Guía para desarrolladores de la versión 1.x

AWS SDK for Java 1.x

AWS SDK for Java 1.x: Guía para desarrolladores de la versión 1.x

Table of Contents

Anunciamos la próxima versión end-of-support para AWS SDK for Java (v1). Se recomienda que migre a AWS SDK for Java versión 2. Para ver las fechas, los detalles adicionales y la información sobre cómo realizar la migración, consulte el anuncio enlazado.

Las traducciones son generadas a través de traducción automática. En caso de conflicto entre la traducción y la version original de inglés, prevalecerá la version en inglés.

Guía para desarrolladores. AWS SDK for Java 1.x

El AWS SDK for Java proporciona una API de Java para los servicios de AWS. Con el SDK, le resultará fácil crear aplicaciones Java que funcionen con Amazon S3, Amazon EC2, DynamoDB y otras soluciones. Añadimos periódicamente nuevos servicios a AWS SDK for Java. Para obtener una lista de los servicios admitidos y las versiones de API que se incluyen con cada versión del SDK, consulte las notas de la versión correspondientes a la versión con la que está trabajando.

Versión 2 del SDK publicada

Eche un vistazo al nuevo AWS SDK for Java 2.x en https://github.com/aws/aws-sdk-java-v2/. Incluye muchas de las características esperadas, como una forma de conectar una implementación HTTP. Para empezar consulte la Guía para desarrolladores del AWS SDK for Java 2.x.

Documentación y recursos adicionales

Además de esta guía, estos son algunos otros recursos online útiles para los desarrolladores de AWS SDK for Java:

- Referencia de la API de AWS SDK for Java
- Blog para desarrolladores de Java
- Foros para desarrolladores de Java
- GitHub:
  - Código fuente de documentación
  - Problemas de documentación
  - Código fuente del SDK
  - Problemas del SDK
  - Ejemplos de SDK
  - Canal de Gitter
- Con la AWS Code Sample Catalog
- @awsforjava (Twitter)
- notas de la versión

Compatibilidad con el IDE de Eclipse

Si desarrolla código utilizando el IDE de Eclipse, puede utilizar AWS Toolkit for Eclipse para añadir AWS SDK for Java a un proyecto de Eclipse existente o crear un nuevo proyecto de AWS SDK for Java. El conjunto de herramientas permite también crear y cargar funciones Lambda, lanzar y monitorizar instancias Amazon EC2 y administrar usuarios y grupos de seguridad de IAM, e incluye un editor de plantillas de AWS CloudFormation y otro contenido.

Para ver la documentación, consulte la Guía del usuario de AWS Toolkit for Eclipse.

Desarrollo de aplicaciones para Android

Si es un desarrollador de Android, Amazon Web Services publica un SDK diseñado específicamente para el desarrollo en Android: Amplify Android (AWS SDK para Android).

Consulta del historial de revisiones del SDK

Para ver el historial de versiones de AWS SDK for Java, incluidos los cambios y los servicios compatibles en cada versión del SDK, consulte las notas de la versión del SDK.

Creación de documentación de referencia de Java para versiones del SDK anteriores

La Referencia de la API AWS SDK for Java representa la compilación más reciente de la versión 1.x del SDK. Si utiliza una compilación anterior de la versión 1.x, puede acceder a la documentación de referencia del SDK correspondiente a la versión que está utilizando.

La forma más sencilla de crear la documentación es utilizar la herramienta de compilación Maven de Apache. Descargue e instale Maven primero si todavía no lo tiene en su sistema y, a continuación, utilice las siguientes instrucciones para crear la documentación de referencia.

1. Busque y seleccione la versión del SDK que está utilizando en la página de versiones del repositorio de SDK en GitHub.
2. Elija el enlace `zip` (para la mayoría de las plataformas, incluido Windows) o `tar.gz` (Linux, macOs O Unix) para descargar el SDK en su equipo.
3. Extraiga el archivo en un directorio local.

4. En la línea de comandos, vaya al directorio donde desempaquetó los archivos y escriba lo siguiente.

```
mvn javadoc:javadoc
```

5. Una vez realizada la compilación, encontrará la documentación HTML generada en el directorio `aws-java-sdk/target/site/apidocs/`.

Introducción

Esta sección proporciona información sobre cómo instalar, configurar y utilizar AWS SDK for Java.

Temas

Configuración básica para trabajar con Servicios de AWS

Información general

Para desarrollar con éxito aplicaciones que accedan a Servicios de AWS utilizando el AWS SDK for Java, se requieren las siguientes condiciones:

- Debe poder iniciar sesión en el portal de acceso a AWS disponible en el AWS IAM Identity Center.
- Los permisos del rol de IAM configurados para el SDK deben permitir el acceso a los Servicios de AWS que requiera su aplicación. Los permisos asociados a la política administrada de AWS PowerUserAccess son suficientes para la mayoría de las necesidades de desarrollo.
- Un entorno de desarrollo con los siguientes elementos:
  - Archivos de configuración compartidos que se configuran de una de las siguientes maneras:
    - El archivo `config` contiene un perfil predeterminado que especifica un Región de AWS.
    - El archivo `credentials` contiene credenciales temporales como parte de un perfil predeterminado.
  - Una instalación adecuada de Java.
  - Una herramienta de automatización de compilaciones, como Maven o Gradle.
  - Un editor de texto para trabajar con código.
  - (Opcional, pero recomendado) Un IDE (entorno de desarrollo integrado) como IntelliJ IDEA Eclipse o NetBeans.

Si utiliza un IDE, también puede integrar AWS Toolkit para trabajar más fácilmente con Servicios de AWS. El AWS Toolkit para IntelliJ y el AWS Toolkit for Eclipse son dos kits de herramientas que puede utilizar para el desarrollo de Java.

> **Important**
>
> En las instrucciones de esta sección de configuración se supone que usted o su organización utilizan el Centro de identidad de IAM. Si su organización utiliza un proveedor de identidad externo que funciona de forma independiente del Centro de identidades de IAM, averigüe cómo puede obtener credenciales temporales para que las utilice el SDK para Java. Siga estas instrucciones para añadir credenciales temporales al archivo `~/.aws/credentials`. Si su proveedor de identidad agrega credenciales temporales automáticamente al archivo `~/.aws/credentials`, asegúrese de que el nombre del perfil sea `[default]` para que no necesite proporcionarlo al SDK o AWS CLI.

Inicie sesión en el portal de acceso a AWS

El portal de acceso a AWS es la ubicación web en la que se inicia sesión manualmente en el Centro de identidades de IAM. El formato de la URL es `d-xxxxxxxxxx.awsapps.com/start` o *`your_subdomain`*`.awsapps.com/start`.

Si no está familiarizado con el portal de acceso a AWS, siga las orientaciones para el acceso a cuentas en el tema de Paso 1 de la autenticación del Centro de identidades IAM de la Guía de referencia de SDK y herramientas de AWS. No siga el paso 2 porque el AWS SDK for Java 1.x no admite la actualización automática de los tokens ni la recuperación automática de las credenciales temporales para el SDK que se describen en el paso 2.

Configurar los archivos de configuración compartidos

Los archivos de configuración compartidos residen en la estación de trabajo de desarrollo y contienen los ajustes básicos que utilizan todos los AWS SDK y la AWS Command Line Interface (CLI). Los archivos de configuración compartidos pueden contener varios ajustes, pero estas instrucciones configuran los elementos básicos necesarios para funcionar con el SDK.

Configuración del archivo compartido **`config`**

El ejemplo siguiente muestra el contenido de un archivo `config` compartido.

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

Para fines de desarrollo, use la Región de AWS más cercana al lugar donde planea ejecutar el código. Para obtener una lista de los códigos de región que se van a usar en el archivo `config`, consulte la guía Referencia general de Amazon Web Services. El ajuste `json` del formato de salida es uno de varios valores posibles.

Siga las instrucciones de esta sección para crear el archivo `config`.

Configure credenciales temporales para el SDK.

Después de tener acceso a un rol Cuenta de AWS y de IAM a través del portal de acceso a AWS, configure su entorno de desarrollo con credenciales temporales para que el SDK tenga acceso.

Pasos para configurar un archivo **`credentials`** local con credenciales temporales

1. Crear un archivo de `credentials` compartido.
2. En el archivo `credentials`, pegue el siguiente texto de marcador de posición hasta que pegue las credenciales temporales que funcionen.

