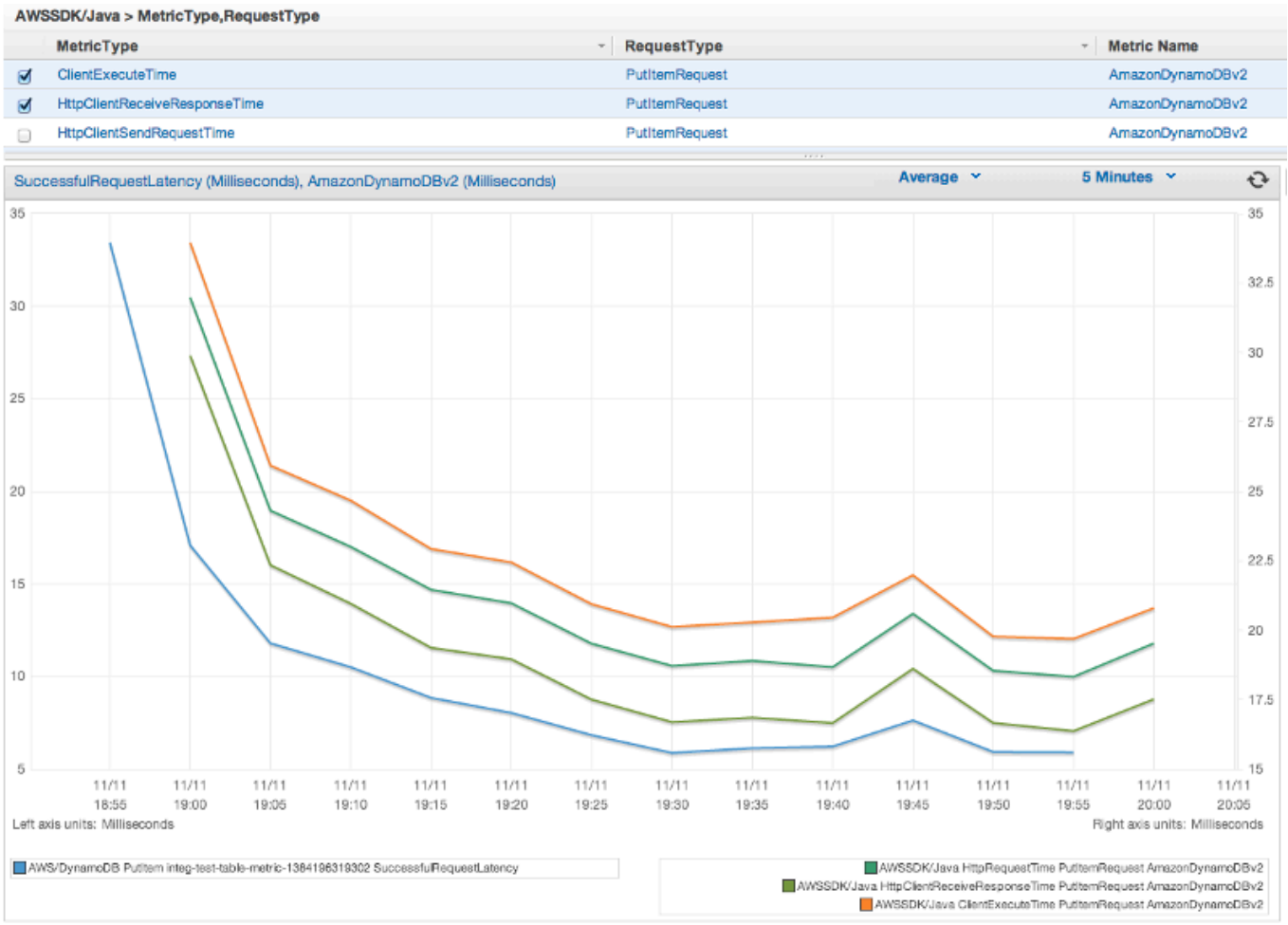
Once you enable the feature, every time there is a service request to AWS from the AWS SDK for Java, metric data points will be generated, queued for statistical summary, and uploaded asynchronously to CloudWatch about once every minute. Once metrics have been uploaded, you can visualize them using the AWS Management Console and set alarms on potential problems such as memory leakage, file descriptor leakage, and so on.

Available Metric Types

The default set of metrics is divided into three major categories:

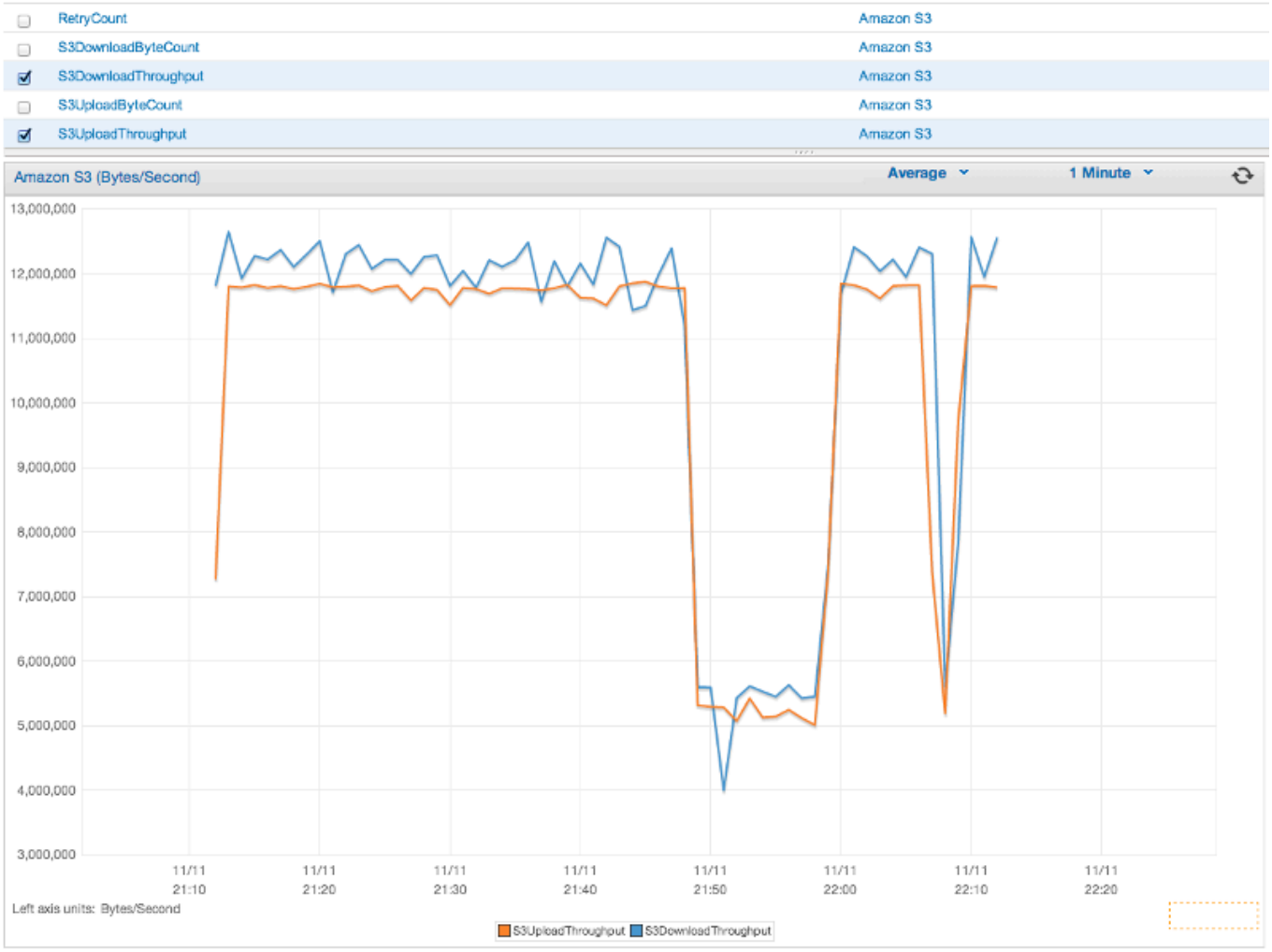
AWS Request Metrics

- Covers areas such as the latency of the HTTP request/response, number of requests, exceptions, and retries.



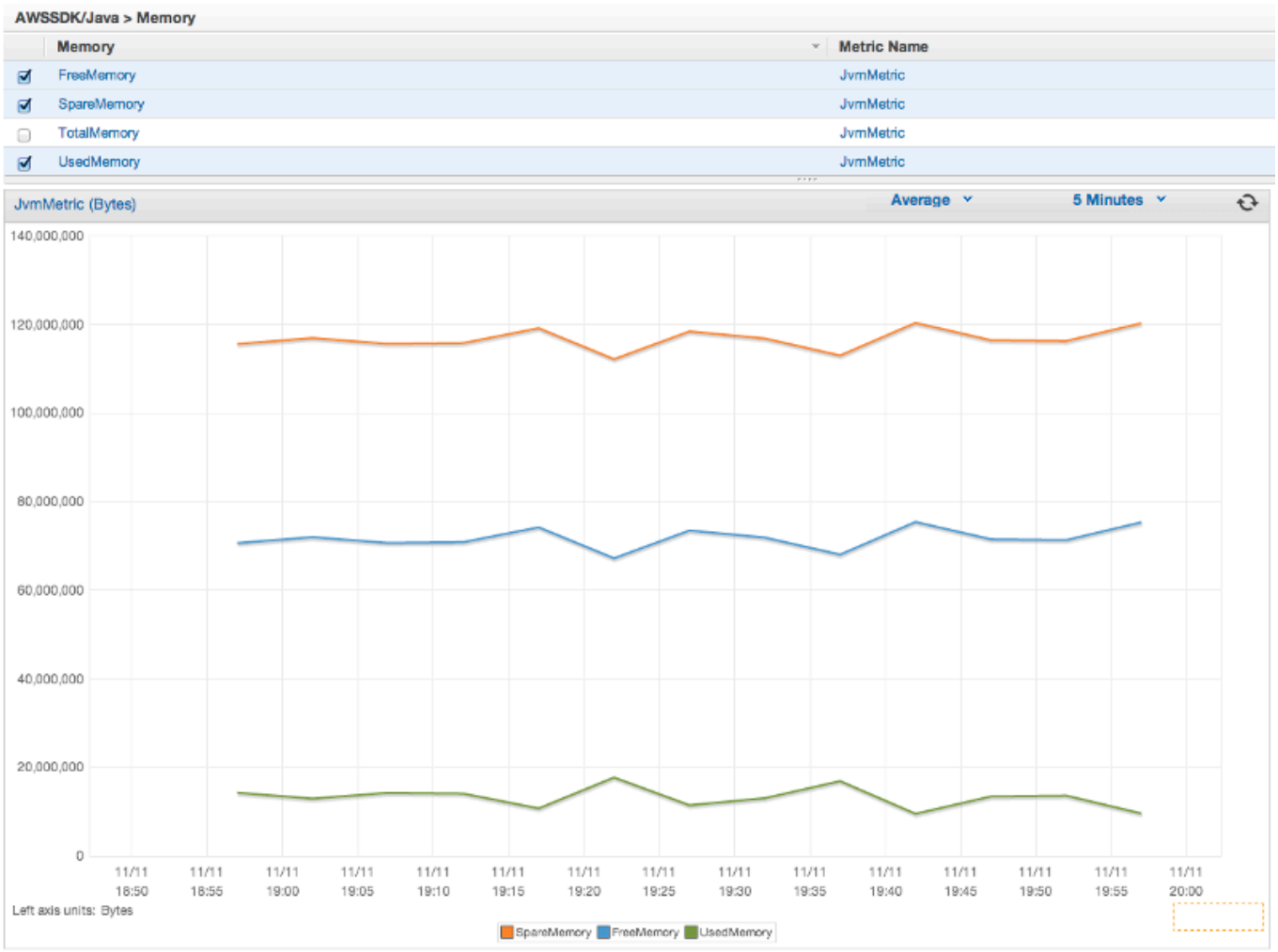
AWS service Metrics

- Include AWS service-specific data, such as the throughput and byte count for S3 uploads and downloads.



Machine Metrics

- Cover the runtime environment, including heap memory, number of threads, and open file descriptors.



If you want to exclude Machine Metrics, add `excludeMachineMetrics` to the system property:

```
-Dcom.amazonaws.sdk.enableDefaultMetrics=credentialFile=/path/
aws.properties,excludeMachineMetrics
```

More Information

- See the amazonaws/metrics package summary for a full list of the predefined core metric types.
- Learn about working with CloudWatch using the AWS SDK for Java in CloudWatch Examples Using the AWS SDK for Java.
- Learn more about performance tuning in Tuning the AWS SDK for Java to Improve Resiliency blog post.

AWS SDK for Java Code Examples

This section provides tutorials and examples of using the AWS SDK for Java v1 to program AWS services.

Find the source code for these examples and others in the AWS documentation code examples repository on GitHub.

To propose a new code example for the AWS documentation team to consider producing, create a new request. The team is looking to produce code examples that cover broader scenarios and use cases, versus simple code snippets that cover only individual API calls. For instructions, see the Contributing guidelines in the code examples respository on GitHub..

AWS SDK for Java 2.x

In 2018, AWS released the AWS SDK for Java 2.x. This guide contains instructions on using the latest Java SDK along with example code.

> **Note**
>
> See Additional Documentation and Resources for more examples and additional resources available for AWS SDK for Java developers!

CloudWatch Examples Using the AWS SDK for Java

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

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.

Topics

Getting Metrics from CloudWatch

Listing Metrics

To list CloudWatch metrics, create a ListMetricsRequest and call the AmazonCloudWatchClient'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 {https---docs-aws-amazon-com-AmazonCloudWatch-latest-monitoring-CW-Support-For-AWS-html}[Amazon CloudWatch Metrics and Dimensions Reference] in the Amazon CloudWatch User Guide.

Imports

```
import com.amazonaws.services.cloudwatch.AmazonCloudWatch;
import com.amazonaws.services.cloudwatch.AmazonCloudWatchClientBuilder;
import com.amazonaws.services.cloudwatch.model.ListMetricsRequest;
import com.amazonaws.services.cloudwatch.model.ListMetricsResult;
import com.amazonaws.services.cloudwatch.model.Metric;
```

Code

```
final AmazonCloudWatch cw =
    AmazonCloudWatchClientBuilder.defaultClient();

ListMetricsRequest request = new ListMetricsRequest()
        .withMetricName(name)
        .withNamespace(namespace);

boolean done = false;

while(!done) {
    ListMetricsResult response = cw.listMetrics(request);

    for(Metric metric : response.getMetrics()) {
        System.out.printf(
            "Retrieved metric %s", metric.getMetricName());
    }

    request.setNextToken(response.getNextToken());

    if(response.getNextToken() == null) {
        done = true;
    }
}
```

The metrics are returned in a ListMetricsResult by calling its `getMetrics` method. The results may be *paged*. To retrieve the next batch of results, call `setNextToken` on the original request object with the return value of the `ListMetricsResult` object's `getNextToken` method, and pass the modified request object back to another call to `listMetrics`.

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 AmazonCloudWatchClient's `putMetricData` method with a PutMetricDataRequest. The `PutMetricDataRequest` must include the custom namespace to use for the data, and information about the data point itself in a MetricDatum object.

> **Note**
>
> You cannot specify a namespace that begins with " AWS ". Namespaces that begin with " AWS " are reserved for use by Amazon Web Services products.

Imports

```
import com.amazonaws.services.cloudwatch.AmazonCloudWatch;
import com.amazonaws.services.cloudwatch.AmazonCloudWatchClientBuilder;
import com.amazonaws.services.cloudwatch.model.Dimension;
import com.amazonaws.services.cloudwatch.model.MetricDatum;
import com.amazonaws.services.cloudwatch.model.PutMetricDataRequest;
import com.amazonaws.services.cloudwatch.model.PutMetricDataResult;
import com.amazonaws.services.cloudwatch.model.StandardUnit;
```

Code

```
final AmazonCloudWatch cw =
    AmazonCloudWatchClientBuilder.defaultClient();

Dimension dimension = new Dimension()
    .withName("UNIQUE_PAGES")
    .withValue("URLS");

MetricDatum datum = new MetricDatum()
    .withMetricName("PAGES_VISITED")
    .withUnit(StandardUnit.None)
    .withValue(data_point)
    .withDimensions(dimension);

PutMetricDataRequest request = new PutMetricDataRequest()
    .withNamespace("SITE/TRAFFIC")
    .withMetricData(datum);
```

```
PutMetricDataResult response = cw.putMetricData(request);
```

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 AmazonCloudWatchClient's `putMetricAlarm` method with a PutMetricAlarmRequest filled with the alarm conditions.

Imports

```
import com.amazonaws.services.cloudwatch.AmazonCloudWatch;
import com.amazonaws.services.cloudwatch.AmazonCloudWatchClientBuilder;
import com.amazonaws.services.cloudwatch.model.ComparisonOperator;
import com.amazonaws.services.cloudwatch.model.Dimension;
import com.amazonaws.services.cloudwatch.model.PutMetricAlarmRequest;
import com.amazonaws.services.cloudwatch.model.PutMetricAlarmResult;
import com.amazonaws.services.cloudwatch.model.StandardUnit;
import com.amazonaws.services.cloudwatch.model.Statistic;
```

Code

```
final AmazonCloudWatch cw =
    AmazonCloudWatchClientBuilder.defaultClient();

Dimension dimension = new Dimension()
    .withName("InstanceId")
    .withValue(instanceId);

PutMetricAlarmRequest request = new PutMetricAlarmRequest()
    .withAlarmName(alarmName)
    .withComparisonOperator(
        ComparisonOperator.GreaterThanThreshold)
    .withEvaluationPeriods(1)
```

```
    .withMetricName("CPUUtilization")
    .withNamespace("{AWS}/EC2")
    .withPeriod(60)
    .withStatistic(Statistic.Average)
    .withThreshold(70.0)
    .withActionsEnabled(false)
    .withAlarmDescription(
        "Alarm when server CPU utilization exceeds 70%")
    .withUnit(StandardUnit.Seconds)
    .withDimensions(dimension);

PutMetricAlarmResult response = cw.putMetricAlarm(request);
```

List Alarms

To list the CloudWatch alarms that you have created, call the AmazonCloudWatchClient's `describeAlarms` method with a DescribeAlarmsRequest that you can use to set options for the result.

Imports

```
import com.amazonaws.services.cloudwatch.AmazonCloudWatch;
import com.amazonaws.services.cloudwatch.AmazonCloudWatchClientBuilder;
import com.amazonaws.services.cloudwatch.model.DescribeAlarmsRequest;
import com.amazonaws.services.cloudwatch.model.DescribeAlarmsResult;
import com.amazonaws.services.cloudwatch.model.MetricAlarm;
```

Code

```
final AmazonCloudWatch cw =
    AmazonCloudWatchClientBuilder.defaultClient();

boolean done = false;
DescribeAlarmsRequest request = new DescribeAlarmsRequest();

while(!done) {

    DescribeAlarmsResult response = cw.describeAlarms(request);

    for(MetricAlarm alarm : response.getMetricAlarms()) {
        System.out.printf("Retrieved alarm %s", alarm.getAlarmName());
    }
```

```
    request.setNextToken(response.getNextToken());

    if(response.getNextToken() == null) {
        done = true;
    }
}
```

The list of alarms can be obtained by calling `getMetricAlarms` on the DescribeAlarmsResult that is returned by `describeAlarms`.

The results may be *paged*. To retrieve the next batch of results, call `setNextToken` on the original request object with the return value of the `DescribeAlarmsResult` object's `getNextToken` method, and pass the modified request object back to another call to `describeAlarms`.

> **Note**
>
> You can also retrieve alarms for a specific metric by using the AmazonCloudWatchClient's `describeAlarmsForMetric` method. Its use is similar to `describeAlarms`.

Delete Alarms

To delete CloudWatch alarms, call the AmazonCloudWatchClient's `deleteAlarms` method with a DeleteAlarmsRequest containing one or more names of alarms that you want to delete.

Imports

```
import com.amazonaws.services.cloudwatch.AmazonCloudWatch;
import com.amazonaws.services.cloudwatch.AmazonCloudWatchClientBuilder;
import com.amazonaws.services.cloudwatch.model.DeleteAlarmsRequest;
import com.amazonaws.services.cloudwatch.model.DeleteAlarmsResult;
```

Code

```
final AmazonCloudWatch cw =
    AmazonCloudWatchClientBuilder.defaultClient();

DeleteAlarmsRequest request = new DeleteAlarmsRequest()
    .withAlarmNames(alarm_name);
```

```
DeleteAlarmsResult response = cw.deleteAlarms(request);
```

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 `setAlarmActions` method when creating an alarm.

Enable Alarm Actions

To enable alarm actions for a CloudWatch alarm, call the AmazonCloudWatchClient's `enableAlarmActions` with a EnableAlarmActionsRequest containing one or more names of alarms whose actions you want to enable.

Imports

```
import com.amazonaws.services.cloudwatch.AmazonCloudWatch;
import com.amazonaws.services.cloudwatch.AmazonCloudWatchClientBuilder;
import com.amazonaws.services.cloudwatch.model.EnableAlarmActionsRequest;
import com.amazonaws.services.cloudwatch.model.EnableAlarmActionsResult;
```

Code

```
final AmazonCloudWatch cw =
    AmazonCloudWatchClientBuilder.defaultClient();

EnableAlarmActionsRequest request = new EnableAlarmActionsRequest()
```

```
    .withAlarmNames(alarm);

EnableAlarmActionsResult response = cw.enableAlarmActions(request);
```

Disable Alarm Actions

To disable alarm actions for a CloudWatch alarm, call the AmazonCloudWatchClient's `disableAlarmActions` with a DisableAlarmActionsRequest containing one or more names of alarms whose actions you want to disable.

Imports

```
import com.amazonaws.services.cloudwatch.AmazonCloudWatch;
import com.amazonaws.services.cloudwatch.AmazonCloudWatchClientBuilder;
import com.amazonaws.services.cloudwatch.model.DisableAlarmActionsRequest;
import com.amazonaws.services.cloudwatch.model.DisableAlarmActionsResult;
```

Code

```
final AmazonCloudWatch cw =
    AmazonCloudWatchClientBuilder.defaultClient();

DisableAlarmActionsRequest request = new DisableAlarmActionsRequest()
    .withAlarmNames(alarmName);

DisableAlarmActionsResult response = cw.disableAlarmActions(request);
```

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 AmazonCloudWatchEventsClient's `putEvents` method with a PutEventsRequest object that contains one or more PutEventsRequestEntry objects that provide details about each event. You can specify several parameters for the entry such as the source and type of the event, resources associated with the event, and so on.

> **Note**
>
> You can specify a maximum of 10 events per call to `putEvents`.

Imports

```
import com.amazonaws.services.cloudwatchevents.AmazonCloudWatchEvents;
import com.amazonaws.services.cloudwatchevents.AmazonCloudWatchEventsClientBuilder;
import com.amazonaws.services.cloudwatchevents.model.PutEventsRequest;
import com.amazonaws.services.cloudwatchevents.model.PutEventsRequestEntry;
import com.amazonaws.services.cloudwatchevents.model.PutEventsResult;
```

Code

```
final AmazonCloudWatchEvents cwe =
    AmazonCloudWatchEventsClientBuilder.defaultClient();

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

PutEventsRequestEntry request_entry = new PutEventsRequestEntry()
    .withDetail(EVENT_DETAILS)
    .withDetailType("sampleSubmitted")
    .withResources(resource_arn)
    .withSource("aws-sdk-java-cloudwatch-example");

PutEventsRequest request = new PutEventsRequest()
    .withEntries(request_entry);

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

Add Rules

To create or update a rule, call the AmazonCloudWatchEventsClient's `putRule` method with a PutRuleRequest with the name of the rule and optional parameters such as the event pattern, IAM role to associate with the rule, and a scheduling expression that describes how often the rule is run.

Imports

```
import com.amazonaws.services.cloudwatchevents.AmazonCloudWatchEvents;
import com.amazonaws.services.cloudwatchevents.AmazonCloudWatchEventsClientBuilder;
import com.amazonaws.services.cloudwatchevents.model.PutRuleRequest;
import com.amazonaws.services.cloudwatchevents.model.PutRuleResult;
import com.amazonaws.services.cloudwatchevents.model.RuleState;
```

Code

```
final AmazonCloudWatchEvents cwe =
    AmazonCloudWatchEventsClientBuilder.defaultClient();

PutRuleRequest request = new PutRuleRequest()
    .withName(rule_name)
    .withRoleArn(role_arn)
    .withScheduleExpression("rate(5 minutes)")
    .withState(RuleState.ENABLED);

PutRuleResult response = cwe.putRule(request);
```

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 AmazonCloudWatchEventsClient's `putTargets` method with a PutTargetsRequest containing the rule to update and a list of targets to add to the rule.

Imports

```
import com.amazonaws.services.cloudwatchevents.AmazonCloudWatchEvents;
import com.amazonaws.services.cloudwatchevents.AmazonCloudWatchEventsClientBuilder;
import com.amazonaws.services.cloudwatchevents.model.PutTargetsRequest;
```

```
import com.amazonaws.services.cloudwatchevents.model.PutTargetsResult;
import com.amazonaws.services.cloudwatchevents.model.Target;
```

Code

```
final AmazonCloudWatchEvents cwe =
    AmazonCloudWatchEventsClientBuilder.defaultClient();

Target target = new Target()
    .withArn(function_arn)
    .withId(target_id);

PutTargetsRequest request = new PutTargetsRequest()
    .withTargets(target)
    .withRule(rule_name);

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

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.

> **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

Use AWS account-based endpoints

DynamoDB offers AWS account-based endpoints that can improve performance by using your AWS account ID to streamline request routing.

To take advantage of this feature, you need to use version 1.12.771 or greater of version 1 of AWS SDK for Java. You can find the latest version of the SDK listed in the Maven central repository. After a supported version of SDK is active, it automatically uses the new endpoints.

If you want to opt out of the account-based routing, you have four options:

- Configure a DynamoDB service client with the `AccountIdEndpointMode` set to `DISABLED`.
- Set an environment variable.
- Set a JVM system property.
- Update the shared AWS config file setting.

The following snippet is an example of how to disable account-based routing by configuring a DynamoDB service client:

```
ClientConfiguration config = new ClientConfiguration()
    .withAccountIdEndpointMode(AccountIdEndpointMode.DISABLED);
AWSCredentialsProvider credentialsProvider = new
 EnvironmentVariableCredentialsProvider();

AmazonDynamoDB dynamodb = AmazonDynamoDBClientBuilder.standard()
    .withClientConfiguration(config)
    .withCredentials(credentialsProvider)
    .withRegion(Regions.US_WEST_2)
    .build();
```

The AWS SDKs and Tools Reference Guide provides more information on the last three configuration options.

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* 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 if your needs change.

Create a Table

Use the DynamoDB client'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. Only define key table attributes when creating your DynamoDB table.

> **Note**
>
> If a table with the name you chose already exists, an AmazonServiceException is thrown.

Imports

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.services.dynamodbv2.model.AttributeDefinition;
import com.amazonaws.services.dynamodbv2.model.CreateTableRequest;
import com.amazonaws.services.dynamodbv2.model.CreateTableResult;
import com.amazonaws.services.dynamodbv2.model.KeySchemaElement;
import com.amazonaws.services.dynamodbv2.model.KeyType;
import com.amazonaws.services.dynamodbv2.model.ProvisionedThroughput;
import com.amazonaws.services.dynamodbv2.model.ScalarAttributeType;
```

Create a Table with a Simple Primary Key

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

Code

```
CreateTableRequest request = new CreateTableRequest()
    .withAttributeDefinitions(new AttributeDefinition(
            "Name", ScalarAttributeType.S))
    .withKeySchema(new KeySchemaElement("Name", KeyType.HASH))
    .withProvisionedThroughput(new ProvisionedThroughput(
            new Long(10), new Long(10)))
    .withTableName(table_name);

final AmazonDynamoDB ddb = AmazonDynamoDBClientBuilder.defaultClient();

try {
    CreateTableResult result = ddb.createTable(request);
    System.out.println(result.getTableDescription().getTableName());
} catch (AmazonServiceException e) {
    System.err.println(e.getErrorMessage());
    System.exit(1);
}
```

See the complete example on GitHub.

Create a Table with a Composite Primary Key

Add another AttributeDefinition and KeySchemaElement to CreateTableRequest.

Code

```
CreateTableRequest request = new CreateTableRequest()
    .withAttributeDefinitions(
          new AttributeDefinition("Language", ScalarAttributeType.S),
          new AttributeDefinition("Greeting", ScalarAttributeType.S))
    .withKeySchema(
          new KeySchemaElement("Language", KeyType.HASH),
          new KeySchemaElement("Greeting", KeyType.RANGE))
    .withProvisionedThroughput(
          new ProvisionedThroughput(new Long(10), new Long(10)))
    .withTableName(table_name);
```

See the complete example on GitHub.

List Tables

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

> **Note**
>
> If the named table doesn't exist for your account and region, a ResourceNotFoundException is thrown.

Imports

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.services.dynamodbv2.model.ListTablesRequest;
import com.amazonaws.services.dynamodbv2.model.ListTablesResult;
```

Code

```
final AmazonDynamoDB ddb = AmazonDynamoDBClientBuilder.defaultClient();

ListTablesRequest request;

boolean more_tables = true;
String last_name = null;
```

```
while(more_tables) {
    try {
        if (last_name == null) {
         request = new ListTablesRequest().withLimit(10);
        }
        else {
         request = new ListTablesRequest()
           .withLimit(10)
           .withExclusiveStartTableName(last_name);
        }

        ListTablesResult table_list = ddb.listTables(request);
        List<String> table_names = table_list.getTableNames();

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

        last_name = table_list.getLastEvaluatedTableName();
        if (last_name == null) {
            more_tables = false;
        }
```

By default, up to 100 tables are returned per call—use `getLastEvaluatedTableName` on the returned ListTablesResult object to get the last table that was evaluated. You can use this value to start the listing after the last returned value of the previous listing.

See the complete example on GitHub.

Describe (Get Information about) a Table

Call the DynamoDB client's `describeTable` method.

> **Note**
>
> If the named table doesn't exist for your account and region, a ResourceNotFoundException is thrown.

Imports

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.services.dynamodbv2.model.AttributeDefinition;
import com.amazonaws.services.dynamodbv2.model.ProvisionedThroughputDescription;
import com.amazonaws.services.dynamodbv2.model.TableDescription;
```

Code

```
final AmazonDynamoDB ddb = AmazonDynamoDBClientBuilder.defaultClient();

try {
    TableDescription table_info =
       ddb.describeTable(table_name).getTable();

    if (table_info != null) {
        System.out.format("Table name  : %s\n",
              table_info.getTableName());
        System.out.format("Table ARN   : %s\n",
              table_info.getTableArn());
        System.out.format("Status      : %s\n",
              table_info.getTableStatus());
        System.out.format("Item count  : %d\n",
              table_info.getItemCount().longValue());
        System.out.format("Size (bytes): %d\n",
              table_info.getTableSizeBytes().longValue());

        ProvisionedThroughputDescription throughput_info =
           table_info.getProvisionedThroughput();
        System.out.println("Throughput");
        System.out.format("  Read Capacity : %d\n",
              throughput_info.getReadCapacityUnits().longValue());
        System.out.format("  Write Capacity: %d\n",
              throughput_info.getWriteCapacityUnits().longValue());

        List<AttributeDefinition> attributes =
           table_info.getAttributeDefinitions();
        System.out.println("Attributes");
        for (AttributeDefinition a : attributes) {
            System.out.format("  %s (%s)\n",
                  a.getAttributeName(), a.getAttributeType());
```

```
        }
    }
} catch (AmazonServiceException e) {
    System.err.println(e.getErrorMessage());
    System.exit(1);
}
```

See the complete example on GitHub.

Modify (Update) a Table

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

> **Note**
>
> If the named table doesn't exist for your account and region, a ResourceNotFoundException is thrown.

Imports

```
import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.services.dynamodbv2.model.ProvisionedThroughput;
import com.amazonaws.AmazonServiceException;
```

Code

```
ProvisionedThroughput table_throughput = new ProvisionedThroughput(
      read_capacity, write_capacity);

final AmazonDynamoDB ddb = AmazonDynamoDBClientBuilder.defaultClient();

try {
    ddb.updateTable(table_name, table_throughput);
} catch (AmazonServiceException e) {
    System.err.println(e.getErrorMessage());
    System.exit(1);
}
```

See the complete example on GitHub.

Delete a Table

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

> **Note**
>
> If the named table doesn't exist for your account and region, a ResourceNotFoundException is thrown.

Imports

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
```

Code

```
final AmazonDynamoDB ddb = AmazonDynamoDBClientBuilder.defaultClient();

try {
    ddb.deleteTable(table_name);
} catch (AmazonServiceException e) {
    System.err.println(e.getErrorMessage());
    System.exit(1);
}
```

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 AmazonDynamoDB's `getItem` method and pass it a GetItemRequest object with the table name and primary key value of the item you want. It returns a GetItemResult object.

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

Imports

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.services.dynamodbv2.model.AttributeValue;
import com.amazonaws.services.dynamodbv2.model.GetItemRequest;
import java.util.HashMap;
import java.util.Map;
```

Code

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

key_to_get.put("DATABASE_NAME", new AttributeValue(name));

GetItemRequest request = null;
if (projection_expression != null) {
    request = new GetItemRequest()
        .withKey(key_to_get)
        .withTableName(table_name)
        .withProjectionExpression(projection_expression);
} else {
    request = new GetItemRequest()
        .withKey(key_to_get)
        .withTableName(table_name);
}

final AmazonDynamoDB ddb = AmazonDynamoDBClientBuilder.defaultClient();

try {
    Map<String,AttributeValue> returned_item =
       ddb.getItem(request).getItem();
    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 (AmazonServiceException 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 com.amazonaws.AmazonServiceException;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.services.dynamodbv2.model.AttributeValue;
import com.amazonaws.services.dynamodbv2.model.ResourceNotFoundException;
import java.util.ArrayList;
```

Code

```
HashMap<String,AttributeValue> item_values =
    new HashMap<String,AttributeValue>();

item_values.put("Name", new AttributeValue(name));
```

```
for (String[] field : extra_fields) {
    item_values.put(field[0], new AttributeValue(field[1]));
}

final AmazonDynamoDB ddb = AmazonDynamoDBClientBuilder.defaultClient();

try {
    ddb.putItem(table_name, item_values);
} 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 (AmazonServiceException e) {
    System.err.println(e.getMessage());
    System.exit(1);
```

See the complete example on GitHub.

Update an Existing Item in a Table

You can update an attribute for an item that already exists in a table by using the AmazonDynamoDB'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 com.amazonaws.AmazonServiceException;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.services.dynamodbv2.model.AttributeAction;
import com.amazonaws.services.dynamodbv2.model.AttributeValue;
import com.amazonaws.services.dynamodbv2.model.AttributeValueUpdate;
import com.amazonaws.services.dynamodbv2.model.ResourceNotFoundException;
import java.util.ArrayList;
```

Code

```
HashMap<String,AttributeValue> item_key =
   new HashMap<String,AttributeValue>();

item_key.put("Name", new AttributeValue(name));

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

for (String[] field : extra_fields) {
    updated_values.put(field[0], new AttributeValueUpdate(
                new AttributeValue(field[1]), AttributeAction.PUT));
}

final AmazonDynamoDB ddb = AmazonDynamoDBClientBuilder.defaultClient();

try {
    ddb.updateItem(table_name, item_key, updated_values);
} catch (ResourceNotFoundException e) {
    System.err.println(e.getMessage());
    System.exit(1);
} catch (AmazonServiceException e) {
    System.err.println(e.getMessage());
    System.exit(1);
```

See the complete example on GitHub.

Use the DynamoDBMapper class

The AWS SDK for Java provides a DynamoDBMapper class, allowing you to map your client-side classes to Amazon DynamoDB tables. To use the DynamoDBMapper class, you define the relationship between items in a DynamoDB table and their corresponding object instances in your code by using annotations (as shown in the following code example). The DynamoDBMapper class enables you to access your tables; perform various create, read, update, and delete (CRUD) operations; and execute queries.

> **Note**
>
> The DynamoDBMapper class does not allow you to create, update, or delete tables.

Imports

```
import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBAttribute;
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBHashKey;
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBMapper;
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBTable;
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBRangeKey;
import com.amazonaws.services.dynamodbv2.model.AmazonDynamoDBException;
```

Code

The following Java code example shows you how to add content to the *Music* table by using the DynamoDBMapper class. After the content is added to the table, notice that an item is loaded by using the *Partition* and *Sort* keys. Then the *Awards* item is updated. For information on creating the *Music* table, see Create a Table in the Amazon DynamoDB Developer Guide.

```
        AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
        MusicItems items = new MusicItems();

        try{
            // Add new content to the Music table
            items.setArtist(artist);
            items.setSongTitle(songTitle);
            items.setAlbumTitle(albumTitle);
            items.setAwards(Integer.parseInt(awards)); //convert to an int

            // Save the item
            DynamoDBMapper mapper = new DynamoDBMapper(client);
            mapper.save(items);

            // Load an item based on the Partition Key and Sort Key
            // Both values need to be passed to the mapper.load method
            String artistName = artist;
            String songQueryTitle = songTitle;

            // Retrieve the item
            MusicItems itemRetrieved = mapper.load(MusicItems.class, artistName,
songQueryTitle);
            System.out.println("Item retrieved:");
            System.out.println(itemRetrieved);
```

```
            // Modify the Award value
            itemRetrieved.setAwards(2);
            mapper.save(itemRetrieved);
            System.out.println("Item updated:");
            System.out.println(itemRetrieved);

            System.out.print("Done");
        } catch (AmazonDynamoDBException e) {
            e.getStackTrace();
        }
    }

    @DynamoDBTable(tableName="Music")
    public static class MusicItems {

        //Set up Data Members that correspond to columns in the Music table
        private String artist;
        private String songTitle;
        private String albumTitle;
        private int awards;

        @DynamoDBHashKey(attributeName="Artist")
        public String getArtist() {
            return this.artist;
        }

        public void setArtist(String artist) {
            this.artist = artist;
        }

        @DynamoDBRangeKey(attributeName="SongTitle")
        public String getSongTitle() {
            return this.songTitle;
        }

        public void setSongTitle(String title) {
            this.songTitle = title;
        }

        @DynamoDBAttribute(attributeName="AlbumTitle")
        public String getAlbumTitle() {
            return this.albumTitle;
        }
```

```
        public void setAlbumTitle(String title) {
            this.albumTitle = title;
        }

        @DynamoDBAttribute(attributeName="Awards")
        public int getAwards() {
            return this.awards;
        }

        public void setAwards(int awards) {
            this.awards = awards;
        }
    }
```

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.

Topics

- Tutorial: Starting an EC2 Instance
- Using IAM Roles to Grant Access to AWS Resources on Amazon EC2
- Tutorial: Amazon EC2 Spot Instances
- Tutorial: Advanced Amazon EC2 Spot Request Management
- Managing Amazon EC2 Instances
- Using Elastic IP Addresses in Amazon EC2
- Use regions and availability zones
- Working with Amazon EC2 Key Pairs
- Working with Security Groups in Amazon EC2

Tutorial: Starting an EC2 Instance

This tutorial demonstrates how to use the AWS SDK for Java to start an EC2 instance.

Topics

Prerequisites

Before you begin, be sure that you have created an AWS account and that you have set up your AWS credentials. For more information, see Getting Started.

Create an Amazon EC2 Security Group

EC2-Classic is retiring

> **Warning**
>
> We are retiring EC2-Classic on August 15, 2022. We recommend that you migrate from EC2-Classic to a VPC. For more information, see the blog post EC2-Classic-Classic Networking is Retiring – Here's How to Prepare.

Create a *security group*, which acts as a virtual firewall that controls the network traffic for one or more EC2 instances. By default, Amazon EC2 associates your instances with a security group that allows no inbound traffic. You can create a security group that allows your EC2 instances to accept certain traffic. For example, if you need to connect to a Linux instance, you must configure the security group to allow SSH traffic. You can create a security group using the Amazon EC2 console or the AWS SDK for Java.

You create a security group for use in either EC2-Classic or EC2-VPC. For more information about EC2-Classic and EC2-VPC, see Supported Platforms in the Amazon EC2 User Guide for Linux Instances.

For more information about creating a security group using the Amazon EC2 console, see Amazon EC2 Security Groups in the Amazon EC2 User Guide for Linux Instances.

1. Create and initialize a CreateSecurityGroupRequest instance. Use the withGroupName method to set the security group name, and the withDescription method to set the security group description, as follows:

```
CreateSecurityGroupRequest csgr = new CreateSecurityGroupRequest();
csgr.withGroupName("JavaSecurityGroup").withDescription("My security group");
```

   The security group name must be unique within the AWS region in which you initialize your Amazon EC2 client. You must use US-ASCII characters for the security group name and description.

2. Pass the request object as a parameter to the createSecurityGroup method. The method returns a CreateSecurityGroupResult object, as follows:

```
CreateSecurityGroupResult createSecurityGroupResult =
    amazonEC2Client.createSecurityGroup(csgr);
```

   If you attempt to create a security group with the same name as an existing security group, `createSecurityGroup` throws an exception.

By default, a new security group does not allow any inbound traffic to your Amazon EC2 instance. To allow inbound traffic, you must explicitly authorize security group ingress. You can authorize ingress for individual IP addresses, for a range of IP addresses, for a specific protocol, and for TCP/UDP ports.

1. Create and initialize an IpPermission instance. Use the withIpv4Ranges method to set the range of IP addresses to authorize ingress for, and use the withIpProtocol method to set the IP protocol. Use the withFromPort and withToPort methods to specify range of ports to authorize ingress for, as follows:

```
IpPermission ipPermission =
    new IpPermission();

IpRange ipRange1 = new IpRange().withCidrIp("111.111.111.111/32");
IpRange ipRange2 = new IpRange().withCidrIp("150.150.150.150/32");
```

```
ipPermission.withIpv4Ranges(Arrays.asList(new IpRange[] {ipRange1, ipRange2}))
            .withIpProtocol("tcp")
            .withFromPort(22)
            .withToPort(22);
```

All the conditions that you specify in the `IpPermission` object must be met in order for ingress to be allowed.

Specify the IP address using CIDR notation. If you specify the protocol as TCP/UDP, you must provide a source port and a destination port. You can authorize ports only if you specify TCP or UDP.

2. Create and initialize an AuthorizeSecurityGroupIngressRequest instance. Use the `withGroupName` method to specify the security group name, and pass the `IpPermission` object you initialized earlier to the withIpPermissions method, as follows:

```
AuthorizeSecurityGroupIngressRequest authorizeSecurityGroupIngressRequest =
    new AuthorizeSecurityGroupIngressRequest();

authorizeSecurityGroupIngressRequest.withGroupName("JavaSecurityGroup")
                                    .withIpPermissions(ipPermission);
```

3. Pass the request object into the authorizeSecurityGroupIngress method, as follows:

```
amazonEC2Client.authorizeSecurityGroupIngress(authorizeSecurityGroupIngressRequest);
```

If you call `authorizeSecurityGroupIngress` with IP addresses for which ingress is already authorized, the method throws an exception. Create and initialize a new `IpPermission` object to authorize ingress for different IPs, ports, and protocols before calling `AuthorizeSecurityGroupIngress`.

Whenever you call the authorizeSecurityGroupIngress or authorizeSecurityGroupEgress methods, a rule is added to your security group.

Create a Key Pair

You must specify a key pair when you launch an EC2 instance and then specify the private key of the key pair when you connect to the instance. You can create a key pair or use an existing key pair that you've used when launching other instances. For more information, see Amazon EC2 Key Pairs in the Amazon EC2 User Guide for Linux Instances.

1. Create and initialize a CreateKeyPairRequest instance. Use the withKeyName method to set the key pair name, as follows:

```
CreateKeyPairRequest createKeyPairRequest = new CreateKeyPairRequest();

createKeyPairRequest.withKeyName(keyName);
```

> **Important**
>
> Key pair names must be unique. If you attempt to create a key pair with the same key name as an existing key pair, you'll get an exception.

2. Pass the request object to the createKeyPair method. The method returns a CreateKeyPairResult instance, as follows:

```
CreateKeyPairResult createKeyPairResult =
  amazonEC2Client.createKeyPair(createKeyPairRequest);
```

3. Call the result object's getKeyPair method to obtain a KeyPair object. Call the `KeyPair` object's getKeyMaterial method to obtain the unencrypted PEM-encoded private key, as follows:

```
KeyPair keyPair = new KeyPair();

keyPair = createKeyPairResult.getKeyPair();

String privateKey = keyPair.getKeyMaterial();
```

Run an Amazon EC2 Instance

Use the following procedure to launch one or more identically configured EC2 instances from the same Amazon Machine Image (AMI). After you create your EC2 instances, you can check their status. After your EC2 instances are running, you can connect to them.

1. Create and initialize a RunInstancesRequest instance. Make sure that the AMI, key pair, and security group that you specify exist in the region that you specified when you created the client object.

```
RunInstancesRequest runInstancesRequest =
   new RunInstancesRequest();
```

```
runInstancesRequest.withImageId("ami-a9d09ed1")
                   .withInstanceType(InstanceType.T1Micro)
                   .withMinCount(1)
                   .withMaxCount(1)
                   .withKeyName("my-key-pair")
                   .withSecurityGroups("my-security-group");
```

withImageId

- The ID of the AMI. To learn how to find public AMIs provided by Amazon or create your own, see Amazon Machine Image (AMI).

withInstanceType

- An instance type that is compatible with the specified AMI. For more information, see Instance Types in the Amazon EC2 User Guide for Linux Instances.

withMinCount

- The minimum number of EC2 instances to launch. If this is more instances than Amazon EC2 can launch in the target Availability Zone, Amazon EC2 launches no instances.

withMaxCount

- The maximum number of EC2 instances to launch. If this is more instances than Amazon EC2 can launch in the target Availability Zone, Amazon EC2 launches the largest possible number of instances above `MinCount`. You can launch between 1 and the maximum number of instances you're allowed for the instance type. For more information, see How many instances can I run in Amazon EC2 in the Amazon EC2 General FAQ.

withKeyName

- The name of the EC2 key pair. If you launch an instance without specifying a key pair, you can't connect to it. For more information, see Create a Key Pair.

withSecurityGroups

- One or more security groups. For more information, see Create an Amazon EC2 Security Group.

2. Launch the instances by passing the request object to the runInstances method. The method returns a RunInstancesResult object, as follows:

```
RunInstancesResult result = amazonEC2Client.runInstances(
                            runInstancesRequest);
```

After your instance is running, you can connect to it using your key pair. For more information, see Connect to Your Linux Instance. in the Amazon EC2 User Guide for Linux Instances.

Using IAM Roles to Grant Access to AWS Resources on Amazon EC2

All requests to Amazon Web Services (AWS) 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 Java SDK 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.

The default provider chain and EC2 instance profiles

If your application creates an AWS client using the default constructor, then the client will search 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. Credentials delivered through the Amazon EC2 container service if the `AWS_CONTAINER_CREDENTIALS_RELATIVE_URI` environment variable is set and security manager has permission to access the variable.
5. In the *instance profile credentials*, which exist within the instance metadata associated with the IAM role for the EC2 instance.
6. Web Identity Token credentials from the environment or container.

The *instance profile credentials* step in the default provider chain is available only when running your application on an Amazon EC2 instance, but 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:

```
AmazonS3 s3 = AmazonS3ClientBuilder.standard()
              .withCredentials(new InstanceProfileCredentialsProvider(false))
```

```
                .build();
```

When using this approach, the SDK retrieves temporary AWS credentials that have the same permissions as those associated with the IAM role 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.

> **Important**
>
> The automatic credentials refresh happens *only* when you use the default client constructor, which creates its own `InstanceProfileCredentialsProvider` as part of the default provider chain, or when you pass an `InstanceProfileCredentialsProvider` instance directly to the client constructor. If you use another method to obtain or pass instance profile credentials, you are responsible for checking for and refreshing expired credentials.

If the client constructor can't find credentials using the credentials provider chain, it will throw an AmazonClientException.

Walkthrough: Using IAM roles for EC2 instances

The following 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.

1. Open the IAM console.
2. In the navigation pane, select **Roles**, then **Create New Role**.
3. Enter a name for the role, then select **Next Step**. Remember this name, since you'll need it when you launch your Amazon EC2 instance.
4. On the **Select Role Type** page, under **AWS service Roles**, select **Amazon EC2** .
5. On the **Set Permissions** page, under **Select Policy Template**, select **Amazon S3 Read Only Access**, then **Next Step**.
6. On the **Review** page, select **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 or the AWS SDK for Java.

- 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'll need to create or use an existing security group and key pair to connect to the instance.

- To launch an Amazon EC2 instance with an IAM role using the AWS SDK for Java, see Run an Amazon EC2 Instance.

Create your Application

Let's build the sample application to run on the EC2 instance. First, create a directory that you can use to hold your tutorial files (for example, `GetS3ObjectApp`).

Next, copy the AWS SDK for Java libraries into your newly-created directory. If you downloaded the AWS SDK for Java to your `~/Downloads` directory, you can copy them using the following commands:

```
cp -r ~/Downloads/aws-java-sdk-{1.7.5}/lib .
cp -r ~/Downloads/aws-java-sdk-{1.7.5}/third-party .
```

Open a new file, call it `GetS3Object.java`, and add the following code:

```
import java.io.*;

import com.amazonaws.auth.*;
import com.amazonaws.services.s3.*;
import com.amazonaws.services.s3.model.*;
import com.amazonaws.AmazonClientException;
import com.amazonaws.AmazonServiceException;
```

```
public class GetS3Object {
  private static final String bucketName = "text-content";
  private static final String key = "text-object.txt";

  public static void main(String[] args) throws IOException
  {
    AmazonS3 s3Client = AmazonS3ClientBuilder.defaultClient();

    try {
      System.out.println("Downloading an object");
      S3Object s3object = s3Client.getObject(
          new GetObjectRequest(bucketName, key));
      displayTextInputStream(s3object.getObjectContent());
    }
    catch(AmazonServiceException ase) {
      System.err.println("Exception was thrown by the service");
    }
    catch(AmazonClientException ace) {
      System.err.println("Exception was thrown by the client");
    }
  }

  private static void displayTextInputStream(InputStream input) throws IOException
  {
    // Read one text line at a time and display.
    BufferedReader reader = new BufferedReader(new InputStreamReader(input));
    while(true)
    {
      String line = reader.readLine();
      if(line == null) break;
      System.out.println( "    " + line );
    }
    System.out.println();
  }
}
```

Open a new file, call it `build.xml`, and add the following lines:

```
<project name="Get {S3} Object" default="run" basedir=".">
  <path id="aws.java.sdk.classpath">
    <fileset dir="./lib" includes="**/*.jar"/>
    <fileset dir="./third-party" includes="**/*.jar"/>
```

```
    <pathelement location="lib"/>
    <pathelement location="."/>
  </path>

  <target name="build">
  <javac debug="true"
    includeantruntime="false"
    srcdir="."
    destdir="."
    classpathref="aws.java.sdk.classpath"/>
  </target>

  <target name="run" depends="build">
    <java classname="GetS3Object" classpathref="aws.java.sdk.classpath" fork="true"/>
  </target>
</project>
```

Build and run the modified program. Note that there are no credentials are stored in the program. Therefore, unless you have your AWS credentials specified already, the code will throw `AmazonServiceException`. For example:

```
$ ant
Buildfile: /path/to/my/GetS3ObjectApp/build.xml

build:
  [javac] Compiling 1 source file to /path/to/my/GetS3ObjectApp

run:
   [java] Downloading an object
   [java] AmazonServiceException

BUILD SUCCESSFUL
```

Transfer the Compiled Program to Your EC2 Instance

Transfer the program to your Amazon EC2 instance using secure copy ( ), along with the AWS SDK for Java libraries. The sequence of commands looks something like the following.

```
scp -p -i {my-key-pair}.pem GetS3Object.class ec2-user@{public_dns}:GetS3Object.class
scp -p -i {my-key-pair}.pem build.xml ec2-user@{public_dns}:build.xml
scp -r -p -i {my-key-pair}.pem lib ec2-user@{public_dns}:lib
scp -r -p -i {my-key-pair}.pem third-party ec2-user@{public_dns}:third-party
```

> **Note**
>
> Depending on the Linux distribution that you used, the *user name* might be "ec2-user", "root", or "ubuntu". To get the public DNS name of your instance, open the EC2 console and look for the **Public DNS** value in the **Description** tab (for example, `ec2-198-51-100-1.compute-1.amazonaws.com`).

In the preceding commands:

- `GetS3Object.class` is your compiled program
- `build.xml` is the ant file used to build and run your program
- the `lib` and `third-party` directories are the corresponding library folders from the AWS SDK for Java.
- The `-r` switch indicates that `scp` should do a recursive copy of all of the contents of the `library` and `third-party` directories in the AWS SDK for Java distribution.
- The `-p` switch indicates that `scp` should preserve the permissions of the source files when it copies them to the destination.

> **Note**
>
> The `-p` switch works only on Linux, macOS, or Unix. If you are copying files from Windows, you may need to fix the file permissions on your instance using the following command:

```
chmod -R u+rwx GetS3Object.class build.xml lib third-party
```

Run the Sample Program on the EC2 Instance

To run the program, connect to your Amazon EC2 instance. For more information, see Connect to Your Linux Instance in the Amazon EC2 User Guide for Linux Instances.

If **ant** is not available on your instance, install it using the following command:

```
sudo yum install ant
```

Then, run the program using `ant` as follows:

```
ant run
```

The program will write the contents of your Amazon S3 object to your command window.

Tutorial: Amazon EC2 Spot Instances

Overview

Spot Instances enable you to bid on unused Amazon Elastic Compute Cloud (Amazon EC2) capacity t up to 90% versus the On-Demand Instance price and run the acquired instances for as long as your bid exceeds the current *Spot Price*. Amazon EC2 changes the Spot Price periodically based on supply and demand, and customers whose bids meet or exceed it gain access to the available Spot Instances. Like On-Demand Instances and Reserved Instances, Spot Instances provide you another option for obtaining more compute capacity.

Spot Instances can significantly lower your Amazon EC2 costs for batch processing, scientific research, image processing, video encoding, data and web crawling, financial analysis, and testing. Additionally, Spot Instances give you access to large amounts of additional capacity in situations where the need for that capacity is not urgent.

To use Spot Instances, place a Spot Instance request specifying the maximum price you are willing to pay per instance hour; this is your bid. If your bid exceeds the current Spot Price, your request is fulfilled and your instances will run until either you choose to terminate them or the Spot Price increases above your bid (whichever is sooner).

It's important to note:

- You will often pay less per hour than your bid. Amazon EC2 adjusts the Spot Price periodically as requests come in and available supply changes. Everyone pays the same Spot Price for that period regardless of whether their bid was higher. Therefore, you might pay less than your bid, but you will never pay more than your bid.
- If you're running Spot Instances and your bid no longer meets or exceeds the current Spot Price, your instances will be terminated. This means that you will want to make sure that your workloads and applications are flexible enough to take advantage of this opportunistic capacity.

Spot Instances perform exactly like other Amazon EC2 instances while running, and like other Amazon EC2 instances, Spot Instances can be terminated when you no longer need them. If you

terminate your instance, you pay for any partial hour used (as you would for On-Demand or Reserved Instances). However, if the Spot Price goes above your bid and your instance is terminated by Amazon EC2, you will not be charged for any partial hour of usage.

This tutorial shows how to use AWS SDK for Java to do the following.

- Submit a Spot Request
- Determine when the Spot Request becomes fulfilled
- Cancel the Spot Request
- Terminate associated instances

Prerequisites

To use this tutorial you must have the AWS SDK for Java installed, as well as having met its basic installation prerequisites. See Set up the AWS SDK for Java for more information.

Step 1: Setting Up Your Credentials

To begin using this code sample, you need to set up AWS credentials. See Set up AWS Credentials and Region for Development for instructions on how to do that.

> **Note**
>
> We recommend that you use the credentials of an IAM user to provide these values. For more information, see Sign Up for AWS and Create an IAM User.

Now that you have configured your settings, you can get started using the code in the example.

Step 2: Setting Up a Security Group

A *security group* acts as a firewall that controls the traffic allowed in and out of a group of instances. By default, an instance is started without any security group, which means that all incoming IP traffic, on any TCP port will be denied. So, before submitting our Spot Request, we will set up a security group that allows the necessary network traffic. For the purposes of this tutorial, we will create a new security group called "GettingStarted" that allows Secure Shell (SSH) traffic from the IP address where you are running your application from. To set up a new security

group, you need to include or run the following code sample that sets up the security group programmatically.

After we create an `AmazonEC2` client object, we create a `CreateSecurityGroupRequest` object with the name, "GettingStarted" and a description for the security group. Then we call the `ec2.createSecurityGroup` API to create the group.

To enable access to the group, we create an `ipPermission` object with the IP address range set to the CIDR representation of the subnet for the local computer; the "/10" suffix on the IP address indicates the subnet for the specified IP address. We also configure the `ipPermission` object with the TCP protocol and port 22 (SSH). The final step is to call `ec2.authorizeSecurityGroupIngress` with the name of our security group and the `ipPermission` object.

```
// Create the AmazonEC2 client so we can call various APIs.
AmazonEC2 ec2 = AmazonEC2ClientBuilder.defaultClient();

// Create a new security group.
try {
    CreateSecurityGroupRequest securityGroupRequest = new
 CreateSecurityGroupRequest("GettingStartedGroup", "Getting Started Security Group");
    ec2.createSecurityGroup(securityGroupRequest);
} catch (AmazonServiceException ase) {
    // Likely this means that the group is already created, so ignore.
    System.out.println(ase.getMessage());
}

String ipAddr = "0.0.0.0/0";

// Get the IP of the current host, so that we can limit the Security
// Group by default to the ip range associated with your subnet.
try {
    InetAddress addr = InetAddress.getLocalHost();

    // Get IP Address
    ipAddr = addr.getHostAddress()+"/10";
} catch (UnknownHostException e) {
}

// Create a range that you would like to populate.
ArrayList<String> ipRanges = new ArrayList<String>();
ipRanges.add(ipAddr);
```

```
// Open up port 22 for TCP traffic to the associated IP
// from above (e.g. ssh traffic).
ArrayList<IpPermission> ipPermissions = new ArrayList<IpPermission> ();
IpPermission ipPermission = new IpPermission();
ipPermission.setIpProtocol("tcp");
ipPermission.setFromPort(new Integer(22));
ipPermission.setToPort(new Integer(22));
ipPermission.setIpRanges(ipRanges);
ipPermissions.add(ipPermission);

try {
    // Authorize the ports to the used.
    AuthorizeSecurityGroupIngressRequest ingressRequest =
        new AuthorizeSecurityGroupIngressRequest("GettingStartedGroup",ipPermissions);
    ec2.authorizeSecurityGroupIngress(ingressRequest);
} catch (AmazonServiceException ase) {
    // Ignore because this likely means the zone has
    // already been authorized.
    System.out.println(ase.getMessage());
}
```

Note you only need to run this application once to create a new security group.

You can also create the security group using the AWS Toolkit for Eclipse. See Managing Security Groups from AWS Cost Explorer for more information.

Step 3: Submitting Your Spot Request

To submit a Spot request, you first need to determine the instance type, Amazon Machine Image (AMI), and maximum bid price you want to use. You must also include the security group we configured previously, so that you can log into the instance if desired.

There are several instance types to choose from; go to Amazon EC2 Instance Types for a complete list. For this tutorial, we will use t1.micro, the cheapest instance type available. Next, we will determine the type of AMI we would like to use. We'll use ami-a9d09ed1, the most up-to-date Amazon Linux AMI available when we wrote this tutorial. The latest AMI may change over time, but you can always determine the latest version AMI by following these steps:

1. Open the Amazon EC2 console.
2. Choose the **Launch Instance** button.

3. The first window displays the AMIs available. The AMI ID is listed next to each AMI title. Alternatively, you can use the `DescribeImages` API, but leveraging that command is outside the scope of this tutorial.

There are many ways to approach bidding for Spot Instances; to get a broad overview of the various approaches you should view the Bidding for Spot Instances video. However, to get started, we'll describe three common strategies: bid to ensure cost is less than on-demand pricing; bid based on the value of the resulting computation; bid so as to acquire computing capacity as quickly as possible.

- *Reduce Cost below On-Demand* You have a batch processing job that will take a number of hours or days to run. However, you are flexible with respect to when it starts and when it completes. You want to see if you can complete it for less cost than with On-Demand Instances. You examine the Spot Price history for instance types using either the AWS Management Console or the Amazon EC2 API. For more information, go to Viewing Spot Price History. After you've analyzed the price history for your desired instance type in a given Availability Zone, you have two alternative approaches for your bid:
  - You could bid at the upper end of the range of Spot Prices (which are still below the On-Demand price), anticipating that your one-time Spot request would most likely be fulfilled and run for enough consecutive compute time to complete the job.
  - Or, you could specify the amount you are willing to pay for Spot Instances as a % of the On-Demand Instance price , and plan to combine many instances launched over time through a persistent request. If the specified price is exceeded, then the Spot Instance will terminate. (We will explain how to automate this task later in this tutorial.)
- *Pay No More than the Value of the Result* You have a data processing job to run. You understand the value of the job's results well enough to know how much they are worth in terms of computing costs. After you've analyzed the Spot Price history for your instance type, you choose a bid price at which the cost of the computing time is no more than the value of the job's results. You create a persistent bid and allow it to run intermittently as the Spot Price fluctuates at or below your bid.
- *Acquire Computing Capacity Quickly* You have an unanticipated, short-term need for additional capacity that is not available through On-Demand Instances. After you've analyzed the Spot Price history for your instance type, you bid above the highest historical price to provide a high likelihood that your request will be fulfilled quickly and continue computing until it completes.

After you choose your bid price, you are ready to request a Spot Instance. For the purposes of this tutorial, we will bid the On-Demand price ($0.03) to maximize the chances that the bid will be fulfilled. You can determine the types of available instances and the On-Demand prices for instances by going to Amazon EC2 Pricing page. While a Spot Instancee is running, you pay the Spot price that's in effect for the time period your instances are running. Spot Instance prices are set by Amazon EC2 and adjust gradually based on long-term trends in supply and demand for Spot Instance capacity. You can also specify the amount you are willing to pay for a Spot Instance as a % of the On-Demand Instance price.To request a Spot Instance, you simply need to build your request with the parameters you chose earlier. We start by creating a `RequestSpotInstanceRequest` object. The request object requires the number of instances you want to start and the bid price. Additionally, you need to set the `LaunchSpecification` for the request, which includes the instance type, AMI ID, and security group you want to use. Once the request is populated, you call the `requestSpotInstances` method on the `AmazonEC2Client` object. The following example shows how to request a Spot Instance.

```
// Create the AmazonEC2 client so we can call various APIs.
AmazonEC2 ec2 = AmazonEC2ClientBuilder.defaultClient();

// Initializes a Spot Instance Request
RequestSpotInstancesRequest requestRequest = new RequestSpotInstancesRequest();

// Request 1 x t1.micro instance with a bid price of $0.03.
requestRequest.setSpotPrice("0.03");
requestRequest.setInstanceCount(Integer.valueOf(1));

// Setup the specifications of the launch. This includes the
// instance type (e.g. t1.micro) and the latest Amazon Linux
// AMI id available. Note, you should always use the latest
// Amazon Linux AMI id or another of your choosing.
LaunchSpecification launchSpecification = new LaunchSpecification();
launchSpecification.setImageId("ami-a9d09ed1");
launchSpecification.setInstanceType(InstanceType.T1Micro);

// Add the security group to the request.
ArrayList<String> securityGroups = new ArrayList<String>();
securityGroups.add("GettingStartedGroup");
launchSpecification.setSecurityGroups(securityGroups);

// Add the launch specifications to the request.
requestRequest.setLaunchSpecification(launchSpecification);
```

```
// Call the RequestSpotInstance API.
RequestSpotInstancesResult requestResult = ec2.requestSpotInstances(requestRequest);
```

Running this code will launch a new Spot Instance Request. There are other options you can use to configure your Spot Requests. To learn more, please visit Tutorial: Advanced Amazon EC2 Spot Request Management or the RequestSpotInstances class in the AWS SDK for Java API Reference.

> **Note**
>
> You will be charged for any Spot Instances that are actually launched, so make sure that you cancel any requests and terminate any instances you launch to reduce any associated fees.

Step 4: Determining the State of Your Spot Request

Next, we want to create code to wait until the Spot request reaches the "active" state before proceeding to the last step. To determine the state of our Spot request, we poll the describeSpotInstanceRequests method for the state of the Spot request ID we want to monitor.

The request ID created in Step 2 is embedded in the response to our `requestSpotInstances` request. The following example code shows how to gather request IDs from the `requestSpotInstances` response and use them to populate an `ArrayList`.

```
// Call the RequestSpotInstance API.
RequestSpotInstancesResult requestResult = ec2.requestSpotInstances(requestRequest);
List<SpotInstanceRequest> requestResponses = requestResult.getSpotInstanceRequests();

// Setup an arraylist to collect all of the request ids we want to
// watch hit the running state.
ArrayList<String> spotInstanceRequestIds = new ArrayList<String>();

// Add all of the request ids to the hashset, so we can determine when they hit the
// active state.
for (SpotInstanceRequest requestResponse : requestResponses) {
    System.out.println("Created Spot Request:
 "+requestResponse.getSpotInstanceRequestId());
    spotInstanceRequestIds.add(requestResponse.getSpotInstanceRequestId());
}
```

To monitor your request ID, call the `describeSpotInstanceRequests` method to determine the state of the request. Then loop until the request is not in the "open" state. Note that we monitor for a state of not "open", rather a state of, say, "active", because the request can go straight to "closed" if there is a problem with your request arguments. The following code example provides the details of how to accomplish this task.

```
// Create a variable that will track whether there are any
// requests still in the open state.
boolean anyOpen;

do {
    // Create the describeRequest object with all of the request ids
    // to monitor (e.g. that we started).
    DescribeSpotInstanceRequestsRequest describeRequest = new
 DescribeSpotInstanceRequestsRequest();
    describeRequest.setSpotInstanceRequestIds(spotInstanceRequestIds);

    // Initialize the anyOpen variable to false - which assumes there
    // are no requests open unless we find one that is still open.
    anyOpen=false;

    try {
        // Retrieve all of the requests we want to monitor.
        DescribeSpotInstanceRequestsResult describeResult =
 ec2.describeSpotInstanceRequests(describeRequest);
        List<SpotInstanceRequest> describeResponses =
 describeResult.getSpotInstanceRequests();

        // Look through each request and determine if they are all in
        // the active state.
        for (SpotInstanceRequest describeResponse : describeResponses) {
            // If the state is open, it hasn't changed since we attempted
            // to request it. There is the potential for it to transition
            // almost immediately to closed or cancelled so we compare
            // against open instead of active.
        if (describeResponse.getState().equals("open")) {
            anyOpen = true;
            break;
        }
    }
} catch (AmazonServiceException e) {
      // If we have an exception, ensure we don't break out of
      // the loop. This prevents the scenario where there was
```

```
      // blip on the wire.
      anyOpen = true;
    }

    try {
        // Sleep for 60 seconds.
        Thread.sleep(60*1000);
    } catch (Exception e) {
        // Do nothing because it woke up early.
    }
} while (anyOpen);
```

After running this code, your Spot Instance Request will have completed or will have failed with an error that will be output to the screen. In either case, we can proceed to the next step to clean up any active requests and terminate any running instances.

Step 5: Cleaning Up Your Spot Requests and Instances

Lastly, we need to clean up our requests and instances. It is important to both cancel any outstanding requests *and* terminate any instances. Just canceling your requests will not terminate your instances, which means that you will continue to pay for them. If you terminate your instances, your Spot requests may be canceled, but there are some scenarios such as if you use persistent bids where terminating your instances is not sufficient to stop your request from being re-fulfilled. Therefore, it is a best practice to both cancel any active bids and terminate any running instances.

The following code demonstrates how to cancel your requests.

```
try {
    // Cancel requests.
    CancelSpotInstanceRequestsRequest cancelRequest =
       new CancelSpotInstanceRequestsRequest(spotInstanceRequestIds);
    ec2.cancelSpotInstanceRequests(cancelRequest);
} catch (AmazonServiceException e) {
    // Write out any exceptions that may have occurred.
    System.out.println("Error cancelling instances");
    System.out.println("Caught Exception: " + e.getMessage());
    System.out.println("Reponse Status Code: " + e.getStatusCode());
    System.out.println("Error Code: " + e.getErrorCode());
    System.out.println("Request ID: " + e.getRequestId());
}
```

To terminate any outstanding instances, you will need the instance ID associated with the request that started them. The following code example takes our original code for monitoring the instances and adds an `ArrayList` in which we store the instance ID associated with the `describeInstance` response.

```
// Create a variable that will track whether there are any requests
// still in the open state.
boolean anyOpen;
// Initialize variables.
ArrayList<String> instanceIds = new ArrayList<String>();

do {
   // Create the describeRequest with all of the request ids to
   // monitor (e.g. that we started).
   DescribeSpotInstanceRequestsRequest describeRequest = new
 DescribeSpotInstanceRequestsRequest();
   describeRequest.setSpotInstanceRequestIds(spotInstanceRequestIds);

   // Initialize the anyOpen variable to false, which assumes there
   // are no requests open unless we find one that is still open.
   anyOpen = false;

   try {
         // Retrieve all of the requests we want to monitor.
         DescribeSpotInstanceRequestsResult describeResult =
            ec2.describeSpotInstanceRequests(describeRequest);

         List<SpotInstanceRequest> describeResponses =
            describeResult.getSpotInstanceRequests();

         // Look through each request and determine if they are all
         // in the active state.
         for (SpotInstanceRequest describeResponse : describeResponses) {
           // If the state is open, it hasn't changed since we
           // attempted to request it. There is the potential for
           // it to transition almost immediately to closed or
           // cancelled so we compare against open instead of active.
           if (describeResponse.getState().equals("open")) {
              anyOpen = true; break;
           }
           // Add the instance id to the list we will
           // eventually terminate.
           instanceIds.add(describeResponse.getInstanceId());
```

```
         }
   } catch (AmazonServiceException e) {
      // If we have an exception, ensure we don't break out
      // of the loop. This prevents the scenario where there
      // was blip on the wire.
      anyOpen = true;
   }

    try {
        // Sleep for 60 seconds.
        Thread.sleep(60*1000);
    } catch (Exception e) {
        // Do nothing because it woke up early.
    }
} while (anyOpen);
```

Using the instance IDs, stored in the `ArrayList`, terminate any running instances using the following code snippet.

```
try {
    // Terminate instances.
    TerminateInstancesRequest terminateRequest = new
 TerminateInstancesRequest(instanceIds);
    ec2.terminateInstances(terminateRequest);
} catch (AmazonServiceException e) {
    // Write out any exceptions that may have occurred.
    System.out.println("Error terminating instances");
    System.out.println("Caught Exception: " + e.getMessage());
    System.out.println("Reponse Status Code: " + e.getStatusCode());
    System.out.println("Error Code: " + e.getErrorCode());
    System.out.println("Request ID: " + e.getRequestId());
}
```

Bringing It All Together

To bring this all together, we provide a more object-oriented approach that combines the preceding steps we showed: initializing the EC2 Client, submitting the Spot Request, determining when the Spot Requests are no longer in the open state, and cleaning up any lingering Spot request and associated instances. We create a class called `Requests` that performs these actions.

We also create a `GettingStarted`App class, which has a main method where we perform the high level function calls. Specifically, we initialize the `Requests` object described previously. We submit

the Spot Instance request. Then we wait for the Spot request to reach the "Active" state. Finally, we clean up the requests and instances.

The complete source code for this example can be viewed or downloaded at GitHub.

Congratulations! You have just completed the getting started tutorial for developing Spot Instance software with the AWS SDK for Java.

Next Steps

Proceed with Tutorial: Advanced Amazon EC2 Spot Request Management.

Tutorial: Advanced Amazon EC2 Spot Request Management

Amazon EC2 Spot Instances allow you to bid on unused Amazon EC2 capacity and run those instances for as long as your bid exceeds the current *spot price*. Amazon EC2 changes the spot price periodically based on supply and demand. For more information about Spot Instances, see Spot Instances in the Amazon EC2 User Guide for Linux Instances.

Prerequisites

To use this tutorial you must have the AWS SDK for Java installed, as well as having met its basic installation prerequisites. See Set up the AWS SDK for Java for more information.

Setting up your credentials

To begin using this code sample, you need to set up AWS credentials. See Set up AWS Credentials and Region for Development for instructions on how to do that.

> **Note**
>
> We recommend that you use the credentials of an IAM user to provide these values. For more information, see Sign Up for AWS and Create an IAM User.

Now that you have configured your settings, you can get started using the code in the example.

Setting up a security group

A security group acts as a firewall that controls the traffic allowed in and out of a group of instances. By default, an instance is started without any security group, which means that all

incoming IP traffic, on any TCP port will be denied. So, before submitting our Spot Request, we will set up a security group that allows the necessary network traffic. For the purposes of this tutorial, we will create a new security group called "GettingStarted" that allows Secure Shell (SSH) traffic from the IP address where you are running your application from. To set up a new security group, you need to include or run the following code sample that sets up the security group programmatically.

After we create an `AmazonEC2` client object, we create a `CreateSecurityGroupRequest` object with the name, "GettingStarted" and a description for the security group. Then we call the `ec2.createSecurityGroup` API to create the group.

To enable access to the group, we create an `ipPermission` object with the IP address range set to the CIDR representation of the subnet for the local computer; the "/10" suffix on the IP address indicates the subnet for the specified IP address. We also configure the `ipPermission` object with the TCP protocol and port 22 (SSH). The final step is to call `ec2 .authorizeSecurityGroupIngress` with the name of our security group and the `ipPermission` object.

(The following code is the same as what we used in the first tutorial.)

```
// Create the AmazonEC2Client object so we can call various APIs.
AmazonEC2 ec2 = AmazonEC2ClientBuilder.standard()
                    .withCredentials(credentials)
                    .build();

// Create a new security group.
try {
    CreateSecurityGroupRequest securityGroupRequest =
        new CreateSecurityGroupRequest("GettingStartedGroup",
        "Getting Started Security Group");
    ec2.createSecurityGroup(securityGroupRequest);
} catch (AmazonServiceException ase) {
    // Likely this means that the group is already created, so ignore.
    System.out.println(ase.getMessage());
}

String ipAddr = "0.0.0.0/0";

// Get the IP of the current host, so that we can limit the Security Group
// by default to the ip range associated with your subnet.
try {
    // Get IP Address
```

```
    InetAddress addr = InetAddress.getLocalHost();
    ipAddr = addr.getHostAddress()+"/10";
}
catch (UnknownHostException e) {
    // Fail here...
}

// Create a range that you would like to populate.
ArrayList<String> ipRanges = new ArrayList<String>();
ipRanges.add(ipAddr);

// Open up port 22 for TCP traffic to the associated IP from
// above (e.g. ssh traffic).
ArrayList<IpPermission> ipPermissions = new ArrayList<IpPermission> ();
IpPermission ipPermission = new IpPermission();
ipPermission.setIpProtocol("tcp");
ipPermission.setFromPort(new Integer(22));
ipPermission.setToPort(new Integer(22));
ipPermission.setIpRanges(ipRanges);
ipPermissions.add(ipPermission);

try {
    // Authorize the ports to the used.
    AuthorizeSecurityGroupIngressRequest ingressRequest =
        new AuthorizeSecurityGroupIngressRequest(
            "GettingStartedGroup",ipPermissions);
    ec2.authorizeSecurityGroupIngress(ingressRequest);
}
catch (AmazonServiceException ase) {
    // Ignore because this likely means the zone has already
    // been authorized.
    System.out.println(ase.getMessage());
}
```

You can view this entire code sample in the `advanced.CreateSecurityGroupApp.java` code sample. Note you only need to run this application once to create a new security group.

> **Note**
>
> You can also create the security group using the AWS Toolkit for Eclipse. See Managing Security Groups from AWS Cost Explorer in the AWS Toolkit for Eclipse User Guide for more information.

Detailed Spot Instance request creation options

As we explained in Tutorial: Amazon EC2 Spot Instances, you need to build your request with an instance type, an Amazon Machine Image (AMI), and maximum bid price.

Let's start by creating a `RequestSpotInstanceRequest` object. The request object requires the number of instances you want and the bid price. Additionally, we need to set the `LaunchSpecification` for the request, which includes the instance type, AMI ID, and security group you want to use. After the request is populated, we call the `requestSpotInstances` method on the `AmazonEC2Client` object. An example of how to request a Spot Instance follows.

(The following code is the same as what we used in the first tutorial.)

```
// Create the AmazonEC2 client so we can call various APIs.
AmazonEC2 ec2 = AmazonEC2ClientBuilder.defaultClient();

// Initializes a Spot Instance Request
RequestSpotInstancesRequest requestRequest = new RequestSpotInstancesRequest();

// Request 1 x t1.micro instance with a bid price of $0.03.
requestRequest.setSpotPrice("0.03");
requestRequest.setInstanceCount(Integer.valueOf(1));

// Set up the specifications of the launch. This includes the
// instance type (e.g. t1.micro) and the latest Amazon Linux
// AMI id available. Note, you should always use the latest
// Amazon Linux AMI id or another of your choosing.
LaunchSpecification launchSpecification = new LaunchSpecification();
launchSpecification.setImageId("ami-a9d09ed1");
launchSpecification.setInstanceType(InstanceType.T1Micro);

// Add the security group to the request.
ArrayList<String> securityGroups = new ArrayList<String>();
securityGroups.add("GettingStartedGroup");
launchSpecification.setSecurityGroups(securityGroups);

// Add the launch specification.
requestRequest.setLaunchSpecification(launchSpecification);

// Call the RequestSpotInstance API.
RequestSpotInstancesResult requestResult =
    ec2.requestSpotInstances(requestRequest);
```

Persistent vs. one-time requests

When building a Spot request, you can specify several optional parameters. The first is whether your request is one-time only or persistent. By default, it is a one-time request. A one-time request can be fulfilled only once, and after the requested instances are terminated, the request will be closed. A persistent request is considered for fulfillment whenever there is no Spot Instance running for the same request. To specify the type of request, you simply need to set the Type on the Spot request. This can be done with the following code.

```
// Retrieves the credentials from an AWSCredentials.properties file.
AWSCredentials credentials = null;
try {
    credentials = new PropertiesCredentials(
        GettingStartedApp.class.getResourceAsStream("AwsCredentials.properties"));
}
catch (IOException e1) {
    System.out.println(
        "Credentials were not properly entered into AwsCredentials.properties.");
    System.out.println(e1.getMessage());
    System.exit(-1);
}

// Create the AmazonEC2 client so we can call various APIs.
AmazonEC2 ec2 = AmazonEC2ClientBuilder.defaultClient();

// Initializes a Spot Instance Request
RequestSpotInstancesRequest requestRequest =
    new RequestSpotInstancesRequest();

// Request 1 x t1.micro instance with a bid price of $0.03.
requestRequest.setSpotPrice("0.03");
requestRequest.setInstanceCount(Integer.valueOf(1));

// Set the type of the bid to persistent.
requestRequest.setType("persistent");

// Set up the specifications of the launch. This includes the
// instance type (e.g. t1.micro) and the latest Amazon Linux
// AMI id available. Note, you should always use the latest
// Amazon Linux AMI id or another of your choosing.
LaunchSpecification launchSpecification = new LaunchSpecification();
launchSpecification.setImageId("ami-a9d09ed1");
launchSpecification.setInstanceType(InstanceType.T1Micro);
```

```
// Add the security group to the request.
ArrayList<String> securityGroups = new ArrayList<String>();
securityGroups.add("GettingStartedGroup");
launchSpecification.setSecurityGroups(securityGroups);

// Add the launch specification.
requestRequest.setLaunchSpecification(launchSpecification);

// Call the RequestSpotInstance API.
RequestSpotInstancesResult requestResult =
    ec2.requestSpotInstances(requestRequest);
```

Limiting the duration of a request

You can also optionally specify the length of time that your request will remain valid. You can specify both a starting and ending time for this period. By default, a Spot request will be considered for fulfillment from the moment it is created until it is either fulfilled or canceled by you. However you can constrain the validity period if you need to. An example of how to specify this period is shown in the following code.

```
// Create the AmazonEC2 client so we can call various APIs.
AmazonEC2 ec2 = AmazonEC2ClientBuilder.defaultClient();

// Initializes a Spot Instance Request
RequestSpotInstancesRequest requestRequest = new RequestSpotInstancesRequest();

// Request 1 x t1.micro instance with a bid price of $0.03.
requestRequest.setSpotPrice("0.03");
requestRequest.setInstanceCount(Integer.valueOf(1));

// Set the valid start time to be two minutes from now.
Calendar cal = Calendar.getInstance();
cal.add(Calendar.MINUTE, 2);
requestRequest.setValidFrom(cal.getTime());

// Set the valid end time to be two minutes and two hours from now.
cal.add(Calendar.HOUR, 2);
requestRequest.setValidUntil(cal.getTime());

// Set up the specifications of the launch. This includes
// the instance type (e.g. t1.micro)
```

```
// and the latest Amazon Linux AMI id available.
// Note, you should always use the latest Amazon
// Linux AMI id or another of your choosing.
LaunchSpecification launchSpecification = new LaunchSpecification();
launchSpecification.setImageId("ami-a9d09ed1");
launchSpecification.setInstanceType("t1.micro");

// Add the security group to the request.
ArrayList<String> securityGroups = new ArrayList<String>();
securityGroups.add("GettingStartedGroup");
launchSpecification.setSecurityGroups(securityGroups);

// Add the launch specification.
requestRequest.setLaunchSpecification(launchSpecification);

// Call the RequestSpotInstance API.
RequestSpotInstancesResult requestResult = ec2.requestSpotInstances(requestRequest);
```

Grouping your Amazon EC2 Spot Instance requests

You have the option of grouping your Spot Instance requests in several different ways. We'll look at the benefits of using launch groups, Availability Zone groups, and placement groups.

If you want to ensure your Spot Instances are all launched and terminated together, then you have the option to leverage a launch group. A launch group is a label that groups a set of bids together. All instances in a launch group are started and terminated together. Note, if instances in a launch group have already been fulfilled, there is no guarantee that new instances launched with the same launch group will also be fulfilled. An example of how to set a Launch Group is shown in the following code example.

```
// Create the AmazonEC2 client so we can call various APIs.
AmazonEC2 ec2 = AmazonEC2ClientBuilder.defaultClient();

// Initializes a Spot Instance Request
RequestSpotInstancesRequest requestRequest = new RequestSpotInstancesRequest();

// Request 5 x t1.micro instance with a bid price of $0.03.
requestRequest.setSpotPrice("0.03");
requestRequest.setInstanceCount(Integer.valueOf(5));

// Set the launch group.
requestRequest.setLaunchGroup("ADVANCED-DEMO-LAUNCH-GROUP");
```

```
// Set up the specifications of the launch. This includes
// the instance type (e.g. t1.micro) and the latest Amazon Linux
// AMI id available. Note, you should always use the latest
// Amazon Linux AMI id or another of your choosing.
LaunchSpecification launchSpecification = new LaunchSpecification();
launchSpecification.setImageId("ami-a9d09ed1");
launchSpecification.setInstanceType(InstanceType.T1Micro);

// Add the security group to the request.
ArrayList<String> securityGroups = new ArrayList<String>();
securityGroups.add("GettingStartedGroup");
launchSpecification.setSecurityGroups(securityGroups);

// Add the launch specification.
requestRequest.setLaunchSpecification(launchSpecification);

// Call the RequestSpotInstance API.
RequestSpotInstancesResult requestResult =
    ec2.requestSpotInstances(requestRequest);
```

If you want to ensure that all instances within a request are launched in the same Availability Zone, and you don't care which one, you can leverage Availability Zone groups. An Availability Zone group is a label that groups a set of instances together in the same Availability Zone. All instances that share an Availability Zone group and are fulfilled at the same time will start in the same Availability Zone. An example of how to set an Availability Zone group follows.