   ```
   [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. Guarde el archivo. El archivo `~/.aws/credentials` debería existir ahora en su sistema de desarrollo local. Este archivo contiene el perfil [predeterminado] que el SDK para Java utiliza si no se especifica un perfil con nombre específico.
4. Inicie sesión en el portal de acceso de AWS.
5. Siga las instrucciones de Actualizar manualmente las credenciales para copiar el rol de IAM desde el portal de acceso de AWS.

   a. Para el paso 4 de las instrucciones vinculadas, elija el nombre del rol de IAM que le concede acceso para sus necesidades de desarrollo. Este rol suele tener un nombre como PowerUserAccess o Developer.

   b. Para el paso 7, seleccione la opción Agregar manualmente un perfil a su archivo de credenciales de AWS y copie el contenido.

6. Pegue las credenciales copiadas en su archivo `credentials` local y elimine cualquier nombre de perfil que se haya pegado. Su archivo debería parecerse a lo siguiente:

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

7. Guarde el archivo `credentials`.

El SDK para Java accederá a estas credenciales temporales cuando cree un cliente de servicio y las utilizará para cada solicitud. La configuración del rol de IAM elegido en el paso 5a determina el tiempo de validez de las credenciales temporales. La duración máxima es de doce horas.

Cuando las credenciales temporales caduquen, repita los pasos del 4 al 7.

Instalar un entorno de desarrollo de Java.

AWS SDK for Java requiere J2SE Development Kit 6.0 o una versión posterior. Puede descargar el software de Java más reciente en http://www.oracle.com/technetwork/java/javase/downloads/.

> **Important**
>
> La versión de Java 1.6 (JS2E 6.0) no admite certificados SSL firmados con SHA256, que son necesarios para todas las conexiones HTTPS con AWS después del 30 de septiembre de 2015.
> Las versiones de Java 1.7 o más recientes incluyen certificados actualizados y no les afecta este problema.

Selección de una JVM

Para conseguir el mejor rendimiento de sus aplicaciones basadas en servidor con AWS SDK for Java, le recomendamos que utilice la versión de 64 bits de la máquina virtual de Java (JVM). Esta JVM se ejecuta únicamente en modo servidor, aunque especifique la opción `-Client` en tiempo de ejecución.

El uso de la versión de 32 bits de la JVM con la opción `-Server` en tiempo de ejecución debería proporcionar un desempeño similar al de la JVM de 64 bits.

Maneras de obtener el AWS SDK for Java

Requisitos previos

Para usar AWS SDK for Java, debe tener:

- Debe poder iniciar sesión en el portal de acceso a AWS disponible en el AWS IAM Identity Center.
- Una instalación adecuada de Java.
- Credenciales temporales configuradas en su archivo `credentials` compartido local.

Consulte el tema the section called “Configuración básica” para obtener instrucciones sobre cómo configurar el uso del SDK para Java.

Usar una herramienta de compilación para gestionar las dependencias del SDK para Java

Le recomendamos usar Apache Maven o Gradle con su proyecto para acceder a las dependencias requeridas del SDK para Java. Esta sección describe cómo utilizar esas herramientas.

Descargar y extraer el SDK (no recomendable)

Recomendamos utilizar una herramienta de compilación para acceder al SDK de su proyecto. Sin embargo, puede descargar un jar precompilado de la última versión del SDK.

> **Note**
>
> Para obtener información acerca de cómo descargar y compilar las versiones anteriores del SDK, consulte Instalación de las versiones anteriores del SDK.

1. Descargar el SDK desde https://sdk-for-java.amazonwebservices.com/latest/aws-java-sdk.zip.
2. Después de descargar el SDK, extraiga el contenido en un directorio local.

El SDK contiene los siguientes directorios:

- `documentation`: contiene la documentación de la API (también disponible en la web: Referencia de la API de AWS SDK for Java).

- `lib`: contiene los archivos `.jar` del SDK.
- `samples`: contiene código de ejemplo funcional que muestra cómo utilizar el SDK.
- `third-party/lib`: contiene bibliotecas de terceros utilizadas por el SDK, como Apache commons registro, AspectJ y la plataforma Spring.

Para usar el SDK, añada la ruta completa de los directorios `lib` y `third-party` a las dependencias en el archivo de compilación y a `CLASSPATH` de Java para ejecutar el código.

Compilar versiones anteriores del SDK desde el código fuente (no se recomienda)

Solo la última versión del SDK se proporciona en el formato precompilado como archivo jar descargable. Sin embargo, puede compilar una versión anterior del SDK a través de Apache Maven (código abierto). Maven descargará todas las dependencias necesarias, y compilará e instalará el SDK en un solo paso. Visite http://maven.apache.org/ para obtener instrucciones de instalación y más información.

1. Vaya a la página de GitHub del SDK en: AWS SDK for Java(GitHub).
2. Elija la etiqueta correspondiente al número de versión del SDK que desee. Por ejemplo, `1.6.10`.
3. Haga clic en el botón Download ZIP (Descargar ZIP) para descargar la versión del SDK que ha seleccionado.
4. Descomprima el archivo en un directorio de su sistema de desarrollo. En muchos sistemas, puede utilizar el administrador gráfico de archivos para realizar esta tarea o la utilidad `unzip` en una ventana de terminal.
5. En una ventana de terminal, vaya al directorio donde descomprimió el código fuente del SDK.
6. Compile e instale el SDK con el siguiente comando (Maven) necesario:

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

   El archivo `.jar` resultante se compila en el directorio `target`.
7. (Opcional) Compile la documentación de referencia de la API usando el siguiente comando:

   ```
   mvn javadoc:javadoc
   ```

   La documentación se compila en el directorio `target/site/apidocs/`.

Usar herramientas de compilación

El uso de herramientas de compilación ayuda a gestionar el desarrollo de los proyectos de Java. Hay varias herramientas de compilación disponibles, pero mostramos cómo ponerlas en marcha con dos herramientas de compilación populares: Maven y Gradle. En este tema, se muestra cómo utilizar estas herramientas de compilación para gestionar las dependencias del SDK para Java que necesita para sus proyectos.

Temas

Usar el SDK con Apache Maven

Puede utilizar Apache Maven para configurar y compilar proyectos de AWS SDK for Java o para compilar el propio SDK.

> **Note**
>
> Debe tener Maven instalado para utilizar las instrucciones de este tema. Si aún no está instalado, visite http://maven.apache.org/ para descargarlo e instalarlo.

Creación de un nuevo paquete de Maven

Para crear un paquete de Maven básico, abra un ventana de terminal (línea de comandos) y ejecute:

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

Sustituya org.example.basicapp por el espacio de nombres del paquete completo de su aplicación y myapp por el nombre del proyecto (este será el nombre del directorio de su proyecto).

De forma predeterminada, crea una plantilla de proyecto que utiliza el arquetipo de inicio rápido, lo que constituye un buen punto de partida para muchos proyectos. Hay más arquetipos disponibles;

visite la página Arquetipos de Maven para obtener una lista de arquetipos del paquete. Puede elegir un determinado arquetipo añadiendo el argumento `-DarchetypeArtifactId` al comando `archetype:generate`. Por ejemplo:

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

> **Note**
>
> Encontrará mucha más información sobre la creación y configuración de proyectos en la Guía de introducción a Maven.

Configuración del SDK como una dependencia de Maven

Para utilizar AWS SDK for Java en el proyecto, tendrá que declararlo como una dependencia en el archivo `pom.xml` del proyecto. A partir de la versión 1.9.0, puede importar componentes individuales o todo el SDK.

Especificación de módulos del SDK individuales

Para seleccionar módulos individuales del SDK, utilice la lista de materiales (BOM) de AWS SDK for Java para Maven, con lo que se asegurará de que los módulos que especifique utilicen la misma versión del SDK y que sean compatibles entre sí.

Para utilizar la lista de materiales, añada una sección `<dependencyManagement>` al archivo `pom.xml` de su aplicación, añadiendo `aws-java-sdk-bom` como una dependencia y especificando la versión del SDK que desee utilizar:

```
<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>
```

Para ver la última versión de la lista de materiales de AWS SDK for Java disponible en Maven Central, visite: https://mvnrepository.com/artifact/com.amazonaws/aws-java-sdk-bom. También puede utilizar esta página para ver qué módulos (dependencias) administrados por la lista de materiales puede incluir en la sección `<dependencies>` del archivo `pom.xml` de su proyecto.

Ahora puede seleccionar los módulos individuales del SDK que desea usar en su aplicación. Como ya ha declarado la versión del SDK en la lista de materiales, no necesita especificar el número de versión de cada componente.

```
<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>
```

También puede consultar el AWS Code Sample Catalog para saber qué dependencias utilizar para un servicio de Servicio de AWS determinado. Consulte el archivo POM en un ejemplo de servicio específico. Por ejemplo, si está interesado en las dependencias del servicio AWS S3, consulte el ejemplo completo en GitHub. (Vea el pom en /java/example_code/s3).

Importación de todos los módulos del SDK

Si desea incluir todo el SDK como una dependencia, no utilice el método de lista de materiales; simplemente declárelo en su archivo `pom.xml` de la manera siguiente:

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

```
</dependencies>
```

Compilación del proyecto

Una vez que haya configurado su proyecto, puede compilarlo mediante el comando `package` de Maven:

```
mvn package
```

Esto creará el archivo `0jar` en el directorio `target`.

Compilación del SDK con Maven

Puede utilizar Apache Maven para compilar el SDK desde el origen. Para ello, descargue el código del SDK de GitHub, desempaquételo localmente y, a continuación, ejecute el siguiente comando de Maven:

```
mvn clean install
```

Usar el SDK con Gradle.

Para administrar las dependencias del SDK en su proyecto Gradle importe la BOM de Maven para el AWS SDK for Java en el archivo `build.gradle` de la aplicación.

> **Note**
>
> En los ejemplos siguientes, sustituya *`1.12.529`* en el archivo de compilación por una versión válida del AWS SDK for Java. Busque la última versión disponible en el repositorio central de Maven.

Configuración del proyecto en Gradle 4.6 o posterior

A partir de Gradle 4.6, puede utilizar la característica de soporte de POM mejorada de Gradle para importar archivos de lista de materiales (BOM) declarando una dependencia en una BOM.

1. Si está utilizando Gradle 5.0 o posterior, vaya al paso 2. De lo contrario, habilite la característica IMPROVED_POM_SUPPORT en el archivo `settings.gradle`.

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

2. Añada la BOM a la sección de dependencias del archivo `build.gradle`.

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

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

3. Especifique los módulos del SDK que desea usar en la sección dependencias. Por ejemplo, el siguiente incluye una dependencia para 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 resuelve automáticamente la versión correcta de las dependencias del SDK con la información de la BOM.

El siguiente es un ejemplo de un archivo `build.gradle` completo que incluye una dependencia para 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**
>
> En el ejemplo anterior, sustituya la dependencia para Amazon S3 por las dependencias de los servicios de AWS que utilizará en su proyecto. Los módulos (dependencias) que administra la BOM de AWS SDK for Java se enumeran en el repositorio central de Maven.

Configuración del proyecto para versiones de Gradle anteriores a 4.6

Las versiones de Gradle anteriores a 4.6 carecen de soporte de BOM nativo. Para administrar dependencias de AWS SDK for Java para su proyecto, use el complemento de administración de dependencias de Spring para Gradle para importar la BOM de Maven para el SDK.

1. Añada el complemento de administración de dependencias a su archivo `build.gradle` de aplicación.

```
buildscript {
    repositories {
        mavenCentral()
    }
    dependencies {
        classpath "io.spring.gradle:dependency-management-plugin:1.0.9.RELEASE"
    }
}

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

2. Añada la lista de materiales a la sección dependencyManagement del archivo.

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

3. Especifique los módulos del SDK que va a usar en la sección dependencies. Por ejemplo, en el siguiente se incluye una dependencia para Amazon S3.

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

Gradle resuelve automáticamente la versión correcta de las dependencias del SDK con la información de la BOM.

El siguiente es un ejemplo de un archivo `build.gradle` completo que incluye una dependencia para 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**
>
> En el ejemplo anterior, sustituya la dependencia para Amazon S3 por las dependencias del servicio de AWS que utilizará en su proyecto. Los módulos (dependencias) que administra la BOM de AWS SDK for Java se enumeran en el repositorio central de Maven.

Para obtener más información sobre cómo especificar las dependencias del SDK mediante la BOM, consulte Uso del SDK con Apache Maven.

Configurar credenciales temporales de AWS y Región de AWS para desarrollo

Para conectarse a cualquiera de los servicios compatibles con AWS SDK for Java, debe proporcionar las credenciales de AWS. Los SDK de AWS y las CLI de AWS usan cadenas de proveedores para buscar las credenciales de AWS en diversos lugares; por ejemplo, en las variables de entorno del sistema o del usuario y en los archivos de configuración de AWS.

En este tema se proporciona información básica acerca de cómo configurar las credenciales de AWS para el desarrollo de aplicaciones locales mediante AWS SDK for Java. Si necesita configurar las credenciales para utilizarlas en una instancia EC2 o si utiliza el IDE de Eclipse para desarrollo, consulte los siguientes temas:

- Cuando utilice una instancia EC2, cree un rol de IAM y conceda acceso a la instancia EC2 a dicho rol como se indica en Uso de roles de IAM para conceder acceso a los recursos de AWS en Amazon EC2.
- Configurar credenciales de AWS en Eclipse mediante AWS Toolkit for Eclipse. Consulte Configuración de credenciales de AWS en la Guía del usuario de AWS Toolkit for Eclipse.

Configurar credenciales temporales

Puede configurar las credenciales temporales para el AWS SDK for Java de varias maneras, pero a continuación se indican los enfoques recomendados:

- Configure las credenciales temporales en el archivo de perfil de credenciales de AWS del sistema local, situado en:
  - `~/.aws/credentials` en Linux, macOS o Unix
  - `C:\Users\USERNAME\.aws\credentials` en Windows

  Consulte las the section called “Configure credenciales temporales para el SDK.” en esta guía para sabre cómo obtener sus credenciales temporales.
- Establezca las variables de entorno `AWS_ACCESS_KEY_ID`, `AWS_SECRET_ACCESS_KEY` y `AWS_SESSION_TOKEN`.