```
// Create the AmazonEC2 client so we can call various APIs.
AmazonEC2 ec2 = AmazonEC2ClientBuilder.defaultClient();

// Initializes a Spot Instance Request
RequestSpotInstancesRequest requestRequest = new RequestSpotInstancesRequest();

// Request 5 x t1.micro instance with a bid price of $0.03.
requestRequest.setSpotPrice("0.03");
requestRequest.setInstanceCount(Integer.valueOf(5));

// Set the availability zone group.
requestRequest.setAvailabilityZoneGroup("ADVANCED-DEMO-AZ-GROUP");

// Set up the specifications of the launch.  This includes the instance
// type (e.g.  t1.micro) and the latest Amazon Linux AMI id available.
// Note, you should always use the latest Amazon Linux AMI id or another
```

```
// of your choosing.
LaunchSpecification launchSpecification = new LaunchSpecification();
launchSpecification.setImageId("ami-a9d09ed1");
launchSpecification.setInstanceType(InstanceType.T1Micro);

// Add the security group to the request.
ArrayList<String> securityGroups = new ArrayList<String>();
securityGroups.add("GettingStartedGroup");
launchSpecification.setSecurityGroups(securityGroups);

// Add the launch specification.
requestRequest.setLaunchSpecification(launchSpecification);

// Call the RequestSpotInstance API.
RequestSpotInstancesResult requestResult =
    ec2.requestSpotInstances(requestRequest);
```

You can specify an Availability Zone that you want for your Spot Instances. The following code example shows you how to set an Availability Zone.

```
// Create the AmazonEC2 client so we can call various APIs.
AmazonEC2 ec2 = AmazonEC2ClientBuilder.defaultClient();

// Initializes a Spot Instance Request
RequestSpotInstancesRequest requestRequest = new RequestSpotInstancesRequest();

// Request 1 x t1.micro instance with a bid price of $0.03.
requestRequest.setSpotPrice("0.03");
requestRequest.setInstanceCount(Integer.valueOf(1));

// Set up the specifications of the launch. This includes the instance
// type (e.g. t1.micro) and the latest Amazon Linux AMI id available.
// Note, you should always use the latest Amazon Linux AMI id or another
// of your choosing.
LaunchSpecification launchSpecification = new LaunchSpecification();
launchSpecification.setImageId("ami-a9d09ed1");
launchSpecification.setInstanceType(InstanceType.T1Micro);

// Add the security group to the request.
ArrayList<String> securityGroups = new ArrayList<String>();
securityGroups.add("GettingStartedGroup");
launchSpecification.setSecurityGroups(securityGroups);
```

```
// Set up the availability zone to use. Note we could retrieve the
// availability zones using the ec2.describeAvailabilityZones() API. For
// this demo we will just use us-east-1a.
SpotPlacement placement = new SpotPlacement("us-east-1b");
launchSpecification.setPlacement(placement);

// Add the launch specification.
requestRequest.setLaunchSpecification(launchSpecification);

// Call the RequestSpotInstance API.
RequestSpotInstancesResult requestResult =
    ec2.requestSpotInstances(requestRequest);
```

Lastly, you can specify a *placement group* if you are using High Performance Computing (HPC) Spot Instances, such as cluster compute instances or cluster GPU instances. Placement groups provide you with lower latency and high-bandwidth connectivity between the instances. An example of how to set a placement group follows.

```
// Create the AmazonEC2 client so we can call various APIs.
AmazonEC2 ec2 = AmazonEC2ClientBuilder.defaultClient();

// Initializes a Spot Instance Request
RequestSpotInstancesRequest requestRequest = new RequestSpotInstancesRequest();

// Request 1 x t1.micro instance with a bid price of $0.03.
requestRequest.setSpotPrice("0.03");
requestRequest.setInstanceCount(Integer.valueOf(1));

// Set up the specifications of the launch. This includes the instance
// type (e.g. t1.micro) and the latest Amazon Linux AMI id available.
// Note, you should always use the latest Amazon Linux AMI id or another
// of your choosing.

LaunchSpecification launchSpecification = new LaunchSpecification();
launchSpecification.setImageId("ami-a9d09ed1");
launchSpecification.setInstanceType(InstanceType.T1Micro);

// Add the security group to the request.
ArrayList<String> securityGroups = new ArrayList<String>();
securityGroups.add("GettingStartedGroup");
launchSpecification.setSecurityGroups(securityGroups);

// Set up the placement group to use with whatever name you desire.
```

```
// For this demo we will just use "ADVANCED-DEMO-PLACEMENT-GROUP".
SpotPlacement placement = new SpotPlacement();
placement.setGroupName("ADVANCED-DEMO-PLACEMENT-GROUP");
launchSpecification.setPlacement(placement);

// Add the launch specification.
requestRequest.setLaunchSpecification(launchSpecification);

// Call the RequestSpotInstance API.
RequestSpotInstancesResult requestResult =
    ec2.requestSpotInstances(requestRequest);
```

All of the parameters shown in this section are optional. It is also important to realize that most of these parameters—with the exception of whether your bid is one-time or persistent—can reduce the likelihood of bid fulfillment. So, it is important to leverage these options only if you need them. All of the preceding code examples are combined into one long code sample, which can be found in the `com.amazonaws.codesamples.advanced.InlineGettingStartedCodeSampleApp.java` class.

How to persist a root partition after interruption or termination

One of the easiest ways to manage interruption of your Spot Instances is to ensure that your data is checkpointed to an Amazon Elastic Block Store (Amazon Amazon EBS) volume on a regular cadence. By checkpointing periodically, if there is an interruption you will lose only the data created since the last checkpoint (assuming no other non-idempotent actions are performed in between). To make this process easier, you can configure your Spot Request to ensure that your root partition will not be deleted on interruption or termination. We've inserted new code in the following example that shows how to enable this scenario.

In the added code, we create a `BlockDeviceMapping` object and set its associated Amazon Elastic Block Store (Amazon EBS) to an Amazon EBS object that we've configured to `not` be deleted if the Spot Instance is terminated. We then add this `BlockDeviceMapping` to the ArrayList of mappings that we include in the launch specification.

```
// Retrieves the credentials from an AWSCredentials.properties file.
AWSCredentials credentials = null;
try {
    credentials = new PropertiesCredentials(
        GettingStartedApp.class.getResourceAsStream("AwsCredentials.properties"));
}
```

```
catch (IOException e1) {
    System.out.println(
        "Credentials were not properly entered into AwsCredentials.properties.");
    System.out.println(e1.getMessage());
    System.exit(-1);
}

// Create the AmazonEC2 client so we can call various APIs.
AmazonEC2 ec2 = AmazonEC2ClientBuilder.defaultClient();

// Initializes a Spot Instance Request
RequestSpotInstancesRequest requestRequest = new RequestSpotInstancesRequest();

// Request 1 x t1.micro instance with a bid price of $0.03.
requestRequest.setSpotPrice("0.03");
requestRequest.setInstanceCount(Integer.valueOf(1));

// Set up the specifications of the launch. This includes the instance
// type (e.g. t1.micro) and the latest Amazon Linux AMI id available.
// Note, you should always use the latest Amazon Linux AMI id or another
// of your choosing.
LaunchSpecification launchSpecification = new LaunchSpecification();
launchSpecification.setImageId("ami-a9d09ed1");
launchSpecification.setInstanceType(InstanceType.T1Micro);

// Add the security group to the request.
ArrayList<String> securityGroups = new ArrayList<String>();
securityGroups.add("GettingStartedGroup");
launchSpecification.setSecurityGroups(securityGroups);

// Create the block device mapping to describe the root partition.
BlockDeviceMapping blockDeviceMapping = new BlockDeviceMapping();
blockDeviceMapping.setDeviceName("/dev/sda1");

// Set the delete on termination flag to false.
EbsBlockDevice ebs = new EbsBlockDevice();
ebs.setDeleteOnTermination(Boolean.FALSE);
blockDeviceMapping.setEbs(ebs);

// Add the block device mapping to the block list.
ArrayList<BlockDeviceMapping> blockList = new ArrayList<BlockDeviceMapping>();
blockList.add(blockDeviceMapping);

// Set the block device mapping configuration in the launch specifications.
```

```
launchSpecification.setBlockDeviceMappings(blockList);

// Add the launch specification.
requestRequest.setLaunchSpecification(launchSpecification);

// Call the RequestSpotInstance API.
RequestSpotInstancesResult requestResult =
    ec2.requestSpotInstances(requestRequest);
```

Assuming you wanted to re-attach this volume to your instance on startup, you can also use the block device mapping settings. Alternatively, if you attached a non-root partition, you can specify the Amazon Amazon EBS volumes you want to attach to your Spot Instance after it resumes. You do this simply by specifying a snapshot ID in your `EbsBlockDevice` and alternative device name in your `BlockDeviceMapping` objects. By leveraging block device mappings, it can be easier to bootstrap your instance.

Using the root partition to checkpoint your critical data is a great way to manage the potential for interruption of your instances. For more methods on managing the potential of interruption, please visit the Managing Interruption video.

How to tag your spot requests and instances

Adding tags to Amazon EC2 resources can simplify the administration of your cloud infrastructure. A form of metadata, tags can be used to create user-friendly names, enhance searchability, and improve coordination between multiple users. You can also use tags to automate scripts and portions of your processes. To read more about tagging Amazon EC2 resources, go to Using Tags in the Amazon EC2 User Guide for Linux Instances.

Tagging requests

To add tags to your spot requests, you need to tag them *after* they have been requested. The return value from `requestSpotInstances()` provides you with a RequestSpotInstancesResult object that you can use to get the spot request IDs for tagging:

```
// Call the RequestSpotInstance API.
RequestSpotInstancesResult requestResult = ec2.requestSpotInstances(requestRequest);
List<SpotInstanceRequest> requestResponses = requestResult.getSpotInstanceRequests();

// A list of request IDs to tag
ArrayList<String> spotInstanceRequestIds = new ArrayList<String>();
```

```
// Add the request ids to the hashset, so we can determine when they hit the
// active state.
for (SpotInstanceRequest requestResponse : requestResponses) {
    System.out.println("Created Spot Request:
 "+requestResponse.getSpotInstanceRequestId());
    spotInstanceRequestIds.add(requestResponse.getSpotInstanceRequestId());
}
```

Once you have the IDs, you can tag the requests by adding their IDs to a CreateTagsRequest and calling the Amazon EC2 client's `createTags()` method:

```
// The list of tags to create
ArrayList<Tag> requestTags = new ArrayList<Tag>();
requestTags.add(new Tag("keyname1","value1"));

// Create the tag request
CreateTagsRequest createTagsRequest_requests = new CreateTagsRequest();
createTagsRequest_requests.setResources(spotInstanceRequestIds);
createTagsRequest_requests.setTags(requestTags);

// Tag the spot request
try {
    ec2.createTags(createTagsRequest_requests);
}
catch (AmazonServiceException e) {
    System.out.println("Error terminating instances");
    System.out.println("Caught Exception: " + e.getMessage());
    System.out.println("Reponse Status Code: " + e.getStatusCode());
    System.out.println("Error Code: " + e.getErrorCode());
    System.out.println("Request ID: " + e.getRequestId());
}
```

Tagging instances

Similarly to spot requests themselves, you can only tag an instance once it has been created, which will happen once the spot request has been met (it is no longer in the *open* state).

You can check the status of your requests by calling the Amazon EC2 client's `describeSpotInstanceRequests()` method with a DescribeSpotInstanceRequestsRequest object. The returned DescribeSpotInstanceRequestsResult object contains a list of SpotInstanceRequest objects that you can use to query the status of your spot requests and obtain their instance IDs once they are no longer in the *open* state.

Once the spot request is no longer open, you can retrieve its instance ID from the `SpotInstanceRequest` object by calling its `getInstanceId()` method.

```
boolean anyOpen; // tracks whether any requests are still open

// a list of instances to tag.
ArrayList<String> instanceIds = new ArrayList<String>();

do {
    DescribeSpotInstanceRequestsRequest describeRequest =
        new DescribeSpotInstanceRequestsRequest();
    describeRequest.setSpotInstanceRequestIds(spotInstanceRequestIds);

    anyOpen=false; // assume no requests are still open

    try {
        // Get the requests to monitor
        DescribeSpotInstanceRequestsResult describeResult =
            ec2.describeSpotInstanceRequests(describeRequest);

        List<SpotInstanceRequest> describeResponses =
            describeResult.getSpotInstanceRequests();

        // are any requests open?
        for (SpotInstanceRequest describeResponse : describeResponses) {
                if (describeResponse.getState().equals("open")) {
                    anyOpen = true;
                    break;
                }
                // get the corresponding instance ID of the spot request
                instanceIds.add(describeResponse.getInstanceId());
        }
    }
    catch (AmazonServiceException e) {
        // Don't break the loop due to an exception (it may be a temporary issue)
        anyOpen = true;
    }

    try {
        Thread.sleep(60*1000); // sleep 60s.
    }
    catch (Exception e) {
        // Do nothing if the thread woke up early.
```

```
    }
} while (anyOpen);
```

Now you can tag the instances that are returned:

```
// Create a list of tags to create
ArrayList<Tag> instanceTags = new ArrayList<Tag>();
instanceTags.add(new Tag("keyname1","value1"));

// Create the tag request
CreateTagsRequest createTagsRequest_instances = new CreateTagsRequest();
createTagsRequest_instances.setResources(instanceIds);
createTagsRequest_instances.setTags(instanceTags);

// Tag the instance
try {
    ec2.createTags(createTagsRequest_instances);
}
catch (AmazonServiceException e) {
    // Write out any exceptions that may have occurred.
    System.out.println("Error terminating instances");
    System.out.println("Caught Exception: " + e.getMessage());
    System.out.println("Reponse Status Code: " + e.getStatusCode());
    System.out.println("Error Code: " + e.getErrorCode());
    System.out.println("Request ID: " + e.getRequestId());
}
```

Canceling spot requests and terminating instances

Canceling a spot request

To cancel a Spot Instance request, call `cancelSpotInstanceRequests` on the Amazon EC2 client with a CancelSpotInstanceRequestsRequest object.

```
try {
    CancelSpotInstanceRequestsRequest cancelRequest = new
 CancelSpotInstanceRequestsRequest(spotInstanceRequestIds);
    ec2.cancelSpotInstanceRequests(cancelRequest);
} catch (AmazonServiceException e) {
    System.out.println("Error cancelling instances");
    System.out.println("Caught Exception: " + e.getMessage());
    System.out.println("Reponse Status Code: " + e.getStatusCode());
```

```
    System.out.println("Error Code: " + e.getErrorCode());
    System.out.println("Request ID: " + e.getRequestId());
}
```

Terminating Spot Instances

You can terminate any Spot Instances that are running by passing their IDs to the Amazon EC2 client's `terminateInstances()` method.

```
try {
    TerminateInstancesRequest terminateRequest = new
 TerminateInstancesRequest(instanceIds);
    ec2.terminateInstances(terminateRequest);
} catch (AmazonServiceException e) {
    System.out.println("Error terminating instances");
    System.out.println("Caught Exception: " + e.getMessage());
    System.out.println("Reponse Status Code: " + e.getStatusCode());
    System.out.println("Error Code: " + e.getErrorCode());
    System.out.println("Request ID: " + e.getRequestId());
}
```

Bringing it all together

To bring this all together, we provide a more object-oriented approach that combines the steps we showed in this tutorial into one easy to use class. We instantiate a class called `Requests` that performs these actions. We also create a `GettingStarted`App class, which has a main method where we perform the high level function calls.

The complete source code for this example can be viewed or downloaded at GitHub.

Congratulations! You've completed the Advanced Request Features tutorial for developing Spot Instance software with the AWS SDK for Java.

Managing Amazon EC2 Instances

Creating an Instance

Create a new Amazon EC2 instance by calling the AmazonEC2Client's `runInstances` method, providing it with a RunInstancesRequest containing the Amazon Machine Image (AMI) to use and an instance type.

Imports

```
import com.amazonaws.services.ec2.AmazonEC2ClientBuilder;
import com.amazonaws.services.ec2.model.InstanceType;
import com.amazonaws.services.ec2.model.RunInstancesRequest;
import com.amazonaws.services.ec2.model.RunInstancesResult;
import com.amazonaws.services.ec2.model.Tag;
```

Code

```
RunInstancesRequest run_request = new RunInstancesRequest()
    .withImageId(ami_id)
    .withInstanceType(InstanceType.T1Micro)
    .withMaxCount(1)
    .withMinCount(1);

RunInstancesResult run_response = ec2.runInstances(run_request);

String reservation_id =
 run_response.getReservation().getInstances().get(0).getInstanceId();
```

See the complete example.

Starting an Instance

To start an Amazon EC2 instance, call the AmazonEC2Client's `startInstances` method, providing it with a StartInstancesRequest containing the ID of the instance to start.

Imports

```
import com.amazonaws.services.ec2.AmazonEC2;
import com.amazonaws.services.ec2.AmazonEC2ClientBuilder;
import com.amazonaws.services.ec2.model.StartInstancesRequest;
```

Code

```
final AmazonEC2 ec2 = AmazonEC2ClientBuilder.defaultClient();

StartInstancesRequest request = new StartInstancesRequest()
    .withInstanceIds(instance_id);
```

```
ec2.startInstances(request);
```

See the complete example.

Stopping an Instance

To stop an Amazon EC2 instance, call the AmazonEC2Client's `stopInstances` method, providing it with a StopInstancesRequest containing the ID of the instance to stop.

Imports

```
import com.amazonaws.services.ec2.AmazonEC2;
import com.amazonaws.services.ec2.AmazonEC2ClientBuilder;
import com.amazonaws.services.ec2.model.StopInstancesRequest;
```

Code

```
final AmazonEC2 ec2 = AmazonEC2ClientBuilder.defaultClient();

StopInstancesRequest request = new StopInstancesRequest()
    .withInstanceIds(instance_id);

ec2.stopInstances(request);
```

See the complete example.

Rebooting an Instance

To reboot an Amazon EC2 instance, call the AmazonEC2Client's `rebootInstances` method, providing it with a RebootInstancesRequest containing the ID of the instance to reboot.

Imports

```
import com.amazonaws.services.ec2.AmazonEC2;
import com.amazonaws.services.ec2.AmazonEC2ClientBuilder;
import com.amazonaws.services.ec2.model.RebootInstancesRequest;
import com.amazonaws.services.ec2.model.RebootInstancesResult;
```

Code

```
final AmazonEC2 ec2 = AmazonEC2ClientBuilder.defaultClient();
```

```
RebootInstancesRequest request = new RebootInstancesRequest()
    .withInstanceIds(instance_id);

RebootInstancesResult response = ec2.rebootInstances(request);
```

See the complete example.

Describing Instances

To list your instances, create a DescribeInstancesRequest and call the AmazonEC2Client's `describeInstances` method. It will return a DescribeInstancesResult 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 `DescribeInstancesResult` class' `getReservations' method, and then call ``getInstances` on each returned Reservation object.

Imports

```
import com.amazonaws.services.ec2.AmazonEC2;
import com.amazonaws.services.ec2.AmazonEC2ClientBuilder;
import com.amazonaws.services.ec2.model.DescribeInstancesRequest;
import com.amazonaws.services.ec2.model.DescribeInstancesResult;
import com.amazonaws.services.ec2.model.Instance;
import com.amazonaws.services.ec2.model.Reservation;
```

Code

```
final AmazonEC2 ec2 = AmazonEC2ClientBuilder.defaultClient();
boolean done = false;

DescribeInstancesRequest request = new DescribeInstancesRequest();
while(!done) {
    DescribeInstancesResult response = ec2.describeInstances(request);

    for(Reservation reservation : response.getReservations()) {
        for(Instance instance : reservation.getInstances()) {
            System.out.printf(
                "Found instance with id %s, " +
```

```
                "AMI %s, " +
                "type %s, " +
                "state %s " +
                "and monitoring state %s",
                instance.getInstanceId(),
                instance.getImageId(),
                instance.getInstanceType(),
                instance.getState().getName(),
                instance.getMonitoring().getState());
        }
    }

    request.setNextToken(response.getNextToken());

    if(response.getNextToken() == null) {
        done = true;
    }
}
```

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

See the complete example.

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 AmazonEC2Client's `monitorInstances` method.

Imports

```
import com.amazonaws.services.ec2.AmazonEC2;
import com.amazonaws.services.ec2.AmazonEC2ClientBuilder;
import com.amazonaws.services.ec2.model.MonitorInstancesRequest;
```

Code

```
final AmazonEC2 ec2 = AmazonEC2ClientBuilder.defaultClient();

MonitorInstancesRequest request = new MonitorInstancesRequest()
        .withInstanceIds(instance_id);

ec2.monitorInstances(request);
```

See the complete example.

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 AmazonEC2Client's `unmonitorInstances` method.

Imports

```
import com.amazonaws.services.ec2.AmazonEC2;
import com.amazonaws.services.ec2.AmazonEC2ClientBuilder;
import com.amazonaws.services.ec2.model.UnmonitorInstancesRequest;
```

Code

```
final AmazonEC2 ec2 = AmazonEC2ClientBuilder.defaultClient();

UnmonitorInstancesRequest request = new UnmonitorInstancesRequest()
    .withInstanceIds(instance_id);

ec2.unmonitorInstances(request);
```

See the complete example.

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
- MonitorInstances in the Amazon EC2 API Reference

- UnmonitorInstances in the Amazon EC2 API Reference

Using Elastic IP Addresses in Amazon EC2

EC2-Classic is retiring

> **Warning**
>
> We are retiring EC2-Classic on August 15, 2022. We recommend that you migrate from EC2-Classic to a VPC. For more information, see the blog post EC2-Classic-Classic Networking is Retiring – Here's How to Prepare.

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 AmazonEC2Client's `allocateAddress` method with an AllocateAddressRequest object containing the network type (classic EC2 or VPC).

The returned AllocateAddressResult 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 AmazonEC2Client's `associateAddress` method.

Imports

```
import com.amazonaws.services.ec2.AmazonEC2;
import com.amazonaws.services.ec2.AmazonEC2ClientBuilder;
import com.amazonaws.services.ec2.model.AllocateAddressRequest;
import com.amazonaws.services.ec2.model.AllocateAddressResult;
import com.amazonaws.services.ec2.model.AssociateAddressRequest;
import com.amazonaws.services.ec2.model.AssociateAddressResult;
import com.amazonaws.services.ec2.model.DomainType;
```

Code

```
final AmazonEC2 ec2 = AmazonEC2ClientBuilder.defaultClient();

AllocateAddressRequest allocate_request = new AllocateAddressRequest()
```

```
    .withDomain(DomainType.Vpc);

AllocateAddressResult allocate_response =
    ec2.allocateAddress(allocate_request);

String allocation_id = allocate_response.getAllocationId();

AssociateAddressRequest associate_request =
    new AssociateAddressRequest()
        .withInstanceId(instance_id)
        .withAllocationId(allocation_id);

AssociateAddressResult associate_response =
    ec2.associateAddress(associate_request);
```

See the complete example.

Describing Elastic IP Addresses

To list the Elastic IP addresses assigned to your account, call the AmazonEC2Client's `describeAddresses` method. It returns a DescribeAddressesResult which you can use to get a list of Address objects that represent the Elastic IP addresses on your account.

Imports

```
import com.amazonaws.services.ec2.AmazonEC2;
import com.amazonaws.services.ec2.AmazonEC2ClientBuilder;
import com.amazonaws.services.ec2.model.Address;
import com.amazonaws.services.ec2.model.DescribeAddressesResult;
```

Code

```
final AmazonEC2 ec2 = AmazonEC2ClientBuilder.defaultClient();

DescribeAddressesResult response = ec2.describeAddresses();

for(Address address : response.getAddresses()) {
    System.out.printf(
            "Found address with public IP %s, " +
            "domain %s, " +
            "allocation id %s " +
            "and NIC id %s",
```

```
            address.getPublicIp(),
            address.getDomain(),
            address.getAllocationId(),
            address.getNetworkInterfaceId());
}
```

See the complete example.

Releasing an Elastic IP Address

To release an Elastic IP address, call the AmazonEC2Client's `releaseAddress` method, passing it a ReleaseAddressRequest containing the allocation ID of the Elastic IP address you want to release.

Imports

```
import com.amazonaws.services.ec2.AmazonEC2;
import com.amazonaws.services.ec2.AmazonEC2ClientBuilder;
import com.amazonaws.services.ec2.model.ReleaseAddressRequest;
import com.amazonaws.services.ec2.model.ReleaseAddressResult;
```

Code

```
final AmazonEC2 ec2 = AmazonEC2ClientBuilder.defaultClient();

ReleaseAddressRequest request = new ReleaseAddressRequest()
    .withAllocationId(alloc_id);

ReleaseAddressResult 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 attempt to release an Elastic IP address that you already released, you'll get an *AuthFailure* error if the address is already allocated to another AWS account.

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 AmazonEC2Client'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.

More Information

- Elastic IP Addresses in the Amazon EC2 User Guide for Linux Instances
- AllocateAddress in the Amazon EC2 API Reference
- DescribeAddresses in the Amazon EC2 API Reference
- ReleaseAddress in the Amazon EC2 API Reference

Use regions and availability zones

Describe regions

To list the Regions available to your account, call the AmazonEC2Client's `describeRegions` method. It returns a DescribeRegionsResult. Call the returned object's `getRegions` method to get a list of Region objects that represent each Region.

Imports

```
import com.amazonaws.services.ec2.AmazonEC2;
import com.amazonaws.services.ec2.AmazonEC2ClientBuilder;
import com.amazonaws.services.ec2.model.DescribeRegionsResult;
import com.amazonaws.services.ec2.model.Region;
import com.amazonaws.services.ec2.model.AvailabilityZone;
import com.amazonaws.services.ec2.model.DescribeAvailabilityZonesResult;
```

Code

```
DescribeRegionsResult regions_response = ec2.describeRegions();

for(Region region : regions_response.getRegions()) {
    System.out.printf(
        "Found region %s " +
        "with endpoint %s",
        region.getRegionName(),
        region.getEndpoint());
}
```

See the complete example.

Describe availability zones

To list each Availability Zone available to your account, call the AmazonEC2Client's `describeAvailabilityZones` method. It returns a DescribeAvailabilityZonesResult. Call its `getAvailabilityZones` method to get a list of AvailabilityZone objects that represent each Availability Zone.

Imports

```
import com.amazonaws.services.ec2.AmazonEC2;
import com.amazonaws.services.ec2.AmazonEC2ClientBuilder;
import com.amazonaws.services.ec2.model.DescribeRegionsResult;
import com.amazonaws.services.ec2.model.Region;
import com.amazonaws.services.ec2.model.AvailabilityZone;
import com.amazonaws.services.ec2.model.DescribeAvailabilityZonesResult;
```

Code

```
DescribeAvailabilityZonesResult zones_response =
    ec2.describeAvailabilityZones();

for(AvailabilityZone zone : zones_response.getAvailabilityZones()) {
    System.out.printf(
        "Found availability zone %s " +
        "with status %s " +
        "in region %s",
        zone.getZoneName(),
        zone.getState(),
        zone.getRegionName());
}
```

See the complete example.

Describe accounts

To describe your account, call the AmazonEC2Client's `describeAccountAttributes` method. This method returns a DescribeAccountAttributesResult object. Invoke this objects `getAccountAttributes` method to get a list of AccountAttribute objects. You can iterate through the list to retrieve an AccountAttribute object.

You can get your account's attribute values by invoking the AccountAttribute object's `getAttributeValues` method. This method returns a list of AccountAttributeValue objects.

You can iterate through this second list to display the value of attributes (see the following code example).

Imports

```
import com.amazonaws.services.ec2.AmazonEC2;
import com.amazonaws.services.ec2.AmazonEC2ClientBuilder;
import com.amazonaws.services.ec2.model.AccountAttributeValue;
import com.amazonaws.services.ec2.model.DescribeAccountAttributesResult;
import com.amazonaws.services.ec2.model.AccountAttribute;
import java.util.List;
import java.util.ListIterator;
```

Code

```
AmazonEC2 ec2 = AmazonEC2ClientBuilder.defaultClient();

try{
    DescribeAccountAttributesResult accountResults = ec2.describeAccountAttributes();
    List<AccountAttribute> accountList = accountResults.getAccountAttributes();

    for (ListIterator iter = accountList.listIterator(); iter.hasNext(); ) {

        AccountAttribute attribute = (AccountAttribute) iter.next();
        System.out.print("\n The name of the attribute is
 "+attribute.getAttributeName());
        List<AccountAttributeValue> values = attribute.getAttributeValues();

         //iterate through the attribute values
        for (ListIterator iterVals = values.listIterator(); iterVals.hasNext(); ) {
            AccountAttributeValue myValue = (AccountAttributeValue) iterVals.next();
            System.out.print("\n The value of the attribute is
 "+myValue.getAttributeValue());
        }
    }
    System.out.print("Done");
}
catch (Exception e)
{
    e.getStackTrace();
}
```

See the complete example on GitHub.

More information

- Regions and Availability Zones in the Amazon EC2 User Guide for Linux Instances
- DescribeRegions in the Amazon EC2 API Reference
- DescribeAvailabilityZones in the Amazon EC2 API Reference

Working with Amazon EC2 Key Pairs

Creating a Key Pair

To create a key pair, call the AmazonEC2Client's `createKeyPair` method with a CreateKeyPairRequest that contains the key's name.

Imports

```
import com.amazonaws.services.ec2.AmazonEC2;
import com.amazonaws.services.ec2.AmazonEC2ClientBuilder;
import com.amazonaws.services.ec2.model.CreateKeyPairRequest;
import com.amazonaws.services.ec2.model.CreateKeyPairResult;
```

Code

```
final AmazonEC2 ec2 = AmazonEC2ClientBuilder.defaultClient();

CreateKeyPairRequest request = new CreateKeyPairRequest()
    .withKeyName(key_name);

CreateKeyPairResult response = ec2.createKeyPair(request);
```

See the complete example.

Describing Key Pairs

To list your key pairs or to get information about them, call the AmazonEC2Client's `describeKeyPairs` method. It returns a DescribeKeyPairsResult that you can use to access the list of key pairs by calling its `getKeyPairs` method, which returns a list of KeyPairInfo objects.

Imports

```
import com.amazonaws.services.ec2.AmazonEC2;
```

```
import com.amazonaws.services.ec2.AmazonEC2ClientBuilder;
import com.amazonaws.services.ec2.model.DescribeKeyPairsResult;
import com.amazonaws.services.ec2.model.KeyPairInfo;
```

Code

```
final AmazonEC2 ec2 = AmazonEC2ClientBuilder.defaultClient();

DescribeKeyPairsResult response = ec2.describeKeyPairs();

for(KeyPairInfo key_pair : response.getKeyPairs()) {
    System.out.printf(
        "Found key pair with name %s " +
        "and fingerprint %s",
        key_pair.getKeyName(),
        key_pair.getKeyFingerprint());
}
```

See the complete example.

Deleting a Key Pair

To delete a key pair, call the AmazonEC2Client's `deleteKeyPair` method, passing it a DeleteKeyPairRequest that contains the name of the key pair to delete.

Imports

```
import com.amazonaws.services.ec2.AmazonEC2;
import com.amazonaws.services.ec2.AmazonEC2ClientBuilder;
import com.amazonaws.services.ec2.model.DeleteKeyPairRequest;
import com.amazonaws.services.ec2.model.DeleteKeyPairResult;
```

Code

```
final AmazonEC2 ec2 = AmazonEC2ClientBuilder.defaultClient();

DeleteKeyPairRequest request = new DeleteKeyPairRequest()
    .withKeyName(key_name);

DeleteKeyPairResult response = ec2.deleteKeyPair(request);
```

See the complete example.

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 AmazonEC2Client's `createSecurityGroup` method with a CreateSecurityGroupRequest that contains the key's name.

Imports

```
import com.amazonaws.services.ec2.AmazonEC2;
import com.amazonaws.services.ec2.AmazonEC2ClientBuilder;
import com.amazonaws.services.ec2.model.CreateSecurityGroupRequest;
import com.amazonaws.services.ec2.model.CreateSecurityGroupResult;
```

Code

```
final AmazonEC2 ec2 = AmazonEC2ClientBuilder.defaultClient();

CreateSecurityGroupRequest create_request = new
    CreateSecurityGroupRequest()
        .withGroupName(group_name)
        .withDescription(group_desc)
        .withVpcId(vpc_id);

CreateSecurityGroupResult create_response =
    ec2.createSecurityGroup(create_request);
```

See the complete example.

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 AmazonEC2Client's `authorizeSecurityGroupIngress` method, providing the name of the security group and the access rules (IpPermission) you want to assign to it within an AuthorizeSecurityGroupIngressRequest object. The following example shows how to add IP permissions to a security group.

Imports

```
import com.amazonaws.services.ec2.AmazonEC2;
import com.amazonaws.services.ec2.AmazonEC2ClientBuilder;
import com.amazonaws.services.ec2.model.CreateSecurityGroupRequest;
import com.amazonaws.services.ec2.model.CreateSecurityGroupResult;
```

Code

```
IpRange ip_range = new IpRange()
    .withCidrIp("0.0.0.0/0");

IpPermission ip_perm = new IpPermission()
    .withIpProtocol("tcp")
    .withToPort(80)
    .withFromPort(80)
    .withIpv4Ranges(ip_range);

IpPermission ip_perm2 = new IpPermission()
    .withIpProtocol("tcp")
    .withToPort(22)
    .withFromPort(22)
    .withIpv4Ranges(ip_range);

AuthorizeSecurityGroupIngressRequest auth_request = new
    AuthorizeSecurityGroupIngressRequest()
        .withGroupName(group_name)
        .withIpPermissions(ip_perm, ip_perm2);

AuthorizeSecurityGroupIngressResult auth_response =
    ec2.authorizeSecurityGroupIngress(auth_request);
```

To add an egress rule to the security group, provide similar data in an AuthorizeSecurityGroupEgressRequest to the AmazonEC2Client's `authorizeSecurityGroupEgress` method.

See the complete example.

Describing Security Groups

To describe your security groups or get information about them, call the AmazonEC2Client's `describeSecurityGroups` method. It returns a DescribeSecurityGroupsResult that you can use to access the list of security groups by calling its `getSecurityGroups` method, which returns a list of SecurityGroup objects.

Imports

```
import com.amazonaws.services.ec2.AmazonEC2;
import com.amazonaws.services.ec2.AmazonEC2ClientBuilder;
import com.amazonaws.services.ec2.model.DescribeSecurityGroupsRequest;
import com.amazonaws.services.ec2.model.DescribeSecurityGroupsResult;
```

Code

```
final String USAGE =
    "To run this example, supply a group id\n" +
    "Ex: DescribeSecurityGroups <group-id>\n";

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

String group_id = args[0];
```

See the complete example.

Deleting a Security Group

To delete a security group, call the AmazonEC2Client's `deleteSecurityGroup` method, passing it a DeleteSecurityGroupRequest that contains the ID of the security group to delete.

Imports

```
import com.amazonaws.services.ec2.AmazonEC2;
import com.amazonaws.services.ec2.AmazonEC2ClientBuilder;
import com.amazonaws.services.ec2.model.DeleteSecurityGroupRequest;
```

```
import com.amazonaws.services.ec2.model.DeleteSecurityGroupResult;
```

Code

```
final AmazonEC2 ec2 = AmazonEC2ClientBuilder.defaultClient();

DeleteSecurityGroupRequest request = new DeleteSecurityGroupRequest()
    .withGroupId(group_id);

DeleteSecurityGroupResult response = ec2.deleteSecurityGroup(request);
```

See the complete example.

More Information

- Amazon EC2 Security Groups in the Amazon EC2 User Guide for Linux Instances
- Authorizing Inbound Traffic for Your Linux Instances in the Amazon EC2 User Guide for Linux Instances
- CreateSecurityGroup in the Amazon EC2 API Reference
- DescribeSecurityGroups in the Amazon EC2 API Reference
- DeleteSecurityGroup in the Amazon EC2 API Reference
- AuthorizeSecurityGroupIngress in the Amazon EC2 API Reference

IAM Examples Using the AWS SDK for Java

This section provides examples of programming IAM by 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

Creating an Access Key

To create an IAM access key, call the AmazonIdentityManagementClient`createAccessKey` method with an CreateAccessKeyRequest object.

`CreateAccessKeyRequest` has two constructors — one that takes a user name and another with no parameters. If you use the version that takes no parameters, you must set the user name using the `withUserName` setter method before passing it to the `createAccessKey` method.

Imports

```
import com.amazonaws.services.identitymanagement.AmazonIdentityManagement;
import com.amazonaws.services.identitymanagement.AmazonIdentityManagementClientBuilder;
import com.amazonaws.services.identitymanagement.model.CreateAccessKeyRequest;
import com.amazonaws.services.identitymanagement.model.CreateAccessKeyResult;
```

Code

```
final AmazonIdentityManagement iam =
    AmazonIdentityManagementClientBuilder.defaultClient();

CreateAccessKeyRequest request = new CreateAccessKeyRequest()
    .withUserName(user);

CreateAccessKeyResult 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 AmazonIdentityManagementClient'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 com.amazonaws.services.identitymanagement.AmazonIdentityManagement;
import com.amazonaws.services.identitymanagement.AmazonIdentityManagementClientBuilder;
import com.amazonaws.services.identitymanagement.model.AccessKeyMetadata;
import com.amazonaws.services.identitymanagement.model.ListAccessKeysRequest;
import com.amazonaws.services.identitymanagement.model.ListAccessKeysResult;
```

Code

```
final AmazonIdentityManagement iam =
    AmazonIdentityManagementClientBuilder.defaultClient();

boolean done = false;
ListAccessKeysRequest request = new ListAccessKeysRequest()
        .withUserName(username);

while (!done) {

    ListAccessKeysResult response = iam.listAccessKeys(request);

    for (AccessKeyMetadata metadata :
            response.getAccessKeyMetadata()) {
        System.out.format("Retrieved access key %s",
                metadata.getAccessKeyId());
    }

    request.setMarker(response.getMarker());
```

```
    if (!response.getIsTruncated()) {
        done = true;
    }
}
```

The results of `listAccessKeys` are paged (with a default maximum of 100 records per call). You can call `getIsTruncated` on the returned ListAccessKeysResult object to see if the query returned fewer results then are available. If so, then call `setMarker` on the `ListAccessKeysRequest` and pass it back to 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 AmazonIdentityManagementClient's `getAccessKeyLastUsed` method with the access key's ID (which can be passed in using a GetAccessKeyLastUsedRequest object, or directly to the overload that takes the access key ID directly.

You can then use the returned GetAccessKeyLastUsedResult object to retrieve the key's last used time.

Imports

```
import com.amazonaws.services.identitymanagement.AmazonIdentityManagement;
import com.amazonaws.services.identitymanagement.AmazonIdentityManagementClientBuilder;
import com.amazonaws.services.identitymanagement.model.GetAccessKeyLastUsedRequest;
import com.amazonaws.services.identitymanagement.model.GetAccessKeyLastUsedResult;
```

Code

```
final AmazonIdentityManagement iam =
    AmazonIdentityManagementClientBuilder.defaultClient();

GetAccessKeyLastUsedRequest request = new GetAccessKeyLastUsedRequest()
    .withAccessKeyId(access_id);

GetAccessKeyLastUsedResult response = iam.getAccessKeyLastUsed(request);

System.out.println("Access key was last used at: " +
        response.getAccessKeyLastUsed().getLastUsedDate());
```

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 AmazonIdentityManagementClient's `updateAccessKey` method.

Imports

```
import com.amazonaws.services.identitymanagement.AmazonIdentityManagement;
import com.amazonaws.services.identitymanagement.AmazonIdentityManagementClientBuilder;
import com.amazonaws.services.identitymanagement.model.UpdateAccessKeyRequest;
import com.amazonaws.services.identitymanagement.model.UpdateAccessKeyResult;
```

Code

```
final AmazonIdentityManagement iam =
    AmazonIdentityManagementClientBuilder.defaultClient();

UpdateAccessKeyRequest request = new UpdateAccessKeyRequest()
    .withAccessKeyId(access_id)
    .withUserName(username)
    .withStatus(status);

UpdateAccessKeyResult response = iam.updateAccessKey(request);
```

See the complete example on GitHub.

Deleting an Access Key

To permanently delete an access key, call the AmazonIdentityManagementClient'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 method instead.

Imports

```
import com.amazonaws.services.identitymanagement.AmazonIdentityManagement;
import com.amazonaws.services.identitymanagement.AmazonIdentityManagementClientBuilder;
import com.amazonaws.services.identitymanagement.model.DeleteAccessKeyRequest;
import com.amazonaws.services.identitymanagement.model.DeleteAccessKeyResult;
```

Code

```
final AmazonIdentityManagement iam =
    AmazonIdentityManagementClientBuilder.defaultClient();

DeleteAccessKeyRequest request = new DeleteAccessKeyRequest()
    .withAccessKeyId(access_key)
    .withUserName(username);

DeleteAccessKeyResult response = iam.deleteAccessKey(request);
```

See the complete example on GitHub.

More Information

- CreateAccessKey in the IAM API Reference
- ListAccessKeys in the IAM API Reference
- GetAccessKeyLastUsed in the IAM API Reference
- UpdateAccessKey in the IAM API Reference
- DeleteAccessKey in the IAM API Reference

Managing IAM Users

Creating a User

Create a new IAM user by providing the user name to the AmazonIdentityManagementClient's `createUser` method, either directly or using a CreateUserRequest object containing the user name.