  Para establecer estas variables en Linux, MacOS, o Unix, utilice :

```
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
```

  Para establecer estas variables en Windows, utilice :

```
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
```

- Para una instancia EC2, especifique un rol de IAM y conceda a la instancia EC2 acceso a dicho rol. Consulte los roles de IAM para Amazon EC2 en la Guía del usuario de Amazon EC2 para instancias de Linux para obtener información detallada sobre su funcionamiento.

Una vez que haya establecido sus credenciales de AWS utilizando alguno de estos métodos, AWS SDK for Java las cargará automáticamente mediante la cadena predeterminada de proveedores de credenciales. Para obtener más información sobre cómo trabajar con credenciales de AWS en sus aplicaciones Java, consulte Uso de credenciales de AWS.

Actualización de credenciales IMDS

El AWS SDK for Java admite seleccionar la actualización de credenciales IMDS en segundo plano cada 1 minuto, independientemente del plazo de vencimiento de la credencial. Esto le permite actualizar las credenciales con más frecuencia y reduce la posibilidad de que no alcanzar IMDS afecte a la disponibilidad de AWS percibida.

```
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();
```

Definir la Región de AWS

Debe definir la región de Región de AWS predeterminada que se usará para obtener acceso a los servicios de AWS con el AWS SDK for Java. Para disfrutar de un rendimiento de red óptimo, elija una región que esté geográficamente cerca de usted (o de sus clientes). Para ver una lista de las regiones, consulte Regiones y puntos de conexión en la Referencia general de Amazon Web Services.

> **Note**
>
> Si no selecciona una región, se usará us-east-1 de forma predeterminada.

Puede utilizar técnicas similares a las de la configuración de credenciales para definir la región de AWS predeterminada:

- Defina la región de Región de AWS en el archivo de configuración de AWS en su sistema local, situado en:
  - ~/.aws/config en Linux, macOS o Unix
  - C:\Users\USERNAME\.aws\config on Windows

  Este archivo debe contener líneas con el siguiente formato:

+

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

+

Sustituya su Región de AWS (por ejemplo, "us-east-1") por su_region_aws.

- Establezca la variable de entorno `AWS_REGION`.

  En Linux, macOS o Unix, utilice :

  ```
  export AWS_REGION=your_aws_region
  ```

  En Windows, use :

  ```
  set AWS_REGION=your_aws_region
  ```

  Donde su_región_de_aws es el nombre de la región de Región de AWS que desee.

Usando el AWS SDK for Java

En esta sección se proporciona información general importante sobre la programación, AWS SDK for Java que se aplica a todos los servicios que pueda utilizar con el SDK.

Para obtener información y ejemplos de programación específicos de un servicio (para Amazon EC2, Amazon S3, Amazon SWF, etc.), consulte Ejemplos de AWS SDK for Java código.

Temas

Mejores prácticas para el AWS desarrollo con AWS SDK for Java

Las siguientes prácticas recomendadas pueden ayudarle a evitar problemas o problemas al desarrollar AWS aplicaciones con AWS SDK for Java. Hemos organizado las prácticas recomendadas por servicio.

S3

Evite ResetExceptions

Al cargar objetos Amazon S3 mediante transmisiones (ya sea a través de un `AmazonS3` cliente o`TransferManager`), es posible que se produzcan problemas de conectividad de red o de tiempo de espera. De forma predeterminada, los AWS SDK for Java intentos de reintentar

realizar transferencias fallidas marcando el flujo de entrada antes del inicio de la transferencia y restableciéndolo antes de volver a intentarlo.

Si la transmisión no admite marcar y restablecer, el SDK lanza una ResetExceptioncuando hay errores transitorios y los reintentos están habilitados.

Práctica recomendada

Le recomendamos que utilice secuencias que admitan operaciones de marcado y restablecimiento.

La forma más fiable de evitar un ResetExceptiones proporcionar datos mediante un archivo o FileInputStream, que AWS SDK for Java puedan gestionar sin verse limitados por los límites de marcar y restablecer.

Si la transmisión no es una, FileInputStreampero admite marcar y restablecer, puedes establecer el límite de marcas mediante el `setReadLimit` método de RequestClientOptions. Su valor predeterminado es 128 KB. Si se establece el valor límite de lectura en un byte mayor que el tamaño de la transmisión, se evitará de forma fiable un ResetException.

Por ejemplo, si el tamaño máximo esperado de una secuencia es de 100 000 bytes, defina el límite de lectura en 100 001 (100 000+1) bytes. Las operaciones de marca y restablecimiento siempre funcionan para 100 000 bytes o menos. Tenga en cuenta que esto puede provocar que algunas secuencias almacenen ese número de bytes en memoria.

Creación de clientes de servicio

Para realizar solicitudes a Amazon Web Services, primero debe crear un objeto de cliente de servicio. La manera recomendada es utilizar el creador de clientes de servicio.

Cada uno Servicio de AWS tiene una interfaz de servicio con métodos para cada acción en la API de servicio. Por ejemplo, la interfaz de servicio de DynamoDB se denomina DBClient. AmazonDynamo Cada interfaz de servicio cuenta con su creador de clientes correspondiente, que puede utilizar para crear una implementación de la interfaz de servicio. La clase de creación de clientes para DynamoDB se denomina DB. AmazonDynamo ClientBuilder

Obtención de un creador de clientes

Para obtener una instancia del creador de clientes, use el método de fábrica estático `standard`, tal y como se muestra en el siguiente ejemplo.

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

Una vez que tenga un creador, puede personalizar las propiedades del cliente mediante el uso de funciones setter Fluent en la API del compilador. Por ejemplo, puede definir una región personalizada y un proveedor de credenciales personalizado, tal y como se indica a continuación.

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

> **Note**
>
> Los métodos `withXXX` Fluent devuelven el objeto `builder` para que pueda encadenar fácilmente las llamadas a los métodos y para simplificar la lectura del código. Después de configurar las propiedades que desee, puede llamar al método `build` para crear el cliente. Una vez que se crea un cliente, este es inmutable y todas las llamadas a `setRegion` o `setEndpoint` producirán un error.

Un creador puede crear varios clientes con la misma configuración. Cuando escriba su aplicación, debe ser consciente de que el creador es mutable y no es seguro para subprocesos.

El código siguiente utiliza el creador como una fábrica de instancias de cliente.

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

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

El generador también expone los setters fluidos para ClientConfigurationy RequestMetricCollector, además, una lista personalizada de RequestHandler 2.

A continuación se presenta un ejemplo completo que invalida todas las propiedades configurables.

```
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();
```

Creación de clientes asíncronos

AWS SDK for Java Tiene clientes asíncronos (o asíncronos) para cada servicio (excepto) y el correspondiente generador de clientes asíncronos para Amazon S3 cada servicio.

Para crear un cliente de DynamoDB asíncrono con el valor predeterminado ExecutorService

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

Además de las opciones de configuración que admite el generador de clientes síncronos (o sincronizados), el cliente asíncrono permite configurar una configuración personalizada ExecutorFactorypara cambiar la que utiliza el cliente asíncrono. `ExecutorService` `ExecutorFactory`es una interfaz funcional, por lo que interopera con las expresiones lambda y las referencias a métodos de Java 8.

Para crear un cliente asíncrono con un ejecutor personalizado

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

Usando DefaultClient

Los creador de clientes síncronos y asíncronos tienen otro método de fábrica denominado `defaultClient`. Este método crea un servicio de cliente con la configuración predeterminada,

utilizando la cadena de proveedores predeterminada para cargar las credenciales y la región Región de AWS. Si las credenciales o la región no se pueden determinar a partir del entorno en el que se ejecuta la aplicación, la llamada a `defaultClient` produce un error. Consulte Trabajar con AWS las credenciales y la Región de AWS selección para obtener más información sobre cómo se determinan las credenciales y la región.

Para crear un cliente de servicio predeterminado

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

Ciclo de vida del cliente

Los clientes de servicio del SDK son seguros para subprocesos y, para obtener el mejor rendimiento, deben tratarse como objetos de larga duración. Cada cliente tiene su propio recurso de grupo de conexiones. Cierre los clientes de forma explícita cuando dejen de ser necesarios para evitar que se filtren recursos.

Para cerrar un cliente de forma explícita, llame al método `shutdown`. Después de llamar a `shutdown`, todos los recursos del cliente se liberan y ya no se puede utilizar el cliente.

Para cerrar un cliente

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

Proporcione credenciales temporales al AWS SDK for Java

Para realizar solicitudes Amazon Web Services, debe proporcionar credenciales AWS temporales para AWS SDK for Java que las utilice cuando llame a los servicios. Puede hacerlo de las siguientes maneras:

- Utilice la cadena predeterminada de proveedores de credenciales (recomendado).
- Utilice un proveedor de credenciales específico o una cadena de proveedores (o cree el suyo propio).
- Proporcione usted mismo las credenciales temporales en código.

Uso de la cadena predeterminada de proveedores de credenciales

Al inicializar un nuevo cliente de servicio sin proporcionar ningún argumento, AWS SDK for Java intenta encontrar credenciales temporales utilizando la cadena de proveedores de credenciales predeterminada implementada por la clase AWSCredentialsProviderChainDefault. La cadena predeterminada de proveedores de credenciales busca las credenciales en este orden:

1. Variables de entorno: `AWS_ACCESS_KEY_ID`, `AWS_SECRET_ACCESS_KEY` y `AWS_SESSION_TOKEN`. AWS SDK for Java Usa la EnvironmentVariableCredentialsProviderclase para cargar estas credenciales.
2. Propiedades del sistema Java: `aws.accessKeyId`, `aws.secretKey` y `aws.sessionToken`. AWS SDK for Java Usa el SystemPropertiesCredentialsProviderpara cargar estas credenciales.
3. Credenciales de Web Identity Token del entorno o contenedor.
4. El archivo de perfiles de credenciales predeterminado, que normalmente se encuentra en `~/.aws/credentials` (la ubicación puede variar según la plataforma) y lo comparten muchos de AWS los SDK y el. AWS CLI AWS SDK for Java Utiliza el ProfileCredentialsProviderpara cargar estas credenciales.

   Puede crear un archivo de credenciales mediante el `aws configure` comando proporcionado por el AWS CLI, o puede crearlo editando el archivo con un editor de texto. Para obtener información sobre el formato del archivo de credenciales, consulte Formato del archivo de credenciales de AWS.
5. Credenciales del contenedor de Amazon ECS: cargadas desde Amazon ECS si se ha establecido la variable de entorno `AWS_CONTAINER_CREDENTIALS_RELATIVE_URI`. AWS SDK for Java Utiliza el ContainerCredentialsProviderpara cargar estas credenciales. Puede especificar la dirección IP para este valor.
6. Credenciales de perfil de instancia: se utilizan en las instancias de EC2 y se entregan a través del servicio de Amazon EC2 metadatos. AWS SDK for Java Utiliza las InstanceProfileCredentialsProviderpara cargar estas credenciales. Puede especificar la dirección IP para este valor.

> **Note**
>
> Las credenciales del perfil de instancia se utilizan únicamente si no se ha establecido `AWS_CONTAINER_CREDENTIALS_RELATIVE_URI`. Consulte EC2 ContainerCredentialsProviderWrapper para obtener más información.

Utilizar credenciales temporales

Para poder utilizar credenciales AWS temporales, deben estar configuradas en al menos una de las ubicaciones anteriores. Para obtener información sobre la configuración de las credenciales, consulte los siguientes temas:

- Para especificar credenciales en el entorno o en el archivo de perfiles de credenciales predeterminado, consulte the section called “Configurar credenciales temporales”.
- Para establecer propiedades del sistema Java, consulte el tutorial System Properties en el sitio web oficial Java Tutorials.
- Para configurar y usar las credenciales del perfil de instancia con sus instancias de EC2, consulte Uso de funciones de IAM para conceder acceso a AWS los recursos en ellas. Amazon EC2

Configurar un perfil de credenciales alternativo

AWS SDK for Java Utiliza el perfil predeterminado de forma predeterminada, pero hay formas de personalizar el perfil que proviene del archivo de credenciales.

Puede usar la variable AWS de entorno Profile para cambiar el perfil cargado por el SDK.

Por ejemplo, en Linux; macOS o Unix ejecutaría el comando siguiente para cambiar el perfil a myProfile.

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

En Windows, usaría el siguiente comando.

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

La configuración de la variable de `AWS_PROFILE` entorno afecta a la carga de credenciales de todos los AWS SDK y herramientas compatibles oficialmente (incluidos el AWS CLI y el AWS Tools for Windows PowerShell). Para cambiar únicamente el perfil de una aplicación Java, puede utilizar la propiedad del sistema `aws.profile` en su lugar.

> **Note**
>
> La variable de entorno prevalece sobre la propiedad del sistema.

Configurar una ubicación del archivo de credenciales alternativa

AWS SDK for Java Carga las credenciales AWS temporales automáticamente desde la ubicación predeterminada del archivo de credenciales. Sin embargo, también puede especificar la ubicación estableciendo la variable de entorno `AWS_CREDENTIAL_PROFILES_FILE` con la ruta completa del archivo de credenciales.

Puede utilizar esta función para cambiar temporalmente la ubicación en la que AWS SDK for Java busca su archivo de credenciales (por ejemplo, configurando esta variable con la línea de comandos). También puede establecer la variable de entorno en el entorno del usuario o del sistema para cambiarlo para el usuario o para todo el sistema.

Para invalidar la ubicación predeterminada del archivo de credenciales

- Defina la variable de `AWS_CREDENTIAL_PROFILES_FILE` entorno en la ubicación del archivo de AWS credenciales.
  - En Linux, macOS o Unix, utilice:

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

  - En Windows, use:

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

Formato de archivo de **`Credentials`**

Siguiendo las instrucciones de la configuración básica de esta guía, el archivo de credenciales debe tener el siguiente formato básico.

```
[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>
```

El nombre de perfil se especifica entre corchetes (por ejemplo, `[default]`), seguido de los campos configurables en ese perfil como pares de clave-valor. Puede tener varios perfiles en su archivo de `credentials`, que puede añadir o editar mediante `aws configure --profile` *`PROFILE_NAME`* para seleccionar el perfil que se va a configurar.

Puede especificar campos adicionales, como `metadata_service_timeout` y `metadata_service_num_attempts`. Estos no se pueden configurar con la CLI; debe editar el archivo manualmente si desea utilizarlos. Para obtener más información sobre el archivo de configuración y sus campos disponibles, consulte Configuración del archivo AWS Command Line Interface en la Guía del AWS Command Line Interface usuario.

Cargar credenciales

Después de definir las credenciales temporales, el SDK las carga mediante la cadena predeterminada de proveedores de credenciales.

Para ello, se crea una instancia de un Servicio de AWS cliente sin proporcionar las credenciales de forma explícita al generador, de la siguiente manera.

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

Especificar un proveedor de credenciales o una cadena de proveedores

Puede especificar un proveedor de credenciales diferente de la cadena de proveedores de credenciales predeterminada mediante el creador del cliente.

Se proporciona una instancia de un proveedor de credenciales o una cadena de proveedores a un creador de clientes que toma una AWSCredentialsProviderinterfaz como entrada. El siguiente ejemplo muestra cómo utilizar credenciales de entorno específicamente.

```
AmazonS3 s3Client = AmazonS3ClientBuilder.standard()
                        .withCredentials(new EnvironmentVariableCredentialsProvider())
                        .build();
```

Para ver la lista completa de proveedores AWS SDK for Java de credenciales y cadenas de proveedores suministrados, consulte Todas las clases de implementación conocidas en. AWSCredentialsProvider

> **Note**
>
> Puede utilizar esta técnica para suministrar proveedores de credenciales o cadenas de proveedores que cree utilizando su propio proveedor de credenciales que implemente la `AWSCredentialsProvider` interfaz o subclasificando la clase. AWSCredentialsProviderChain

Especificar explícitamente credenciales temporales

Si la cadena de credenciales predeterminada o un proveedor o cadena de proveedores específicos o personalizados no funcionan para su código, puede configurar credenciales que proporcione explícitamente. Si ha obtenido credenciales temporales utilizando este método AWS STS, utilice este método para especificar las credenciales de acceso. AWS

1. Cree una instancia de la BasicSessionCredentialsclase y suministre la clave de AWS acceso, la clave AWS secreta y el token de AWS sesión que el SDK utilizará para la conexión.
2. Crea una AWSStaticCredentialsProvidercon el `AWSCredentials` objeto.
3. Configure el creador del cliente con `AWSStaticCredentialsProvider` y compile el cliente.

A continuación, se muestra un ejemplo.

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

Más información

- Regístrese AWS y cree un usuario de IAM
- Configure AWS las credenciales y la región para el desarrollo
- Uso de las funciones de IAM para conceder acceso a AWS los recursos en Amazon EC2

Región de AWS Selección

Las regiones le permiten acceder a AWS los servicios que residen físicamente en un área geográfica específica. Esto puede ser útil para evitar redundancias y para que sus datos y aplicaciones se ejecuten cerca del lugar desde donde usted y sus usuarios obtendrán acceso a ellos.

Comprobación de la disponibilidad del servicio en una región

Para ver si una determinada región Servicio de AWS está disponible en una región, usa el `isServiceSupported` método de la región que quieras usar.

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

Consulte la documentación de la clase Regions para saber las regiones que puede especificar y use el prefijo de punto de enlace del servicio que desea consultar. Cada prefijo de punto de enlace del servicio se define en la interfaz del servicio. Por ejemplo, el prefijo del DynamoDB punto final se define en la AmazonDynamobase de datos.

Selección de una región

A partir de la versión 1.4 del AWS SDK for Java, puedes especificar el nombre de una región y el SDK elegirá automáticamente el punto de conexión adecuado para ti. Para elegir usted mismo el punto de enlace, consulte Selección de un punto de enlace específico.

Para configurar de forma explícita una región, le recomendamos que utilice la enumeración Regions. Esta es una enumeración de todas las regiones disponibles públicamente. Para crear un cliente con una región desde la enumeración, utilice el siguiente código.

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

Si la región que intenta utilizar no está en la enumeración `Regions`, puede configurar la región mediante una cadena que represente el nombre de la región.

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

```
                    .build();
```

> **Note**
>
> Después de compilar un cliente con el creador, este es inmutable y la región no puede modificarse. Si trabaja con varios Regiones de AWS para el mismo servicio, debe crear varios clientes, uno por región.

Selección de un punto de enlace específico

Cada AWS cliente se puede configurar para usar un punto final específico dentro de una región llamando al `withEndpointConfiguration` método al crear el cliente.

Por ejemplo, para configurar el Amazon S3 cliente para que utilice la región de Europa (Irlanda), utilice el siguiente código.

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

Consulte Regiones y puntos de enlace para ver la lista actual de regiones y sus puntos de enlace correspondientes para todos los AWS servicios.

Determinar automáticamente la región desde el entorno

> **Important**
>
> Esta sección solo se aplica cuando se utiliza un generador de clientes para acceder AWS a los servicios. AWS los clientes creados mediante el constructor de clientes no determinarán automáticamente la región a partir del entorno y, en su lugar, utilizarán la región del SDK predeterminada (usEast1).

Cuando se ejecuta en Lambda Amazon EC2 o Lambda, es posible que desee configurar los clientes para que usen la misma región en la que se ejecuta el código. De esta forma, el código se desacopla

del entorno en el que se está ejecutando y es más sencillo implementar la aplicación en varias regiones para reducir la latencia o la redundancia.

Debe utilizar creadores de clientes para que el SDK detecte automáticamente la región en la que se ejecuta el código.

Para utilizar la cadena predeterminada de proveedores de credenciales o regiones para determinar la región a partir del entorno, use el método `defaultClient` del creador del cliente.

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

Es lo mismo que usar `standard` seguido de `build`.

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

Si no configura explícitamente una región mediante los métodos `withRegion`, el SDK consulta la cadena predeterminada de proveedores de regiones para intentar determinar la región que se va a usar.

Cadena predeterminada de proveedores de regiones

A continuación se muestra el proceso de búsqueda de regiones:

1. Cualquier región explícita establecida mediante `withRegion` o `setRegion` en el propio creador prevalece sobre todas las demás.
2. Se comprueba la variable de entorno `AWS_REGION`. Si se ha establecido, se usa esa región para configurar el cliente.

   > **Note**
   >
   > El Lambda contenedor establece esta variable de entorno.

3. El SDK comprueba el archivo de configuración AWS compartido (que normalmente se encuentra en`~/.aws/config`). Si la propiedad region está presente, el SDK la utiliza.
   - La variable de entorno `AWS_CONFIG_FILE` se puede utilizar para personalizar la ubicación del archivo de configuración compartida.
   - La variable de entorno `AWS_PROFILE` o la propiedad del sistema `aws.profile` se pueden utilizar para personalizar el perfil que carga el SDK.

4. El SDK intenta usar el servicio de metadatos de la Amazon EC2 instancia para determinar la región de la Amazon EC2 instancia que se está ejecutando actualmente.
5. Si el SDK todavía no ha encontrado una región en ese momento, la creación del cliente produce una excepción.

Al desarrollar AWS aplicaciones, un enfoque habitual consiste en utilizar el archivo de configuración compartido (que se describe en la sección Uso de la cadena de proveedores de credenciales predeterminada) para establecer la región para el desarrollo local, y utilizar la cadena de proveedores de regiones predeterminada para determinar la región cuando se ejecuta en la AWS infraestructura. Esto simplifica enormemente la creación del cliente y dota de portabilidad a su aplicación.

Tratamiento de excepciones

Entender cómo y cuándo se AWS SDK for Java producen excepciones es importante para crear aplicaciones de alta calidad mediante el SDK. En las siguientes secciones se describen los diferentes casos de excepciones que produce el SDK y cómo tratarlas correctamente.

¿Por qué usar excepciones no controladas?

AWS SDK for Java Utiliza excepciones en tiempo de ejecución (o no comprobadas) en lugar de excepciones comprobadas por los siguientes motivos:

- Para permitir a los desarrolladores un control minucioso de los errores que desean administrar sin obligarles a abordar casos excepcionales que no les preocupan (o que les obligan a detallar su código en exceso)
- Para evitar problemas de escalabilidad inherentes a las excepciones controladas en aplicaciones grandes

En general, las excepciones controladas funcionan bien a pequeña escala, pero pueden ser problemáticas cuando las aplicaciones crecen y se vuelven más complejas.