Imports

```
import com.amazonaws.services.identitymanagement.AmazonIdentityManagement;
```

```
import com.amazonaws.services.identitymanagement.AmazonIdentityManagementClientBuilder;
import com.amazonaws.services.identitymanagement.model.CreateUserRequest;
import com.amazonaws.services.identitymanagement.model.CreateUserResult;
```

Code

```
final AmazonIdentityManagement iam =
    AmazonIdentityManagementClientBuilder.defaultClient();

CreateUserRequest request = new CreateUserRequest()
    .withUserName(username);

CreateUserResult response = iam.createUser(request);
```

See the complete example on GitHub.

Listing Users

To list the IAM users for your account, create a new ListUsersRequest and pass it to the AmazonIdentityManagementClient's `listUsers` method. You can retrieve the list of users by calling `getUsers` on the returned ListUsersResult 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 `getIsTruncated` method. If it returns `true`, then call the request object's `setMarker()` method, passing it the return value of the response object's `getMarker()` method.

Imports

```
import com.amazonaws.services.identitymanagement.AmazonIdentityManagement;
import com.amazonaws.services.identitymanagement.AmazonIdentityManagementClientBuilder;
import com.amazonaws.services.identitymanagement.model.ListUsersRequest;
import com.amazonaws.services.identitymanagement.model.ListUsersResult;
import com.amazonaws.services.identitymanagement.model.User;
```

Code

```
final AmazonIdentityManagement iam =
    AmazonIdentityManagementClientBuilder.defaultClient();
```

```
boolean done = false;
ListUsersRequest request = new ListUsersRequest();

while(!done) {
    ListUsersResult response = iam.listUsers(request);

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

    request.setMarker(response.getMarker());

    if(!response.getIsTruncated()) {
        done = true;
    }
}
```

See the complete example on GitHub.

Updating a User

To update a user, call the AmazonIdentityManagementClient object's `updateUser` method, which takes a UpdateUserRequest object that you can use to change the user's *name* or *path*.

Imports

```
import com.amazonaws.services.identitymanagement.AmazonIdentityManagement;
import com.amazonaws.services.identitymanagement.AmazonIdentityManagementClientBuilder;
import com.amazonaws.services.identitymanagement.model.UpdateUserRequest;
import com.amazonaws.services.identitymanagement.model.UpdateUserResult;
```

Code

```
final AmazonIdentityManagement iam =
    AmazonIdentityManagementClientBuilder.defaultClient();

UpdateUserRequest request = new UpdateUserRequest()
    .withUserName(cur_name)
    .withNewUserName(new_name);

UpdateUserResult response = iam.updateUser(request);
```

See the complete example on GitHub.

Deleting a User

To delete a user, call the AmazonIdentityManagementClient's `deleteUser` request with a UpdateUserRequest object set with the user name to delete.

Imports

```
import com.amazonaws.services.identitymanagement.AmazonIdentityManagement;
import com.amazonaws.services.identitymanagement.AmazonIdentityManagementClientBuilder;
import com.amazonaws.services.identitymanagement.model.DeleteConflictException;
import com.amazonaws.services.identitymanagement.model.DeleteUserRequest;
```

Code

```
final AmazonIdentityManagement iam =
    AmazonIdentityManagementClientBuilder.defaultClient();

DeleteUserRequest request = new DeleteUserRequest()
    .withUserName(username);

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 AmazonIdentityManagementClient's `createAccountAlias` method with a CreateAccountAliasRequest object that contains the alias name.

Imports

```
import com.amazonaws.services.identitymanagement.AmazonIdentityManagement;
import com.amazonaws.services.identitymanagement.AmazonIdentityManagementClientBuilder;
import com.amazonaws.services.identitymanagement.model.CreateAccountAliasRequest;
import com.amazonaws.services.identitymanagement.model.CreateAccountAliasResult;
```

Code

```
final AmazonIdentityManagement iam =
    AmazonIdentityManagementClientBuilder.defaultClient();

CreateAccountAliasRequest request = new CreateAccountAliasRequest()
    .withAccountAlias(alias);

CreateAccountAliasResult response = iam.createAccountAlias(request);
```

See the complete example on GitHub.

Listing Account Aliases

To list your account's alias, if any, call the AmazonIdentityManagementClient's `listAccountAliases` method.

> **Note**
>
> The returned ListAccountAliasesResult supports the same `getIsTruncated` and `getMarker` methods as other AWS SDK for Java *list* methods, but an AWS account can have only *one* account alias.

imports

```
import com.amazonaws.services.identitymanagement.AmazonIdentityManagement;
import com.amazonaws.services.identitymanagement.AmazonIdentityManagementClientBuilder;
import com.amazonaws.services.identitymanagement.model.ListAccountAliasesResult;
```

code

```
final AmazonIdentityManagement iam =
    AmazonIdentityManagementClientBuilder.defaultClient();

ListAccountAliasesResult response = iam.listAccountAliases();

for (String alias : response.getAccountAliases()) {
    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 AmazonIdentityManagementClient's `deleteAccountAlias` method. When deleting an account alias, you must supply its name using a DeleteAccountAliasRequest object.

imports

```
import com.amazonaws.services.identitymanagement.AmazonIdentityManagement;
import com.amazonaws.services.identitymanagement.AmazonIdentityManagementClientBuilder;
import com.amazonaws.services.identitymanagement.model.DeleteAccountAliasRequest;
import com.amazonaws.services.identitymanagement.model.DeleteAccountAliasResult;
```

Code

```
final AmazonIdentityManagement iam =
    AmazonIdentityManagementClientBuilder.defaultClient();

DeleteAccountAliasRequest request = new DeleteAccountAliasRequest()
    .withAccountAlias(alias);

DeleteAccountAliasResult response = iam.deleteAccountAlias(request);
```

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 AmazonIdentityManagementClient's `createPolicy` method.

Imports

```
import com.amazonaws.services.identitymanagement.AmazonIdentityManagement;
import com.amazonaws.services.identitymanagement.AmazonIdentityManagementClientBuilder;
import com.amazonaws.services.identitymanagement.model.CreatePolicyRequest;
import com.amazonaws.services.identitymanagement.model.CreatePolicyResult;
```

Code

```
final AmazonIdentityManagement iam =
    AmazonIdentityManagementClientBuilder.defaultClient();

CreatePolicyRequest request = new CreatePolicyRequest()
    .withPolicyName(policy_name)
```

```
    .withPolicyDocument(POLICY_DOCUMENT);

CreatePolicyResult response = iam.createPolicy(request);
```

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

```
public static final String POLICY_DOCUMENT =
    "{" +
    "  \"Version\": \"2012-10-17\"," +
    "  \"Statement\": [" +
    "    {" +
    "        \"Effect\": \"Allow\"," +
    "        \"Action\": \"logs:CreateLogGroup\"," +
    "        \"Resource\": \"%s\"" +
    "    }," +
    "    {" +
    "        \"Effect\": \"Allow\"," +
    "        \"Action\": [" +
    "            \"dynamodb:DeleteItem\"," +
    "            \"dynamodb:GetItem\"," +
    "            \"dynamodb:PutItem\"," +
    "            \"dynamodb:Scan\"," +
    "            \"dynamodb:UpdateItem\"" +
    "       ]," +
    "       \"Resource\": \"RESOURCE_ARN\"" +
    "    }" +
    "   ]" +
    "}";
```

See the complete example on GitHub.

Getting a Policy

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

Imports

```
import com.amazonaws.services.identitymanagement.AmazonIdentityManagement;
import com.amazonaws.services.identitymanagement.AmazonIdentityManagementClientBuilder;
import com.amazonaws.services.identitymanagement.model.GetPolicyRequest;
```

```
import com.amazonaws.services.identitymanagement.model.GetPolicyResult;
```

Code

```
final AmazonIdentityManagement iam =
    AmazonIdentityManagementClientBuilder.defaultClient();

GetPolicyRequest request = new GetPolicyRequest()
    .withPolicyArn(policy_arn);

GetPolicyResult response = iam.getPolicy(request);
```

See the complete example on GitHub.

Attaching a Role Policy

You can attach a policy to an IAMhttp://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles.html[role] by calling the AmazonIdentityManagementClient's `attachRolePolicy` method, providing it with the role name and policy ARN in an AttachRolePolicyRequest.

Imports

```
import com.amazonaws.services.identitymanagement.AmazonIdentityManagement;
import com.amazonaws.services.identitymanagement.AmazonIdentityManagementClientBuilder;
import com.amazonaws.services.identitymanagement.model.AttachRolePolicyRequest;
import com.amazonaws.services.identitymanagement.model.AttachedPolicy;
```

Code

```
final AmazonIdentityManagement iam =
    AmazonIdentityManagementClientBuilder.defaultClient();

AttachRolePolicyRequest attach_request =
    new AttachRolePolicyRequest()
        .withRoleName(role_name)
        .withPolicyArn(POLICY_ARN);

iam.attachRolePolicy(attach_request);
```

See the complete example on GitHub.

Listing Attached Role Policies

List attached policies on a role by calling the AmazonIdentityManagementClient's `listAttachedRolePolicies` method. It takes a ListAttachedRolePoliciesRequest object that contains the role name to list the policies for.

Call `getAttachedPolicies` on the returned ListAttachedRolePoliciesResult object to get the list of attached policies. Results may be truncated; if the `ListAttachedRolePoliciesResult` object's `getIsTruncated` method returns `true`, call the `ListAttachedRolePoliciesRequest` object's `setMarker` method and use it to call `listAttachedRolePolicies` again to get the next batch of results.

Imports

```
import com.amazonaws.services.identitymanagement.AmazonIdentityManagement;
import com.amazonaws.services.identitymanagement.AmazonIdentityManagementClientBuilder;
import com.amazonaws.services.identitymanagement.model.ListAttachedRolePoliciesRequest;
import com.amazonaws.services.identitymanagement.model.ListAttachedRolePoliciesResult;
import java.util.ArrayList;
import java.util.List;
import java.util.stream.Collectors;
```

Code

```
final AmazonIdentityManagement iam =
    AmazonIdentityManagementClientBuilder.defaultClient();

ListAttachedRolePoliciesRequest request =
    new ListAttachedRolePoliciesRequest()
        .withRoleName(role_name);

List<AttachedPolicy> matching_policies = new ArrayList<>();

boolean done = false;

while(!done) {
    ListAttachedRolePoliciesResult response =
        iam.listAttachedRolePolicies(request);

    matching_policies.addAll(
            response.getAttachedPolicies()
```

```
                    .stream()
                    .filter(p -> p.getPolicyName().equals(role_name))
                    .collect(Collectors.toList()));

    if(!response.getIsTruncated()) {
        done = true;
    }
    request.setMarker(response.getMarker());
}
```

See the complete example on GitHub.

Detaching a Role Policy

To detach a policy from a role, call the AmazonIdentityManagementClient's `detachRolePolicy` method, providing it with the role name and policy ARN in a DetachRolePolicyRequest.

Imports

```
import com.amazonaws.services.identitymanagement.AmazonIdentityManagement;
import com.amazonaws.services.identitymanagement.AmazonIdentityManagementClientBuilder;
import com.amazonaws.services.identitymanagement.model.DetachRolePolicyRequest;
import com.amazonaws.services.identitymanagement.model.DetachRolePolicyResult;
```

Code

```
final AmazonIdentityManagement iam =
    AmazonIdentityManagementClientBuilder.defaultClient();

DetachRolePolicyRequest request = new DetachRolePolicyRequest()
    .withRoleName(role_name)
    .withPolicyArn(policy_arn);

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

See the complete example on GitHub.

More Information

- Overview of IAM Policies in the IAM User Guide.
- AWS IAM Policy Reference in the IAM User Guide.

- CreatePolicy in the IAM API Reference
- GetPolicy in the IAM API Reference
- AttachRolePolicy in the IAM API Reference
- ListAttachedRolePolicies in the IAM API Reference
- DetachRolePolicy in the IAM API Reference

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 AmazonIdentityManagementClient's `getServerCertificate` method, passing it a GetServerCertificateRequest with the certificate's name.

Imports

```
import com.amazonaws.services.identitymanagement.AmazonIdentityManagement;
import com.amazonaws.services.identitymanagement.AmazonIdentityManagementClientBuilder;
import com.amazonaws.services.identitymanagement.model.GetServerCertificateRequest;
import com.amazonaws.services.identitymanagement.model.GetServerCertificateResult;
```

Code

```
final AmazonIdentityManagement iam =
    AmazonIdentityManagementClientBuilder.defaultClient();

GetServerCertificateRequest request = new GetServerCertificateRequest()
            .withServerCertificateName(cert_name);
```

```
GetServerCertificateResult response = iam.getServerCertificate(request);
```

See the complete example on GitHub.

Listing Server Certificates

To list your server certificates, call the AmazonIdentityManagementClient's `listServerCertificates` method with a ListServerCertificatesRequest. It returns a ListServerCertificatesResult.

Call the returned `ListServerCertificateResult` object's `getServerCertificateMetadataList` method to get a list of ServerCertificateMetadata objects that you can use to get information about each certificate.

Results may be truncated; if the `ListServerCertificateResult` object's `getIsTruncated` method returns `true`, call the `ListServerCertificatesRequest` object's `setMarker` method and use it to call `listServerCertificates` again to get the next batch of results.

Imports

```
import com.amazonaws.services.identitymanagement.AmazonIdentityManagement;
import com.amazonaws.services.identitymanagement.AmazonIdentityManagementClientBuilder;
import com.amazonaws.services.identitymanagement.model.ListServerCertificatesRequest;
import com.amazonaws.services.identitymanagement.model.ListServerCertificatesResult;
import com.amazonaws.services.identitymanagement.model.ServerCertificateMetadata;
```

Code

```
final AmazonIdentityManagement iam =
    AmazonIdentityManagementClientBuilder.defaultClient();

boolean done = false;
ListServerCertificatesRequest request =
        new ListServerCertificatesRequest();

while(!done) {

    ListServerCertificatesResult response =
        iam.listServerCertificates(request);
```

```
    for(ServerCertificateMetadata metadata :
            response.getServerCertificateMetadataList()) {
        System.out.printf("Retrieved server certificate %s",
                metadata.getServerCertificateName());
    }

    request.setMarker(response.getMarker());

    if(!response.getIsTruncated()) {
        done = true;
    }
}
```

See the complete example on GitHub.

Updating a Server Certificate

You can update a server certificate's name or path by calling the AmazonIdentityManagementClient's `updateServerCertificate` method. It takes a UpdateServerCertificateRequest object set with the server certificate's current name and either a new name or new path to use.

Imports

```
import com.amazonaws.services.identitymanagement.AmazonIdentityManagement;
import com.amazonaws.services.identitymanagement.AmazonIdentityManagementClientBuilder;
import com.amazonaws.services.identitymanagement.model.UpdateServerCertificateRequest;
import com.amazonaws.services.identitymanagement.model.UpdateServerCertificateResult;
```

Code

```
final AmazonIdentityManagement iam =
    AmazonIdentityManagementClientBuilder.defaultClient();

UpdateServerCertificateRequest request =
    new UpdateServerCertificateRequest()
        .withServerCertificateName(cur_name)
        .withNewServerCertificateName(new_name);

UpdateServerCertificateResult response =
```

```
    iam.updateServerCertificate(request);
```

See the complete example on GitHub.

Deleting a Server Certificate

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

Imports

```
import com.amazonaws.services.identitymanagement.AmazonIdentityManagement;
import com.amazonaws.services.identitymanagement.AmazonIdentityManagementClientBuilder;
import com.amazonaws.services.identitymanagement.model.DeleteServerCertificateRequest;
import com.amazonaws.services.identitymanagement.model.DeleteServerCertificateResult;
```

Code

```
final AmazonIdentityManagement iam =
    AmazonIdentityManagementClientBuilder.defaultClient();

DeleteServerCertificateRequest request =
    new DeleteServerCertificateRequest()
        .withServerCertificateName(cert_name);

DeleteServerCertificateResult 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

Lambda Examples Using the AWS SDK for Java

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

> **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

- Invoking, Listing, and Deleting Lambda Functions

Invoking, Listing, and Deleting Lambda Functions

This section provides examples of programming with the Lambda service client by using the AWS SDK for Java. To learn how to create a Lambda function, see How to Create AWS Lambda functions.

Topics

- Invoke a function
- List functions
- Delete a function

Invoke a function

You can invoke a Lambda function by creating an AWSLambda object and invoking its `invoke` method. Create an InvokeRequest object to specify additional information such as the function name and the payload to pass to the Lambda function. Function names appear as *arn:aws:lambda:us-east-1:555556330391:function:HelloFunction*. You can retrieve the value by looking at the function in the AWS Management Console.

To pass payload data to a function, invoke the InvokeRequest object's `withPayload` method and specify a String in JSON format, as shown in the following code example.

Imports

```
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.lambda.AWSLambda;
import com.amazonaws.services.lambda.AWSLambdaClientBuilder;
import com.amazonaws.services.lambda.model.InvokeRequest;
import com.amazonaws.services.lambda.model.InvokeResult;
import com.amazonaws.services.lambda.model.ServiceException;

import java.nio.charset.StandardCharsets;
```

Code

The following code example demonstrates how to invoke a Lambda function.

```
        String functionName = args[0];

        InvokeRequest invokeRequest = new InvokeRequest()
                .withFunctionName(functionName)
                .withPayload("{\n" +
                        " \"Hello \": \"Paris\",\n" +
                        " \"countryCode\": \"FR\"\n" +
                        "}");
        InvokeResult invokeResult = null;

        try {
            AWSLambda awsLambda = AWSLambdaClientBuilder.standard()
                    .withCredentials(new ProfileCredentialsProvider())
                    .withRegion(Regions.US_WEST_2).build();

            invokeResult = awsLambda.invoke(invokeRequest);

            String ans = new String(invokeResult.getPayload().array(),
 StandardCharsets.UTF_8);

            //write out the return value
            System.out.println(ans);

        } catch (ServiceException e) {
            System.out.println(e);
        }
```

```
        System.out.println(invokeResult.getStatusCode());
```

See the complete example on Github.

List functions

Build an AWSLambda object and invoke its `listFunctions` method. This method returns a ListFunctionsResult object. You can invoke this object's `getFunctions` method to return a list of FunctionConfiguration objects. You can iterate through the list to retrieve information about the functions. For example, the following Java code example shows how to get each function name.

Imports

```
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.lambda.AWSLambda;
import com.amazonaws.services.lambda.AWSLambdaClientBuilder;
import com.amazonaws.services.lambda.model.FunctionConfiguration;
import com.amazonaws.services.lambda.model.ListFunctionsResult;
import com.amazonaws.services.lambda.model.ServiceException;
import java.util.Iterator;
import java.util.List;
```

Code

The following Java code example demonstrates how to retrieve a list of Lambda function names.

```
        ListFunctionsResult functionResult = null;

        try {
            AWSLambda awsLambda = AWSLambdaClientBuilder.standard()
                    .withCredentials(new ProfileCredentialsProvider())
                    .withRegion(Regions.US_WEST_2).build();

            functionResult = awsLambda.listFunctions();

            List<FunctionConfiguration> list = functionResult.getFunctions();

            for (Iterator iter = list.iterator(); iter.hasNext(); ) {
                FunctionConfiguration config = (FunctionConfiguration)iter.next();

                System.out.println("The function name is "+config.getFunctionName());
```

```
            }

        } catch (ServiceException e) {
            System.out.println(e);
        }
```

See the complete example on Github.

Delete a function

Build an AWSLambda object and invoke its `deleteFunction` method. Create a DeleteFunctionRequest object and pass it to the `deleteFunction` method. This object contains information such as the name of the function to delete. Function names appear as *arn:aws:lambda:us-east-1:555556330391:function:HelloFunction*. You can retrieve the value by looking at the function in the AWS Management Console.

Imports

```
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.lambda.AWSLambda;
import com.amazonaws.services.lambda.AWSLambdaClientBuilder;
import com.amazonaws.services.lambda.model.ServiceException;
import com.amazonaws.services.lambda.model.DeleteFunctionRequest;
```

Code

The following Java code demonstrates how to delete a Lambda function.

```
        String functionName = args[0];
        try {
            AWSLambda awsLambda = AWSLambdaClientBuilder.standard()
                    .withCredentials(new ProfileCredentialsProvider())
                    .withRegion(Regions.US_WEST_2).build();

            DeleteFunctionRequest delFunc = new DeleteFunctionRequest();
            delFunc.withFunctionName(functionName);

            //Delete the function
            awsLambda.deleteFunction(delFunc);
            System.out.println("The function is deleted");
```

```
        } catch (ServiceException e) {
            System.out.println(e);
        }
```

See the complete example on Github.

Amazon Pinpoint Examples Using the AWS SDK for Java

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

> **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 and Deleting Apps in Amazon Pinpoint

An app is an Amazon Pinpoint project in which you define the audience for a distinct application, and you engage this audience with tailored messages. The examples on this page demonstrate how to create a new app or delete an existing one.

Create an App

Create a new app in Amazon Pinpoint by providing an app name to the CreateAppRequest object, and then passing that object to the AmazonPinpointClient's `createApp` method.

Imports

```
import com.amazonaws.services.pinpoint.AmazonPinpoint;
import com.amazonaws.services.pinpoint.AmazonPinpointClientBuilder;
import com.amazonaws.services.pinpoint.model.CreateAppRequest;
import com.amazonaws.services.pinpoint.model.CreateAppResult;
import com.amazonaws.services.pinpoint.model.CreateApplicationRequest;
```

Code

```
CreateApplicationRequest appRequest = new CreateApplicationRequest()
  .withName(appName);

CreateAppRequest request = new CreateAppRequest();
request.withCreateApplicationRequest(appRequest);
CreateAppResult result = pinpoint.createApp(request);
```

See the complete example on GitHub.

Delete an App

To delete an app, call the AmazonPinpointClient's `deleteApp` request with a DeleteAppRequest object that's set with the app name to delete.

Imports

```
import com.amazonaws.services.pinpoint.AmazonPinpoint;
import com.amazonaws.services.pinpoint.AmazonPinpointClientBuilder;
```

Code

```
DeleteAppRequest deleteRequest = new DeleteAppRequest()
  .withApplicationId(appID);

pinpoint.deleteApp(deleteRequest);
```

See the complete example on GitHub.

More Information

- Apps in the Amazon Pinpoint API Reference
- App in the Amazon Pinpoint API Reference

Creating Endpoints in Amazon Pinpoint

An endpoint uniquely identifies a user device to which you can send push notifications with Amazon Pinpoint. If your app is enabled with Amazon Pinpoint support, your app automatically registers an endpoint with Amazon Pinpoint when a new user opens your app. The following example demonstrates how to add a new endpoint programmatically.

Create an Endpoint

Create a new endpoint in Amazon Pinpoint by providing the endpoint data in an EndpointRequest object.

Imports

```
import com.amazonaws.services.pinpoint.AmazonPinpoint;
import com.amazonaws.services.pinpoint.AmazonPinpointClientBuilder;
import com.amazonaws.services.pinpoint.model.UpdateEndpointRequest;
import com.amazonaws.services.pinpoint.model.UpdateEndpointResult;
import com.amazonaws.services.pinpoint.model.EndpointDemographic;
import com.amazonaws.services.pinpoint.model.EndpointLocation;
import com.amazonaws.services.pinpoint.model.EndpointRequest;
import com.amazonaws.services.pinpoint.model.EndpointResponse;
import com.amazonaws.services.pinpoint.model.EndpointUser;
import com.amazonaws.services.pinpoint.model.GetEndpointRequest;
import com.amazonaws.services.pinpoint.model.GetEndpointResult;
```

Code

```
HashMap<String, List<String>> customAttributes = new HashMap<>();
List<String> favoriteTeams = new ArrayList<>();
favoriteTeams.add("Lakers");
favoriteTeams.add("Warriors");
customAttributes.put("team", favoriteTeams);


EndpointDemographic demographic = new EndpointDemographic()
        .withAppVersion("1.0")
        .withMake("apple")
        .withModel("iPhone")
        .withModelVersion("7")
        .withPlatform("ios")
        .withPlatformVersion("10.1.1")
```

```
        .withTimezone("America/Los_Angeles");

EndpointLocation location = new EndpointLocation()
        .withCity("Los Angeles")
        .withCountry("US")
        .withLatitude(34.0)
        .withLongitude(-118.2)
        .withPostalCode("90068")
        .withRegion("CA");

Map<String,Double> metrics = new HashMap<>();
metrics.put("health", 100.00);
metrics.put("luck", 75.00);

EndpointUser user = new EndpointUser()
        .withUserId(UUID.randomUUID().toString());

DateFormat df = new SimpleDateFormat("yyyy-MM-dd'T'HH:mm'Z'"); // Quoted "Z" to
 indicate UTC, no timezone offset
String nowAsISO = df.format(new Date());

EndpointRequest endpointRequest = new EndpointRequest()
        .withAddress(UUID.randomUUID().toString())
        .withAttributes(customAttributes)
        .withChannelType("APNS")
        .withDemographic(demographic)
        .withEffectiveDate(nowAsISO)
        .withLocation(location)
        .withMetrics(metrics)
        .withOptOut("NONE")
        .withRequestId(UUID.randomUUID().toString())
        .withUser(user);
```

Then create an UpdateEndpointRequest object with that EndpointRequest object. Finally, pass the UpdateEndpointRequest object to the AmazonPinpointClient's `updateEndpoint` method.

Code

```
UpdateEndpointRequest updateEndpointRequest = new UpdateEndpointRequest()
        .withApplicationId(appId)
        .withEndpointId(endpointId)
        .withEndpointRequest(endpointRequest);
```

```
UpdateEndpointResult updateEndpointResponse =
 client.updateEndpoint(updateEndpointRequest);
System.out.println("Update Endpoint Response: " +
 updateEndpointResponse.getMessageBody());
```

See the complete example on GitHub.

More Information

- Adding Endpoint in the Amazon Pinpoint Developer Guide
- Endpoint in the Amazon Pinpoint API Reference

Creating Segments in Amazon Pinpoint

A user segment represents a subset of your users that's based on shared characteristics, such as how recently a user opened your app or which device they use. The following example demonstrates how to define a segment of users.

Create a Segment

Create a new segment in Amazon Pinpoint by defining dimensions of the segment in a SegmentDimensions object.

Imports

```
import com.amazonaws.services.pinpoint.AmazonPinpoint;
import com.amazonaws.services.pinpoint.AmazonPinpointClientBuilder;
import com.amazonaws.services.pinpoint.model.CreateSegmentRequest;
import com.amazonaws.services.pinpoint.model.CreateSegmentResult;
import com.amazonaws.services.pinpoint.model.AttributeDimension;
import com.amazonaws.services.pinpoint.model.AttributeType;
import com.amazonaws.services.pinpoint.model.RecencyDimension;
import com.amazonaws.services.pinpoint.model.SegmentBehaviors;
import com.amazonaws.services.pinpoint.model.SegmentDemographics;
import com.amazonaws.services.pinpoint.model.SegmentDimensions;
import com.amazonaws.services.pinpoint.model.SegmentLocation;
import com.amazonaws.services.pinpoint.model.SegmentResponse;
import com.amazonaws.services.pinpoint.model.WriteSegmentRequest;
```

Code

```
Pinpoint pinpoint =
 AmazonPinpointClientBuilder.standard().withRegion(Regions.US_EAST_1).build();
Map<String, AttributeDimension> segmentAttributes = new HashMap<>();
segmentAttributes.put("Team", new
 AttributeDimension().withAttributeType(AttributeType.INCLUSIVE).withValues("Lakers"));

SegmentBehaviors segmentBehaviors = new SegmentBehaviors();
SegmentDemographics segmentDemographics = new SegmentDemographics();
SegmentLocation segmentLocation = new SegmentLocation();

RecencyDimension recencyDimension = new RecencyDimension();
recencyDimension.withDuration("DAY_30").withRecencyType("ACTIVE");
segmentBehaviors.setRecency(recencyDimension);

SegmentDimensions dimensions = new SegmentDimensions()
        .withAttributes(segmentAttributes)
        .withBehavior(segmentBehaviors)
        .withDemographic(segmentDemographics)
        .withLocation(segmentLocation);
```

Next set the SegmentDimensions object in a WriteSegmentRequest, which in turn is used to create a CreateSegmentRequest object. Then pass the CreateSegmentRequest object to the AmazonPinpointClient's `createSegment` method.

Code

```
WriteSegmentRequest writeSegmentRequest = new WriteSegmentRequest()
        .withName("MySegment").withDimensions(dimensions);

CreateSegmentRequest createSegmentRequest = new CreateSegmentRequest()
        .withApplicationId(appId).withWriteSegmentRequest(writeSegmentRequest);

CreateSegmentResult createSegmentResult = client.createSegment(createSegmentRequest);
```

See the complete example on GitHub.

More Information

- Amazon Pinpoint Segments in the Amazon Pinpoint User Guide
- Creating Segments in the Amazon Pinpoint Developer Guide
- Segments in the Amazon Pinpoint API Reference

- Segment in the Amazon Pinpoint API Reference

Creating Campaigns in Amazon Pinpoint

You can use campaigns to help increase engagement between your app and your users. You can create a campaign to reach out to a particular segment of your users with tailored messages or special promotions. This example demonstrates how to create a new standard campaign that sends a custom push notification to a specified segment.

Create a Campaign

Before creating a new campaign, you must define a Schedule and a Message and set these values in a WriteCampaignRequest object.

Imports

```
import com.amazonaws.services.pinpoint.AmazonPinpoint;
import com.amazonaws.services.pinpoint.AmazonPinpointClientBuilder;
import com.amazonaws.services.pinpoint.model.CreateCampaignRequest;
import com.amazonaws.services.pinpoint.model.CreateCampaignResult;
import com.amazonaws.services.pinpoint.model.Action;
import com.amazonaws.services.pinpoint.model.CampaignResponse;
import com.amazonaws.services.pinpoint.model.Message;
import com.amazonaws.services.pinpoint.model.MessageConfiguration;
import com.amazonaws.services.pinpoint.model.Schedule;
import com.amazonaws.services.pinpoint.model.WriteCampaignRequest;
```

Code

```
Schedule schedule = new Schedule()
        .withStartTime("IMMEDIATE");

Message defaultMessage = new Message()
        .withAction(Action.OPEN_APP)
        .withBody("My message body.")
        .withTitle("My message title.");

MessageConfiguration messageConfiguration = new MessageConfiguration()
        .withDefaultMessage(defaultMessage);

WriteCampaignRequest request = new WriteCampaignRequest()
```

```
        .withDescription("My description.")
        .withSchedule(schedule)
        .withSegmentId(segmentId)
        .withName("MyCampaign")
        .withMessageConfiguration(messageConfiguration);
```

Then create a new campaign in Amazon Pinpoint by providing the WriteCampaignRequest with the campaign configuration to a CreateCampaignRequest object. Finally, pass the CreateCampaignRequest object to the AmazonPinpointClient's `createCampaign` method.

Code

```
CreateCampaignRequest createCampaignRequest = new CreateCampaignRequest()
        .withApplicationId(appId).withWriteCampaignRequest(request);

CreateCampaignResult result = client.createCampaign(createCampaignRequest);
```

See the complete example on GitHub.

More Information

- Amazon Pinpoint Campaigns in the Amazon Pinpoint User Guide
- Creating Campaigns in the Amazon Pinpoint Developer Guide
- Campaigns in the Amazon Pinpoint API Reference
- Campaign in the Amazon Pinpoint API Reference
- Campaign Activities in the Amazon Pinpoint API Reference
- Campaign Versions in the Amazon Pinpoint API Reference
- Campaign Version in the Amazon Pinpoint API Reference

Updating Channels in Amazon Pinpoint

A channel defines the types of platforms to which you can deliver messages. This example shows how to use the APNs channel to send a message.

Update a Channel

Enable a channel in Amazon Pinpoint by providing an app ID and a request object of the channel type you want to update. This example updates the APNs channel, which requires the

APNSChannelRequest object. Set these in the UpdateApnsChannelRequest and pass that object to the AmazonPinpointClient's `updateApnsChannel` method.

Imports

```
import com.amazonaws.services.pinpoint.AmazonPinpoint;
import com.amazonaws.services.pinpoint.AmazonPinpointClientBuilder;
import com.amazonaws.services.pinpoint.model.APNSChannelRequest;
import com.amazonaws.services.pinpoint.model.APNSChannelResponse;
import com.amazonaws.services.pinpoint.model.GetApnsChannelRequest;
import com.amazonaws.services.pinpoint.model.GetApnsChannelResult;
import com.amazonaws.services.pinpoint.model.UpdateApnsChannelRequest;
import com.amazonaws.services.pinpoint.model.UpdateApnsChannelResult;
```

Code

```
APNSChannelRequest request = new APNSChannelRequest()
  .withEnabled(enabled);

UpdateApnsChannelRequest updateRequest = new UpdateApnsChannelRequest()
  .withAPNSChannelRequest(request)
  .withApplicationId(appId);
UpdateApnsChannelResult result = client.updateApnsChannel(updateRequest);
```

See the complete example on GitHub.

More Information

- Amazon Pinpoint Channels in the Amazon Pinpoint User Guide
- ADM Channel in the Amazon Pinpoint API Reference
- APNs Channel in the Amazon Pinpoint API Reference
- APNs Sandbox Channel in the Amazon Pinpoint API Reference
- APNs VoIP Channel in the Amazon Pinpoint API Reference
- APNs VoIP Sandbox Channel in the Amazon Pinpoint API Reference
- Baidu Channel in the Amazon Pinpoint API Reference
- Email Channel in the Amazon Pinpoint API Reference
- GCM Channel in the Amazon Pinpoint API Reference

- SMS Channel in the Amazon Pinpoint API Reference

Amazon S3 Examples Using the AWS SDK for Java

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

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

Topics

Creating, Listing, and Deleting Amazon S3 Buckets

Every object (file) in Amazon S3 must reside within a *bucket*, which represents a collection (container) of objects. Each bucket is known by a *key* (name), which must be unique. For detailed information about buckets and their configuration, see Working with Amazon S3 Buckets in the Amazon Simple Storage Service User Guide.

> **Note**
>
> Best Practice
> We recommend that you enable the AbortIncompleteMultipartUpload lifecycle rule on your Amazon S3 buckets.

This rule directs Amazon S3 to abort multipart uploads that don't complete within a specified number of days after being initiated. When the set time limit is exceeded, Amazon S3 aborts the upload and then deletes the incomplete upload data.
For more information, see Lifecycle Configuration for a Bucket with Versioning in the Amazon S3 User Guide.

Note

These code examples assume that you understand the material in Using the AWS SDK for Java and have configured default AWS credentials using the information in Set up AWS Credentials and Region for Development.

Create a Bucket

Use the AmazonS3 client's `createBucket` method. The new Bucket is returned. The `createBucket` method will raise an exception if the bucket already exists.

Note

To check whether a bucket already exists before attempting to create one with the same name, call the `doesBucketExist` method. It will return `true` if the bucket exists, and `false` otherwise.

Imports

```
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.AmazonS3Exception;
import com.amazonaws.services.s3.model.Bucket;

import java.util.List;
```

Code

```
if (s3.doesBucketExistV2(bucket_name)) {
```

```
    System.out.format("Bucket %s already exists.\n", bucket_name);
    b = getBucket(bucket_name);
} else {
    try {
        b = s3.createBucket(bucket_name);
    } catch (AmazonS3Exception e) {
        System.err.println(e.getErrorMessage());
    }
}
return b;
```

See the complete example on GitHub.

List Buckets

Use the AmazonS3 client's `listBucket` method. If successful, a list of Bucket is returned.

Imports

```
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.Bucket;

import java.util.List;
```

Code

```
List<Bucket> buckets = s3.listBuckets();
System.out.println("Your {S3} buckets are:");
for (Bucket b : buckets) {
    System.out.println("* " + b.getName());
}
```

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 an error will result. If you have a versioned bucket, you must also delete any versioned objects associated with the bucket.

Note

The complete example includes each of these steps in order, providing a complete solution for deleting an Amazon S3 bucket and its contents.

Topics

- Remove Objects from an Unversioned Bucket Before Deleting It
- Remove Objects from a Versioned Bucket Before Deleting It
- Delete an Empty Bucket

Remove Objects from an Unversioned Bucket Before Deleting It

Use the AmazonS3 client's `listObjects` method to retrieve the list of objects and `deleteObject` to delete each one.

Imports

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.*;

import java.util.Iterator;
```

Code

```
System.out.println(" - removing objects from bucket");
ObjectListing object_listing = s3.listObjects(bucket_name);
while (true) {
    for (Iterator<?> iterator =
         object_listing.getObjectSummaries().iterator();
         iterator.hasNext(); ) {
        S3ObjectSummary summary = (S3ObjectSummary) iterator.next();
        s3.deleteObject(bucket_name, summary.getKey());
    }

    // more object_listing to retrieve?
    if (object_listing.isTruncated()) {
```

```
        object_listing = s3.listNextBatchOfObjects(object_listing);
    } else {
        break;
    }
}
```

See the complete example on GitHub.

Remove Objects from a Versioned Bucket Before Deleting It

If you're using a versioned bucket, you also need to remove any stored versions of the objects in the bucket before the bucket can be deleted.

Using a pattern similar to the one used when removing objects within a bucket, remove versioned objects by using the AmazonS3 client's `listVersions` method to list any versioned objects, and then `deleteVersion` to delete each one.

Imports

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.*;

import java.util.Iterator;
```

Code

```
System.out.println(" - removing versions from bucket");
VersionListing version_listing = s3.listVersions(
        new ListVersionsRequest().withBucketName(bucket_name));
while (true) {
    for (Iterator<?> iterator =
         version_listing.getVersionSummaries().iterator();
         iterator.hasNext(); ) {
        S3VersionSummary vs = (S3VersionSummary) iterator.next();
        s3.deleteVersion(
                bucket_name, vs.getKey(), vs.getVersionId());
    }

    if (version_listing.isTruncated()) {
        version_listing = s3.listNextBatchOfVersions(
```

```
                version_listing);
    } else {
        break;
    }
}
```

See the complete example on GitHub.

Delete an Empty Bucket

Once you remove the objects from a bucket (including any versioned objects), you can delete the bucket itself by using the AmazonS3 client's `deleteBucket` method.

Imports

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.*;

import java.util.Iterator;
```

Code

```
System.out.println(" OK, bucket ready to delete!");
s3.deleteBucket(bucket_name);
```

See the complete example on GitHub.

Performing Operations on Amazon S3 Objects

An Amazon S3 object represents a *file* or collection of data. Every object must reside within a bucket.

> **Note**
>
> These code examples assume that you understand the material in Using the AWS SDK for Java and have configured default AWS credentials using the information in Set up AWS Credentials and Region for Development.

Topics

Upload an Object

Use the AmazonS3 client's `putObject` method, supplying a bucket name, key name, and file to upload. *The bucket must exist, or an error will result.*

Imports

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
```

Code

```
System.out.format("Uploading %s to S3 bucket %s...\n", file_path, bucket_name);
final AmazonS3 s3 =
 AmazonS3ClientBuilder.standard().withRegion(Regions.DEFAULT_REGION).build();
try {
    s3.putObject(bucket_name, key_name, new File(file_path));
} catch (AmazonServiceException e) {
    System.err.println(e.getErrorMessage());
    System.exit(1);
}
```

See the complete example on GitHub.

List Objects

To get a list of objects within a bucket, use the AmazonS3 client's `listObjects` method, supplying the name of a bucket.

The `listObjects` method returns an ObjectListing object that provides information about the objects in the bucket. To list the object names (keys), use the `getObjectSummaries` method to get a List of S3ObjectSummary objects, each of which represents a single object in the bucket. Then call its `getKey` method to retrieve the object's name.

Imports

```
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.ListObjectsV2Result;
import com.amazonaws.services.s3.model.S3ObjectSummary;
```

Code

```
System.out.format("Objects in S3 bucket %s:\n", bucket_name);
final AmazonS3 s3 =
 AmazonS3ClientBuilder.standard().withRegion(Regions.DEFAULT_REGION).build();
ListObjectsV2Result result = s3.listObjectsV2(bucket_name);
List<S3ObjectSummary> objects = result.getObjectSummaries();
for (S3ObjectSummary os : objects) {
    System.out.println("* " + os.getKey());
}
```

See the complete example on GitHub.

Download an Object

Use the AmazonS3 client's `getObject` method, passing it the name of a bucket and object to download. If successful, the method returns an S3Object. *The specified bucket and object key must exist, or an error will result.*

You can get the object's contents by calling `getObjectContent` on the `S3Object`. This returns an S3ObjectInputStream that behaves as a standard Java `InputStream` object.

The following example downloads an object from S3 and saves its contents to a file (using the same name as the object's key).

Imports

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.regions.Regions;
```

```
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.S3Object;
import com.amazonaws.services.s3.model.S3ObjectInputStream;

import java.io.File;
```

Code

```
System.out.format("Downloading %s from S3 bucket %s...\n", key_name, bucket_name);
final AmazonS3 s3 =
 AmazonS3ClientBuilder.standard().withRegion(Regions.DEFAULT_REGION).build();
try {
    S3Object o = s3.getObject(bucket_name, key_name);
    S3ObjectInputStream s3is = o.getObjectContent();
    FileOutputStream fos = new FileOutputStream(new File(key_name));
    byte[] read_buf = new byte[1024];
    int read_len = 0;
    while ((read_len = s3is.read(read_buf)) > 0) {
        fos.write(read_buf, 0, read_len);
    }
    s3is.close();
    fos.close();
} catch (AmazonServiceException e) {
    System.err.println(e.getErrorMessage());
    System.exit(1);
} catch (FileNotFoundException e) {
    System.err.println(e.getMessage());
    System.exit(1);
} catch (IOException e) {
    System.err.println(e.getMessage());
    System.exit(1);
}
```

See the complete example on GitHub.

Copy, Move, or Rename Objects

You can copy an object from one bucket to another by using the AmazonS3 client's `copyObject` method. It takes the name of the bucket to copy from, the object to copy, and the destination bucket name.

Imports

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.regions.Regions;
```

Code

```
try {
    s3.copyObject(from_bucket, object_key, to_bucket, object_key);
} catch (AmazonServiceException e) {
    System.err.println(e.getErrorMessage());
    System.exit(1);
}
System.out.println("Done!");
```

See the complete example on GitHub.