Para obtener más información sobre el uso de excepciones controladas y no controladas, consulte:

- Excepciones no controladas: la controversia
- The Trouble with Checked Exceptions

- Java's checked exceptions were a mistake (and here's what I would like to do about it)

AmazonServiceException (y subclases)

AmazonServiceExceptiones la excepción más común que experimentará al AWS SDK for Java usar. Esta excepción representa una respuesta de error de un servicio de Servicio de AWS. Por ejemplo, si intenta terminar una Amazon EC2 instancia que no existe, EC2 devolverá una respuesta de error y todos los detalles de esa respuesta de error se incluirán en la `AmazonServiceException` que se arroje. En algunos casos, se produce una subclase de `AmazonServiceException` para permitir a los desarrolladores un control minucioso del tratamiento de casos de error a través de bloques de captura.

Cuando encuentres una`AmazonServiceException`, sabrás que tu solicitud se ha enviado correctamente a esa dirección Servicio de AWS , pero que no se ha podido procesar correctamente. Esto puede ser debido a errores en los parámetros de la solicitud o a problemas en el servicio.

`AmazonServiceException` proporciona información como:

- Código de estado HTTP devuelto
- Código AWS de error devuelto
- Mensaje de error detallado del servicio
- AWS ID de solicitud de la solicitud fallida

`AmazonServiceException`también incluye información sobre si la solicitud fallida fue culpa de la persona que llamó (una solicitud con valores no válidos) o fue culpa Servicio de AWS de la persona que llamó (un error de servicio interno).

AmazonClientException

AmazonClientExceptionindica que se ha producido un problema en el código del cliente de Java, ya sea al intentar enviar una solicitud AWS o al analizar una respuesta desde AWSél. Un `AmazonClientException` es generalmente más grave que uno e indica un problema importante que impide que el cliente realice llamadas de servicio a AWS los servicios. `AmazonServiceException` Por ejemplo, `AmazonClientException` si no hay ninguna conexión de red disponible cuando se intenta llamar a una operación en uno de los clientes. AWS SDK for Java

Programación asíncrona

Puede utilizar métodos síncronos o asíncronos para llamar a las operaciones de los servicios. AWS Los métodos síncronos bloquean la ejecución de los subprocesos hasta que el cliente recibe una respuesta del servicio. Los métodos asíncronos terminan de ejecutarse inmediatamente, devolviendo el control al subproceso que realiza la llamada sin esperar una respuesta.

Como un método asíncrono termina de ejecutarse antes de que haya una respuesta disponible, necesita una forma de obtener la respuesta cuando esté lista. AWS SDK for Java Proporciona dos formas: objetos futuros y métodos de devolución de llamada.

Objetos Future de Java

Los métodos asíncronos AWS SDK for Java devuelven un objeto Future que contiene los resultados de la operación asincrónica en el futuro.

Llame al método `Future` `isDone()` para saber si el servicio ya ha proporcionado un objeto de respuesta. Cuando la respuesta esté lista, podrá obtener el objeto de respuesta llamando al método `Future` `get()`. Puede utilizar este mecanismo para sondear periódicamente los resultados de las operaciones asíncronas mientras su aplicación sigue trabajando en otras cosas.

Este es un ejemplo de una operación asíncrona que llama a una Lambda función y recibe una que puede contener un objeto. `Future` InvokeResult El objeto `InvokeResult` solo se recupera cuando `isDone()` es `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);
    }
}
```

Devoluciones de llamadas asíncronas

Además de utilizar el `Future` objeto Java para supervisar el estado de las solicitudes asincrónicas, el SDK también permite implementar una clase que utilice la interfaz. AsyncHandler

`AsyncHandler`proporciona dos métodos a los que se invoca en función de cómo se haya completado la solicitud: y. on`Success` on`Error`

La principal ventaja del enfoque de la interfaz de devolución de llamada es que se le permite sondear el objeto `Future` para saber si la solicitud se ha completado. El código puede iniciar inmediatamente su siguiente actividad y usar el SDK para llamar a su identificador en el momento oportuno.

```
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(".");
        }
    }
}
```

Prácticas recomendadas

Ejecución de la devolución de llamada

Su implementación de `AsyncHandler` se ejecuta dentro del grupo de subprocesos propiedad del cliente asíncrono. El código breve que se ejecuta rápidamente es más apropiado para su implementación de `AsyncHandler`. El código de bloqueo o de ejecución prolongada de sus métodos de identificador puede provocar controversia en el grupo de subprocesos usado por el cliente asíncrono y pueden impedir que el cliente ejecute las solicitudes. Si tiene una tarea de ejecución prolongada que debe iniciarse desde una devolución de llamada, permita que la devolución de llamada ejecute su tarea en un nuevo subproceso o en un grupo de subprocesos administrado por su aplicación.

Configuración del grupo de subprocesos

Los clientes asíncronos del servidor AWS SDK for Java proporcionan un grupo de subprocesos predeterminado que debería funcionar para la mayoría de las aplicaciones. Puede implementar una personalizada ExecutorServicey pasarla a clientes AWS SDK for Java asíncronos para tener un mayor control sobre la forma en que se administran los grupos de subprocesos.

Por ejemplo, puedes proporcionar una `ExecutorService` implementación que utilice una personalización ThreadFactorypara controlar el nombre de los subprocesos del grupo o para registrar información adicional sobre el uso de los subprocesos.

Acceso asíncrono

La TransferManagerclase del SDK ofrece soporte asíncrono para trabajar con ellos. Amazon S3`TransferManager`gestiona las cargas y descargas asíncronas, proporciona informes detallados sobre el progreso de las transferencias y admite la devolución de llamadas a distintos eventos.

Registro de llamadas AWS SDK for Java

AWS SDK for Java Está equipado con Apache Commons Logging, que es una capa de abstracción que permite el uso de cualquiera de los varios sistemas de registro en tiempo de ejecución.

Los sistemas de registro admitidos incluyen Java Logging Framework y Apache Log4j, entre otros. En este tema se muestra cómo utilizar Log4j. Puede utilizar la funcionalidad de registro del SDK sin necesidad de realizar cambios en el código de la aplicación.

Para obtener más información sobre Log4j, consulte el sitio web de Apache.

> **Note**
>
> Este tema se centra en Log4j 1.x. Log4j2 no admite directamente Apache Commons Logging, pero ofrece un adaptador que dirige automáticamente las llamadas de registro a Log4j2 utilizando la interfaz de Apache Commons Logging. Para obtener más información, consulte Commons Logging Bridge en la documentación de Log4j2.

Descarga del archivo JAR de Log4J

Para utilizar Log4j con el SDK, debe descargar el archivo JAR de Log4j JAR del sitio web de Apache. El SDK no incluye el archivo JAR. Copie el archivo JAR en una ubicación que esté en el classpath.

Log4j usa un archivo de configuración, log4j.properties. A continuación, se muestran ejemplos de archivos de configuración. Copie este archivo de configuración en un directorio del classpath. El archivo JAR de Log4j y el archivo log4j.properties no necesitan estar en el mismo directorio.

El archivo de configuración log4j.properties especifica propiedades, como el nivel de registro, dónde se envía la salida del registro (por ejemplo, a un archivo o a la consola) y el formato de la salida. El nivel de registro es el detalle de la salida que genera el registrador. Log4j admite el concepto de varias jerarquías de registro. El nivel de registro se define de forma independiente para cada jerarquía. Las siguientes dos jerarquías de registro están disponibles en AWS SDK for Java:

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

Definición del classpath

El archivo JAR de Log4j y el archivo log4j.properties deben estar en el classpath. Si utiliza Apache Ant, establezca el classpath en el elemento `path` en su archivo Ant. El ejemplo siguiente muestra un elemento de la ruta del archivo Ant para el ejemplo de Amazon S3 incluido en el SDK.

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

Si utiliza el IDE de Eclipse, puede establecer el classpath abriendo el menú y yendo a Project (Proyecto) | Properties (Propiedades) | Java Build Path (Ruta de compilación de Java).

Errores y advertencias específicos del servicio

Le recomendamos que siempre deje la jerarquía del registrador "com.amazonaws" establecida en "WARN" para identificar los mensajes importantes de las bibliotecas cliente. Por ejemplo, si el Amazon S3 cliente detecta que su aplicación no ha cerrado correctamente `InputStream` y podría estar filtrando recursos, el cliente S3 lo notificará mediante un mensaje de advertencia en los registros. Esto también garantiza que se registren los mensajes si el cliente tiene algún problema con el tratamiento de las solicitudes o respuestas.

El siguiente archivo log4j.properties establece `rootLogger` en WARN, lo que hace que se incluyan los mensajes de advertencia y de error de todos los registradores de la jerarquía "com.amazonaws". Otra opción consiste en establecer de forma explícita el registrador com.amazonaws en 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
```

Registro de resumen de solicitudes y respuestas

Cada solicitud a an Servicio de AWS genera un identificador de AWS solicitud único que resulta útil si tienes algún problema con la forma en que an Servicio de AWS gestiona una solicitud. AWS Se puede acceder a los ID de solicitud mediante programación a través de los objetos Exception del SDK para cualquier llamada de servicio fallida, y también se pueden informar a través del nivel de registro DEBUG, en el registrador «com.amazonaws.request».

El siguiente archivo log4j.properties permite obtener un resumen de las solicitudes y respuestas, incluidos los ID de las solicitudes. AWS

```
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
```

Este es un ejemplo del resultado del registro.

```
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
```

Registro detallado en red

En algunos casos, puede resultar útil ver las solicitudes y respuestas exactas que AWS SDK for Java envía y recibe. No deberías habilitar este registro en los sistemas de producción, ya que escribir solicitudes o respuestas de gran tamaño (por ejemplo, un archivo en el que se está cargando Amazon S3) o respuestas puede ralentizar considerablemente la aplicación. Si realmente necesita acceder a esta información, puede habilitarla temporalmente a través del registrador Apache HttpClient 4. La activación del nivel DEBUG en el registrador `org.apache.http.wire` permite registrar todos los datos de las solicitudes y respuestas.

El siguiente archivo log4j.properties activa el registro completo en Apache HttpClient 4 y solo debe activarse temporalmente, ya que puede tener un impacto significativo en el rendimiento de la aplicación.

```
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
```

Registro de métricas de latencia

Si está solucionando problemas y desea ver métricas; por ejemplo, qué proceso está tardando más o si es mayor la latencia del cliente o del servidor, el registrador de latencia puede resultarle muy útil. Para habilitar este registrador, configure el registrador `com.amazonaws.latency` en DEBUG.

> **Note**
>
> Este registrador solo está disponible si se habilitan las métricas de SDK. Para obtener más información sobre el paquete de métricas de SDK, consulte Habilitación de métricas para el 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
```

Este es un ejemplo del resultado del registro.

```
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]
```

Configuración de los clientes

AWS SDK for Java Esto le permite cambiar la configuración predeterminada del cliente, lo que resulta útil cuando desea:

- Conectarse a Internet a través del proxy
- Cambiar la configuración del transporte HTTP, como el tiempo de espera y los reintentos de solicitud de conexión
- Especificar sugerencias del tamaño del búfer del socket TCP

Configuración del proxy

Al crear un objeto de cliente, puede pasar un ClientConfigurationobjeto opcional para personalizar la configuración del cliente.

Si se conecta a Internet a través de un servidor proxy, tendrá que configurar las opciones del servidor proxy (host del proxy, puerto y nombre de usuario/contraseña) a través del objeto `ClientConfiguration`.

Configuración del transporte HTTP

Puede configurar varias opciones de transporte HTTP mediante el ClientConfigurationobjeto. De vez en cuando se añaden nuevas opciones; para ver la lista completa de opciones que puede recuperar o configurar, consulte la referencia de la AWS SDK for Java API.

> **Note**
>
> Cada uno de los valores configurables tiene un valor predeterminado definido por una constante. Para obtener una lista de los valores constantes de`ClientConfiguration`, consulte Valores de campo constantes en la referencia de la AWS SDK for Java API.

Número máximo de conexiones

Puede establecer el número máximo permitido de conexiones HTTP abiertas mediante ClientConfiguration. setMaxConnectionsmétodo.

> **Important**
>
> Para evitar problemas de conexión y un desempeño deficiente, establezca el número máximo de conexiones en el número de conexiones simultáneas. Para ver el valor máximo de conexiones predeterminado, consulta los valores de campo constantes en la referencia de la AWS SDK for Java API.

Tiempos de espera y tratamiento de errores

Puede configurar opciones relacionadas con los tiempos de espera y el tratamiento de errores con conexiones HTTP.

- Tiempo de espera de conexión

  El tiempo de espera de conexión es la cantidad de tiempo (en milisegundos) que la conexión HTTP esperará a que se establezca una conexión antes de desistir. El valor predeterminado es 10 000 ms.

  Para establecer este valor usted mismo, utilice ClientConfiguration. setConnectionTimeoutmétodo.
- Tiempo de vida (TTL) de la conexión

De forma predeterminada, el SDK intentará volver a utilizar las conexiones HTTP siempre que sea posible. En situaciones en las que no es posible establecer una conexión con un servidor que está fuera de servicio, disponer de un TTL finito puede ayudarle a recuperar la aplicación. Por ejemplo, si configura un TTL de 15 minutos, se asegurará de que aunque tenga establecida una conexión con un servidor que está experimentando problemas, se restablecerá una conexión con un nuevo servidor en un plazo de 15 minutos.

Para configurar el TTL de la conexión HTTP, utilice el método ClientConfiguration.setConnectionTTL.

- Número máximo de reintentos con error

  El número máximo predeterminado de reintentos para errores recuperables es 3. Puede establecer un valor diferente mediante. ClientConfiguration setMaxErrorMétodo de reintento.

Dirección local

Para establecer la dirección local a la que se enlazará el cliente HTTP, utiliceClientConfiguration.setLocalAddress.

Sugerencias del tamaño del búfer del socket TCP

Los usuarios avanzados que deseen ajustar los parámetros TCP de bajo nivel también pueden configurar sugerencias sobre el tamaño del búfer TCP a través del ClientConfigurationobjeto. La mayoría de los usuarios nunca necesitarás ajustar estos valores, pero se proporcionan para los usuarios avanzados.

Los tamaños del búfer TCP óptimos para una aplicación dependen en gran medida de la red y de la configuración y funciones del sistema operativo. Por ejemplo, la mayoría de los sistemas operativos modernos proporcionan lógica de ajuste automático para los tamaños del búfer TCP. Esto puede afectar considerablemente al desempeño de las conexiones TCP que se mantienen abiertas el tiempo suficiente para que el ajuste automático optimice los tamaños del búfer.

Los tamaños de búfer grandes (p. ej., 2 MB) permiten al sistema operativo almacenar más datos en memoria sin requerir que el servidor remoto confirme la recepción de esa información y pueden ser especialmente útiles cuando la red tiene alta latencia.

Esto es solo una sugerencia y puede que el sistema operativo no la aplique. Cuando se utiliza esta opción, los usuarios deben comprobar siempre los límites y valores predeterminados del sistema

operativo. La mayoría de los sistemas operativos tienen un límite máximo de tamaño del búfer TCP configurado y no permiten que se supere ese límite a menos que se aumente explícitamente el límite máximo del tamaño del búfer TCP.

Hay muchos recursos disponibles que le pueden ayudar a configurar los tamaños del búfer TCP y las opciones de TCP específicas del sistema operativo, incluidas las siguientes:

- Host Tuning

Política de control de acceso

AWS las políticas de control de acceso le permiten especificar controles de acceso detallados en sus recursos. AWS Una política de control de acceso se compone de un conjunto de instrucciones, con el siguiente formato:

La cuenta A tiene permiso para realizar la acción B en el recurso C donde se aplica la condición D.

Donde:

- A es el principal: el Cuenta de AWS que solicita el acceso o la modificación de uno de sus recursos. AWS
- B es la acción: la forma en que se accede a un AWS recurso o se lo modifica, por ejemplo, enviando un mensaje a una Amazon SQS cola o almacenando un objeto en un Amazon S3 depósito.
- C es el recurso: la AWS entidad a la que el principal quiere acceder, como una Amazon SQS cola o un objeto almacenado. Amazon S3
- D es un conjunto de condiciones: las limitaciones opcionales que se especifican para permitir o denegar el acceso al recurso de la entidad principal. Hay muchas condiciones expresivas disponibles, algunas específicas de cada servicio. Por ejemplo, puede utilizar condiciones de fecha para permitir el acceso a los recursos únicamente después o antes de un momento específico.

Amazon S3 Ejemplo

En el siguiente ejemplo, se muestra una política que permite a cualquier persona acceder a todos los objetos de un depósito, pero restringe el acceso a la carga de objetos a ese depósito a dos tipos Cuenta de AWS específicos (además de la cuenta del propietario del depósito).

```
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 Ejemplo

Un uso común de las políticas es autorizar una Amazon SQS cola para recibir mensajes de un tema de Amazon SNS.

```
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));
```

Ejemplo de Amazon SNS

Algunos servicios ofrecen condiciones adicionales que pueden utilizarse en las políticas. Amazon SNS establece condiciones para permitir o denegar suscripciones a temas de SNS según el protocolo (por ejemplo, correo electrónico, HTTP, HTTPS Amazon SQS) y el punto final (por ejemplo, dirección de correo electrónico, URL, Amazon SQS ARN) de la solicitud de suscripción a un tema.

```
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()));
```

Configuración del TTL de JVM para las búsquedas de nombres DNS

La máquina virtual de Java (JVM) almacena en caché las búsquedas de nombres DNS. Cuando la JVM convierte un nombre de host en una dirección IP, guarda en caché la dirección IP durante un período de tiempo específico, conocido como TTL. time-to-live

Como AWS los recursos utilizan entradas de nombres DNS que cambian de vez en cuando, le recomendamos que configure la JVM con un valor TTL no superior a 60 segundos. Con esto se asegurará de que cuando cambie la dirección IP de un recurso, su aplicación pueda recibir y utilizar la nueva dirección IP del recurso volviendo a consultar el DNS.

En algunas configuraciones de Java, el TTL predeterminado de JVM está establecido de forma que nunca se actualicen las entradas DNS hasta que se reinicie la JVM. Por lo tanto, si la dirección IP de un AWS recurso cambia mientras la aplicación aún está en ejecución, no podrá usar ese recurso hasta que reinicie manualmente la JVM y se actualice la información de IP almacenada en caché. En este caso, es fundamental establecer el TTL de la JVM de forma que actualice periódicamente la información de las direcciones IP almacenada en caché.

> **Note**
>
> El TTL predeterminado puede variar en función de la versión de su JVM y de si un administrador de seguridad está instalado. Muchas JVM proporcionan un TTL predeterminado inferior a 60 segundos. Si utiliza una de estas JVM y no usa un administrador de seguridad, puede omitir el resto de este tema.

Cómo configurar el TTL de JVM

Para modificar el TTL de JVM, establezca el valor de la propiedad networkaddress.cache.ttl. Utilice uno de los siguientes métodos, en función de sus necesidades:

- globalmente, para todas las aplicaciones que utilizan la JVM. Defina `networkaddress.cache.ttl` en el archivo `$JAVA_HOME/jre/lib/security/java.security` para Java 8 o `$JAVA_HOME/conf/security/java.security` en el archivo para Java 11 o superior:

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

- solo para su aplicación, establezca `networkaddress.cache.ttl` en el código de inicialización de su aplicación:

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

Habilitación de métricas para AWS SDK for Java

AWS SDK for Java Pueden generar métricas para la visualización y el monitoreo con Amazon CloudWatch que midan:

- el rendimiento de su aplicación al acceder AWS
- el rendimiento de sus máquinas virtuales jVM cuando se utilizan con AWS
- los detalles del entorno en tiempo de ejecución, como la memoria del montón, el número de subprocesos y los descriptores de archivos abiertos