> **Note**
>
> You can use `copyObject` with `deleteObject` to **move** or **rename** an object, by first copying the object to a new name (you can use the same bucket as both the source and destination) and then deleting the object from its old location.

Delete an Object

Use the AmazonS3 client's `deleteObject` method, passing it the name of a bucket and object to delete. *The specified bucket and object key must exist, or an error will result.*

Imports

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.regions.Regions;
```

Code

```
final AmazonS3 s3 =
 AmazonS3ClientBuilder.standard().withRegion(Regions.DEFAULT_REGION).build();
try {
    s3.deleteObject(bucket_name, object_key);
} catch (AmazonServiceException e) {
    System.err.println(e.getErrorMessage());
    System.exit(1);
```

```
}
```

See the complete example on GitHub.

Delete Multiple Objects at Once

Using the AmazonS3 client's `deleteObjects` method, you can delete multiple objects from the same bucket by passing their names to the link:sdk-for-java/v1/reference/com/amazonaws/services/s3/model/DeleteObjectsRequest.html method.

Imports

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
```

Code

```
final AmazonS3 s3 =
 AmazonS3ClientBuilder.standard().withRegion(Regions.DEFAULT_REGION).build();
try {
    DeleteObjectsRequest dor = new DeleteObjectsRequest(bucket_name)
            .withKeys(object_keys);
    s3.deleteObjects(dor);
} catch (AmazonServiceException e) {
    System.err.println(e.getErrorMessage());
    System.exit(1);
}
```

See the complete example on GitHub.

Managing Amazon S3 Access Permissions for Buckets and Objects

You can use access control lists (ACLs) for Amazon S3 buckets and objects for fine-grained control over your Amazon S3 resources.

> **Note**
>
> These code examples assume that you understand the material in Using the AWS SDK for Java and have configured default AWS credentials using the information in Set up AWS Credentials and Region for Development.

Get the Access Control List for a Bucket

To get the current ACL for a bucket, call the AmazonS3's `getBucketAcl` method, passing it the *bucket name* to query. This method returns an AccessControlList object. To get each access grant in the list, call its `getGrantsAsList` method, which will return a standard Java list of Grant objects.

Imports

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.AccessControlList;
import com.amazonaws.services.s3.model.Grant;
```

Code

```
final AmazonS3 s3 =
 AmazonS3ClientBuilder.standard().withRegion(Regions.DEFAULT_REGION).build();
try {
    AccessControlList acl = s3.getBucketAcl(bucket_name);
    List<Grant> grants = acl.getGrantsAsList();
    for (Grant grant : grants) {
        System.out.format("  %s: %s\n", grant.getGrantee().getIdentifier(),
                grant.getPermission().toString());
    }
} catch (AmazonServiceException e) {
    System.err.println(e.getErrorMessage());
    System.exit(1);
}
```

See the complete example on GitHub.

Set the Access Control List for a Bucket

To add or modify permissions to an ACL for a bucket, call the AmazonS3's `setBucketAcl` method. It takes an AccessControlList object that contains a list of grantees and access levels to set.

Imports

```
import com.amazonaws.AmazonServiceException;
```

```
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.AccessControlList;
import com.amazonaws.services.s3.model.EmailAddressGrantee;
```

Code

```
final AmazonS3 s3 =
 AmazonS3ClientBuilder.standard().withRegion(Regions.DEFAULT_REGION).build();
try {
    // get the current ACL
    AccessControlList acl = s3.getBucketAcl(bucket_name);
    // set access for the grantee
    EmailAddressGrantee grantee = new EmailAddressGrantee(email);
    Permission permission = Permission.valueOf(access);
    acl.grantPermission(grantee, permission);
    s3.setBucketAcl(bucket_name, acl);
} catch (AmazonServiceException e) {
    System.err.println(e.getErrorMessage());
    System.exit(1);
}
```

> **Note**
>
> You can provide the grantee's unique identifier directly using the Grantee class, or use the EmailAddressGrantee class to set the grantee by email, as we've done here.

See the complete example on GitHub.

Get the Access Control List for an Object

To get the current ACL for an object, call the AmazonS3's `getObjectAcl` method, passing it the *bucket name* and *object name* to query. Like `getBucketAcl`, this method returns an AccessControlList object that you can use to examine each Grant.

Imports

```
import com.amazonaws.AmazonServiceException;
```

```
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.AccessControlList;
import com.amazonaws.services.s3.model.Grant;
```

Code

```
try {
    AccessControlList acl = s3.getObjectAcl(bucket_name, object_key);
    List<Grant> grants = acl.getGrantsAsList();
    for (Grant grant : grants) {
        System.out.format("  %s: %s\n", grant.getGrantee().getIdentifier(),
                grant.getPermission().toString());
    }
} catch (AmazonServiceException e) {
    System.err.println(e.getErrorMessage());
    System.exit(1);
}
```

See the complete example on GitHub.

Set the Access Control List for an Object

To add or modify permissions to an ACL for an object, call the AmazonS3's `setObjectAcl` method. It takes an AccessControlList object that contains a list of grantees and access levels to set.

Imports

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.AccessControlList;
import com.amazonaws.services.s3.model.EmailAddressGrantee;
```

Code

```
    try {
        // get the current ACL
        AccessControlList acl = s3.getObjectAcl(bucket_name, object_key);
```

```
        // set access for the grantee
        EmailAddressGrantee grantee = new EmailAddressGrantee(email);
        Permission permission = Permission.valueOf(access);
        acl.grantPermission(grantee, permission);
        s3.setObjectAcl(bucket_name, object_key, acl);
    } catch (AmazonServiceException e) {
        System.err.println(e.getErrorMessage());
        System.exit(1);
    }
}
```

Note

You can provide the grantee's unique identifier directly using the Grantee class, or use the EmailAddressGrantee class to set the grantee by email, as we've done here.

See the complete example on GitHub.

More Information

- GET Bucket acl in the Amazon S3 API Reference
- PUT Bucket acl in the Amazon S3 API Reference
- GET Object acl in the Amazon S3 API Reference
- PUT Object acl in the Amazon S3 API Reference

Managing Access to Amazon S3 Buckets Using Bucket Policies

You can set, get, or delete a *bucket policy* to manage access to your Amazon S3 buckets.

Set a Bucket Policy

You can set the bucket policy for a particular S3 bucket by:

- Calling the AmazonS3 client's `setBucketPolicy` and providing it with a SetBucketPolicyRequest
- Setting the policy directly by using the `setBucketPolicy` overload that takes a bucket name and policy text (in JSON format)

Imports

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.auth.policy.Policy;
import com.amazonaws.auth.policy.Principal;
```

Code

```
    s3.setBucketPolicy(bucket_name, policy_text);
} catch (AmazonServiceException e) {
    System.err.println(e.getErrorMessage());
    System.exit(1);
}
```

Use the Policy Class to Generate or Validate a Policy

When providing a bucket policy to `setBucketPolicy`, you can do the following:

- Specify the policy directly as a string of JSON-formatted text
- Build the policy using the Policy class

By using the `Policy` class, you don't have to be concerned about correctly formatting your text string. To get the JSON policy text from the `Policy` class, use its `toJson` method.

Imports

```
import com.amazonaws.auth.policy.Resource;
import com.amazonaws.auth.policy.Statement;
import com.amazonaws.auth.policy.actions.S3Actions;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
```

Code

```
        new Statement(Statement.Effect.Allow)
                .withPrincipals(Principal.AllUsers)
                .withActions(S3Actions.GetObject)
                .withResources(new Resource(
                        "{region-arn}s3:::" + bucket_name + "/*")));
```

```
return bucket_policy.toJson();
```

The `Policy` class also provides a `fromJson` method that can attempt to build a policy using a passed-in JSON string. The method validates it to ensure that the text can be transformed into a valid policy structure, and will fail with an `IllegalArgumentException` if the policy text is invalid.

```
Policy bucket_policy = null;
try {
    bucket_policy = Policy.fromJson(file_text.toString());
} catch (IllegalArgumentException e) {
    System.out.format("Invalid policy text in file: \"%s\"",
            policy_file);
    System.out.println(e.getMessage());
}
```

You can use this technique to prevalidate a policy that you read in from a file or other means.

See the complete example on GitHub.

Get a Bucket Policy

To retrieve the policy for an Amazon S3 bucket, call the AmazonS3 client's `getBucketPolicy` method, passing it the name of the bucket to get the policy from.

Imports

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
```

Code

```
  try {
      BucketPolicy bucket_policy = s3.getBucketPolicy(bucket_name);
      policy_text = bucket_policy.getPolicyText();
  } catch (AmazonServiceException e) {
      System.err.println(e.getErrorMessage());
      System.exit(1);
  }
```

If the named bucket doesn't exist, if you don't have access to it, or if it has no bucket policy, an `AmazonServiceException` is thrown.

See the complete example on GitHub.

Delete a Bucket Policy

To delete a bucket policy, call the AmazonS3 client's `deleteBucketPolicy`, providing it with the bucket name.

Imports

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
```

Code

```
try {
    s3.deleteBucketPolicy(bucket_name);
} catch (AmazonServiceException e) {
    System.err.println(e.getErrorMessage());
    System.exit(1);
}
```

This method succeeds even if the bucket doesn't already have a policy. If you specify a bucket name that doesn't exist or if you don't have access to the bucket, an `AmazonServiceException` is thrown.

See the complete example on GitHub.

More Info

- Access Policy Language Overview in the Amazon Simple Storage Service User Guide
- Bucket Policy Examples in the Amazon Simple Storage Service User Guide

Using TransferManager for Amazon S3 Operations

You can use the AWS SDK for Java TransferManager class to reliably transfer files from the local environment to Amazon S3 and to copy objects from one S3 location to another.

`TransferManager` can get the progress of a transfer and pause or resume uploads and downloads.

> **Note**
>
> Best Practice
> We recommend that you enable the AbortIncompleteMultipartUpload lifecycle rule on your Amazon S3 buckets.
> This rule directs Amazon S3 to abort multipart uploads that don't complete within a specified number of days after being initiated. When the set time limit is exceeded, Amazon S3 aborts the upload and then deletes the incomplete upload data.
> For more information, see Lifecycle Configuration for a Bucket with Versioning in the Amazon S3 User Guide.

> **Note**
>
> These code examples assume that you understand the material in Using the AWS SDK for Java and have configured default AWS credentials using the information in Set up AWS Credentials and Region for Development.

Upload Files and Directories

TransferManager can upload files, file lists, and directories to any Amazon S3 buckets that you've previously created.

Topics

- Upload a Single File
- Upload a List of Files
- Upload a Directory

Upload a Single File

Call TransferManager's `upload` method, providing an Amazon S3 bucket name, a key (object) name, and a standard Java File object that represents the file to upload.

Imports

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.services.s3.transfer.MultipleFileUpload;
import com.amazonaws.services.s3.transfer.TransferManager;
import com.amazonaws.services.s3.transfer.TransferManagerBuilder;
import com.amazonaws.services.s3.transfer.Upload;

import java.io.File;
import java.util.ArrayList;
import java.util.Arrays;
```

Code

```
File f = new File(file_path);
TransferManager xfer_mgr = TransferManagerBuilder.standard().build();
try {
    Upload xfer = xfer_mgr.upload(bucket_name, key_name, f);
    // loop with Transfer.isDone()
    XferMgrProgress.showTransferProgress(xfer);
    //  or block with Transfer.waitForCompletion()
    XferMgrProgress.waitForCompletion(xfer);
} catch (AmazonServiceException e) {
    System.err.println(e.getErrorMessage());
    System.exit(1);
}
xfer_mgr.shutdownNow();
```

The `upload` method returns *immediately*, providing an `Upload` object to use to check the transfer state or to wait for it to complete.

See Wait for a Transfer to Complete for information about using `waitForCompletion` to successfully complete a transfer before calling TransferManager's `shutdownNow` method. While waiting for the transfer to complete, you can poll or listen for updates about its status and progress. See Get Transfer Status and Progress for more information.

See the complete example on GitHub.

Upload a List of Files

To upload multiple files in one operation, call the TransferManager`uploadFileList` method, providing the following:

- An Amazon S3 bucket name

- A *key prefix* to prepend to the names of the created objects (the path within the bucket in which to place the objects)
- A File object that represents the relative directory from which to create file paths
- A List object containing a set of File objects to upload

Imports

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.services.s3.transfer.MultipleFileUpload;
import com.amazonaws.services.s3.transfer.TransferManager;
import com.amazonaws.services.s3.transfer.TransferManagerBuilder;
import com.amazonaws.services.s3.transfer.Upload;

import java.io.File;
import java.util.ArrayList;
import java.util.Arrays;
```

Code

```
ArrayList<File> files = new ArrayList<File>();
for (String path : file_paths) {
    files.add(new File(path));
}

TransferManager xfer_mgr = TransferManagerBuilder.standard().build();
try {
    MultipleFileUpload xfer = xfer_mgr.uploadFileList(bucket_name,
            key_prefix, new File("."), files);
    // loop with Transfer.isDone()
    XferMgrProgress.showTransferProgress(xfer);
    // or block with Transfer.waitForCompletion()
    XferMgrProgress.waitForCompletion(xfer);
} catch (AmazonServiceException e) {
    System.err.println(e.getErrorMessage());
    System.exit(1);
}
xfer_mgr.shutdownNow();
```

See Wait for a Transfer to Complete for information about using `waitForCompletion` to successfully complete a transfer before calling TransferManager's `shutdownNow` method. While

waiting for the transfer to complete, you can poll or listen for updates about its status and progress. See Get Transfer Status and Progress for more information.

The MultipleFileUpload object returned by `uploadFileList` can be used to query the transfer state or progress. See Poll the Current Progress of a Transfer and Get Transfer Progress with a ProgressListener for more information.

You can also use `MultipleFileUpload`'s `getSubTransfers` method to get the individual `Upload` objects for each file being transferred. For more information, see Get the Progress of Subtransfers.

See the complete example on GitHub.

Upload a Directory

You can use TransferManager's `uploadDirectory` method to upload an entire directory of files, with the option to copy files in subdirectories recursively. You provide an Amazon S3 bucket name, an S3 key prefix, a File object representing the local directory to copy, and a `boolean` value indicating whether you want to copy subdirectories recursively (*true* or *false*).

Imports

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.services.s3.transfer.MultipleFileUpload;
import com.amazonaws.services.s3.transfer.TransferManager;
import com.amazonaws.services.s3.transfer.TransferManagerBuilder;
import com.amazonaws.services.s3.transfer.Upload;

import java.io.File;
import java.util.ArrayList;
import java.util.Arrays;
```

Code

```
TransferManager xfer_mgr = TransferManagerBuilder.standard().build();
try {
    MultipleFileUpload xfer = xfer_mgr.uploadDirectory(bucket_name,
            key_prefix, new File(dir_path), recursive);
    // loop with Transfer.isDone()
    XferMgrProgress.showTransferProgress(xfer);
    // or block with Transfer.waitForCompletion()
    XferMgrProgress.waitForCompletion(xfer);
```

```
} catch (AmazonServiceException e) {
    System.err.println(e.getErrorMessage());
    System.exit(1);
}
xfer_mgr.shutdownNow();
```

See Wait for a Transfer to Complete for information about using `waitForCompletion` to successfully complete a transfer before calling TransferManager's `shutdownNow` method. While waiting for the transfer to complete, you can poll or listen for updates about its status and progress. See Get Transfer Status and Progress for more information.

The MultipleFileUpload object returned by `uploadFileList` can be used to query the transfer state or progress. See Poll the Current Progress of a Transfer and Get Transfer Progress with a ProgressListener for more information.

You can also use `MultipleFileUpload`'s `getSubTransfers` method to get the individual `Upload` objects for each file being transferred. For more information, see Get the Progress of Subtransfers.

See the complete example on GitHub.

Download Files or Directories

Use the TransferManager class to download either a single file (Amazon S3 object) or a directory (an Amazon S3 bucket name followed by an object prefix) from Amazon S3.

Topics

- Download a Single File
- Download a Directory

Download a Single File

Use the TransferManager's `download` method, providing the Amazon S3 bucket name containing the object you want to download, the key (object) name, and a File object that represents the file to create on your local system.

Imports

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.services.s3.transfer.Download;
```

```
import com.amazonaws.services.s3.transfer.MultipleFileDownload;
import com.amazonaws.services.s3.transfer.TransferManager;
import com.amazonaws.services.s3.transfer.TransferManagerBuilder;

import java.io.File;
```

Code

```
File f = new File(file_path);
TransferManager xfer_mgr = TransferManagerBuilder.standard().build();
try {
    Download xfer = xfer_mgr.download(bucket_name, key_name, f);
    // loop with Transfer.isDone()
    XferMgrProgress.showTransferProgress(xfer);
    // or block with Transfer.waitForCompletion()
    XferMgrProgress.waitForCompletion(xfer);
} catch (AmazonServiceException e) {
    System.err.println(e.getErrorMessage());
    System.exit(1);
}
xfer_mgr.shutdownNow();
```

See Wait for a Transfer to Complete for information about using `waitForCompletion` to successfully complete a transfer before calling TransferManager's `shutdownNow` method. While waiting for the transfer to complete, you can poll or listen for updates about its status and progress. See Get Transfer Status and Progress for more information.

See the complete example on GitHub.

Download a Directory

To download a set of files that share a common key prefix (analogous to a directory on a file system) from Amazon S3, use the TransferManager`downloadDirectory` method. The method takes the Amazon S3 bucket name containing the objects you want to download, the object prefix shared by all of the objects, and a File object that represents the directory to download the files into on your local system. If the named directory doesn't exist yet, it will be created.

Imports

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.services.s3.transfer.Download;
```

```
import com.amazonaws.services.s3.transfer.MultipleFileDownload;
import com.amazonaws.services.s3.transfer.TransferManager;
import com.amazonaws.services.s3.transfer.TransferManagerBuilder;

import java.io.File;
```

Code

```
TransferManager xfer_mgr = TransferManagerBuilder.standard().build();

try {
    MultipleFileDownload xfer = xfer_mgr.downloadDirectory(
            bucket_name, key_prefix, new File(dir_path));
    // loop with Transfer.isDone()
    XferMgrProgress.showTransferProgress(xfer);
    // or block with Transfer.waitForCompletion()
    XferMgrProgress.waitForCompletion(xfer);
} catch (AmazonServiceException e) {
    System.err.println(e.getErrorMessage());
    System.exit(1);
}
xfer_mgr.shutdownNow();
```

See Wait for a Transfer to Complete for information about using `waitForCompletion` to successfully complete a transfer before calling TransferManager's `shutdownNow` method. While waiting for the transfer to complete, you can poll or listen for updates about its status and progress. See Get Transfer Status and Progress for more information.

See the complete example on GitHub.

Copy Objects

To copy an object from one S3 bucket to another, use the TransferManager`copy` method.

Imports

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.services.s3.transfer.Copy;
import com.amazonaws.services.s3.transfer.TransferManager;
import com.amazonaws.services.s3.transfer.TransferManagerBuilder;
```

Code

```
System.out.println("Copying s3 object: " + from_key);
System.out.println("      from bucket: " + from_bucket);
System.out.println("     to s3 object: " + to_key);
System.out.println("        in bucket: " + to_bucket);

TransferManager xfer_mgr = TransferManagerBuilder.standard().build();
try {
    Copy xfer = xfer_mgr.copy(from_bucket, from_key, to_bucket, to_key);
    // loop with Transfer.isDone()
    XferMgrProgress.showTransferProgress(xfer);
    // or block with Transfer.waitForCompletion()
    XferMgrProgress.waitForCompletion(xfer);
} catch (AmazonServiceException e) {
    System.err.println(e.getErrorMessage());
    System.exit(1);
}
xfer_mgr.shutdownNow();
```

See the complete example on GitHub.

Wait for a Transfer to Complete

If your application (or thread) can block until the transfer completes, you can use the Transfer interface's `waitForCompletion` method to block until the transfer is complete or an exception occurs.

```
try {
    xfer.waitForCompletion();
} catch (AmazonServiceException e) {
    System.err.println("Amazon service error: " + e.getMessage());
    System.exit(1);
} catch (AmazonClientException e) {
    System.err.println("Amazon client error: " + e.getMessage());
    System.exit(1);
} catch (InterruptedException e) {
    System.err.println("Transfer interrupted: " + e.getMessage());
    System.exit(1);
}
```

You get progress of transfers if you poll for events *before* calling `waitForCompletion`, implement a polling mechanism on a separate thread, or receive progress updates asynchronously using a ProgressListener.

See the complete example on GitHub.

Get Transfer Status and Progress

Each of the classes returned by the TransferManagerupload*, download*, and copy methods returns an instance of one of the following classes, depending on whether it's a single-file or multiple-file operation.

Class	Returned by
Copy	copy
Download	download
MultipleFileDownload	downloadDirectory
Upload	upload
MultipleFileUpload	uploadFileList ,uploadDirectory

All of these classes implement the Transfer interface. Transfer provides useful methods to get the progress of a transfer, pause or resume the transfer, and get the transfer's current or final status.

Topics

- Poll the Current Progress of a Transfer
- Get Transfer Progress with a ProgressListener
- Get the Progress of Subtransfers

Poll the Current Progress of a Transfer

This loop prints the progress of a transfer, examines its current progress while running and, when complete, prints its final state.

Imports

```
import com.amazonaws.AmazonClientException;
```

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.event.ProgressEvent;
import com.amazonaws.event.ProgressListener;
import com.amazonaws.services.s3.transfer.*;
import com.amazonaws.services.s3.transfer.Transfer.TransferState;

import java.io.File;
import java.util.ArrayList;
import java.util.Collection;
```

Code

```
// print the transfer's human-readable description
System.out.println(xfer.getDescription());
// print an empty progress bar...
printProgressBar(0.0);
// update the progress bar while the xfer is ongoing.
do {
    try {
        Thread.sleep(100);
    } catch (InterruptedException e) {
        return;
    }
    // Note: so_far and total aren't used, they're just for
    // documentation purposes.
    TransferProgress progress = xfer.getProgress();
    long so_far = progress.getBytesTransferred();
    long total = progress.getTotalBytesToTransfer();
    double pct = progress.getPercentTransferred();
    eraseProgressBar();
    printProgressBar(pct);
} while (xfer.isDone() == false);
// print the final state of the transfer.
TransferState xfer_state = xfer.getState();
System.out.println(": " + xfer_state);
```

See the complete example on GitHub.

Get Transfer Progress with a ProgressListener

You can attach a ProgressListener to any transfer by using the Transfer interface's `addProgressListener` method.

A ProgressListener requires only one method, `progressChanged`, which takes a ProgressEvent object. You can use the object to get the total bytes of the operation by calling its `getBytes` method, and the number of bytes transferred so far by calling `getBytesTransferred`.

Imports

```
import com.amazonaws.AmazonClientException;
import com.amazonaws.AmazonServiceException;
import com.amazonaws.event.ProgressEvent;
import com.amazonaws.event.ProgressListener;
import com.amazonaws.services.s3.transfer.*;
import com.amazonaws.services.s3.transfer.Transfer.TransferState;

import java.io.File;
import java.util.ArrayList;
import java.util.Collection;
```

Code

```
File f = new File(file_path);
TransferManager xfer_mgr = TransferManagerBuilder.standard().build();
try {
    Upload u = xfer_mgr.upload(bucket_name, key_name, f);
    // print an empty progress bar...
    printProgressBar(0.0);
    u.addProgressListener(new ProgressListener() {
        public void progressChanged(ProgressEvent e) {
            double pct = e.getBytesTransferred() * 100.0 / e.getBytes();
            eraseProgressBar();
            printProgressBar(pct);
        }
    });
    // block with Transfer.waitForCompletion()
    XferMgrProgress.waitForCompletion(u);
    // print the final state of the transfer.
    TransferState xfer_state = u.getState();
    System.out.println(": " + xfer_state);
} catch (AmazonServiceException e) {
    System.err.println(e.getErrorMessage());
    System.exit(1);
}
xfer_mgr.shutdownNow();
```

See the complete example on GitHub.

Get the Progress of Subtransfers

The MultipleFileUpload class can return information about its subtransfers by calling its `getSubTransfers` method. It returns an unmodifiable Collection of Upload objects that provide the individual transfer status and progress of each sub-transfer.

Imports

```
import com.amazonaws.AmazonClientException;
import com.amazonaws.AmazonServiceException;
import com.amazonaws.event.ProgressEvent;
import com.amazonaws.event.ProgressListener;
import com.amazonaws.services.s3.transfer.*;
import com.amazonaws.services.s3.transfer.Transfer.TransferState;

import java.io.File;
import java.util.ArrayList;
import java.util.Collection;
```

Code

```
Collection<? extends Upload> sub_xfers = new ArrayList<Upload>();
sub_xfers = multi_upload.getSubTransfers();

do {
    System.out.println("\nSubtransfer progress:\n");
    for (Upload u : sub_xfers) {
        System.out.println("  " + u.getDescription());
        if (u.isDone()) {
            TransferState xfer_state = u.getState();
            System.out.println("  " + xfer_state);
        } else {
            TransferProgress progress = u.getProgress();
            double pct = progress.getPercentTransferred();
            printProgressBar(pct);
            System.out.println();
        }
    }

    // wait a bit before the next update.
```

```
    try {
        Thread.sleep(200);
    } catch (InterruptedException e) {
        return;
    }
} while (multi_upload.isDone() == false);
// print the final state of the transfer.
TransferState xfer_state = multi_upload.getState();
System.out.println("\nMultipleFileUpload " + xfer_state);
```

See the complete example on GitHub.

More Info

- Object Keys in the Amazon Simple Storage Service User Guide

Configuring an Amazon S3 Bucket as a Website

You can configure an Amazon S3 bucket to behave as a website. To do this, you need to set its website configuration.

> **Note**
>
> These code examples assume that you understand the material in Using the AWS SDK for Java and have configured default AWS credentials using the information in Set up AWS Credentials and Region for Development.

Set a Bucket's Website Configuration

To set an Amazon S3 bucket's website configuration, call the AmazonS3's `setWebsiteConfiguration` method with the bucket name to set the configuration for, and a BucketWebsiteConfiguration object containing the bucket's website configuration.

Setting an index document is *required*; all other parameters are optional.

Imports

```
import com.amazonaws.AmazonServiceException;
```

```
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.BucketWebsiteConfiguration;
```

Code

```
    String bucket_name, String index_doc, String error_doc) {
BucketWebsiteConfiguration website_config = null;

if (index_doc == null) {
    website_config = new BucketWebsiteConfiguration();
} else if (error_doc == null) {
    website_config = new BucketWebsiteConfiguration(index_doc);
} else {
    website_config = new BucketWebsiteConfiguration(index_doc, error_doc);
}

final AmazonS3 s3 =
 AmazonS3ClientBuilder.standard().withRegion(Regions.DEFAULT_REGION).build();
try {
    s3.setBucketWebsiteConfiguration(bucket_name, website_config);
} catch (AmazonServiceException e) {
    System.out.format(
            "Failed to set website configuration for bucket '%s'!\n",
            bucket_name);
    System.err.println(e.getErrorMessage());
    System.exit(1);
}
```

Note

Setting a website configuration does not modify the access permissions for your bucket. To make your files visible on the web, you will also need to set a *bucket policy* that allows public read access to the files in the bucket. For more information, see Managing Access to Amazon S3 Buckets Using Bucket Policies.

See the complete example on GitHub.

Get a Bucket's Website Configuration

To get an Amazon S3 bucket's website configuration, call the AmazonS3's `getWebsiteConfiguration` method with the name of the bucket to retrieve the configuration for.

The configuration will be returned as a BucketWebsiteConfiguration object. If there is no website configuration for the bucket, then `null` will be returned.

Imports

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.BucketWebsiteConfiguration;
```

Code

```
final AmazonS3 s3 =
 AmazonS3ClientBuilder.standard().withRegion(Regions.DEFAULT_REGION).build();
try {
    BucketWebsiteConfiguration config =
            s3.getBucketWebsiteConfiguration(bucket_name);
    if (config == null) {
        System.out.println("No website configuration found!");
    } else {
        System.out.format("Index document: %s\n",
                config.getIndexDocumentSuffix());
        System.out.format("Error document: %s\n",
                config.getErrorDocument());
    }
} catch (AmazonServiceException e) {
    System.err.println(e.getErrorMessage());
    System.out.println("Failed to get website configuration!");
    System.exit(1);
}
```

See the complete example on GitHub.

Delete a Bucket's Website Configuration

To delete an Amazon S3 bucket's website configuration, call the AmazonS3's `deleteWebsiteConfiguration` method with the name of the bucket to delete the configuration from.

Imports

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
```

Code

```
final AmazonS3 s3 =
 AmazonS3ClientBuilder.standard().withRegion(Regions.DEFAULT_REGION).build();
try {
    s3.deleteBucketWebsiteConfiguration(bucket_name);
} catch (AmazonServiceException e) {
    System.err.println(e.getErrorMessage());
    System.out.println("Failed to delete website configuration!");
    System.exit(1);
}
```

See the complete example on GitHub.

More Information

- PUT Bucket website in the Amazon S3 API Reference
- GET Bucket website in the Amazon S3 API Reference
- DELETE Bucket website in the Amazon S3 API Reference

Use Amazon S3 client-side encryption

Encrypting data using the Amazon S3 encryption client is one way you can provide an additional layer of protection for sensitive information you store in Amazon S3. The examples in this section demonstrate how to create and configure the Amazon S3 encryption client for your application.

If you are new to cryptography, see the Cryptography Basics in the AWS KMS Developer Guide for a basic overview of cryptography terms and algorithms. For information about cryptography support across all AWS SDKs, see AWS SDK Support for Amazon S3 Client-Side Encryption in the Amazon Web Services General Reference.

> **Note**
>
> These code examples assume that you understand the material in Using the AWS SDK for Java and have configured default AWS credentials using the information in Set up AWS Credentials and Region for Development.

If you are using version 1.11.836 or earlier of the AWS SDK for Java, see Amazon S3 Encryption Client Migration for information on migrating your applications to later versions. If you cannot migrate, see this complete example on GitHub.

Otherwise, if you are using version 1.11.837 or later of the AWS SDK for Java, explore the example topics listed below to use Amazon S3 client-side encryption.

Topics

Amazon S3 client-side encryption with client master keys

The following examples use the AmazonS3EncryptionClientV2Builder class to create an Amazon S3 client with client-side encryption enabled. Once enabled, any objects you upload to Amazon S3 using this client will be encrypted. Any objects you get from Amazon S3 using this client will automatically be decrypted.

> **Note**
>
> The following examples demonstrate using the Amazon S3 client-side encryption with customer-managed client master keys. To learn how to use encryption with AWS KMS managed keys, see Amazon S3 client-side encryption with AWS KMS managed keys.

You can choose from two encryption modes when enabling client-side Amazon S3 encryption: strict authenticated or authenticated. The following sections show how to enable each type. To learn which algorithms each mode uses, see the CryptoMode definition.

Required imports

Import the following classes for these examples.

Imports

```
import com.amazonaws.ClientConfiguration;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3EncryptionClientV2Builder;
import com.amazonaws.services.s3.AmazonS3EncryptionV2;
import com.amazonaws.services.s3.model.CryptoConfigurationV2;
import com.amazonaws.services.s3.model.CryptoMode;
import com.amazonaws.services.s3.model.EncryptionMaterials;
import com.amazonaws.services.s3.model.StaticEncryptionMaterialsProvider;
```

Strict authenticated encryption

Strict authenticated encryption is the default mode if no `CryptoMode` is specified.

To explicitly enable this mode, specify the `StrictAuthenticatedEncryption` value in the `withCryptoConfiguration` method.

> **Note**
>
> To use client-side authenticated encryption, you must include the latest Bouncy Castle jar file in the classpath of your application.

Code

```
AmazonS3EncryptionV2 s3Encryption = AmazonS3EncryptionClientV2Builder.standard()
         .withRegion(Regions.US_WEST_2)
         .withCryptoConfiguration(new
 CryptoConfigurationV2().withCryptoMode((CryptoMode.StrictAuthenticatedEncryption)))
         .withEncryptionMaterialsProvider(new StaticEncryptionMaterialsProvider(new
 EncryptionMaterials(secretKey)))
         .build();
```

```
s3Encryption.putObject(bucket_name, ENCRYPTED_KEY2, "This is the 2nd content to
 encrypt");
```

Authenticated encryption mode

When you use `AuthenticatedEncryption` mode, an improved key wrapping algorithm is applied during encryption. When decrypting in this mode, the algorithm can verify the integrity of the decrypted object and throw an exception if the check fails. For more details about how authenticated encryption works, see the Amazon S3 Client-Side Authenticated Encryption blog post.

> **Note**
>
> To use client-side authenticated encryption, you must include the latest Bouncy Castle jar file in the classpath of your application.

To enable this mode, specify the `AuthenticatedEncryption` value in the `withCryptoConfiguration` method.

Code

```
AmazonS3EncryptionV2 s3EncryptionClientV2 =
 AmazonS3EncryptionClientV2Builder.standard()
         .withRegion(Regions.DEFAULT_REGION)
         .withClientConfiguration(new ClientConfiguration())
         .withCryptoConfiguration(new
 CryptoConfigurationV2().withCryptoMode(CryptoMode.AuthenticatedEncryption))
         .withEncryptionMaterialsProvider(new StaticEncryptionMaterialsProvider(new
 EncryptionMaterials(secretKey)))
         .build();

s3EncryptionClientV2.putObject(bucket_name, ENCRYPTED_KEY1, "This is the 1st content to
 encrypt");
```

Amazon S3 client-side encryption with AWS KMS managed keys

The following examples use the AmazonS3EncryptionClientV2Builder class to create an Amazon S3 client with client-side encryption enabled. Once configured, any objects you upload to Amazon S3 using this client will be encrypted. Any objects you get from Amazon S3 using this client are automatically decrypted.

> **Note**
>
> The following examples demonstrate how to use the Amazon S3 client-side encryption with AWS KMS managed keys. To learn how to use encryption with your own keys, see Amazon S3 client-side encryption with client master keys.

You can choose from two encryption modes when enabling client-side Amazon S3 encryption: strict authenticated or authenticated. The following sections show how to enable each type. To learn which algorithms each mode uses, see the CryptoMode definition.

Required imports

Import the following classes for these examples.

Imports

```
import com.amazonaws.ClientConfiguration;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.kms.AWSKMS;
import com.amazonaws.services.kms.AWSKMSClientBuilder;
import com.amazonaws.services.kms.model.GenerateDataKeyRequest;
import com.amazonaws.services.kms.model.GenerateDataKeyResult;
import com.amazonaws.services.s3.AmazonS3EncryptionClientV2Builder;
import com.amazonaws.services.s3.AmazonS3EncryptionV2;
import com.amazonaws.services.s3.model.CryptoConfigurationV2;
import com.amazonaws.services.s3.model.CryptoMode;
import com.amazonaws.services.s3.model.EncryptionMaterials;
import com.amazonaws.services.s3.model.KMSEncryptionMaterialsProvider;
```

Strict authenticated encryption

Strict authenticated encryption is the default mode if no `CryptoMode` is specified.

To explicitly enable this mode, specify the `StrictAuthenticatedEncryption` value in the `withCryptoConfiguration` method.

> **Note**
>
> To use client-side authenticated encryption, you must include the latest Bouncy Castle jar file in the classpath of your application.

Code

```
AmazonS3EncryptionV2 s3Encryption = AmazonS3EncryptionClientV2Builder.standard()
         .withRegion(Regions.US_WEST_2)
         .withCryptoConfiguration(new
 CryptoConfigurationV2().withCryptoMode((CryptoMode.StrictAuthenticatedEncryption)))
         .withEncryptionMaterialsProvider(new KMSEncryptionMaterialsProvider(keyId))
         .build();

s3Encryption.putObject(bucket_name, ENCRYPTED_KEY3, "This is the 3rd content to encrypt
 with a key created in the {console}");
System.out.println(s3Encryption.getObjectAsString(bucket_name, ENCRYPTED_KEY3));
```

Call the `putObject` method on the Amazon S3 encryption client to upload objects.

Code

```
s3Encryption.putObject(bucket_name, ENCRYPTED_KEY3, "This is the 3rd content to encrypt
 with a key created in the {console}");
```

You can retrieve the object using the same client. This example calls the `getObjectAsString` method to retrieve the string that was stored.

Code

```
System.out.println(s3Encryption.getObjectAsString(bucket_name, ENCRYPTED_KEY3));
```

Authenticated encryption mode

When you use `AuthenticatedEncryption` mode, an improved key wrapping algorithm is applied during encryption. When decrypting in this mode, the algorithm can verify the integrity of the decrypted object and throw an exception if the check fails. For more details about how authenticated encryption works, see the Amazon S3 Client-Side Authenticated Encryption blog post.

> **Note**
>
> To use client-side authenticated encryption, you must include the latest Bouncy Castle jar file in the classpath of your application.

To enable this mode, specify the `AuthenticatedEncryption` value in the `withCryptoConfiguration` method.

Code

```
AmazonS3EncryptionV2 s3Encryption = AmazonS3EncryptionClientV2Builder.standard()
         .withRegion(Regions.US_WEST_2)
         .withCryptoConfiguration(new
 CryptoConfigurationV2().withCryptoMode((CryptoMode.AuthenticatedEncryption)))
         .withEncryptionMaterialsProvider(new KMSEncryptionMaterialsProvider(keyId))
         .build();
```

Configuring the AWS KMS client

The Amazon S3 encryption client creates a AWS KMS client by default, unless one is explicitly specified.

To set the region for this automatically-created AWS KMS client, set the `awsKmsRegion`.

Code

```
Region kmsRegion = Region.getRegion(Regions.AP_NORTHEAST_1);

AmazonS3EncryptionV2 s3Encryption = AmazonS3EncryptionClientV2Builder.standard()
        .withRegion(Regions.US_WEST_2)
        .withCryptoConfiguration(new
 CryptoConfigurationV2().withAwsKmsRegion(kmsRegion))
        .withEncryptionMaterialsProvider(new KMSEncryptionMaterialsProvider(keyId))
        .build();
```

Alternatively, you can use your own AWS KMS client to initialize the encryption client.

Code

```
AWSKMS kmsClient = AWSKMSClientBuilder.standard()
        .withRegion(Regions.US_WEST_2);
        .build();

AmazonS3EncryptionV2 s3Encryption = AmazonS3EncryptionClientV2Builder.standard()
        .withRegion(Regions.US_WEST_2)
        .withKmsClient(kmsClient)
```

```
        .withCryptoConfiguration(new
CryptoConfigurationV2().withCryptoMode((CryptoMode.AuthenticatedEncryption)))
        .withEncryptionMaterialsProvider(new KMSEncryptionMaterialsProvider(keyId))
        .build();
```

Amazon SQS Examples Using the AWS SDK for Java

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

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

Topics

Working with Amazon SQS Message Queues

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

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

Create a Queue

Use the AmazonSQS client's `createQueue` method, providing a CreateQueueRequest object that describes the queue parameters.

Imports

```
import com.amazonaws.services.sqs.AmazonSQS;
import com.amazonaws.services.sqs.AmazonSQSClientBuilder;
import com.amazonaws.services.sqs.model.AmazonSQSException;
import com.amazonaws.services.sqs.model.CreateQueueRequest;
```

Code

```
AmazonSQS sqs = AmazonSQSClientBuilder.defaultClient();
CreateQueueRequest create_request = new CreateQueueRequest(QUEUE_NAME)
        .addAttributesEntry("DelaySeconds", "60")
        .addAttributesEntry("MessageRetentionPeriod", "86400");

try {
    sqs.createQueue(create_request);
} catch (AmazonSQSException e) {
    if (!e.getErrorCode().equals("QueueAlreadyExists")) {
        throw e;
    }
}
```

You can use the simplified form of `createQueue`, which needs only a queue name, to create a standard queue.

```
sqs.createQueue("MyQueue" + new Date().getTime());
```

See the complete example on GitHub.

Listing Queues

To list the Amazon SQS queues for your account, call the AmazonSQS client's `listQueues` method.

Imports

```
import com.amazonaws.services.sqs.AmazonSQS;
import com.amazonaws.services.sqs.AmazonSQSClientBuilder;
import com.amazonaws.services.sqs.model.ListQueuesResult;
```

Code

```
AmazonSQS sqs = AmazonSQSClientBuilder.defaultClient();
ListQueuesResult lq_result = sqs.listQueues();
System.out.println("Your SQS Queue URLs:");
for (String url : lq_result.getQueueUrls()) {
    System.out.println(url);
}
```

Using the `listQueues` overload without any parameters returns *all queues*. You can filter the returned results by passing it a `ListQueuesRequest` object.

Imports

```
import com.amazonaws.services.sqs.AmazonSQS;
import com.amazonaws.services.sqs.AmazonSQSClientBuilder;
import com.amazonaws.services.sqs.model.ListQueuesRequest;
```

Code

```
AmazonSQS sqs = AmazonSQSClientBuilder.defaultClient();
String name_prefix = "Queue";
lq_result = sqs.listQueues(new ListQueuesRequest(name_prefix));
System.out.println("Queue URLs with prefix: " + name_prefix);
for (String url : lq_result.getQueueUrls()) {
    System.out.println(url);
}
```

See the complete example on GitHub.

Get the URL for a Queue

Call the AmazonSQS client's `getQueueUrl` method.

Imports

```
import com.amazonaws.services.sqs.AmazonSQS;
import com.amazonaws.services.sqs.AmazonSQSClientBuilder;
```

Code

```
AmazonSQS sqs = AmazonSQSClientBuilder.defaultClient();
```

```
String queue_url = sqs.getQueueUrl(QUEUE_NAME).getQueueUrl();
```

See the complete example on GitHub.

Delete a Queue

Provide the queue's URL to the AmazonSQS client's `deleteQueue` method.