Cómo habilitar la generación de métricas de SDK

Debe agregar la siguiente dependencia de Maven para permitir que el SDK envíe métricas a. 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>
```

*Sustituya el número de versión por la última versión del SDK disponible en Maven Central.

AWS SDK for Java las métricas están deshabilitadas de forma predeterminada. Para habilitarlas en el entorno de desarrollo local, incluya una propiedad del sistema que apunte al archivo de credenciales de seguridad de AWS cuando inicie la JVM. Por ejemplo:

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

Debe especificar la ruta al archivo de credenciales para que el SDK pueda cargar los puntos de datos recopilados CloudWatch para analizarlos más adelante.

> **Note**
>
> Si accedes AWS desde una Amazon EC2 instancia mediante el servicio de metadatos de la Amazon EC2 instancia, no necesitas especificar un archivo de credenciales. En este caso, solo debe especificar:
>
> ```
> -Dcom.amazonaws.sdk.enableDefaultMetrics
> ```

Todas las métricas capturadas por el AWS SDK for Java se encuentran en el espacio de nombres AWSSDK/Java y se cargan en la región CloudWatch predeterminada (us-east-1). Para cambiar la región, especifíquela usando el atributo `cloudwatchRegion` en la propiedad del sistema. Por ejemplo, para establecer la CloudWatch región en us-east-1, usa:

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

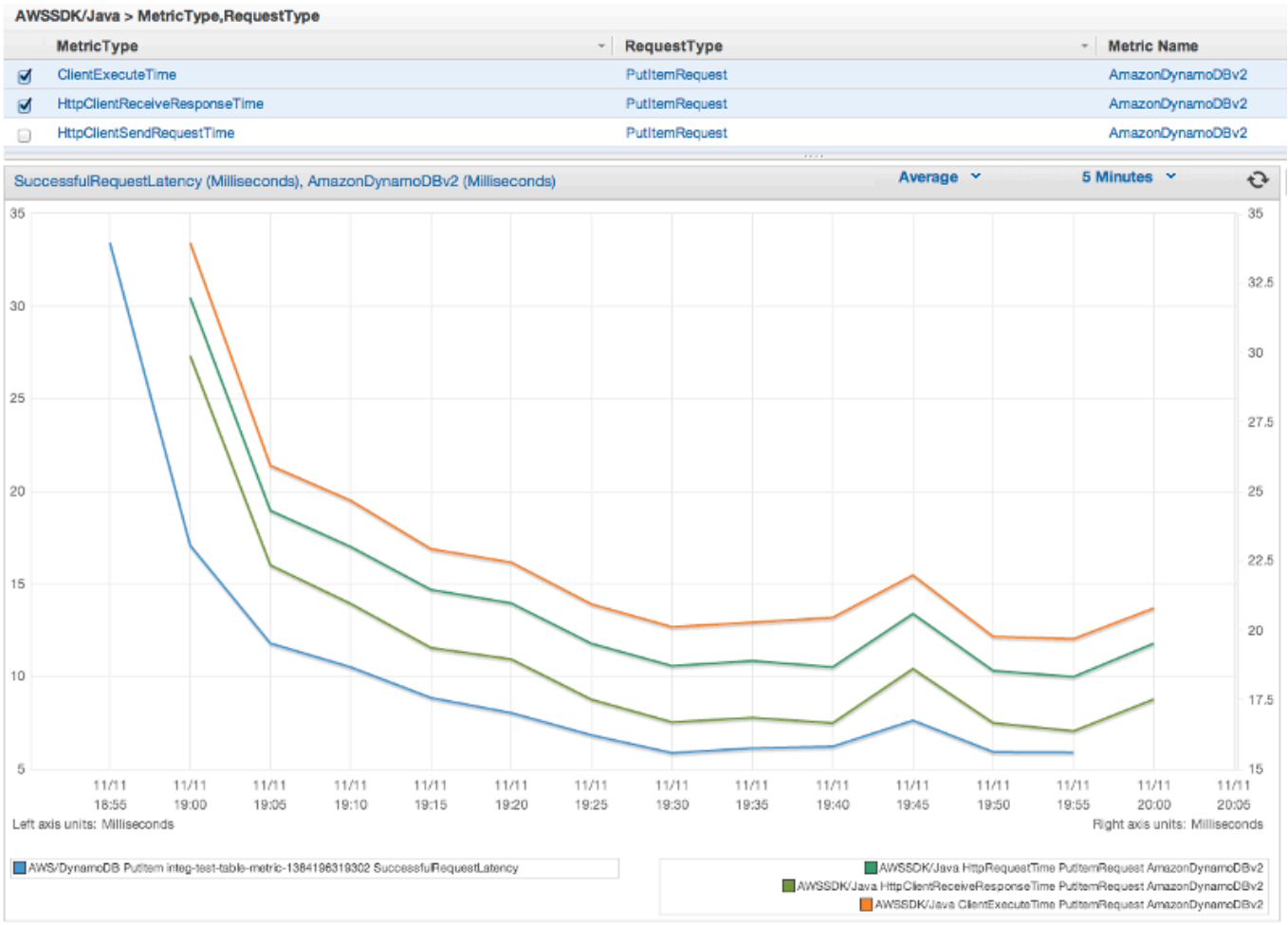
Una vez que active la función, cada vez que se reciba una solicitud de servicio AWS desde el AWS SDK for Java, se generarán puntos de datos métricos, se pondrán en cola para obtener un resumen estadístico y se cargarán de forma asíncrona aproximadamente una vez cada minuto. CloudWatch Una vez que se hayan cargado las métricas, puede consultarlas mediante la consola de administración de AWS Management Console y definir alarmas para posibles problemas como fuga de memoria, fuga de descriptores de archivos, etc.

Tipos de métricas disponibles

El conjunto predeterminado de métricas se divide en tres categorías principales:

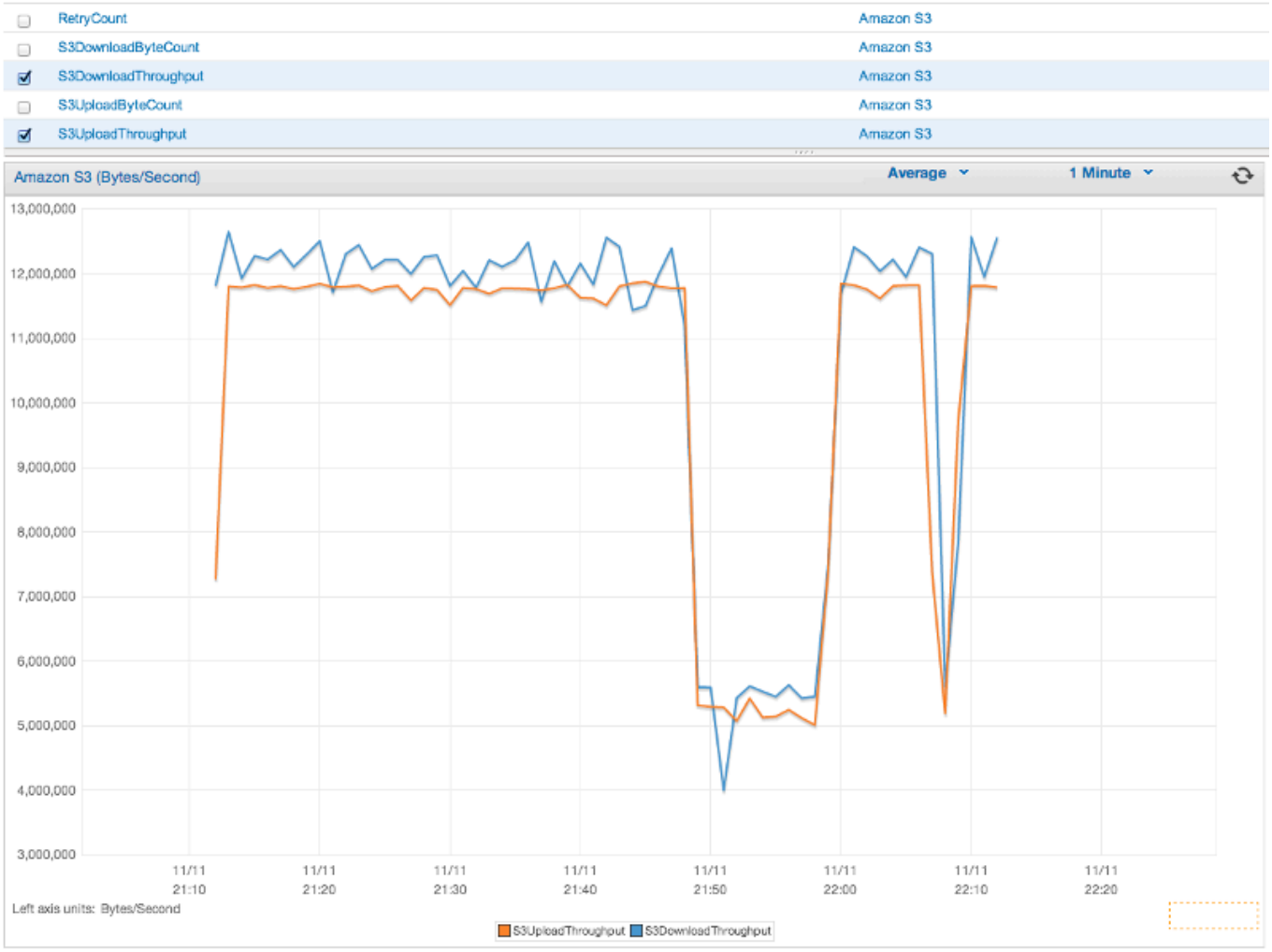
AWS Solicita métricas

- Cubren áreas como la latencia de la solicitud/respuesta HTTP, el número de solicitudes, las excepciones y los reintentos.



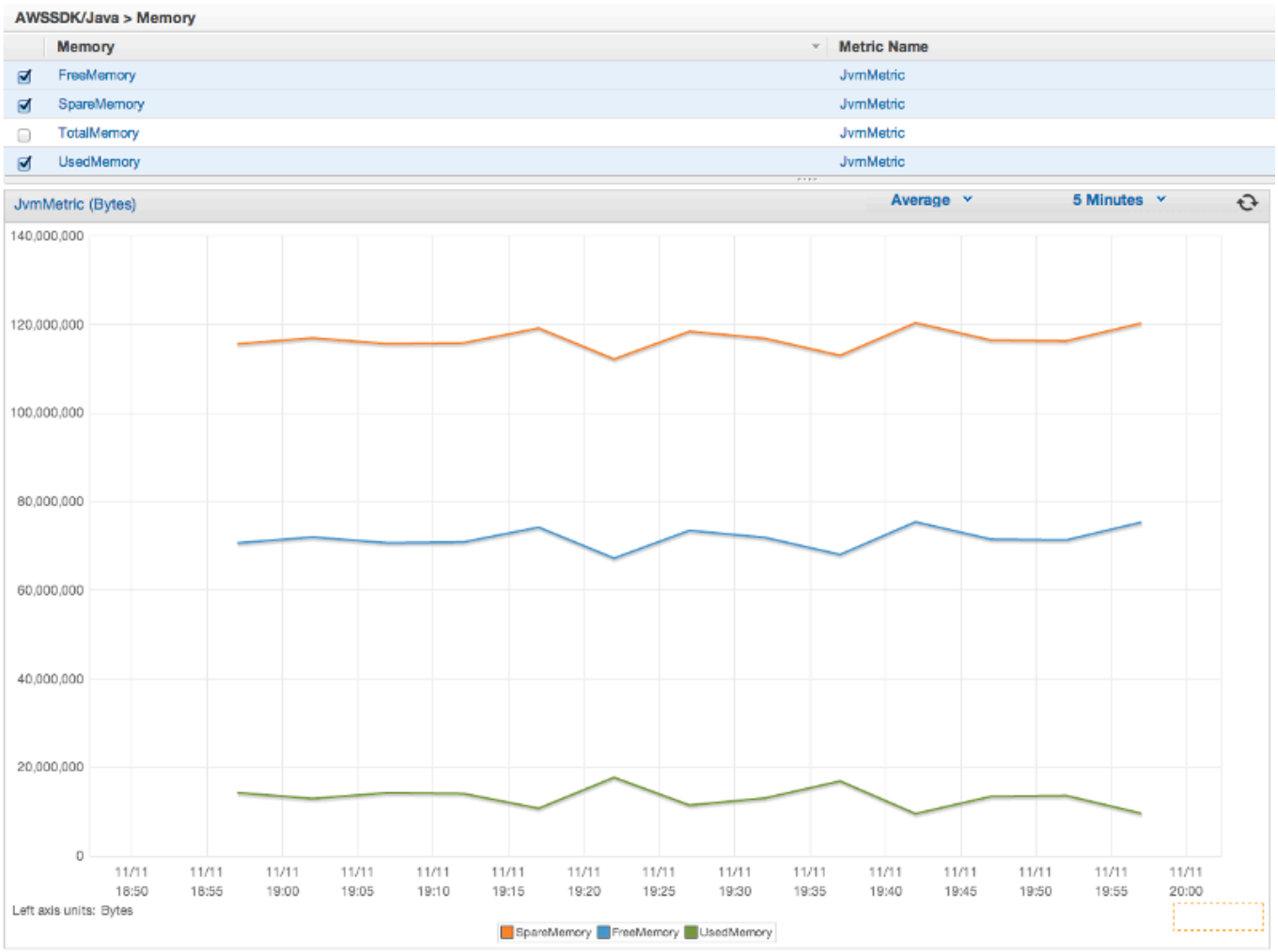
Servicio de AWS Métricas

- Incluya datos Servicio de AWS específicos, como el rendimiento y el recuento de bytes de las cargas y descargas de S3.



Métricas de la máquina

- Cubren el entorno en tiempo de ejecución, como la memoria del montón, el número de subprocesos y los descriptores de archivos abiertos.



Si desea excluir las métricas de la máquina, añada `excludeMachineMetrics` a la propiedad del sistema:

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

Más información

- Consulte amazonaws/metrics package summary para ver una lista completa de los principales tipos de métricas predefinidas.
- Obtenga información sobre cómo trabajar con el CloudWatch uso del AWS SDK for Java en los CloudWatch ejemplos de uso del AWS SDK for Java.

- Obtenga más información sobre el ajuste del rendimiento en la entrada del blog Tuning the AWS SDK for Java to Improve Resiliency.

Ejemplos de código de AWS SDK for Java

En esta sección se proporcionan tutoriales y ejemplos de cómo utilizar AWS SDK for Java v1 para programar servicios de AWS.

Busque el código fuente de estos ejemplos y otros en la documentación de AWS repositorio de ejemplos de código en GitHub.

Para proponer un nuevo ejemplo de código para que el equipo de documentación de AWS considere la posibilidad de crearlo, cree una nueva solicitud. El equipo está buscando crear ejemplos de código que abarquen situaciones y casos de uso más amplios, en comparación con fragmentos de código sencillos que tratan solo llamadas a la API individuales. Para obtener instrucciones, consulte las pautas de contribución en el repositorio de ejemplos de código en GitHub.

AWS SDK for Java 2.x

En 2018, AWS lanzó el AWS SDK for Java 2.x. Esta guía contiene instrucciones sobre el uso de la versión más reciente del SDK de Java, junto con un código de ejemplo.

> **Note**
>
> Consulte Documentación y recursos adicionales para obtener más ejemplos y recursos adicionales disponibles para los desarrolladores de AWS SDK for Java.

Ejemplos de CloudWatch con AWS SDK for Java

En esta sección se proporcionan ejemplos de programación en CloudWatch mediante AWS SDK for Java.

Amazon CloudWatch monitorea sus recursos Amazon Web Services (AWS) y las aplicaciones que ejecuta en AWS en tiempo real. Puede utilizar CloudWatch para recopilar y hacer un seguimiento de métricas, que son las variables que puede medir en los recursos y aplicaciones. Las alarmas de CloudWatch envían notificaciones o efectúan cambios automáticamente en los recursos que está supervisando basándose en las reglas que defina.

Para obtener más información sobre CloudWatch, consulte la Amazon CloudWatchGuía del usuario de .

> **Note**
>
> Los ejemplos incluyen únicamente el código necesario para demostrar cada técnica. El código de ejemplo completo está disponible en GitHub. Desde allí, puede descargar un único archivo de código fuente o clonar el repositorio localmente para obtener todos los ejemplos para compilarlos y ejecutarlos.

Temas

Obtención de métricas de CloudWatch

Mostrar métricas

Para enumerar las métricas de CloudWatch, cree un objeto ListMetricsRequest y llame al método `listMetrics` del AmazonCloudWatchClient. Puede utilizar el objeto `ListMetricsRequest` para filtrar las métricas devueltas por espacio de nombres, nombre de métrica o dimensiones.

> **Note**
>
> Puede encontrar una lista de las métricas y dimensiones publicadas por los servicios de AWS en {https://--docs-aws-amazon-com-AmazonCloudWatch-latest-Monitoring-CW-Support-for-AWS-html}Referencia de métricas y dimensiones de Amazon CloudWatch] de la Guía del usuario de Amazon CloudWatch.

Importaciones