Imports

```
import com.amazonaws.services.sqs.AmazonSQS;
import com.amazonaws.services.sqs.AmazonSQSClientBuilder;
```

Code

```
AmazonSQS sqs = AmazonSQSClientBuilder.defaultClient();
sqs.deleteQueue(queue_url);
```

See the complete example 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

This topic describes how to send, receive and delete Amazon SQS messages. Messages are always delivered using an SQS Queue.

Send a Message

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

Imports

```
import com.amazonaws.services.sqs.AmazonSQS;
import com.amazonaws.services.sqs.AmazonSQSClientBuilder;
import com.amazonaws.services.sqs.model.SendMessageRequest;
```

Code

```
SendMessageRequest send_msg_request = new SendMessageRequest()
        .withQueueUrl(queueUrl)
        .withMessageBody("hello world")
        .withDelaySeconds(5);
sqs.sendMessage(send_msg_request);
```

See the complete example on GitHub.

Send Multiple Messages at Once

You can send more than one message in a single request. To send multiple messages, use the AmazonSQS client's `sendMessageBatch` method, which takes a SendMessageBatchRequest containing the queue URL and a list of messages (each one a SendMessageBatchRequestEntry) to send. You can also set an optional delay value per message.

Imports

```
import com.amazonaws.services.sqs.model.SendMessageBatchRequest;
import com.amazonaws.services.sqs.model.SendMessageBatchRequestEntry;
```

Code

```
SendMessageBatchRequest send_batch_request = new SendMessageBatchRequest()
        .withQueueUrl(queueUrl)
        .withEntries(
                new SendMessageBatchRequestEntry(
                        "msg_1", "Hello from message 1"),
                new SendMessageBatchRequestEntry(
                        "msg_2", "Hello from message 2")
                        .withDelaySeconds(10));
sqs.sendMessageBatch(send_batch_request);
```

See the complete example on GitHub.

Receive Messages

Retrieve any messages that are currently in the queue by calling the AmazonSQS client's `receiveMessage` method, passing it the queue's URL. Messages are returned as a list of Message objects.

Imports

```
import com.amazonaws.services.sqs.AmazonSQSClientBuilder;
import com.amazonaws.services.sqs.model.AmazonSQSException;
import com.amazonaws.services.sqs.model.SendMessageBatchRequest;
```

Code

```
List<Message> messages = sqs.receiveMessage(queueUrl).getMessages();
```

Delete Messages 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 AmazonSQS client's `deleteMessage` method.

Code

```
for (Message m : messages) {
    sqs.deleteMessage(queueUrl, m.getReceiptHandle());
}
```

See the complete example 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

Enabling Long Polling for Amazon SQS Message Queues

Amazon SQS uses *short polling* by default, querying only a subset of the servers—based on a weighted random distribution—to determine whether any messages are available for inclusion in the response.

Long polling helps reduce your cost of using Amazon SQS by reducing the number of empty responses when there are no messages available to return in reply to a ReceiveMessage request sent to an Amazon SQS queue and eliminating false empty responses.

> **Note**
>
> You can set a long polling frequency from *1-20 seconds*.

Enabling Long Polling when Creating a Queue

To enable long polling when creating an Amazon SQS queue, set the `ReceiveMessageWaitTimeSeconds` attribute on the CreateQueueRequest object before calling the AmazonSQS class' `createQueue` method.

Imports

```
import com.amazonaws.services.sqs.AmazonSQS;
import com.amazonaws.services.sqs.AmazonSQSClientBuilder;
import com.amazonaws.services.sqs.model.AmazonSQSException;
import com.amazonaws.services.sqs.model.CreateQueueRequest;
```

Code

```
final AmazonSQS sqs = AmazonSQSClientBuilder.defaultClient();

// Enable long polling when creating a queue
CreateQueueRequest create_request = new CreateQueueRequest()
        .withQueueName(queue_name)
        .addAttributesEntry("ReceiveMessageWaitTimeSeconds", "20");

try {
    sqs.createQueue(create_request);
} catch (AmazonSQSException e) {
```

```
    if (!e.getErrorCode().equals("QueueAlreadyExists")) {
        throw e;
    }
}
```

See the complete example on GitHub.

Enabling Long Polling on an Existing Queue

In addition to enabling long polling when creating a queue, you can also enable it on an existing queue by setting `ReceiveMessageWaitTimeSeconds` on the SetQueueAttributesRequest before calling the AmazonSQS class' `setQueueAttributes` method.

Imports

```
import com.amazonaws.services.sqs.model.SetQueueAttributesRequest;
```

Code

```
SetQueueAttributesRequest set_attrs_request = new SetQueueAttributesRequest()
        .withQueueUrl(queue_url)
        .addAttributesEntry("ReceiveMessageWaitTimeSeconds", "20");
sqs.setQueueAttributes(set_attrs_request);
```

See the complete example on GitHub.

Enabling Long Polling on Message Receipt

You can enable long polling when receiving a message by setting the wait time in seconds on the ReceiveMessageRequest that you supply to the AmazonSQS class' `receiveMessage` method.

> **Note**
>
> You should make sure that the AWS client's request timeout is larger than the maximum long poll time (20s) so that your `receiveMessage` requests don't time out while waiting for the next poll event!

Imports

```
import com.amazonaws.services.sqs.model.ReceiveMessageRequest;
```

Code

```
ReceiveMessageRequest receive_request = new ReceiveMessageRequest()
        .withQueueUrl(queue_url)
        .withWaitTimeSeconds(20);
sqs.receiveMessage(receive_request);
```

See the complete example on GitHub.

More Info

- Amazon SQS Long Polling in the Amazon SQS Developer Guide
- CreateQueue in the Amazon SQS API Reference
- ReceiveMessage in the Amazon SQS API Reference
- SetQueueAttributes in the Amazon SQS API Reference

Setting Visibility Timeout in Amazon SQS

When a message is received in Amazon SQS, it remains on the queue until it's deleted in order to ensure receipt. A message that was received, but not deleted, will be available in subsequent requests after a given *visibility timeout* to help prevent the message from being received more than once before it can be processed and deleted.

> **Note**
>
> When using standard queues, visibility timeout isn't a guarantee against receiving a message twice. If you are using a standard queue, be sure that your code can handle the case where the same message has been delivered more than once.

Setting the Message Visibility Timeout for a Single Message

When you have received a message, you can modify its visibility timeout by passing its receipt handle in a ChangeMessageVisibilityRequest that you pass to the AmazonSQS class' `changeMessageVisibility` method.

Imports

```
import com.amazonaws.services.sqs.AmazonSQS;
import com.amazonaws.services.sqs.AmazonSQSClientBuilder;
```

Code

```
AmazonSQS sqs = AmazonSQSClientBuilder.defaultClient();

// Get the receipt handle for the first message in the queue.
String receipt = sqs.receiveMessage(queue_url)
                    .getMessages()
                    .get(0)
                    .getReceiptHandle();

sqs.changeMessageVisibility(queue_url, receipt, timeout);
```

See the complete example on GitHub.

Setting the Message Visibility Timeout for Multiple Messages at Once

To set the message visibility timeout for multiple messages at once, create a list of ChangeMessageVisibilityBatchRequestEntry objects, each containing a unique ID string and a receipt handle. Then, pass the list to the Amazon SQS client class' `changeMessageVisibilityBatch` method.

Imports

```
import com.amazonaws.services.sqs.AmazonSQS;
import com.amazonaws.services.sqs.AmazonSQSClientBuilder;
import com.amazonaws.services.sqs.model.ChangeMessageVisibilityBatchRequestEntry;
import java.util.ArrayList;
import java.util.List;
```

Code

```
AmazonSQS sqs = AmazonSQSClientBuilder.defaultClient();

List<ChangeMessageVisibilityBatchRequestEntry> entries =
    new ArrayList<ChangeMessageVisibilityBatchRequestEntry>();
```

```
entries.add(new ChangeMessageVisibilityBatchRequestEntry(
            "unique_id_msg1",
            sqs.receiveMessage(queue_url)
               .getMessages()
               .get(0)
               .getReceiptHandle())
        .withVisibilityTimeout(timeout));

entries.add(new ChangeMessageVisibilityBatchRequestEntry(
            "unique_id_msg2",
            sqs.receiveMessage(queue_url)
               .getMessages()
               .get(0)
               .getReceiptHandle())
        .withVisibilityTimeout(timeout + 200));

sqs.changeMessageVisibilityBatch(queue_url, entries);
```

See the complete example on GitHub.

More Info

- Visibility Timeout in the Amazon SQS Developer Guide
- SetQueueAttributes in the Amazon SQS API Reference
- GetQueueAttributes in the Amazon SQS API Reference
- ReceiveMessage in the Amazon SQS API Reference
- ChangeMessageVisibility in the Amazon SQS API Reference
- ChangeMessageVisibilityBatch in the Amazon SQS API Reference

Using Dead Letter Queues in Amazon SQS

Amazon SQS provides support for *dead letter queues*. A dead letter queue is a queue that other (source) queues can target for messages that can't be processed successfully. You can set aside and isolate these messages in the dead letter queue to determine why their processing did not succeed.

Creating a Dead Letter Queue

A dead letter queue is created the same way as a regular queue, but it has the following restrictions:

- A dead letter queue must be the same type of queue (FIFO or standard) as the source queue.
- A dead letter queue must be created using the same AWS account and region as the source queue.

Here we create two identical Amazon SQS queues, one of which will serve as the dead letter queue:

Imports

```
import com.amazonaws.services.sqs.AmazonSQS;
import com.amazonaws.services.sqs.AmazonSQSClientBuilder;
import com.amazonaws.services.sqs.model.AmazonSQSException;
```

Code

```
final AmazonSQS sqs = AmazonSQSClientBuilder.defaultClient();

// Create source queue
try {
    sqs.createQueue(src_queue_name);
} catch (AmazonSQSException e) {
    if (!e.getErrorCode().equals("QueueAlreadyExists")) {
        throw e;
    }
}

// Create dead-letter queue
try {
    sqs.createQueue(dl_queue_name);
} catch (AmazonSQSException e) {
    if (!e.getErrorCode().equals("QueueAlreadyExists")) {
        throw e;
    }
}
```

See the complete example on GitHub.

Designating a Dead Letter Queue for a Source Queue

To designate a dead letter queue, you must first create a *redrive policy*, and then set the policy in the queue's attributes. A redrive policy is specified in JSON, and specifies the ARN of the dead

letter queue and the maximum number of times the message can be received and not processed before it's sent to the dead letter queue.

To set the redrive policy for your source queue, call the AmazonSQS class' `setQueueAttributes` method with a SetQueueAttributesRequest object for which you've set the `RedrivePolicy` attribute with your JSON redrive policy.

Imports

```
import com.amazonaws.services.sqs.model.GetQueueAttributesRequest;
import com.amazonaws.services.sqs.model.GetQueueAttributesResult;
import com.amazonaws.services.sqs.model.SetQueueAttributesRequest;
```

Code

```
String dl_queue_url = sqs.getQueueUrl(dl_queue_name)
                         .getQueueUrl();

GetQueueAttributesResult queue_attrs = sqs.getQueueAttributes(
        new GetQueueAttributesRequest(dl_queue_url)
            .withAttributeNames("QueueArn"));

String dl_queue_arn = queue_attrs.getAttributes().get("QueueArn");

// Set dead letter queue with redrive policy on source queue.
String src_queue_url = sqs.getQueueUrl(src_queue_name)
                          .getQueueUrl();

SetQueueAttributesRequest request = new SetQueueAttributesRequest()
        .withQueueUrl(src_queue_url)
        .addAttributesEntry("RedrivePolicy",
                "{\"maxReceiveCount\":\"5\", \"deadLetterTargetArn\":\""
                + dl_queue_arn + "\"}");

sqs.setQueueAttributes(request);
```

See the complete example on GitHub.

More Info

- Using Amazon SQS Dead Letter Queues in the Amazon SQS Developer Guide

- SetQueueAttributes in the Amazon SQS API Reference

Amazon SWF Examples Using the AWS SDK for Java

Amazon SWF is a workflow-management service that helps developers build and scale distributed workflows that can have parallel or sequential steps consisting of activities, child workflows or even Lambda tasks.

There are two ways to work with Amazon SWF using the AWS SDK for Java, by using the SWF *client* object, or by using the AWS Flow Framework for Java. The AWS Flow Framework for Java is more difficult to configure initially, since it makes heavy use of annotations and relies on additional libraries such as AspectJ and the Spring Framework. However, for large or complex projects, you will save coding time by using the AWS Flow Framework for Java. For more information, see the AWS Flow Framework for Java Developer Guide.

This section provides examples of programming Amazon SWF by using the AWS SDK for Java client directly.

Topics

SWF basics

These are general patterns for working with Amazon SWF using the AWS SDK for Java. It is meant primarily for reference. For a more complete introductory tutorial, see Building a Simple Amazon SWF Application.

Dependencies

Basic Amazon SWF applications will require the following dependencies, which are included with the AWS SDK for Java:

- aws-java-sdk-1.12.*.jar
- commons-logging-1.2.*.jar
- httpclient-4.3.*.jar
- httpcore-4.3.*.jar
- jackson-annotations-2.12.*.jar
- jackson-core-2.12.*.jar
- jackson-databind-2.12.*.jar
- joda-time-2.8.*.jar

> **Note**
>
> The version numbers of these packages will differ depending on the version of the SDK that you have, but the versions that are supplied with the SDK have been tested for compatibility, and are the ones you should use.

AWS Flow Framework for Java applications require additional setup, *and* additional dependencies. See the AWS Flow Framework for Java Developer Guide for more information about using the framework.

Imports

In general, you can use the following imports for code development:

```
import com.amazonaws.services.simpleworkflow.AmazonSimpleWorkflowClientBuilder;
import com.amazonaws.services.simpleworkflow.model.*;
```

It's a good practice to import only the classes you require, however. You will likely end up specifying particular classes in the `com.amazonaws.services.simpleworkflow.model` workspace:

```
import com.amazonaws.services.simpleworkflow.model.PollForActivityTaskRequest;
import com.amazonaws.services.simpleworkflow.model.RespondActivityTaskCompletedRequest;
import com.amazonaws.services.simpleworkflow.model.RespondActivityTaskFailedRequest;
import com.amazonaws.services.simpleworkflow.model.TaskList;
```

If you are using the AWS Flow Framework for Java, you will import classes from the `com.amazonaws.services.simpleworkflow.flow` workspace. For example:

```
import com.amazonaws.services.simpleworkflow.AmazonSimpleWorkflow;
import com.amazonaws.services.simpleworkflow.flow.ActivityWorker;
```

> **Note**
>
> The AWS Flow Framework for Java has additional requirements beyond those of the base AWS SDK for Java. For more information, see the AWS Flow Framework for Java Developer Guide.

Using the SWF client class

Your basic interface to Amazon SWF is through either the AmazonSimpleWorkflowClient or AmazonSimpleWorkflowAsyncClient classes. The main difference between these is that the `\*AsyncClient` class return Future objects for concurrent (asynchronous) programming.

```
AmazonSimpleWorkflowClient swf = AmazonSimpleWorkflowClientBuilder.defaultClient();
```

Building a Simple Amazon SWF Application

This topic will introduce you to programming Amazon SWF applications with the AWS SDK for Java, while presenting a few important concepts along the way.

About the example

The example project will create a workflow with a single activity that accepts workflow data passed through the AWS cloud (In the tradition of HelloWorld, it'll be the name of someone to greet) and then prints a greeting in response.

While this seems very simple on the surface, Amazon SWF applications consist of a number of parts working together:

- A **domain**, used as a logical container for your workflow execution data.
- One or more **workflows** which represent code components that define logical order of execution of your workflow's activities and child workflows.

- A **workflow worker**, also known as a *decider*, that polls for decision tasks and schedules activities or child workflows in response.
- One or more **activities**, each of which represents a unit of work in the workflow.
- An **activity worker** that polls for activity tasks and runs activity methods in response.
- One or more **task lists**, which are queues maintained by Amazon SWF used to issue requests to the workflow and activity workers. Tasks on a task list meant for workflow workers are called *decision tasks*. Those meant for activity workers are called *activity tasks*.
- A **workflow starter** that begins your workflow execution.

Behind the scenes, Amazon SWF orchestrates the operation of these components, coordinating their flow from the AWS cloud, passing data between them, handling timeouts and heartbeat notifications, and logging workflow execution history.

Prerequisites

Development environment

The development environment used in this tutorial consists of:

- The AWS SDK for Java.
- Apache Maven (3.3.1).
- JDK 1.7 or later. This tutorial was developed and tested using JDK 1.8.0.
- A good Java text editor (your choice).

> **Note**
>
> If you use a different build system than Maven, you can still create a project using the appropriate steps for your environment and use the concepts provided here to follow along. More information about configuring and using the AWS SDK for Java with various build systems is provided in Getting Started.
> Likewise, but with more effort, the steps shown here can be implemented using any of the AWS SDKs with support for Amazon SWF.

All of the necessary external dependencies are included with the AWS SDK for Java, so there's nothing additional to download.

AWS Access

To successfully work through this tutorial, you must have access to the AWS access portal as described in the basic setup section of this guide.

The instructions describe how to access temporary credentials that you copy and paste to your local shared `credentials` file. The temporary credentials that you paste must be associated with an IAM role in AWS IAM Identity Center that has permissions to access Amazon SWF. After pasting the temporary credentials, your `credentials` file will look similar to the following.

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

These temporary credentials are associated with the `default` profile.

Create a SWF project

1. Start a new project with Maven:

   ```
   mvn archetype:generate -DartifactId=helloswf \
   -DgroupId=aws.example.helloswf -DinteractiveMode=false
   ```

   This will create a new project with a standard maven project structure:

   ```
   helloswf
   ### pom.xml
   ### src
       ### main
       #   ### java
       #       ### aws
       #           ### example
       #               ### helloswf
       #                   ### App.java
       ### test
           ### ...
   ```

   You can ignore or delete the `test` directory and all it contains, we won't be using it for this tutorial. You can also delete `App.java`, since we'll be replacing it with new classes.

2. Edit the project's `pom.xml` file and add the **aws-java-sdk-simpleworkflow** module by adding a dependency for it within the `<dependencies>` block.

```
<dependencies>
  <dependency>
    <groupId>com.amazonaws</groupId>
    <artifactId>aws-java-sdk-simpleworkflow</artifactId>
    <version>1.11.1000</version>
  </dependency>
</dependencies>
```

3. *Make sure that Maven builds your project with JDK 1.7+ support*. Add the following to your project (either before or after the `<dependencies>` block) in `pom.xml`:

```
<build>
  <plugins>
    <plugin>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>3.6.1</version>
      <configuration>
          <source>1.8</source>
          <target>1.8</target>
      </configuration>
    </plugin>
  </plugins>
</build>
```

Code the project

The example project will consist of four separate applications, which we'll visit one by one:

- **HelloTypes.java**--contains the project's domain, activity and workflow type data, shared with the other components. It also handles registering these types with SWF.
- **ActivityWorker.java**--contains the activity worker, which polls for activity tasks and runs activities in response.
- **WorkflowWorker.java**--contains the workflow worker (decider), which polls for decision tasks and schedules new activities.

- **WorkflowStarter.java**--contains the workflow starter, which starts a new workflow execution, which will cause SWF to start generating decision and workflow tasks for your workers to consume.

Common steps for all source files

All of the files that you create to house your Java classes will have a few things in common. In the interest of time, these steps *will be implied every time you add a new file to the project*:

1. Create the file in the in the project's `src/main/java/aws/example/helloswf/` directory.
2. Add a `package` declaration to the beginning of each file to declare its namespace. The example project uses:

```
package aws.example.helloswf;
```

3. Add `import` declarations for the AmazonSimpleWorkflowClient class and for multiple classes in the `com.amazonaws.services.simpleworkflow.model` namespace. To simplify things, we'll use:

```
import com.amazonaws.regions.Regions;
import com.amazonaws.services.simpleworkflow.AmazonSimpleWorkflow;
import com.amazonaws.services.simpleworkflow.AmazonSimpleWorkflowClientBuilder;
import com.amazonaws.services.simpleworkflow.model.*;
```

Register a domain, workflow and activity types

We'll begin by creating a new executable class, `HelloTypes.java`. This file will contain shared data that different parts of your workflow will need to know about, such as the name and version of your activity and workflow types, the domain name and the task list name.

1. Open your text editor and create the file `HelloTypes.java`, adding a package declaration and imports according to the common steps.
2. Declare the `HelloTypes` class and provide it with values to use for your registered activity and workflow types:

```
    public static final String DOMAIN = "HelloDomain";
    public static final String TASKLIST = "HelloTasklist";
    public static final String WORKFLOW = "HelloWorkflow";
```

```
    public static final String WORKFLOW_VERSION = "1.0";
    public static final String ACTIVITY = "HelloActivity";
    public static final String ACTIVITY_VERSION = "1.0";
```

These values will be used throughout the code.

3. After the String declarations, create an instance of the AmazonSimpleWorkflowClient class. This is the basic interface to the Amazon SWF methods provided by the AWS SDK for Java.

```
private static final AmazonSimpleWorkflow swf =

 AmazonSimpleWorkflowClientBuilder.standard().withRegion(Regions.DEFAULT_REGION).build();
```

The previous snippet assumes that temporary credentials are associated with the `default` profile. If you use a different profile, modify the code above as follows and replace *`profile_name`* with the name of actual profile name.

```
private static final AmazonSimpleWorkflow swf =
        AmazonSimpleWorkflowClientBuilder
                .standard()
                .withCredentials(new ProfileCredentialsProvider("profile_name"))
                .withRegion(Regions.DEFAULT_REGION)
                .build();
```

4. Add a new function to register a SWF domain. A *domain* is a logical container for a number of related SWF activity and workflow types. SWF components can only communicate with each other if they exist within the same domain.

```
    try {
        System.out.println("** Registering the domain '" + DOMAIN + "'.");
        swf.registerDomain(new RegisterDomainRequest()
            .withName(DOMAIN)
            .withWorkflowExecutionRetentionPeriodInDays("1"));
    } catch (DomainAlreadyExistsException e) {
        System.out.println("** Domain already exists!");
    }
```

When you register a domain, you provide it with a *name* (any set of 1 - 256 characters excluding `:`, `/`, `|`, control characters or the literal string '`arn') and a *retention period*, which is the number of days that Amazon SWF will keep your workflow's execution history data after a workflow

execution has completed. The maximum workflow execution retention period is 90 days. See RegisterDomainRequest for more information.

If a domain with that name already exists, a DomainAlreadyExistsException is raised. Because we're unconcerned if the domain has already been created, we can ignore the exception.

> **Note**
>
> This code demonstrates a common pattern when working with AWS SDK for Java methods, data for the method is supplied by a class in the `simpleworkflow.model` namespace, which you instantiate and populate using the chainable `0with*` methods.

5. Add a function to register a new activity type. An *activity* represents a unit of work in your workflow.

```
try {
    System.out.println("** Registering the activity type '" + ACTIVITY +
        "-" + ACTIVITY_VERSION + "'.");
    swf.registerActivityType(new RegisterActivityTypeRequest()
        .withDomain(DOMAIN)
        .withName(ACTIVITY)
        .withVersion(ACTIVITY_VERSION)
        .withDefaultTaskList(new TaskList().withName(TASKLIST))
        .withDefaultTaskScheduleToStartTimeout("30")
        .withDefaultTaskStartToCloseTimeout("600")
        .withDefaultTaskScheduleToCloseTimeout("630")
        .withDefaultTaskHeartbeatTimeout("10"));
} catch (TypeAlreadyExistsException e) {
    System.out.println("** Activity type already exists!");
}
```

An activity type is identified by a *name* and a *version*, which are used to uniquely identify the activity from any others in the domain that it's registered in. Activities also contain a number of optional parameters, such as the default task-list used to receive tasks and data from SWF and a number of different timeouts that you can use to place constraints upon how long different parts of the activity execution can take. See RegisterActivityTypeRequest for more information.

> **Note**
>
> All timeout values are specified in *seconds*. See Amazon SWF Timeout Types for a full description of how timeouts affect your workflow executions.

If the activity type that you're trying to register already exists, an TypeAlreadyExistsException is raised. . Add a function to register a new workflow type. A *workflow*, also known as a *decider* represents the logic of your workflow's execution.

+

```
    try {
        System.out.println("** Registering the workflow type '" + WORKFLOW +
            "-" + WORKFLOW_VERSION + "'.");
        swf.registerWorkflowType(new RegisterWorkflowTypeRequest()
            .withDomain(DOMAIN)
            .withName(WORKFLOW)
            .withVersion(WORKFLOW_VERSION)
            .withDefaultChildPolicy(ChildPolicy.TERMINATE)
            .withDefaultTaskList(new TaskList().withName(TASKLIST))
            .withDefaultTaskStartToCloseTimeout("30"));
    } catch (TypeAlreadyExistsException e) {
        System.out.println("** Workflow type already exists!");
    }
```

+

Similar to activity types, workflow types are identified by a *name* and a *version* and also have configurable timeouts. See RegisterWorkflowTypeRequest for more information.

+

If the workflow type that you're trying to register already exists, an TypeAlreadyExistsException is raised. . Finally, make the class executable by providing it a `main` method, which will register the domain, the activity type, and the workflow type in turn:

+

```
    registerDomain();
```

```
    registerWorkflowType();
    registerActivityType();
```

You can build and run the application now to run the registration script, or continue with coding the activity and workflow workers. Once the domain, workflow and activity have been registered, you won't need to run this again—these types persist until you deprecate them yourself.

Implement the activity worker

An *activity* is the basic unit of work in a workflow. A workflow provides the logic, scheduling activities to be run (or other actions to be taken) in response to decision tasks. A typical workflow usually consists of a number of activities that can run synchronously, asynchronously, or a combination of both.

The *activity worker* is the bit of code that polls for activity tasks that are generated by Amazon SWF in response to workflow decisions. When it receives an activity task, it runs the corresponding activity and returns a success/failure response back to the workflow.

We'll implement a simple activity worker that drives a single activity.

1. Open your text editor and create the file `ActivityWorker.java`, adding a package declaration and imports according to the common steps.

   ```
   import com.amazonaws.regions.Regions;
   import com.amazonaws.services.simpleworkflow.AmazonSimpleWorkflow;
   import com.amazonaws.services.simpleworkflow.AmazonSimpleWorkflowClientBuilder;
   import com.amazonaws.services.simpleworkflow.model.*;
   ```

2. Add the `ActivityWorker` class to the file, and give it a data member to hold a SWF client that we'll use to interact with Amazon SWF:

   ```
       private static final AmazonSimpleWorkflow swf =

    AmazonSimpleWorkflowClientBuilder.standard().withRegion(Regions.DEFAULT_REGION).build();
   ```

3. Add the method that we'll use as an activity:

   ```
   private static String sayHello(String input) throws Throwable {
       return "Hello, " + input + "!";
   }
   ```

The activity simply takes a string, combines it into a greeting and returns the result. Although there is little chance that this activity will raise an exception, it's a good idea to design activities that can raise an error if something goes wrong.

4. Add a `main` method that we'll use as the activity task polling method. We'll start it by adding some code to poll the task list for activity tasks:

```
        System.out.println("Polling for an activity task from the tasklist '"
                + HelloTypes.TASKLIST + "' in the domain '" +
                HelloTypes.DOMAIN + "'.");

        ActivityTask task = swf.pollForActivityTask(
            new PollForActivityTaskRequest()
                .withDomain(HelloTypes.DOMAIN)
                .withTaskList(
                    new TaskList().withName(HelloTypes.TASKLIST)));

        String task_token = task.getTaskToken();
```

The activity receives tasks from Amazon SWF by calling the SWF client's `pollForActivityTask` method, specifying the domain and task list to use in the passed-in PollForActivityTaskRequest.

Once a task is received, we retrieve a unique identifier for it by calling the task's `getTaskToken` method.

5. Next, write some code to process the tasks that come in. Add the following to your `main` method, right after the code that polls for the task and retrieves its task token.

```
    if (task_token != null) {
        String result = null;
        Throwable error = null;

        try {
            System.out.println("Executing the activity task with input '" +
                    task.getInput() + "'.");
            result = sayHello(task.getInput());
        } catch (Throwable th) {
            error = th;
        }

        if (error == null) {
```

```
            System.out.println("The activity task succeeded with result '"
                    + result + "'.");
            swf.respondActivityTaskCompleted(
                new RespondActivityTaskCompletedRequest()
                    .withTaskToken(task_token)
                    .withResult(result));
        } else {
            System.out.println("The activity task failed with the error '"
                    + error.getClass().getSimpleName() + "'.");
            swf.respondActivityTaskFailed(
                new RespondActivityTaskFailedRequest()
                    .withTaskToken(task_token)
                    .withReason(error.getClass().getSimpleName())
                    .withDetails(error.getMessage()));
        }
    }
```

If the task token is not `null`, then we can start running the activity method (`sayHello`), providing it with the input data that was sent with the task.

If the task *succeeded* (no error was generated), then the worker responds to SWF by calling the SWF client's `respondActivityTaskCompleted` method with a RespondActivityTaskCompletedRequest object containing the task token and the activity's result data.

On the other hand, if the task *failed*, then we respond by calling the `respondActivityTaskFailed` method with a RespondActivityTaskFailedRequest object, passing it the task token and information about the error.

> **Note**
>
> This activity will not shut down gracefully if killed. Although it is beyond the scope of this tutorial, an alternative implementation of this activity worker is provided in the accompanying topic, Shutting Down Activity and Workflow Workers Gracefully.

Implement the workflow worker

Your workflow logic resides in a piece of code known as a **workflow worker**. The workflow worker polls for decision tasks that are sent by Amazon SWF in the domain, and on the default tasklist, that the workflow type was registered with.

When the workflow worker receives a task, it makes some sort of decision (usually whether to schedule a new activity or not) and takes an appropriate action (such as scheduling the activity).

1. Open your text editor and create the file `WorkflowWorker.java`, adding a package declaration and imports according to the common steps.
2. Add a few additional imports to the file:

```
import com.amazonaws.regions.Regions;
import com.amazonaws.services.simpleworkflow.AmazonSimpleWorkflow;
import com.amazonaws.services.simpleworkflow.AmazonSimpleWorkflowClientBuilder;
import com.amazonaws.services.simpleworkflow.model.*;
import java.util.ArrayList;
import java.util.List;
import java.util.UUID;
```

3. Declare the `WorkflowWorker` class, and create an instance of the AmazonSimpleWorkflowClient class used to access SWF methods.

```
    private static final AmazonSimpleWorkflow swf =

AmazonSimpleWorkflowClientBuilder.standard().withRegion(Regions.DEFAULT_REGION).build();
```

4. Add the `main` method. The method loops continuously, polling for decision tasks using the SWF client's `pollForDecisionTask` method. The PollForDecisionTaskRequest provides the details.

```
    PollForDecisionTaskRequest task_request =
        new PollForDecisionTaskRequest()
            .withDomain(HelloTypes.DOMAIN)
            .withTaskList(new TaskList().withName(HelloTypes.TASKLIST));

    while (true) {
        System.out.println(
                "Polling for a decision task from the tasklist '" +
                HelloTypes.TASKLIST + "' in the domain '" +
                HelloTypes.DOMAIN + "'.");
```

```
        DecisionTask task = swf.pollForDecisionTask(task_request);

        String taskToken = task.getTaskToken();
        if (taskToken != null) {
            try {
                executeDecisionTask(taskToken, task.getEvents());
            } catch (Throwable th) {
                th.printStackTrace();
            }
        }
    }
```

Once a task is received, we call its `getTaskToken` method, which returns a string that can be used to identify the task. If the returned token is not `null`, then we process it further in the `executeDecisionTask` method, passing it the task token and the list of HistoryEvent objects sent with the task.

5. Add the `executeDecisionTask` method, taking the task token (a `String`) and the `HistoryEvent` list.

```
    List<Decision> decisions = new ArrayList<Decision>();
    String workflow_input = null;
    int scheduled_activities = 0;
    int open_activities = 0;
    boolean activity_completed = false;
    String result = null;
```

We also set up some data members to keep track of things such as:

- A list of Decision objects used to report the results of processing the task.
- A String to hold workflow input provided by the "WorkflowExecutionStarted" event
- a count of the scheduled and open (running) activities to avoid scheduling the same activity when it has already been scheduled or is currently running.
- a boolean to indicate that the activity has completed.
- A String to hold the activity results, for returning it as our workflow result.

6. Next, add some code to `executeDecisionTask` to process the `HistoryEvent` objects that were sent with the task, based on the event type reported by the `getEventType` method.

```
System.out.println("Executing the decision task for the history events: [");
for (HistoryEvent event : events) {
```

```
    System.out.println("  " + event);
    switch(event.getEventType()) {
        case "WorkflowExecutionStarted":
            workflow_input =
                event.getWorkflowExecutionStartedEventAttributes()
                     .getInput();
            break;
        case "ActivityTaskScheduled":
            scheduled_activities++;
            break;
        case "ScheduleActivityTaskFailed":
            scheduled_activities--;
            break;
        case "ActivityTaskStarted":
            scheduled_activities--;
            open_activities++;
            break;
        case "ActivityTaskCompleted":
            open_activities--;
            activity_completed = true;
            result = event.getActivityTaskCompletedEventAttributes()
                          .getResult();
            break;
        case "ActivityTaskFailed":
            open_activities--;
            break;
        case "ActivityTaskTimedOut":
            open_activities--;
            break;
    }
}
System.out.println("]");
```

For the purposes of our workflow, we are most interested in:

- the "WorkflowExecutionStarted" event, which indicates that the workflow execution has started (typically meaning that you should run the first activity in the workflow), and that provides the initial input provided to the workflow. In this case, it's the name portion of our greeting, so it's saved in a String for use when scheduling the activity to run.
- the "ActivityTaskCompleted" event, which is sent once the scheduled activity is complete. The event data also includes the return value of the completed activity. Since we have only one activity, we'll use that value as the result of the entire workflow.

The other event types can be used if your workflow requires them. See the HistoryEvent class description for information about each event type.

+ NOTE: Strings in `switch` statements were introduced in Java 7. If you're using an earlier version of Java, you can make use of the EventType class to convert the `String` returned by `history_event.getType()` to an enum value and then back to a `String` if necessary:

```
EventType et = EventType.fromValue(event.getEventType());
```

1. After the `switch` statement, add more code to respond with an appropriate *decision* based on the task that was received.

```
if (activity_completed) {
    decisions.add(
        new Decision()
            .withDecisionType(DecisionType.CompleteWorkflowExecution)
            .withCompleteWorkflowExecutionDecisionAttributes(
                new CompleteWorkflowExecutionDecisionAttributes()
                    .withResult(result)));
} else {
    if (open_activities == 0 && scheduled_activities == 0) {

        ScheduleActivityTaskDecisionAttributes attrs =
            new ScheduleActivityTaskDecisionAttributes()
                .withActivityType(new ActivityType()
                    .withName(HelloTypes.ACTIVITY)
                    .withVersion(HelloTypes.ACTIVITY_VERSION))
                .withActivityId(UUID.randomUUID().toString())
                .withInput(workflow_input);

        decisions.add(
                new Decision()
                    .withDecisionType(DecisionType.ScheduleActivityTask)
                    .withScheduleActivityTaskDecisionAttributes(attrs));
    } else {
        // an instance of HelloActivity is already scheduled or running. Do nothing,
 another
        // task will be scheduled once the activity completes, fails or times out
    }
}
```

```
System.out.println("Exiting the decision task with the decisions " + decisions);
```

- If the activity hasn't been scheduled yet, we respond with a `ScheduleActivityTask` decision, which provides information in a ScheduleActivityTaskDecisionAttributes structure about the activity that Amazon SWF should schedule next, also including any data that Amazon SWF should send to the activity.
- If the activity was completed, then we consider the entire workflow completed and respond with a `CompletedWorkflowExecution` decision, filling in a CompleteWorkflowExecutionDecisionAttributes structure to provide details about the completed workflow. In this case, we return the result of the activity.

In either case, the decision information is added to the `Decision` list that was declared at the top of the method.

2. Complete the decision task by returning the list of `Decision` objects collected while processing the task. Add this code at the end of the `executeDecisionTask` method that we've been writing:

```
swf.respondDecisionTaskCompleted(
    new RespondDecisionTaskCompletedRequest()
        .withTaskToken(taskToken)
        .withDecisions(decisions));
```

The SWF client's `respondDecisionTaskCompleted` method takes the task token that identifies the task as well as the list of `Decision` objects.

Implement the workflow starter

Finally, we'll write some code to start the workflow execution.

1. Open your text editor and create the file `WorkflowStarter.java`, adding a package declaration and imports according to the common steps.
2. Add the `WorkflowStarter` class:

```
package aws.example.helloswf;


import com.amazonaws.regions.Regions;
```

```
import com.amazonaws.services.simpleworkflow.AmazonSimpleWorkflow;
import com.amazonaws.services.simpleworkflow.AmazonSimpleWorkflowClientBuilder;
import com.amazonaws.services.simpleworkflow.model.*;

public class WorkflowStarter {
    private static final AmazonSimpleWorkflow swf =

 AmazonSimpleWorkflowClientBuilder.standard().withRegion(Regions.DEFAULT_REGION).build();
    public static final String WORKFLOW_EXECUTION = "HelloWorldWorkflowExecution";

    public static void main(String[] args) {
        String workflow_input = "{SWF}";
        if (args.length > 0) {
            workflow_input = args[0];
        }

        System.out.println("Starting the workflow execution '" + WORKFLOW_EXECUTION +
                "' with input '" + workflow_input + "'.");

        WorkflowType wf_type = new WorkflowType()
            .withName(HelloTypes.WORKFLOW)
            .withVersion(HelloTypes.WORKFLOW_VERSION);

        Run run = swf.startWorkflowExecution(new StartWorkflowExecutionRequest()
            .withDomain(HelloTypes.DOMAIN)
            .withWorkflowType(wf_type)
            .withWorkflowId(WORKFLOW_EXECUTION)
            .withInput(workflow_input)
            .withExecutionStartToCloseTimeout("90"));

        System.out.println("Workflow execution started with the run id '" +
                run.getRunId() + "'.");
    }
}
```

The `WorkflowStarter` class consists of a single method, `main`, which takes an optional argument passed on the command-line as input data for the workflow.

The SWF client method, `startWorkflowExecution`, takes a StartWorkflowExecutionRequest object as input. Here, in addition to specifying the domain and workflow type to run, we provide it with:

- a human-readable workflow execution name

- workflow input data (provided on the command-line in our example)
- a timeout value that represents how long, in seconds, that the entire workflow should take to run.

The Run object that `startWorkflowExecution` returns provides a *run ID*, a value that can be used to identify this particular workflow execution in Amazon SWF's history of your workflow executions.

+ NOTE: The run ID is generated by Amazon SWF, and is *not* the same as the workflow execution name that you pass in when starting the workflow execution.

Build the example

To build the example project with Maven, go to the `helloswf` directory and type:

```
mvn package
```

The resulting `helloswf-1.0.jar` will be generated in the `target` directory.

Run the example

The example consists of four separate executable classes, which are run independently of each other.

> **Note**
>
> If you are using a Linux, macOS, or Unix system, you can run all of them, one after another, in a single terminal window. If you are running Windows, you should open two additional command-line instances and navigate to the `helloswf` directory in each.

Setting the Java classpath

Although Maven has handled the dependencies for you, to run the example, you'll need to provide the AWS SDK library and its dependencies on your Java classpath. You can either set the `CLASSPATH` environment variable to the location of your AWS SDK libraries and the `third-party/lib` directory in the SDK, which includes necessary dependencies:

```
export CLASSPATH='target/helloswf-1.0.jar:/path/to/sdk/lib/*:/path/to/sdk/third-party/
lib/*'
java example.swf.hello.HelloTypes
```

or use the **`java`** command's `-cp` option to set the classpath while running each applications.

```
java -cp target/helloswf-1.0.jar:/path/to/sdk/lib/*:/path/to/sdk/third-party/lib/* \
  example.swf.hello.HelloTypes
```

The style that you use is up to you. If you had no trouble building the code, both then try to run the examples and get a series of "NoClassDefFound" errors, it is likely because the classpath is set incorrectly.

Register the domain, workflow and activity types

Before running your workers and the workflow starter, you'll need to register the domain and your workflow and activity types. The code to do this was implemented in Register a domain workflow and activity types.

After building, and if you've set the CLASSPATH, you can run the registration code by executing the command:

```
    echo 'Supply the name of one of the example classes as an argument.'
```

Start the activity and workflow workers

Now that the types have been registered, you can start the activity and workflow workers. These will continue to run and poll for tasks until they are killed, so you should either run them in separate terminal windows, or, if you're running on Linux, macOS, or Unix you can use the & operator to cause each of them to spawn a separate process when run.

```
    echo 'If there are arguments to the class, put them in quotes after the class
 name.'
    exit 1
```

If you're running these commands in separate windows, omit the final & operator from each line.

Start the workflow execution

Now that your activity and workflow workers are polling, you can start the workflow execution. This process will run until the workflow returns a completed status. You should run it in a new terminal window (unless you ran your workers as new spawned processes by using the & operator).

```
fi
```

> **Note**
>
> If you want to provide your own input data, which will be passed first to the workflow and then to the activity, add it to the command-line. For example:
>
> ```
> echo "## Running $className..."
> ```

Once you begin the workflow execution, you should start seeing output delivered by both workers and by the workflow execution itself. When the workflow finally completes, its output will be printed to the screen.

Complete source for this example

You can browse the complete source for this example on Github in the *aws-java-developer-guide* repository.

For more information

- The workers presented here can result in lost tasks if they are shutdown while a workflow poll is still going on. To find out how to shut down workers gracefully, see Shutting Down Activity and Workflow Workers Gracefully.
- To learn more about Amazon SWF, visit the Amazon SWF home page or view the Amazon SWF Developer Guide.
- You can use the AWS Flow Framework for Java to write more complex workflows in an elegant Java style using annotations. To learn more, see the AWS Flow Framework for Java Developer Guide.

Lambda Tasks

As an alternative to, or in conjunction with, Amazon SWF activities, you can use Lambda functions to represent units of work in your workflows, and schedule them similarly to activities.

This topic focuses on how to implement Amazon SWF Lambda tasks using the AWS SDK for Java. For more information about Lambda tasks in general, see AWS Lambda Tasks in the Amazon SWF Developer Guide.

Set up a cross-service IAM role to run your Lambda function

Before Amazon SWF can run your Lambda function, you need to set up an IAM role to give Amazon SWF permission to run Lambda functions on your behalf. For complete information about how to do this, see AWS Lambda Tasks.

You will need the Amazon Resource Name (ARN) of this IAM role when you register a workflow that will use Lambda tasks.

Create a Lambda function

You can write Lambda functions in a number of different languages, including Java. For complete information about how to author, deploy and use Lambda functions, see the AWS Lambda Developer Guide.

> **Note**
>
> It doesn't matter what language you use to write your Lambda function, it can be scheduled and run by *any* Amazon SWF workflow, regardless of the language that your workflow code is written in. Amazon SWF handles the details of running the function and passing data to and from it.

Here's a simple Lambda function that could be used in place of the activity in Building a Simple Amazon SWF Application.

- This version is written in JavaScript, which can be entered directly using the AWS Management Console:

```
exports.handler = function(event, context) {
```

```
    context.succeed("Hello, " + event.who + "!");
};
```

- Here is the same function written in Java, which you could also deploy and run on Lambda:

```
package example.swf.hellolambda;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.util.json.JSONException;
import com.amazonaws.util.json.JSONObject;

public class SwfHelloLambdaFunction implements RequestHandler<Object, Object> {
    @Override
    public Object handleRequest(Object input, Context context) {
        String who = "{SWF}";
        if (input != null) {
            JSONObject jso = null;
            try {
                jso = new JSONObject(input.toString());
                who = jso.getString("who");
            } catch (JSONException e) {
                e.printStackTrace();
            }
        }
        return ("Hello, " + who + "!");
    }
}
```

> **Note**
>
> To learn more about deploying Java functions to Lambda, see Creating a Deployment Package (Java) in the AWS Lambda Developer Guide. You will also want to look at the section titled Programming Model for Authoring Lambda Functions in Java.

Lambda functions take an *event* or *input* object as the first parameter, and a *context* object as the second, which provides information about the request to run the Lambda function. This particular function expects input to be in JSON, with a `who` field set to the name used to create the greeting.

Register a workflow for use with Lambda

For a workflow to schedule a Lambda function, you must provide the name of the IAM role that provides Amazon SWF with permission to invoke Lambda functions. You can set this during workflow registration by using the `withDefaultLambdaRole` or `setDefaultLambdaRole` methods of RegisterWorkflowTypeRequest.

```
System.out.println("** Registering the workflow type '" + WORKFLOW + "-" +
 WORKFLOW_VERSION
        + "'.");
try {
    swf.registerWorkflowType(new RegisterWorkflowTypeRequest()
        .withDomain(DOMAIN)
        .withName(WORKFLOW)
        .withDefaultLambdaRole(lambda_role_arn)
        .withVersion(WORKFLOW_VERSION)
        .withDefaultChildPolicy(ChildPolicy.TERMINATE)
        .withDefaultTaskList(new TaskList().withName(TASKLIST))
        .withDefaultTaskStartToCloseTimeout("30"));
}
catch (TypeAlreadyExistsException e) {
```

Schedule a Lambda task

Schedule a Lambda task is similar to scheduling an activity. You provide a Decision with a `ScheduleLambdaFunction` DecisionType and with ScheduleLambdaFunctionDecisionAttributes.

```
running_functions == 0 && scheduled_functions == 0) {
AWSLambda lam = AWSLambdaClientBuilder.defaultClient();
GetFunctionConfigurationResult function_config =
    lam.getFunctionConfiguration(
            new GetFunctionConfigurationRequest()
                .withFunctionName("HelloFunction"));
String function_arn = function_config.getFunctionArn();

ScheduleLambdaFunctionDecisionAttributes attrs =
    new ScheduleLambdaFunctionDecisionAttributes()
        .withId("HelloFunction (Lambda task example)")
        .withName(function_arn)
        .withInput(workflow_input);

decisions.add(
```

In the `ScheduleLambdaFuntionDecisionAttributes`, you must supply a *name*, which is the ARN of the Lambda function to call, and an *id*, which is the name that Amazon SWF will use to identify the Lambda function in history logs.

You can also provide optional *input* for the Lambda function and set its *start to close timeout* value, which is the number of seconds that the Lambda function is allowed to run before generating a `LambdaFunctionTimedOut` event.

> **Note**
>
> This code uses the AWSLambdaClient to retrieve the ARN of the Lambda function, given the function name. You can use this technique to avoid hard-coding the full ARN (which includes your AWS account ID) in your code.

Handle Lambda function events in your decider

Lambda tasks will generate a number of events that you can take action on when polling for decision tasks in your workflow worker, corresponding to the lifecycle of your Lambda task, with EventType values such as `LambdaFunctionScheduled`, `LambdaFunctionStarted`, and `LambdaFunctionCompleted`. If the Lambda function fails, or takes longer to run than its set timeout value, you will receive either a `LambdaFunctionFailed` or `LambdaFunctionTimedOut` event type, respectively.

```
boolean function_completed = false;
String result = null;

System.out.println("Executing the decision task for the history events: [");
for (HistoryEvent event : events) {
    System.out.println("  " + event);
    EventType event_type = EventType.fromValue(event.getEventType());
    switch(event_type) {
    case WorkflowExecutionStarted:
        workflow_input =
            event.getWorkflowExecutionStartedEventAttributes()
                 .getInput();
        break;
    case LambdaFunctionScheduled:
        scheduled_functions++;
        break;
    case ScheduleLambdaFunctionFailed:
```

```
        scheduled_functions--;
        break;
    case LambdaFunctionStarted:
        scheduled_functions--;
        running_functions++;
        break;
    case LambdaFunctionCompleted:
        running_functions--;
        function_completed = true;
        result = event.getLambdaFunctionCompletedEventAttributes()
                      .getResult();
        break;
    case LambdaFunctionFailed:
        running_functions--;
        break;
    case LambdaFunctionTimedOut:
        running_functions--;
        break;
```

Receive output from your Lambda function

When you receive a `LambdaFunctionCompleted`EventType, you can retrieve your 0 function's return value by first calling `getLambdaFunctionCompletedEventAttributes` on the HistoryEvent to get a LambdaFunctionCompletedEventAttributes object, and then calling its `getResult` method to retrieve the output of the Lambda function:

```
 LambdaFunctionCompleted:
running_functions--;
```

Complete source for this example

You can browse the *complete source :github:`<awsdocs/aws-java-developer-guide/tree/master/doc_source/snippets/helloswf_lambda/>* for this example on Github in the *aws-java-developer-guide* repository.

Shutting Down Activity and Workflow Workers Gracefully

The Building a Simple Amazon SWF Application topic provided a complete implementation of a simple workflow application consisting of a registration application, an activity and workflow worker, and a workflow starter.

Worker classes are designed to run continuously, polling for tasks sent by Amazon SWF in order to run activities or return decisions. Once a poll request is made, Amazon SWF records the poller and will attempt to assign a task to it.

If the workflow worker is terminated during a long poll, Amazon SWF may still try to send a task to the terminated worker, resulting in a lost task (until the task times out).

One way to handle this situation is to wait for all long poll requests to return before the worker terminates.

In this topic, we'll rewrite the activity worker from `helloswf`, using Java's shutdown hooks to attempt a graceful shutdown of the activity worker.

Here is the complete code:

```
import java.util.concurrent.CountDownLatch;
import java.util.concurrent.TimeUnit;

import com.amazonaws.regions.Regions;
import com.amazonaws.services.simpleworkflow.AmazonSimpleWorkflow;
import com.amazonaws.services.simpleworkflow.AmazonSimpleWorkflowClientBuilder;
import com.amazonaws.services.simpleworkflow.model.ActivityTask;
import com.amazonaws.services.simpleworkflow.model.PollForActivityTaskRequest;
import com.amazonaws.services.simpleworkflow.model.RespondActivityTaskCompletedRequest;
import com.amazonaws.services.simpleworkflow.model.RespondActivityTaskFailedRequest;
import com.amazonaws.services.simpleworkflow.model.TaskList;

public class ActivityWorkerWithGracefulShutdown {

    private static final AmazonSimpleWorkflow swf =

 AmazonSimpleWorkflowClientBuilder.standard().withRegion(Regions.DEFAULT_REGION).build();
    private static final CountDownLatch waitForTermination = new CountDownLatch(1);
    private static volatile boolean terminate = false;

    private static String executeActivityTask(String input) throws Throwable {
        return "Hello, " + input + "!";
    }

    public static void main(String[] args) {
        Runtime.getRuntime().addShutdownHook(new Thread() {
            @Override
            public void run() {
```

```
                try {
                    terminate = true;
                    System.out.println("Waiting for the current poll request" +
                            " to return before shutting down.");
                    waitForTermination.await(60, TimeUnit.SECONDS);
                }
                catch (InterruptedException e) {
                    // ignore
                }
            }
        });
        try {
            pollAndExecute();
        }
        finally {
            waitForTermination.countDown();
        }
    }

    public static void pollAndExecute() {
        while (!terminate) {
            System.out.println("Polling for an activity task from the tasklist '"
                    + HelloTypes.TASKLIST + "' in the domain '" +
                    HelloTypes.DOMAIN + "'.");

            ActivityTask task = swf.pollForActivityTask(new
PollForActivityTaskRequest()
                .withDomain(HelloTypes.DOMAIN)
                .withTaskList(new TaskList().withName(HelloTypes.TASKLIST)));

            String taskToken = task.getTaskToken();

            if (taskToken != null) {
                String result = null;
                Throwable error = null;

                try {
                    System.out.println("Executing the activity task with input '"
                            + task.getInput() + "'.");
                    result = executeActivityTask(task.getInput());
                }
                catch (Throwable th) {
                    error = th;
                }
```

```
                if (error == null) {
                    System.out.println("The activity task succeeded with result '"
                            + result + "'.");
                    swf.respondActivityTaskCompleted(
                        new RespondActivityTaskCompletedRequest()
                            .withTaskToken(taskToken)
                            .withResult(result));
                }
                else {
                    System.out.println("The activity task failed with the error '"
                            + error.getClass().getSimpleName() + "'.");
                    swf.respondActivityTaskFailed(
                        new RespondActivityTaskFailedRequest()
                            .withTaskToken(taskToken)
                            .withReason(error.getClass().getSimpleName())
                            .withDetails(error.getMessage()));
                }
            }
        }
    }
}
```

In this version, the polling code that was in the `main` function in the original version has been moved into its own method, `pollAndExecute`.

The `main` function now uses a CountDownLatch in conjunction with a shutdown hook to cause the thread to wait for up to 60 seconds after its termination is requested before letting the thread shut down.

Registering Domains

Every workflow and activity in Amazon SWF needs a *domain* to run in.

1. Create a new RegisterDomainRequest object, providing it with at least the domain name and workflow execution retention period (these parameters are both required).
2. Call the AmazonSimpleWorkflowClient.registerDomain method with the *RegisterDomainRequest* object.
3. Catch the DomainAlreadyExistsException if the domain you're requesting already exists (in which case, no action is usually required).

The following code demonstrates this procedure:

```
public void register_swf_domain(AmazonSimpleWorkflowClient swf, String name)
{
    RegisterDomainRequest request = new RegisterDomainRequest().withName(name);
    request.setWorkflowExecutionRetentionPeriodInDays("10");
    try
    {
        swf.registerDomain(request);
    }
    catch (DomainAlreadyExistsException e)
    {
        System.out.println("Domain already exists!");
    }
}
```

Listing Domains

You can list the Amazon SWF domains associated with your account and AWS region by registration type.

1. Create a ListDomainsRequest object, and specify the registration status of the domains that you're interested in—this is required.
2. Call AmazonSimpleWorkflowClient.listDomains with the *ListDomainRequest* object. Results are provided in a DomainInfos object.
3. Call getDomainInfos on the returned object to get a list of DomainInfo objects.
4. Call getName on each *DomainInfo* object to get its name.

The following code demonstrates this procedure:

```
public void list_swf_domains(AmazonSimpleWorkflowClient swf)
{
    ListDomainsRequest request = new ListDomainsRequest();
    request.setRegistrationStatus("REGISTERED");
    DomainInfos domains = swf.listDomains(request);
    System.out.println("Current Domains:");
    for (DomainInfo di : domains.getDomainInfos())
    {
        System.out.println(" * " + di.getName());
```

```
    }
}
```

Code Samples included with the SDK

The AWS SDK for Java comes packaged with code samples that demonstrate many of the features of the SDK in buildable, runnable programs. You can study or modify these to implement your own AWS solutions using the AWS SDK for Java.

How to Get the Samples

The AWS SDK for Java code samples are provided in the *samples* directory of the SDK. If you downloaded and installed the SDK using the information in Set up the AWS SDK for Java, you already have the samples on your system.

You can also view the latest samples on the AWS SDK for Java GitHub repository, in the src/samples directory.

Building and Running the Samples Using the Command Line

The samples include Ant build scripts so that you can easily build and run them from the command line. Each sample also contains a README file in HTML format that contains information specific to each sample.

> **Note**
>
> If you're browsing the sample code on GitHub, click the **Raw** button in the source code display when viewing the sample's README.html file. In raw mode, the HTML will render as intended in your browser.

Prerequisites

Before running any of the AWS SDK for Java samples, you need to set your AWS credentials in the environment or with the AWS CLI, as specified in Set up AWS Credentials and Region for Development. The samples use the default credential provider chain whenever possible. So by setting your credentials in this way, you can avoid the risky practice of inserting your AWS

credentials in files within the source code directory (where they may inadvertently be checked in and shared publicly).

Running the Samples

1. Change to the directory containing the sample's code. For example, if you're in the root directory of the AWS SDK download and want to run the `AwsConsoleApp` sample, you would type:

   ```
   cd samples/AwsConsoleApp
   ```

2. Build and run the sample with Ant. The default build target performs both actions, so you can just enter:

   ```
   ant
   ```

The sample prints information to standard output—for example:

```
===========================================

Welcome to the {AWS} Java SDK!

===========================================
You have access to 4 Availability Zones.

You have 0 {EC2} instance(s) running.

You have 13 Amazon SimpleDB domain(s) containing a total of 62 items.

You have 23 {S3} bucket(s), containing 44 objects with a total size of 154767691 bytes.
```

Building and Running the Samples Using the Eclipse IDE

If you use the AWS Toolkit for Eclipse, you can also start a new project in Eclipse based on the AWS SDK for Java or add the SDK to an existing Java project.

Prerequisites

After installing the AWS Toolkit for Eclipse, we recommend configuring the Toolkit with your security credentials. You can do this anytime by choosing **Preferences** from the **Window** menu in Eclipse, and then choosing the **AWS Toolkit** section.

Running the Samples

1. Open Eclipse.
2. Create a new AWS Java project. In Eclipse, on the **File** menu, choose **New**, and then click **Project**. The **New Project** wizard opens.
3. Expand the **AWS** category, then choose **AWS Java Project**.
4. Choose **Next**. The project settings page is displayed.
5. Enter a name in the **Project Name** box. The AWS SDK for Java Samples group displays the samples available in the SDK, as described previously.
6. Select the samples you want to include in your project by selecting each check box.
7. Enter your AWS credentials. If you've already configured the AWS Toolkit for Eclipse with your credentials, this is automatically filled in.
8. Choose **Finish**. The project is created and added to the **Project Explorer**.
9. Choose the sample `.java` file you want to run. For example, for the Amazon S3 sample, choose `S3Sample.java`.
10. Choose **Run** from the **Run** menu.
11. Right-click the project in **Project Explorer**, point to **Build Path**, and then choose **Add Libraries**.
12. Choose **AWS Java SDK**, choose **Next**, and then follow the remaining on-screen instructions.

Security for the AWS SDK for Java

Cloud security at Amazon Web Services (AWS) is the highest priority. As an AWS customer, you benefit from a data center and network architecture that is built to meet the requirements of the most security-sensitive organizations. Security is a shared responsibility between AWS and you. The Shared Responsibility Model describes this as Security of the Cloud and Security in the Cloud.

Security of the Cloud – AWS is responsible for protecting the infrastructure that runs all of the services offered in the AWS Cloud and providing you with services that you can use securely. Our security responsibility is the highest priority at AWS, and the effectiveness of our security is regularly tested and verified by third-party auditors as part of the AWS Compliance Programs.

Security in the Cloud – Your responsibility is determined by the AWS service you are using, and other factors including the sensitivity of your data, your organization's requirements, and applicable laws and regulations.

This AWS product or service follows the shared responsibility model through the specific Amazon Web Services (AWS) services it supports. For AWS service security information, see the AWS service security documentation page and AWS services that are in scope of AWS compliance efforts by compliance program.

Topics

Data protection in AWS SDK for Java 1.x

The shared responsibility model applies to data protection in this AWS product or service. As described in this model, AWS is responsible for protecting the global infrastructure that runs all of the AWS Cloud. You are responsible for maintaining control over your content that is hosted on this infrastructure. This content includes the security configuration and management tasks for the

AWS services that you use. For more information about data privacy, see the Data Privacy FAQ. For information about data protection in Europe, see the AWS Shared Responsibility Model and GDPR blog post on the AWS Security Blog.

For data protection purposes, we recommend that you protect AWS account credentials and set up individual user accounts with AWS Identity and Access Management (IAM). That way each user is given only the permissions necessary to fulfill their job duties. We also recommend that you secure your data in the following ways:

- Use multi-factor authentication (MFA) with each account.
- Use SSL/TLS to communicate with AWS resources.
- Set up API and user activity logging with AWS CloudTrail.
- Use AWS encryption solutions, with all default security controls within AWS services.
- Use advanced managed security services such as Amazon Macie, which assists in discovering and securing personal data that is stored in Amazon S3.
- If you require FIPS 140-2 validated cryptographic modules when accessing AWS through a command line interface or an API, use a FIPS endpoint. For more information about the available FIPS endpoints, see Federal Information Processing Standard (FIPS) 140-2.

We strongly recommend that you never put sensitive identifying information, such as your customers' account numbers, into free-form fields such as a **Name** field. This includes when you work with this AWS product or service or other AWS services using the console, API, AWS CLI, or AWS SDKs. Any data that you enter into this AWS product or service or other services might get picked up for inclusion in diagnostic logs. When you provide a URL to an external server, don't include credentials information in the URL to validate your request to that server.

AWS SDK for Java support for TLS

The following information applies only to Java SSL implementation (the default SSL implementation in the AWS SDK for Java). If you're using a different SSL implementation, see your specific SSL implementation to learn how to enforce TLS versions.

How to check the TLS version

Consult your Java virtual machine (JVM) provider's documentation to determine which TLS versions are supported on your platform. For some JVMs, the following code will print which SSL versions are supported.

```
System.out.println(Arrays.toString(SSLContext.getDefault().getSupportedSSLParameters().getProto
```

To see the SSL handshake in action and what version of TLS is used, you can use the system property **javax.net.debug**.

```
java app.jar -Djavax.net.debug=ssl
```

> **Note**
>
> TLS 1.3 is incompatible with SDK for Java versions 1.9.5 to 1.10.31. For more information, see the following blog post.
> https://aws.amazon.com/blogs/developer/tls-1-3-incompatibility-with-aws-sdk-for-java-versions-1-9-5-to-1-10-31/

Enforcing a minimum TLS version

The SDK always prefers the latest TLS version supported by the platform and service. If you wish to enforce a specific minimum TLS version, consult your JVM's documentation. For OpenJDK-based JVMs, you can use the system property `jdk.tls.client.protocols`.

```
java app.jar -Djdk.tls.client.protocols=PROTOCOLS
```

Consult your JVM's documentation for the supported values of PROTOCOLS.

Identity and Access Management

AWS Identity and Access Management (IAM) is an AWS service that helps an administrator securely control access to AWS resources. IAM administrators control who can be *authenticated* (signed in) and *authorized* (have permissions) to use AWS resources. IAM is an AWS service that you can use with no additional charge.

Topics

- Managing access using policies
- How AWS services work with IAM
- Troubleshooting AWS identity and access

Audience

How you use AWS Identity and Access Management (IAM) differs, depending on the work that you do in AWS.

Service user – If you use AWS services to do your job, then your administrator provides you with the credentials and permissions that you need. As you use more AWS features to do your work, you might need additional permissions. Understanding how access is managed can help you request the right permissions from your administrator. If you cannot access a feature in AWS, see Troubleshooting AWS identity and access or the user guide of the AWS service you are using.

Service administrator – If you're in charge of AWS resources at your company, you probably have full access to AWS. It's your job to determine which AWS features and resources your service users should access. You must then submit requests to your IAM administrator to change the permissions of your service users. Review the information on this page to understand the basic concepts of IAM. To learn more about how your company can use IAM with AWS, see the user guide of the AWS service you are using.

IAM administrator – If you're an IAM administrator, you might want to learn details about how you can write policies to manage access to AWS. To view example AWS identity-based policies that you can use in IAM, see the user guide of the AWS service you are using.

Authenticating with identities

Authentication is how you sign in to AWS using your identity credentials. You must be *authenticated* (signed in to AWS) as the AWS account root user, as an IAM user, or by assuming an IAM role.

You can sign in to AWS as a federated identity by using credentials provided through an identity source. AWS IAM Identity Center (IAM Identity Center) users, your company's single sign-on authentication, and your Google or Facebook credentials are examples of federated identities. When you sign in as a federated identity, your administrator previously set up identity federation using IAM roles. When you access AWS by using federation, you are indirectly assuming a role.

Depending on the type of user you are, you can sign in to the AWS Management Console or the AWS access portal. For more information about signing in to AWS, see How to sign in to your AWS account in the *AWS Sign-In User Guide*.

If you access AWS programmatically, AWS provides a software development kit (SDK) and a command line interface (CLI) to cryptographically sign your requests by using your credentials. If you don't use AWS tools, you must sign requests yourself. For more information about using the recommended method to sign requests yourself, see AWS Signature Version 4 for API requests in the *IAM User Guide*.

Regardless of the authentication method that you use, you might be required to provide additional security information. For example, AWS recommends that you use multi-factor authentication (MFA) to increase the security of your account. To learn more, see Multi-factor authentication in the *AWS IAM Identity Center User Guide* and AWS Multi-factor authentication in IAM in the *IAM User Guide*.

AWS account root user

When you create an AWS account, you begin with one sign-in identity that has complete access to all AWS services and resources in the account. This identity is called the AWS account *root user* and is accessed by signing in with the email address and password that you used to create the account. We strongly recommend that you don't use the root user for your everyday tasks. Safeguard your root user credentials and use them to perform the tasks that only the root user can perform. For the complete list of tasks that require you to sign in as the root user, see Tasks that require root user credentials in the *IAM User Guide*.

Federated identity

As a best practice, require human users, including users that require administrator access, to use federation with an identity provider to access AWS services by using temporary credentials.

A *federated identity* is a user from your enterprise user directory, a web identity provider, the AWS Directory Service, the Identity Center directory, or any user that accesses AWS services by using credentials provided through an identity source. When federated identities access AWS accounts, they assume roles, and the roles provide temporary credentials.

For centralized access management, we recommend that you use AWS IAM Identity Center. You can create users and groups in IAM Identity Center, or you can connect and synchronize to a set of users and groups in your own identity source for use across all your AWS accounts and applications. For

information about IAM Identity Center, see What is IAM Identity Center? in the *AWS IAM Identity Center User Guide*.

IAM users and groups

An *IAM user* is an identity within your AWS account that has specific permissions for a single person or application. Where possible, we recommend relying on temporary credentials instead of creating IAM users who have long-term credentials such as passwords and access keys. However, if you have specific use cases that require long-term credentials with IAM users, we recommend that you rotate access keys. For more information, see Rotate access keys regularly for use cases that require long-term credentials in the *IAM User Guide*.

An *IAM group* is an identity that specifies a collection of IAM users. You can't sign in as a group. You can use groups to specify permissions for multiple users at a time. Groups make permissions easier to manage for large sets of users. For example, you could have a group named *IAMAdmins* and give that group permissions to administer IAM resources.

Users are different from roles. A user is uniquely associated with one person or application, but a role is intended to be assumable by anyone who needs it. Users have permanent long-term credentials, but roles provide temporary credentials. To learn more, see Use cases for IAM users in the *IAM User Guide*.

IAM roles

An *IAM role* is an identity within your AWS account that has specific permissions. It is similar to an IAM user, but is not associated with a specific person. To temporarily assume an IAM role in the AWS Management Console, you can switch from a user to an IAM role (console). You can assume a role by calling an AWS CLI or AWS API operation or by using a custom URL. For more information about methods for using roles, see Methods to assume a role in the *IAM User Guide*.

IAM roles with temporary credentials are useful in the following situations:

- **Federated user access** – To assign permissions to a federated identity, you create a role and define permissions for the role. When a federated identity authenticates, the identity is associated with the role and is granted the permissions that are defined by the role. For information about roles for federation, see Create a role for a third-party identity provider (federation) in the *IAM User Guide*. If you use IAM Identity Center, you configure a permission set. To control what your identities can access after they authenticate, IAM Identity Center correlates the permission set to a role in IAM. For information about permissions sets, see Permission sets in the *AWS IAM Identity Center User Guide*.

- **Temporary IAM user permissions** – An IAM user or role can assume an IAM role to temporarily take on different permissions for a specific task.
- **Cross-account access** – You can use an IAM role to allow someone (a trusted principal) in a different account to access resources in your account. Roles are the primary way to grant cross-account access. However, with some AWS services, you can attach a policy directly to a resource (instead of using a role as a proxy). To learn the difference between roles and resource-based policies for cross-account access, see Cross account resource access in IAM in the *IAM User Guide*.
- **Cross-service access** – Some AWS services use features in other AWS services. For example, when you make a call in a service, it's common for that service to run applications in Amazon EC2 or store objects in Amazon S3. A service might do this using the calling principal's permissions, using a service role, or using a service-linked role.
  - **Forward access sessions (FAS)** – When you use an IAM user or role to perform actions in AWS, you are considered a principal. When you use some services, you might perform an action that then initiates another action in a different service. FAS uses the permissions of the principal calling an AWS service, combined with the requesting AWS service to make requests to downstream services. FAS requests are only made when a service receives a request that requires interactions with other AWS services or resources to complete. In this case, you must have permissions to perform both actions. For policy details when making FAS requests, see Forward access sessions.
  - **Service role** – A service role is an IAM role that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see Create a role to delegate permissions to an AWS service in the *IAM User Guide*.
  - **Service-linked role** – A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your AWS account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.
- **Applications running on Amazon EC2** – You can use an IAM role to manage temporary credentials for applications that are running on an EC2 instance and making AWS CLI or AWS API requests. This is preferable to storing access keys within the EC2 instance. To assign an AWS role to an EC2 instance and make it available to all of its applications, you create an instance profile that is attached to the instance. An instance profile contains the role and enables programs that are running on the EC2 instance to get temporary credentials. For more information, see Use an IAM role to grant permissions to applications running on Amazon EC2 instances in the *IAM User Guide*.

Managing access using policies

You control access in AWS by creating policies and attaching them to AWS identities or resources. A policy is an object in AWS that, when associated with an identity or resource, defines their permissions. AWS evaluates these policies when a principal (user, root user, or role session) makes a request. Permissions in the policies determine whether the request is allowed or denied. Most policies are stored in AWS as JSON documents. For more information about the structure and contents of JSON policy documents, see Overview of JSON policies in the *IAM User Guide*.

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

By default, users and roles have no permissions. To grant users permission to perform actions on the resources that they need, an IAM administrator can create IAM policies. The administrator can then add the IAM policies to roles, and users can assume the roles.

IAM policies define permissions for an action regardless of the method that you use to perform the operation. For example, suppose that you have a policy that allows the `iam:GetRole` action. A user with that policy can get role information from the AWS Management Console, the AWS CLI, or the AWS API.

Identity-based policies

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Define custom IAM permissions with customer managed policies in the *IAM User Guide*.

Identity-based policies can be further categorized as *inline policies* or *managed policies*. Inline policies are embedded directly into a single user, group, or role. Managed policies are standalone policies that you can attach to multiple users, groups, and roles in your AWS account. Managed policies include AWS managed policies and customer managed policies. To learn how to choose between a managed policy or an inline policy, see Choose between managed policies and inline policies in the *IAM User Guide*.

Resource-based policies

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM *role trust policies* and Amazon S3 *bucket policies*. In services that

support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must specify a principal in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.

Resource-based policies are inline policies that are located in that service. You can't use AWS managed policies from IAM in a resource-based policy.

Access control lists (ACLs)

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

Amazon S3, AWS WAF, and Amazon VPC are examples of services that support ACLs. To learn more about ACLs, see Access control list (ACL) overview in the *Amazon Simple Storage Service Developer Guide*.

Other policy types

AWS supports additional, less-common policy types. These policy types can set the maximum permissions granted to you by the more common policy types.

- **Permissions boundaries** – A permissions boundary is an advanced feature in which you set the maximum permissions that an identity-based policy can grant to an IAM entity (IAM user or role). You can set a permissions boundary for an entity. The resulting permissions are the intersection of an entity's identity-based policies and its permissions boundaries. Resource-based policies that specify the user or role in the `Principal` field are not limited by the permissions boundary. An explicit deny in any of these policies overrides the allow. For more information about permissions boundaries, see Permissions boundaries for IAM entities in the *IAM User Guide*.
- **Service control policies (SCPs)** – SCPs are JSON policies that specify the maximum permissions for an organization or organizational unit (OU) in AWS Organizations. AWS Organizations is a service for grouping and centrally managing multiple AWS accounts that your business owns. If you enable all features in an organization, then you can apply service control policies (SCPs) to any or all of your accounts. The SCP limits permissions for entities in member accounts, including each AWS account root user. For more information about Organizations and SCPs, see Service control policies in the *AWS Organizations User Guide*.

- **Resource control policies (RCPs)** – RCPs are JSON policies that you can use to set the maximum available permissions for resources in your accounts without updating the IAM policies attached to each resource that you own. The RCP limits permissions for resources in member accounts and can impact the effective permissions for identities, including the AWS account root user, regardless of whether they belong to your organization. For more information about Organizations and RCPs, including a list of AWS services that support RCPs, see Resource control policies (RCPs) in the *AWS Organizations User Guide*.
- **Session policies** – Session policies are advanced policies that you pass as a parameter when you programmatically create a temporary session for a role or federated user. The resulting session's permissions are the intersection of the user or role's identity-based policies and the session policies. Permissions can also come from a resource-based policy. An explicit deny in any of these policies overrides the allow. For more information, see Session policies in the *IAM User Guide*.

Multiple policy types

When multiple types of policies apply to a request, the resulting permissions are more complicated to understand. To learn how AWS determines whether to allow a request when multiple policy types are involved, see Policy evaluation logic in the *IAM User Guide*.

How AWS services work with IAM

To get a high-level view of how AWS services work with most IAM features, see AWS services that work with IAM in the *IAM User Guide*.

To learn how to use a specific AWS service with IAM, see the security section of the relevant service's User Guide.

Troubleshooting AWS identity and access

Use the following information to help you diagnose and fix common issues that you might encounter when working with AWS and IAM.

Topics

- I am not authorized to perform an action in AWS
- I am not authorized to perform iam:PassRole
- I want to allow people outside of my AWS account to access my AWS resources

I am not authorized to perform an action in AWS

If you receive an error that you're not authorized to perform an action, your policies must be updated to allow you to perform the action.

The following example error occurs when the `mateojackson` IAM user tries to use the console to view details about a fictional *`my-example-widget`* resource but doesn't have the fictional `awes:`*`GetWidget`* permissions.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 awes:GetWidget on resource: my-example-widget
```

In this case, the policy for the `mateojackson` user must be updated to allow access to the *`my-example-widget`* resource by using the `awes:`*`GetWidget`* action.

If you need help, contact your AWS administrator. Your administrator is the person who provided you with your sign-in credentials.

I am not authorized to perform iam:PassRole

If you receive an error that you're not authorized to perform the `iam:PassRole` action, your policies must be updated to allow you to pass a role to AWS.

Some AWS services allow you to pass an existing role to that service instead of creating a new service role or service-linked role. To do this, you must have permissions to pass the role to the service.

The following example error occurs when an IAM user named `marymajor` tries to use the console to perform an action in AWS. However, the action requires the service to have permissions that are granted by a service role. Mary does not have permissions to pass the role to the service.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

In this case, Mary's policies must be updated to allow her to perform the `iam:PassRole` action.

If you need help, contact your AWS administrator. Your administrator is the person who provided you with your sign-in credentials.

I want to allow people outside of my AWS account to access my AWS resources

You can create a role that users in other accounts or people outside of your organization can use to access your resources. You can specify who is trusted to assume the role. For services that support resource-based policies or access control lists (ACLs), you can use those policies to grant people access to your resources.

To learn more, consult the following:

- To learn whether AWS supports these features, see How AWS services work with IAM.
- To learn how to provide access to your resources across AWS accounts that you own, see Providing access to an IAM user in another AWS account that you own in the *IAM User Guide*.
- To learn how to provide access to your resources to third-party AWS accounts, see Providing access to AWS accounts owned by third parties in the *IAM User Guide*.
- To learn how to provide access through identity federation, see Providing access to externally authenticated users (identity federation) in the *IAM User Guide*.
- To learn the difference between using roles and resource-based policies for cross-account access, see Cross account resource access in IAM in the *IAM User Guide*.

Compliance Validation for this AWS Product or Service

To learn whether an AWS service is within the scope of specific compliance programs, see AWS services in Scope by Compliance Program and choose the compliance program that you are interested in. For general information, see AWS Compliance Programs.

You can download third-party audit reports using AWS Artifact. For more information, see Downloading Reports in AWS Artifact.

Your compliance responsibility when using AWS services is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. AWS provides the following resources to help with compliance:

- Security Compliance & Governance – These solution implementation guides discuss architectural considerations and provide steps for deploying security and compliance features.
- HIPAA Eligible Services Reference – Lists HIPAA eligible services. Not all AWS services are HIPAA eligible.

- AWS Compliance Resources – This collection of workbooks and guides might apply to your industry and location.
- AWS Customer Compliance Guides – Understand the shared responsibility model through the lens of compliance. The guides summarize the best practices for securing AWS services and map the guidance to security controls across multiple frameworks (including National Institute of Standards and Technology (NIST), Payment Card Industry Security Standards Council (PCI), and International Organization for Standardization (ISO)).
- Evaluating Resources with Rules in the *AWS Config Developer Guide* – The AWS Config service assesses how well your resource configurations comply with internal practices, industry guidelines, and regulations.
- AWS Security Hub – This AWS service provides a comprehensive view of your security state within AWS. Security Hub uses security controls to evaluate your AWS resources and to check your compliance against security industry standards and best practices. For a list of supported services and controls, see Security Hub controls reference.
- Amazon GuardDuty – This AWS service detects potential threats to your AWS accounts, workloads, containers, and data by monitoring your environment for suspicious and malicious activities. GuardDuty can help you address various compliance requirements, like PCI DSS, by meeting intrusion detection requirements mandated by certain compliance frameworks.
- AWS Audit Manager – This AWS service helps you continuously audit your AWS usage to simplify how you manage risk and compliance with regulations and industry standards.

This AWS product or service follows the shared responsibility model through the specific Amazon Web Services (AWS) services it supports. For AWS service security information, see the AWS service security documentation page and AWS services that are in scope of AWS compliance efforts by compliance program.

Resilience for this AWS Product or Service

The AWS global infrastructure is built around AWS Regions and Availability Zones.

AWS Regions provide multiple physically separated and isolated Availability Zones, which are connected with low-latency, high-throughput, and highly redundant networking.

With Availability Zones, you can design and operate applications and databases that automatically fail over between zones without interruption. Availability Zones are more highly available, fault tolerant, and scalable than traditional single or multiple data center infrastructures.

For more information about AWS Regions and Availability Zones, see AWS Global Infrastructure.

This AWS product or service follows the shared responsibility model through the specific Amazon Web Services (AWS) services it supports. For AWS service security information, see the AWS service security documentation page and AWS services that are in scope of AWS compliance efforts by compliance program.

Infrastructure Security for this AWS Product or Service

This AWS product or service uses managed services, and therefore is protected by the AWS global network security. For information about AWS security services and how AWS protects infrastructure, see AWS Cloud Security. To design your AWS environment using the best practices for infrastructure security, see Infrastructure Protection in *Security Pillar AWS Well-Architected Framework*.

You use AWS published API calls to access this AWS Product or Service through the network. Clients must support the following:

- Transport Layer Security (TLS). We require TLS 1.2 and recommend TLS 1.3.
- Cipher suites with perfect forward secrecy (PFS) such as DHE (Ephemeral Diffie-Hellman) or ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). Most modern systems such as Java 7 and later support these modes.

Additionally, requests must be signed by using an access key ID and a secret access key that is associated with an IAM principal. Or you can use the AWS Security Token Service (AWS STS) to generate temporary security credentials to sign requests.

This AWS product or service follows the shared responsibility model through the specific Amazon Web Services (AWS) services it supports. For AWS service security information, see the AWS service security documentation page and AWS services that are in scope of AWS compliance efforts by compliance program.

Amazon S3 Encryption Client Migration

This topic shows you how to migrate your applications from Version 1 (V1) of the Amazon Simple Storage Service (Amazon S3) encryption client to Version 2 (V2) and ensure application availability throughout the migration process.

Prerequisites

Amazon S3 client-side encryption requires the following:

- Java 8 or later installed in your application environment. The AWS SDK for Java works with the Oracle Java SE Development Kit and with distributions of Open Java Development Kit (OpenJDK) such as Amazon Corretto, Red Hat OpenJDK, and AdoptOpenJDK.
- The Bouncy Castle Crypto package. You can place the Bouncy Castle .jar file on the classpath of your application environment, or add a dependency on the artifactId `bcprov-ext-jdk15on` (with the groupId of `org.bouncycastle`) to your Maven `pom.xml` file.

Migration Overview

This migration happens in two phases:

1. **Update existing clients to read new formats.** Update your application to use version 1.11.837 or later of the AWS SDK for Java and redeploy the application. This enables the Amazon S3 client-side encryption service clients in your application to decrypt objects created by V2 service clients. If your application uses multiple AWS SDKs, you must update each SDK separately.
2. **Migrate encryption and decryption clients to V2.** Once all of your V1 encryption clients can read V2 encryption formats, update the Amazon S3 client-side encryption and decryption clients in your application code to use their V2 equivalents.

Update Existing Clients to Read New Formats

The V2 encryption client uses encryption algorithms that older versions of the AWS SDK for Java do not support.

The first step in the migration is to update your V1 encryption clients to use version 1.11.837 or later of the AWS SDK for Java. (We recommend that you update to the latest release version, which you can find in the Java API Reference version 1.x.) To do so, update the dependency in your project configuration. After your project configuration is updated, rebuild your project and redeploy it.

Once you have completed these steps, your application's V1 encryption clients will be able to read objects written by V2 encryption clients.

Update the Dependency in Your Project Configuration

Modify your project configuration file (for example, pom.xml or build.gradle) to use version 1.11.837 or later of the AWS SDK for Java. Then, rebuild your project and redeploy it.

Completing this step before deploying new application code helps to ensure that encryption and decryption operations remain consistent across your fleet during the migration process.

Example Using Maven

Snippet from a pom.xml file:

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>com.amazonaws</groupId>
      <artifactId>aws-java-sdk-bom</artifactId>
      <version>1.11.837</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
```

Example Using Gradle

Snippet from a build.gradle file:

```
dependencies {
  implementation platform('com.amazonaws:aws-java-sdk-bom:1.11.837')
  implementation 'com.amazonaws:aws-java-sdk-s3'
}
```

Migrate Encryption and Decryption Clients to V2

Once your project has been updated with the latest SDK version, you can modify your application code to use the V2 client. To do so, first update your code to use the new service client builder. Then provide encryption materials using a method on the builder that has been renamed, and configure your service client further as needed.

These code snippets demonstrate how to use client-side encryption with the AWS SDK for Java, and provide comparisons between the V1 and V2 encryption clients.

V1

```
// minimal configuration in V1; default CryptoMode.EncryptionOnly.
EncryptionMaterialsProvider encryptionMaterialsProvider = ...
AmazonS3Encryption encryptionClient = AmazonS3EncryptionClient.encryptionBuilder()
             .withEncryptionMaterials(encryptionMaterialsProvider)
             .build();
```

V2

```
// minimal configuration in V2; default CryptoMode.StrictAuthenticatedEncryption.
EncryptionMaterialsProvider encryptionMaterialsProvider = ...
AmazonS3EncryptionV2 encryptionClient = AmazonS3EncryptionClientV2.encryptionBuilder()
             .withEncryptionMaterialsProvider(encryptionMaterialsProvider)
             .withCryptoConfiguration(new CryptoConfigurationV2()
                           // The following setting allows the client to read V1
 encrypted objects
                           .withCryptoMode(CryptoMode.AuthenticatedEncryption)
             )
             .build();
```

The above example sets the `cryptoMode` to `AuthenticatedEncryption`. This is a setting that allows a V2 encryption client to read objects that have been written by a V1 encryption client. If your client does not need the capability to read objects written by a V1 client, then we recommend using the default setting of `StrictAuthenticatedEncryption` instead.

Construct a V2 Encryption Client

The V2 encryption client can be constructed by calling *AmazonS3EncryptionClientV2.encryptionBuilder().*

You can replace all of your existing V1 encryption clients with V2 encryption clients. A V2 encryption client will always be able to read any object that has been written by a V1 encryption client as long as you permit it to do so by configuring the V2 encryption client to use the `AuthenticatedEncryption`cryptoMode.

Creating a new V2 encryption client is very similar to how you create a V1 encryption client. However, there are a few differences:

- You will use a `CryptoConfigurationV2` object to configure the client instead of a `CryptoConfiguration` object. This parameter is required.

- The default `cryptoMode` setting for the V2 encryption client is `StrictAuthenticatedEncryption`. For the V1 encryption client it is `EncryptionOnly`.
- The method *withEncryptionMaterials()* on the encryption client builder has been renamed to *withEncryptionMaterialsProvider()*. This is merely a cosmetic change that more accurately reflects the argument type. You must use the new method when you configure your service client.

> **Note**
>
> When decrypting with AES-GCM, read the entire object to the end before you start using the decrypted data. This is to verify that the object has not been modified since it was encrypted.

Use Encryption Materials Providers

You can continue to use the same encryption materials providers and encryption materials objects you are already using with the V1 encryption client. These classes are responsible for providing the keys the encryption client uses to secure your data. They can be used interchangeably with both the V2 and the V1 encryption client.

Configure the V2 Encryption Client

The V2 encryption client is configured with a `CryptoConfigurationV2` object. This object can be constructed by calling its default constructor and then modifying its properties as required from the defaults.

The default values for `CryptoConfigurationV2` are:

- `cryptoMode` = `CryptoMode.StrictAuthenticatedEncryption`
- `storageMode` = `CryptoStorageMode.ObjectMetadata`
- `secureRandom` = instance of `SecureRandom`
- `rangeGetMode` = `CryptoRangeGetMode.DISABLED`
- `unsafeUndecryptableObjectPassthrough` = `false`

Note that *EncryptionOnly* is not a supported `cryptoMode` in the V2 encryption client. The V2 encryption client will always encrypt content using authenticated encryption, and protects content encrypting keys (CEKs) using V2 `KeyWrap` objects.

The following example demonstrates how to specify the crypto configuration in V1, and how to instantiate a *CryptoConfigurationV2* object to pass to the V2 encryption client builder.

V1

```
CryptoConfiguration cryptoConfiguration = new CryptoConfiguration()
        .withCryptoMode(CryptoMode.StrictAuthenticatedEncryption);
```

V2

```
CryptoConfigurationV2 cryptoConfiguration = new CryptoConfigurationV2()
        .withCryptoMode(CryptoMode.StrictAuthenticatedEncryption);
```

Additional Examples

The following examples demonstrate how to address specific use cases related to a migration from V1 to V2.

Configure a Service Client to Read Objects Created by the V1 Encryption Client

To read objects that were previously written using a V1 encryption client, set the `cryptoMode` to `AuthenticatedEncryption`. The following code snippet demonstrates how to construct a configuration object with this setting.

```
CryptoConfigurationV2 cryptoConfiguration = new CryptoConfigurationV2()
        .withCryptoMode(CryptoMode.AuthenticatedEncryption);
```

Configure a Service Client to Get Byte Ranges of Objects

To be able to `get` a range of bytes from an encrypted S3 object, enable the new configuration setting `rangeGetMode`. This setting is disabled on the V2 encryption client by default. Note that even when enabled, a ranged `get` only works on objects that have been encrypted using algorithms supported by the `cryptoMode` setting of the client. For more information, see CryptoRangeGetMode in the AWS SDK for Java API Reference.

If you plan to use the Amazon S3 TransferManager to perform multipart downloads of encrypted Amazon S3 objects using the V2 encryption client, then you must first enable the `rangeGetMode` setting on the V2 encryption client.

The following code snippet demonstrates how to configure the V2 client for performing a ranged `get`.

```
// Allows range gets using AES/CTR, for V2 encrypted objects only
CryptoConfigurationV2 cryptoConfiguration = new CryptoConfigurationV2()
       .withRangeGetMode(CryptoRangeGetMode.ALL);

// Allows range gets using AES/CTR and AES/CBC, for V1 and V2 objects
CryptoConfigurationV2 cryptoConfiguration = new CryptoConfigurationV2()
       .withCryptoMode(CryptoMode.AuthenticatedEncryption)
       .withRangeGetMode(CryptoRangeGetMode.ALL);
```

OpenPGP key for the AWS SDK for Java

All publicly available Maven artifacts for the AWS SDK for Java are signed using the OpenPGP standard. The public key that you need to verify the signature of an artifact is available in the following section.

Current key

The following table shows OpenPGP key information for the current releases of the SDK for Java 1x and SDK for Java 2.x.

Key ID	0xAC107B386692DADD
Type	RSA
Size	4096/4096
Created	2016-06-30
Expires	**2025-10-04**
User ID	AWS SDKs and Tools <aws-dr-tools@amazon.com>
Key fingerprint	FEB9 209F 2F2F 3F46 6484 1E55 AC10 7B38 6692 DADD

To copy the following OpenPGP public key for the SDK for Java to the clipboard, select the "Copy" icon in the upper right corner.

```
-----BEGIN PGP PUBLIC KEY BLOCK-----
Comment: Hostname:
Version: Hockeypuck 2.2

xsFNBFd1gAUBEACqbmmFbxdJgz1lD7wrlskQA1LLuSAC4p8ny9u/D2zLR8Ynk3Yz
mzJuQ+Kfjne2t+xTDex6MPJlMYpOviSWsX2psgvdmeyUpW9ap0lrThNYkc+W5fRc
buFehfbi9LSATZGJi8RG0sCCr5FsYVz0gEk85M2+PeM24cXhQIOZtQUjswX/pdk/
KduGtZASqNAYLKROmRODzUuaokLPo24pfm9bnr1RnRtwt5ktPAA5bM9ZZaGKriej
kT2lPffbBjp8F5AZvmGLtNm2Cmg4FKBvI04SQjy2jjrQ3wBzi5Lc9HTxDuHK/rtV
```

```
u6PewUe2WPlnxlXenhMZU1UK4YoSB9E9StQ2VxQiySLHSdxR7Ma4WgYdVLn9bOie
nj3QxLuQ1ZUKF79ES6JaM4tOz1gGcQeU1+UklgjFLuKwmzWRdEIFfxMyvH6qgKnd
U+DioH5mcUwhwffAAsuIJyAdMIEUYh7IfzJJXQf+fF+XfOCl6byOJFWrIGQkAzMu
CEvaCfwtHC2Lpzo33/WRFeMAuzzd0QJ4uz4xFFvaSOSZHMLHWI9YV/+Pea3X99Ms
0Nlek/LolAJh67MynHeVBOHKrq+fluorWepQivctzN6Y1NOkx5naTPGGaKWK7G2q
TbcY5SMnkIWfLFSougj0Fvmjczq8iZRwYxWA+i+LQvsR9WEXEiQffIWRoQARAQAB
zSxBV1MgU0RLcyBhbmQgVG9vbHMgPGF3cy1kci10b29sc0BhbWF6b24uY29tPsLB
lAQTAQoAPgIbAwULCQgHAwUVCgkICwUWAgMBAAIeAQIXgBYhBP65IJ8vLz9GZIQe
VawQezhmktrdBQJnABcIBQkRa8qDAAoJEKwQezhmktrdl1MQAIwEuDar3OTxkfTa
cPNKDnNzxaWqrxZ3FTQ+PyrHhQ6usxxrvDKJS+uCjE9bmWHVFU1R4yQNF+72lJdw
5UhX0u+ZgT9afApE65uAZuwLhPsz8upXT8C6VeKXh3shdw7qXi2hrwtM1a0Pls4O
Cs2C9rLUDMJTySrVDDVwpnaAB+8DcFrs9bIt5Q3gdOUatdzDvcb7QKh9jUvzCpbE
cInb1epDN5MRzowMR4iU2VV1RzLxCvm7CQSyXfgf0DFLkXWiknhOq9eINmytJFG/
ntFdiZFkNZ5hP709loywdfNrmqB6PsF8BPGFh8gKw1pjowrfHpv6cNIqShmA76LT
3OHViOlqGFB7obffq//eZGPR0oYJFDrOdD2CFRoHnP3N++AfkA4SRN7eXwyoz6Pk
Do9WNIEEkAcp6PGvv7AokogDo/4OqmxgC6fN+3BTOstWpv4FlD4NxOZWsTs49wxg
kPlCCVf8t75aZZkcjXng1eClZZQ5SBlRtSB7gMqtP7MIn2J5w8spNbs5xQvJc76u
NvzwEasPkY+UcHd05RddOUwoKqDerLUG7YqdlNCJoQR1mBIgZButbQlMyaZcmQqO
iR0kwDi9h6Dl6fnUb2dFCNJw+eDHvsjG8HI3IVZMl0bUQ2kmmwr1O2YQ1ynJQm0l
lMllI4hFU8/lHNHm8ie5darpVXQEwsGUBBMBCgA+AhsDBQsJCAcDBRUKCQgLBRYC
AwEAAh4BAheAFiEE/rkgny8vP0ZkhB5VrBB7OGaS2t0FAmUkSiIFCQ+P/Z0ACgkQ
rBB7OGaS2t3HVg//S+/KbeOBf+nCdHsrWtp9kxvWIpAGvQhIbxx1tp/impfm+5Rm
fKPD0KX+g42fuMmOdDE4gjO4GjGd7ZY3bx+0zbDSdVebzmYCbPZ/BDP990oPKidd
w6Gl8PaIyqfuARKOESBETvAwNgwO4t2ocjs4pYZV+CuHvESYpqkuHjmHtye6ajZW
Mv24NhjVo4EfP33dPugTjXLjeuGT7qQpsYV3a66juHmPVkXwuPqxh9wTNc5TU6FG
UPSfIGMPL0xha7Rg2i5zvRaAxx4bHqGO8IAz/l/E/tJkV5xnt494HQam9UDbiFI0
Q0TSve1R6S45/UjQW6cycyduHtk72s9ipa9YM0ilTdLgKMWFjzYv4h4qeYvLw3oB
JGQew+I0I4dIrwL/TKet33EuFfwmyT9MaJBhqV6geFaQ0uVmwvzpAcvxIoSqSpkJ
B/kASqCEM/o0QZLuWc56cDsmMisD0ouVPt+c1Zk7AWLlf6j8LKYTbK0QxLRh/eeZ
jhSf3HnpaCfonbOoHmeo5d/o3EZ0AiA4GbT3xgScoIgXOT7KGgOWmtWkdyd3Zvl/
o6q6Hwpg4RsqCkwnfNm5ZIvmTAYXWc2hiICSicxrP0fek5Cc4xVgJR5RMNGyI7+m
ut1SE2WvLhMCwEyl5ecWF0tUze8VBlWkHJp0Y4k2ado39Zq/DZrTRQYEVrjCwX0E
EwEKACcCGwMFCwkIBwMFFQoJCAsFFgIDAQACHgECF4AFAmEyoZoFCQueVRUACgkQ
rBB7OGaS2t3axA//dgcz5T4Fs7LWdIQB/KRnvX64IzaGQcwt3FBhYH+sFaSaD+lu
752vi3jlGxMBKs2NFxk6e2U8xBEir3vfKYd+mZ5L6egXC9MYfvMO/UFEFH3A+t3V
dTOJK4RQcaL9+rFRVdDmZuifN9OFfy25d66JCZ5OiqgHTQViVmbrGw2cQdyNWXrq
YLgg+gktadkmzis4J6hF/le8NOBfrG3n+QthFl/v2ppYYW9pmmxzUIf6tAlR1Vr8
PhPukjuFfrPLRL3XPiK4LkdlSI5MX7Llq4RkcZN1NY1LWS8699wJOLRcr8aQYvzZ
Jm7tfZUaekJ5ScXJWJEaWT4poMbWxXINj6VwE+DqKWvjkzoxBXSIdLk4XThA1dIq
3n5SfMkfuWV2A2xHqJtEzI1XeTm/d/JRxIG8hjIs5FNMGJUSNANJuTVA2putCVf0
JbP4QlafXoEVOYwW/EFJY+brjQadwc/knf/QxszDcKb3THuTxR80A0h6ZysmtLEg
PCaD1xUCDdr4P7DGOtV7yMaDR6O8QKmb7TKzCKCPnHouqPT0DhSB2MRq437+mfSe
EGga/sPMxe0z8ugO2CnrCf3eplU3ZOy4eeQSThTKe0Hp5YlsF1cdZSvjt7GJaHR2
9A22nAJY9PojtlM+0Dkr/PH6r2brv3sEuACRNhzqCWhAve+zVnVLeb+Fk1rCwX0E
```

```
EwEKACcCGwMFCwkIBwMFFQoJCAsFFgIDAQACHgECF4AFAmCavPYFCQsGcHEACgkQ
rBB7OGaS2t3fqg//asHYSTBI4IoLibSF5ZJ8NaLo4EqgRtsO0W1zr8fCoFbxI+yI
qWrNXR7eoFj8tWO7Tj6S0kQr4QZcucLeILfox6CtDzO3f3WQTH9m/0si5U4Jf3RA
gBd0vwxVSSSNEsIUUfylOBHLVwlhaTVfh1h1dZhzb18LA1Vqu0100GGxvaG3dUl4
oMSRSK8EeaSO4hIM8ScjVyhjsuPRm1jlwd1EXZ5mkHRGRMGmwi3XysJjIeQRFmuG
a9uPes7I/K85r7X91BLz+G/mkSZsrHxzLxF0mSZtQdhq08GyMeQlJavs7sb1UVbg
ag30JEAjgqRsNIQOMIvl2/amrRJ7DUyQu5YkZQsAyCIhSVZw6zlJlkYVKSw1Cu1I
VX1n/Aahx5wwaQZA1dDTolX8WvL9O/KmEGWbKUSov40uoRwSOeXP3QhW75kD4zT0
n3pSUc7tXka8GAz5Ar+r90l4wc9as2WjWADo3CX8cF0zQOrN1naMQ++xBIAG9ms8
y3L4ECoRqvjCpjUAfhflxwSM7uclCgFBINFKSLV+QGxzWfHjg3+bSQd0hrRn9j4z
Di9aJmFESQ6iykbUjiUFrcI1NUCFKz2awaxh+Sq8UkYcvuxljxdK/zYYEG8DSdQ2
uwlssHCjYVpAb16hq1EAaSqnhv86020G4Z6JCg3AUZt9RlMBpK2Pn/NmPSzCwX0E
EwEKACcCGwMFCwkIBwMFFQoJCAsFFgIDAQACHgECF4AFAl771vAFCQlnimUACgkQ
rBB7OGaS2t35Jw/9GhpQquJVUCAsc4az7+ad8q2d9OUKXOLl2x0j3/ofS8umZJYr
8KXZxPqBqvLjlUyAB5Ur4fWVBdp9iP4Vj74jmRih7YatK8heGmNoUAnI1/90qV5O
ypFObfvWLxvcUNizmS/LYKdNuYcHwyw4bFyTz9gd3XH6Dxak7YiAXz3JgJIXcIB3
ELfpsduzpncLeVwz/dKhYX5iJ7Y/xd4Ew8Ian8FyNDNRwcXobTpPAMNiGLG5xSLq
8RgQfkm07BVVDtu5tPw4V46gnBXGXNjPhSirGMoNbKZqtPO7TcBQhPQ9c95x73hs
kAqZKHrwi1B7cfy1I8y5sRFbPa0RXeVWQKEMZdwLsFhCbEex6iXHP+rMK0nSJf0G
9lF2eJk140n8K7wzakOnjvN00VfN/9D+xD2lCXCzbYbaDXX2tYOd4GS72fW9M/zT
96tIH5UtMGMOICuUJjnL34EbzBuxwTEnJKhDGQH2P+eUzM9jmuCTRLLGw2P0Zuof
6GsSNLf+5pw3BIvEZw5PYT1N6i3ml9MQ7p9BWr7flCF8CU/xzyPN7TVb/677Be7V
SL1rNFLNi0IdqcbLJ63pTWaUGkCSqRnMfq3cIDlzNz2LoVvOO8VVmdtFRkhHN8hJ
h7CUXlaqBOfaJhV3FqA9G8sXmSN3r3pusZb13kfCKsJUZorThWxdaUBuUH3CwX0E
EwEKACcFAld1gAUCGwMFCQeGH4AFCwkIBwMFFQoJCAsFFgIDAQACHgECF4AACgkQ
rBB7OGaS2t1aKA//dBaXto7zUFRbJvLgcSE3hJ0ChYZj8Uuo7iYK26mVycyBFQJK
Iv2XYFIJMwK1Rx1Ja1jA6BIKE3aYyx2LVTYU1Hke826dhH+kV2qN9C6t/VmYpV4b
n/i64po7EzD17ykrm5gB+z47uCvyC79/r8Ouns1mTf/JUql/hYJj2hf1MDWr07/X
Wc1zBLjlT93tg/43Vgz1wFdrUOcNx1+bQGXKTOi4AXSp3UvCL+2YsQ2/JsPf7ZzZ
awSuUVkZJrllp87S1MhZeHmTrCHkV1FHJyudLJErcH0337Jpd8xDRek7Klrx2pAS
cKIiyeAMIk/G1OuExYqEEVFQzMr9ZlMNuhNBJAMr5hVGsTgrwy2hlIrBktXav07d
OleSlsqw9ViZ7F6t90x5l+NkwXVsLsGYSzuNyIfomNuoCgA+cfM3TjzVp41qsg1J
WJc9Bavaan2pKF6Lb9Fq8u3HZk2u+YZbvZkqkXwTdZZQOkEmoVV4Y1G86bdpmPyj
8eV7C02NxPii4l+qV8qJQu/6DsA0QwMtBMUNODm3BF2+ZmUHuhMGxq9/4vDE8heE
ffYhHtNftV6JwwzGZmeZkrYA1P9AGLeVp/6iNUe8H5/oPvh4s2rRmqN+L/dQU1ix
iOAT5iAKoRQGkduXrWc4fAY5KxDB9qna4oqXO6QP8rEflI8ELcNgYEj4oJvOwU0E
V3WABQEQALzM0Cs9Zvd08xOEvbEBj59LrS9d0HVKQ61gmkNakWC+jR35VD6FXpe6
UYAcBLrEbVYfKw9P0p6MhFKAsb570JoznKGzE1rVYUZQzhD0RKje35rvkajvEcjG
AWMLTjr87pWHeD0389ER64bzORncfa/l+YP56PI+CThb2wUvTTONGJkPQUpVhH+P
256cQL/Y0Fwu4XLerpwN+YKgMQ47raRcydobPeSfMQr9fVKRyOzFEOrvNpCVDUqi
77d0gLDLjHlIlDyOX5554S8XYLb91eYOiFvnu2pTCKiiExRCSYK29mAQePKlTCCn
QxOjbmBbGS8mVIkpQ5vpvXvzpY3JIjMXaDGqWSQSYGXhECyxCR5eOtKYbCwwPIc2
rIl5gW6yXyw9pKmj5XafTP7YHTvRSr7CZ/VLkDkWl6AfQ9nPOg1mjwjpDFpmN71h
JlSKMaZkh0QGV5FW3dK+GLwxiWdqx3htbZErWyvumWQF/xBF7puKJBEXcoM5KfkJ
```

```
uZekBwcnVkfNFF2RdkM1ALq8InGzLXc7ROuEm0BXVirfju7JRtWLb3UhJWCuhRW2
muyYegSTkag5MduD1IJK37GL8WIlAL65taYgZegUoxHdSaEOefOhspxuduz8d33z
UV1WCFhi+r/+BMCQmTRbF8ao7fTC1dGd084DRP6qE/dMT4u0ZEn7ABEBAAHCwXwE
GAEKACYCGwwWIQT+uSCfLy8/RmSEHlWsEHs4ZpLa3QUCZwAXCwUJEWvKhQAKCRCs
EHs4ZpLa3XtzD/9dwi1qffV70UTq8w/21jn1owHp09jxP7WHTmPWHE0BW5yFIWlV
A1gKN6Ym0dw+LvS5WOKJaRnyewUyBxWvZsn6Wlb5qzY7nmCOKJpYtuCUPwiqjXWP
EM8c/v0MojSuwMOXBAViLvOFhgdUrHn1lk962XvWAW++4DXFh2deaV0163IFMRmO
PNPDAiPWBVqvBANIh2sLRZ5gd1BXwpVrd+x8tzyr69YrN7hutPlCyPEUM9//mcEh
vFPsbW/iOx/foCE3NXhQm/rSMKecVn5csXBV2JOlMzi+8txYNrSBLkjbSB1AvTQ1
aG3+nCNCgM2XDLyoj0IrgZ1To4Ay5gmTOR+msY/cfoIuKFYenmtxy6jM8o5uSZHg
hoClrx9IA98hhGQ73G2r5EDpXuU/uCXn53Sswj65bl9IssfqEIoji/FonkkpEgeg
bGXFDUnrhicDO/WOzqpXf2Fa0DQWY+Vc/pt52ftBFgwzCNIUYDKUhCHPnZ0wtLtd
N2fkXHNiCavCDZlOud7FHHwmRNdj2q1uKxe4m+pFYmKwAU/H+Htkz9Gjsj+ZKedY
nnfai2s2gQOrbfwvV9VdhCWSuLK17ZnGTtiJuOUQIlV8n6QQJpohd3mVgmynu6gQ
uKw0YS2RuEUFv0vOg2tASA+4EM/SBUpGhudODLA4b5wO4gKmh1B1HqQrIsLBfAQY
AQoAJgIbDBYhBP65IJ8vLz9GZIQeVawQezhmktrdBQJlJEokBQkPj/2fAAoJEKwQ
ezhmktrdwMAP/RpFylIL4yhgscBOEnQ7e3No8OraNk0z/YhSd125N/uQVEU94JGQ
rrvQ+4Lfve2laPweBDO18/A0CsmOyHPVQMA0a2vx8ItVdIcNc8iFkP4AJ1922lOq
i0Vh0b1UeZnlfK9+Qvq4PQ2lhWJr0uzyL/S38REsAT1I25sfJOP+RCaR1MH9dm85
E56Lee6uZR8SkGuiL6kGpPh6fWTNij3bICjth1iSSCL2HCOW8lvcwSldDu2EfILU
QCSqfSG7bF8dFk+nKhzhVXOUks3XGjLdICxZewU5ycryitpfRgARgZs2A43gshdi
fiKaX6Ksan03uhKDrLhDHNj2y07PUrFo8ggtlRpV/PrlB/UqCsC9FUOixbD+n4ZF
Sqov2qwelLj0f4mZ6yiLsTDUOFPrdkOlHTJZl7AF0zXZMM6CvaCUaJCKx9GVdSrR
+LI4wLQonPrTnXavhkC4intlqSX8ZQNLhEggdE8YwMEJn59R/nVIT3i5WzYph5R9
P4Vz3Yn7jRqM8wAyEbHkA8s45fMRi9akWSw93H5nWukcmfkt3UEbmka3BQg3HKWP
6TvhfI28euM8qqjbPilfkpEBjnChYVk2Rgn0P8zA7Q5kCo293kwJL9c3RDjMPcxI
45ktKvBTZftsDt1Z718LwW7Q3VQiGiKvo1XLMuV7Z51fmydfUPcrnv17wsFlBBgB
CgAPAhsMBQJhMqGaBQkLnlUVAAoJEKwQezhmktrdbhwQAITmFb67XIUZswr3TREd
Q7ZCLG4EDyfTsW8n75r6A9OqsR+z68nC2Sm7e8mKQFFPwjHPOhsGhHtCOTZtQk7O
jbwyL4N3uxDyEvOfbckH5WzOejZcG7KKQrqAiWJJ7q6CH/zOnVurySjVyzJpy/wL
WpVAcF/uaW5ZhlFCXqePaEzsUBJ757qsr2ho14BV4seT1RSQ9nneTZ0Hhab3wqXP
4qDTo8+zkTvNo9YbeZ1qj62l1+QGIUBTP5MEdXCuC1e4FQ3f6vnXxmB86cUPx7cl
/y2rIjeiOdkKgPeUjNWWSzxS2jYehL5we7gvaSwmEvJ74pV+/3Hs+TxX39XtYFwj
k9I795idnsS5l1dAW3yoI3HBQsYa3US7bpH4g3yZMkstc3bHJ6X54PMCd8Skb+N3
FE8+zGduDmDTKitumiWVVxEFGIwsLAcpWPxecI2AMIMGfMheURYsdvD/yvCbCB29
0KwCSrDVkAG9N2VorNzd7KUeTPTMN1bg2d1lF6u5sQeTN5KVaGd7xE1OXME2wA2D
T3+EsAQytriFbcWm3s8Ugbc9BXMmKBfjlvKu6+Fr6Mgvf/txn56M2SyXBCFQ5OFt
qTFuAFIRv+nayk5tx5Eg1iA7u3dbB1jH3yxGH1B7TeQypA5BqD3x72b7vbXkeci3
1Kz035LYoT5/yTK5sGvacIvCwsFlBBgBCgAPAhsMBQJgmrz3BQkLBnByAAoJEKwQ
ezhmktrdLmgP/1FkWkYhxACnkagRv09mpPl2STbu0B3zYKFBALm/Wa7vKDz18dgC
S3BxDSlpnhZS8QA3VjmbOAZvaDnsN1UJ0f3Qao5136G/UXPnmFIwN612szP0K6nF
PEsotzIzRlJo3S+WkBfiKaQDIDgSxtUxJzOwufz76xibmKRhJ5ChMDCvxmIaoNle
tKRxFT77OupnnyaQs22UsueqrZJ0resgTVnNeF4A1+lU59pFuAlf971SVLr472LP
Uj8mPJihF2ukL1Hdz3F7+kYlpOJRmLk9fo4dlZHBUPiZ1ML/U2yhQfW+Y6tW71vf
```

```
izAxJWF7se6QT+UT5Pji6cohMSERVoYt8e2jFjs0PiPcrjU3mJEx4hAEEVIbP9RY
eKC4CL/UGYAtJkUjd85vKZUHYr7NWZQKLAKqpAPQUMKrIKLEHuz/doq2CCamstLI
vcBgg8EjLJnl3SBesFt/1DCWZeummqz3omQKRl9EHU2cIzIf0Cv/IEysnmbpSpjZ
DX8Fqjtezoq1qiyrLFR7YN1VDPBCHYfqDagw10nlrWFJqT6VqfslmdMdTBRWYVEB
0GUxrkyI+APdi0M2634/410b1ptkqyTIr1KIg1J/qsSiKVcBZSOYFW/rskxYcPPT
wpKYaycEYtOdkS6FPcnehJOOlB+F32WVq2bs2Ps8we6KhjjaYS4Iv4dwwsFlBBgB
CgAPAhsMBQJe+9W/BQkJZ4k1AAoJEKwQezhmktrdvzMQAI6BBj3c2r4bDpV3TwkX
dQ+UCa/E/zUhFds9XKfGb3a5IzRdPUwT+KrAZyiYrr2NSMOzhl/VtqJL18YCYsxO
Ob/TB1hDM+IZiI5gH0cHKhDYKTnNSGP09P/pJAlvHQend9CdZE9J9jwkczfS+bz6
mVxkxpi73fTDox9duesOLsS2/ntRzA0wqhDdaaavRvhAEf9vavCWVrNZmq22WVsU
lnIPxNWGGzWn85JYI6uAi4f4/ABFkry69/c0cvbr0P8qgCmeCuGmX4f0j7qRg77A
+mSueBDx8RK0O2ol021B7b8IcVizj+lpsRQN0oa+i+mFG+o6vtD1ZYhQude4N5sR
RybcLclxjSCoZs5q9JfTpbB2n7pSf/UD3ytwnt9kpD4Vv9dTGAPB83bjL+QK6e3A
XM10jxFE5jSFSr94E40kK80YcIR5jLqsg2f610ENY5drMSA4zuDFDL1Y2ChfjgjZ
uNoFbPHGt/8DfWTVOochVnikA7ggKjz20+RjvwyrHhRMAftO8MMh9UV28pdL+H53
oOtOV0u5aoTbcNqdYQy9B2Bw4lfmj2fi6Dpl+vnZp6hOm0CWiJVW/dtilppYjuxd
w5Kj+9IxZYaBNYH4l1pMT+BsvMDqGzXxDIL89NnY5BkMvqEKnjXSHGRWYMz0xigf
51YKbfQnEQ1oz5bRQndntRQWwsFlBBgBCgAPBQJXdYAFAhsMBQkHhh+AAAoJEKwQ
ezhmktrdTyEP/0HOVWHwQsaWjMrGj0OOMFzxGUo8SBmYYTBs29VM8wBGDsPkYCje
ZzU16i9iqDpDqxpyqmTigcjHV8CDx/6xsMBLG2yKaKZ4m3+YnOQf/sQkyCvqiyMF
9mS7pDYWy+mPhPuw8TDIfiqgVhzjSpIMFWPqxVjn6KKbPN/QASr3Pf0cuP6qpHG+
NAM6Q5dYkCebyvwzLmg1sVnil6iSyJd1jBj3D34XrgWS9buyxBB2CjIM76WxfNVi
J9zAaPI78X9v6PpDGn0kg6oLzrusrvBjoZknKQm0SZ+41fx6xvrTPs8uPEzevzJB
lkke6kw9+KagY8mrVX1ZenRg+sY/4vxJreYWQeql67ggx+wFjKDcfhZA7m7OLHOD
ysrGVCLcmuinUBaNlHmLDcGYXZ+kMCoXfObpuCVByQmNJgEb47EIFlx/+TEeNHKM
O+22xL1atFzXfkEVZck+NghLZyFDhS3g1bma7puU7r752uiJjA6Iv8+kHDXi+/V7
GNpuiEFUYh69QQ2//CS5H51osC/Bkb9evSn/Lp8dMubtWAaXDGJMgw9vqZ55N02N
K0fvF/IKHnGkvH28rv0OPCvOWTA/MClv28y0PrSvcvMXnduLtkBEX7TISMPW+n+O
Ta63/z4YFfEZ7sFLrEm3Q3vJMN3mE5i3cw+JGXPSu0nTtgqk/oZv//SS
=bboB
-----END PGP PUBLIC KEY BLOCK-----
```

Historical keys

> **Important**
>
> New keys are created before the previous ones expire. As a result, at any given moment in time, more than one key may be valid. Keys are used to sign artifacts starting the day they are created, so use the more recently-issued key when the validities of keys overlap.

The following table shows older OpenPGP key information for releases of the SDK for Java 1x and SDK for Java 2.x.

Key ID	0xAC107B386692DADD
Type	RSA
Size	4096/4096
Created	2016-06-30
Expires	2024-10-08
User ID	AWS SDKs and Tools <aws-dr-tools@amazon.com>
Key fingerprint	FEB9 209F 2F2F 3F46 6484 1E55 AC10 7B38 6692 DADD

To copy the following OpenPGP public key for the SDK for Java to the clipboard, select the "Copy" icon in the upper right corner.

```
-----BEGIN PGP PUBLIC KEY BLOCK-----

xsFNBFd1gAUBEACqbmmFbxdJgz1lD7wrlskQA1LLuSAC4p8ny9u/D2zLR8Ynk3Yz
mzJuQ+Kfjne2t+xTDex6MPJlMYpOviSWsX2psgvdmeyUpW9ap0lrThNYkc+W5fRc
buFehfbi9LSATZGJi8RG0sCCr5FsYVz0gEk85M2+PeM24cXhQIOZtQUjswX/pdk/
KduGtZASqNAYLKROmRODzUuaokLPo24pfm9bnr1RnRtwt5ktPAA5bM9ZZaGKriej
kT2lPffbBjp8F5AZvmGLtNm2Cmg4FKBvI04SQjy2jjrQ3wBzi5Lc9HTxDuHK/rtV
u6PewUe2WPlnxlXenhMZU1UK4YoSB9E9StQ2VxQiySLHSdxR7Ma4WgYdVLn9bOie
nj3QxLuQ1ZUKF79ES6JaM4tOz1gGcQeU1+UklgjFLuKwmzWRdEIFfxMyvH6qgKnd
U+DioH5mcUwhwffAAsuIJyAdMIEUYh7IfzJJXQf+fF+XfOCl6byOJFWrIGQkAzMu
CEvaCfwtHC2Lpzo33/WRFeMAuzzd0QJ4uz4xFFvaSOSZHMLHWI9YV/+Pea3X99Ms
0Nlek/LolAJh67MynHeVBOHKrq+fluorWepQivctzN6Y1NOkx5naTPGGaKWK7G2q
TbcY5SMnkIWfLFSougj0Fvmjczq8iZRwYxWA+i+LQvsR9WEXEiQffIWRoQARAQAB
zsFNBFd1gAUBEAC8zNArPWb3dPMThL2xAY+fS60vXdB1SkOtYJpDWpFgvo0d+VQ+
hV6XulGAHAS6xG1WHysPT9KejIRSgLG+e9CaM5yhsxNa1WFGUM4Q9ESo3t+a75Go
7xHIxgFjC046/O6Vh3g9N/PREeuG8zkZ3H2v5fmD+ejyPgk4W9sFL00zjRiZD0FK
VYR/j9uenEC/2NBcLuFy3q6cDfmCoDEOO62kXMnaGz3knzEK/X1SkcjsxRDq7zaQ
lQ1Kou+3dICwy4x5SJQ8jl+eeeEvF2C2/dXmDohb57tqUwioohMUQkmCtvZgEHjy
pUwgp0MTo25gWxkvJlSJKUOb6b1786WNySIzF2gxqlkkEmBl4RAssQkeXjrSmGws
```

```
MDyHNqyJeYFusl8sPaSpo+V2n0z+2B070Uq+wmf1S5A5FpegH0PZzzoNZo8I6Qxa
Zje9YSZUijGmZIdEBleRVt3Svhi8MYlnasd4bW2RK1sr7plkBf8QRe6biiQRF3KD
OSn5CbmXpAcHJ1ZHzRRdkXZDNQC6vCJxsy13O0TrhJtAV1Yq347uyUbVi291ISVg
roUVtprsmHoEk5GoOTHbg9SCSt+xi/FiJQC+ubWmIGXoFKMR3UmhDnnzobKcbnbs
/Hd981FdVghYYvq//gTAkJk0WxfGqO30wtXRndPOA0T+qhP3TE+LtGRJ+wARAQAB
wsFlBBgBCgAPBQJXdYAFAhsMBQkHhh+AAAoJEKwQezhmktrdTyEP/0HOVWHwQsaW
jMrGj0OOMFzxGUo8SBmYYTBs29VM8wBGDsPkYCjeZzU16i9iqDpDqxpyqmTigcjH
V8CDx/6xsMBLG2yKaKZ4m3+YnOQf/sQkyCvqiyMF9mS7pDYWy+mPhPuw8TDIfiqg
VhzjSpIMFWPqxVjn6KKbPN/QASr3Pf0cuP6qpHG+NAM6Q5dYkCebyvwzLmg1sVni
l6iSyJd1jBj3D34XrgWS9buyxBB2CjIM76WxfNViJ9zAaPI78X9v6PpDGn0kg6oL
zrusrvBjoZknKQm0SZ+41fx6xvrTPs8uPEzevzJBlkke6kw9+KagY8mrVX1ZenRg
+sY/4vxJreYWQeql67ggx+wFjKDcfhZA7m7OLHODysrGVCLcmuinUBaNlHmLDcGY
XZ+kMCoXfObpuCVByQmNJgEb47EIFlx/+TEeNHKMO+22xL1atFzXfkEVZck+NghL
ZyFDhS3g1bma7puU7r752uiJjA6Iv8+kHDXi+/V7GNpuiEFUYh69QQ2//CS5H51o
sC/Bkb9evSn/Lp8dMubtWAaXDGJMgw9vqZ55N02NK0fvF/IKHnGkvH28rv0OPCvO
WTA/MClv28y0PrSvcvMXnduLtkBEX7TISMPW+n+OTa63/z4YFfEZ7sFLrEm3Q3vJ
MN3mE5i3cw+JGXPSu0nTtgqk/oZv//SS
=Z9u3
-----END PGP PUBLIC KEY BLOCK-----
```

Document History

This page lists important changes to the AWS SDK for Java Developer Guide over the course of its history.

This guide was published on: October 5, 2024.

October 5, 2024

- Update current OpenPGP key information.

September 4, 2024

Add information about AWS account-based endpoints for DynamoDB. See, the section called "Use AWS account-based endpoints".

May 21, 2024, 2024

Remove instructions to set `networkaddress.cache.ttl` security property by using a java command-line system property. See, How to set the JVM TTL.

January 12, 2024

Add banner that announces the end of support for AWS SDK for Java v1.x.

December 6, 2023

- Provide current OpenPGP key.

March 14, 2023

- Updated guide to align with the IAM best practices. For more information, see Security best practices in IAM.

July 28, 2022

- Added an alert that EC2-Classic is retiring on August 15, 2022.

Mar 22, 2018

- Removed managing Tomcat sessions in DynamoDB example as that tool is no longer supported.

Nov 2, 2017

- Added cryptography examples for Amazon S3 encryption client, including new topics: Use Amazon S3 client-side encryption and Amazon S3 client-side encryption with AWS KMS managed keys and Amazon S3 client-side encryption with client master keys.

Apr 14, 2017

- Made a number of updates to the Amazon S3 Examples Using the AWS SDK for Java section, including new topics: Managing Amazon S3 Access Permissions for Buckets and Objects and Configuring an Amazon S3 Bucket as a Website.

Apr 04, 2017

- A new topic, Enabling Metrics for the AWS SDK for Java describes how to generate application and SDK performance metrics for the AWS SDK for Java.

Apr 03, 2017

- Added new CloudWatch examples to the CloudWatch Examples Using the AWS SDK for Java section: Getting Metrics from CloudWatch, Publishing Custom Metric Data, Working with CloudWatch Alarms, Using Alarm Actions in CloudWatch, and Sending Events to CloudWatch

Mar 27, 2017

- Added more Amazon EC2 examples to the Amazon EC2 Examples Using the AWS SDK for Java section: Managing Amazon EC2 Instances, Using Elastic IP Addresses in Amazon EC2, Use regions and availability zones, Working with Amazon EC2 Key Pairs, and Working with Security Groups in Amazon EC2.

Mar 21, 2017

- Added a new set of IAM examples to the IAM Examples Using the AWS SDK for Java section: Managing IAM Access Keys, Managing IAM Users, Using IAM Account Aliases, Working with IAM Policies, and Working with IAM Server Certificates

Mar 13, 2017

- Added three new topics to the Amazon SQS section: Enabling Long Polling for Amazon SQS Message Queues, Setting Visibility Timeout in Amazon SQS, and Using Dead Letter Queues in Amazon SQS.

Jan 26, 2017

- Added a new Amazon S3 topic, Using TransferManager for Amazon S3 Operations, and a new Best Practices for AWS Development with the AWS SDK for Java topic in the Using the AWS SDK for Java section.

Jan 16, 2017

- Added a new Amazon S3 topic, Managing Access to Amazon S3 Buckets Using Bucket Policies, and two new Amazon SQS topics, Working with Amazon SQS Message Queues and Sending Receiving and Deleting Amazon SQS Messages.

Dec 16, 2016

- Added new example topics for DynamoDB: Working with Tables in DynamoDB and Working with Items in DynamoDB.

Sep 26, 2016

- The topics in the **Advanced** section have been moved into Using the AWS SDK for Java, since they really are central to using the SDK.

Aug 25, 2016

- A new topic, Creating Service Clients, has been added to Using the AWS SDK for Java, which demonstrates how to use *client builders* to simplify the creation of AWS service clients.

  The AWS SDK for Java Code Examples section has been updated with new examples for S3 which are backed by a repository on GitHub that contains the complete example code.

May 02, 2016

- A new topic, Asynchronous Programming, has been added to the Using the AWS SDK for Java section, describing how to work with asynchronous client methods that return `Future` objects or that take an `AsyncHandler`.

Apr 26, 2016

- The *SSL Certificate Requirements* topic has been removed, since it is no longer relevant. Support for SHA-1 signed certificates was deprecated in 2015 and the site that housed the test scripts has been removed.

Mar 14, 2016

- Added a new topic to the Amazon SWF section: Lambda Tasks, which describes how to implement a Amazon SWF workflow that calls Lambda functions as tasks as an alternative to using traditional Amazon SWF activities.

Mar 04, 2016

- The Amazon SWF Examples Using the AWS SDK for Java section has been updated with new content:
  - Amazon SWF Basics- Provides basic information about how to include SWF in your projects.
  - Building a Simple Amazon SWF Application- A new tutorial that provides step-by-step guidance for Java developers new to Amazon SWF.
  - Shutting Down Activity and Workflow Workers Gracefully- Describes how you can gracefully shut down Amazon SWF worker classes using Java's concurrency classes.

Feb 23, 2016

- The source for the AWS SDK for Java Developer Guide has been moved to aws-java-developer-guide.

Dec 28, 2015

- the section called "Set the JVM TTL for DNS name lookups" has been moved from **Advanced** into Using the AWS SDK for Java, and has been rewritten for clarity.

  Using the SDK with Apache Maven has been updated with information about how to include the SDK's bill of materials (BOM) in your project.

Aug 04, 2015

- *SSL Certificate Requirements* is a new topic in the Getting Started section that describes AWS' move to SHA256-signed certificates for SSL connections, and how to fix early 1.6 and previous Java environments to use these certificates, which are *required* for AWS access after September 30, 2015.

  > **Note**
  >
  > Java 1.7+ is already capable of working with SHA256-signed certificates.

May 14, 2014

- The introduction and getting started material has been heavily revised to support the new guide structure and now includes guidance about how to Set up AWS Credentials and Region for Development.

  The discussion of code samples has been moved into its own topic in the Additional Documentation and Resources section.

  Information about how to view the SDK revision history has been moved into the introduction.

May 9, 2014

- The overall structure of the AWS SDK for Java documentation has been simplified, and the Getting Started and Additional Documentation and Resources topics have been updated.

  New topics have been added:

  - Working with AWS Credentials- discusses the various ways that you can specify credentials for use with the AWS SDK for Java.

- Using IAM Roles to Grant Access to AWS Resources on Amazon EC2- provides information about how to securely specify credentials for applications running on EC2 instances.

Sep 9, 2013

- This topic, *Document History*, tracks changes to the AWS SDK for Java Developer Guide. It is intended as a companion to the release notes history.