```
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;
    }
}
```

Las métricas se devuelven en un objeto ListMetricsResult llamando a su método `getMetrics`. Los resultados puede que estén paginados. Para recuperar el siguiente lote de resultados, llame a `setNextToken` en el objeto de la solicitud original con el valor devuelto del método `getNextToken` del objeto `ListMetricsResult` y pase el objeto de la solicitud modificado a otra llamada a `listMetrics`.

Más información

- ListMetrics en la Referencia de la API de Amazon CloudWatch

Publicación de datos de métricas personalizadas

Algunos servicios de AWS publican sus propias métricas en espacios de nombres que comienzan por "AWS". También puede publicar datos de métricas personalizadas usando su propio espacio de nombres (siempre y cuando no comience por "AWS").

Publicación de datos de métricas personalizadas

Para publicar sus propios datos de métricas, llame al método `putMetricData` de AmazonCloudWatchClient con un objeto PutMetricDataRequest. El `PutMetricDataRequest` debe incluir el espacio de nombres personalizado que se va a usar para los datos e información sobre el propio punto de datos en un objeto MetricDatum.

> **Note**
>
> No puede especificar un espacio de nombres que comience por "AWS". Los espacios de nombres que comienzan por "AWS" están reservados para su uso por los productos de Amazon Web Services.

Importaciones

```
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);
```

Más información

- Uso de Métricas de Amazon CloudWatch en la Guía del usuario de Amazon CloudWatch.
- Espacios de nombres de AWS en la Guía del usuario de Amazon CloudWatch.
- PutMetricData en la Referencia de la API de Amazon CloudWatch.

Uso de alarmas de CloudWatch

Crear una alarma

Para crear una alarma basada en una métrica de CloudWatch, llame al método `putMetricAlarm` de AmazonCloudWatchClient con un objeto PutMetricAlarmRequest en el que se especifiquen las condiciones de la alarma.

Importaciones

```
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);
```

Mostrar alarmas

Para mostrar las alarmas de CloudWatch que ha creado, llame al método `describeAlarms` de AmazonCloudWatchClient con un objeto DescribeAlarmsRequest que puede utilizar para establecer opciones para el resultado.

Importaciones

```
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;
    }
}
```

La lista de alarmas se puede obtener llamando a `getMetricAlarms` en el objeto DescribeAlarmsResult que devuelve `describeAlarms`.

Los resultados puede que estén paginados. Para recuperar el siguiente lote de resultados, llame a `setNextToken` en el objeto de la solicitud original con el valor devuelto del método `getNextToken` del objeto `DescribeAlarmsResult` y pase el objeto de la solicitud modificado a otra llamada a `describeAlarms`.

> **Note**
>
> También puede recuperar alarmas para una métrica específica mediante el método `describeAlarmsForMetric` de AmazonCloudWatchClient. Su uso es similar a `describeAlarms`.

Eliminar alarmas

Para eliminar alarmas de CloudWatch, llame al método `deleteAlarms` de AmazonCloudWatchClient con un objeto DeleteAlarmsRequest que contenga uno o más nombres de alarma que desea eliminar.

Importaciones

```
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);
```

Más información

- Creación de una capa de Amazon CloudWatch en la Guía del usuario de Amazon CloudWatch
- PutMetricalArm en la Referencia de la API de Amazon CloudWatch
- DescribeAlarms en la referencia de la API de Amazon CloudWatch
- DeleteAlarms en la referencia de la API de Amazon CloudWatch

Uso de acciones de alarma en CloudWatch

Mediante las acciones de alarma, puede crear alarmas que realicen acciones como detener, terminar, reiniciar, o recuperar automáticamente instancias de Amazon EC2.

> **Note**
>
> Las acciones de alarma se pueden añadir a una alarma mediante el método de PutMetricAlarmRequest`setAlarmActions` al crear una alarma.

Habilitar acciones de alarma

Para habilitar acciones de alarma para una alarma de CloudWatch, llame al `enableAlarmActions` de AmazonCloudWatchClient con un objeto EnableAlarmActionsRequest que contenga uno o varios nombres de alarma cuyas acciones desee habilitar.

Importaciones

```
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);
```

Deshabilitar acciones de alarma

Para deshabilitar acciones de alarma para una alarma de CloudWatch, llame al `disableAlarmActions` del AmazonCloudWatchClient con un objeto DisableAlarmActionsRequest que contenga uno o varios nombres de alarma cuyas acciones desee deshabilitar.

Importaciones

```
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);
```

Más información

- Crear alarmas para detener, terminar, reiniciar o recuperar una instancia en la Guía del usuario de Amazon CloudWatch

- PutMetricalArm en la Referencia de la API de Amazon CloudWatch
- EnableAlarmActions en la Referencia de la API de Amazon CloudWatch
- DisableAlarmActions en la Referencia de la API de Amazon CloudWatch

Envío de eventos a CloudWatch

CloudWatch permite la transmisión casi en tiempo real de eventos del sistema que describen cambios en los recursos de AWS a instancias Amazon EC2, funciones Lambda, secuencias de Kinesis, tareas de Amazon ECS, máquinas de estado de Step Functions, temas de Amazon SNS, colas de Amazon SQS o destinos integrados. Mediante reglas sencillas, puede asignar los eventos y dirigirlos a una o más secuencias o funciones de destino.

Añadir eventos

Para añadir eventos de CloudWatch personalizados, llame al método `putEvents` del cliente AmazonCloudWatchEventsClient con un objeto PutEventsRequest que contenga uno o varios objetos PutEventsRequestEntry que proporcionen detalles sobre cada evento. Puede especificar varios parámetros para la entrada como el origen y el tipo del evento, los recursos asociados con el evento, etc.

> **Note**
>
> Puede especificar un máximo de 10 eventos para cada llamada a `putEvents`.

Importaciones

```
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);
```

Añadir reglas

Para crear o actualizar una regla, llame al método `putRule` de AmazonCloudWatchEventsClient con un objeto PutRuleRequest con el nombre de la regla y parámetros opcionales como el patrón del evento, el rol de IAM que se va a asociar a la regla y una expresión de programación que describa con qué frecuencia se ejecuta la regla.

Importaciones

```
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);
```

Añadir destinos

Los destinos son los recursos que se invocan cuando se activa una regla. Algunos destinos de ejemplo son instancias Amazon EC2, funciones Lambda, secuencias de Kinesis, tareas de Amazon ECS, máquinas de estado de Step Functions y destinos integrados.

Para añadir una regla a un destino, llame al método `putTargets` de AmazonCloudWatchEventsClient con un objeto PutTargetsRequest que contenga la regla para actualizar y la lista de destinos que se van a añadir a la regla.

Importaciones

```
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);
```

Más información

- Añadir eventos con PutEvents en la Guía del usuario del Amazon CloudWatch Events
- Programar expresiones para reglas en la Guía del usuario del Amazon CloudWatch Events
- Tipos de eventos para los eventos de CloudWatch en la Guía del usuario del Amazon CloudWatch Events

- Eventos y patrones de eventos en la Guía del usuario del Amazon CloudWatch Events
- PutEvents en la Referencia de la API de Amazon CloudWatch Events
- PutTargets en la Referencia de la API de Amazon CloudWatch Events
- PutRule en la Referencia de la API de Amazon CloudWatch Events

Ejemplos de DynamoDB usando la AWS SDK for Java

En esta sección se proporcionan ejemplos de programación en DynamoDB mediante AWS SDK for Java.

> **Note**
>
> Los ejemplos incluyen únicamente el código necesario para demostrar cada técnica. El código de ejemplo completo está disponible en GitHub. Desde allí, puede descargar un único archivo de código fuente o clonar el repositorio localmente para obtener todos los ejemplos para compilarlos y ejecutarlos.

Temas

- Uso de tablas en DynamoDB
- Uso de elementos en DynamoDB

Uso de tablas en DynamoDB

Las tablas son los contenedores de todos los elementos de una base de datos de DynamoDB. Para poder añadir o eliminar datos de DynamoDB, debe crear una tabla.

Para cada tabla, debe definir:

- Un nombre de tabla que sea único para su cuenta y región.
- Una clave principal para la que cada valor debe ser único; no puede haber dos elementos de la tabla que tengan el mismo valor de clave principal.

  Una clave principal puede ser simple, formada por una sola clave de partición (HASH) o compuesta, formada por una clave de partición y una clave de ordenación (RANGE).

Cada valor de clave tiene un tipo de datos asociado enumerado por la clase ScalarAttributeType. El valor de clave puede ser binario (B), numérico (N) o una cadena (S). Para obtener más información, consulte Reglas de nomenclatura y tipos de datos en la Guía para desarrolladores de Amazon DynamoDB.

- Valores de rendimiento aprovisionado que definan el número de unidades de capacidad de lectura/escritura reservadas para la tabla.

> **Note**
>
> Los precios de Amazon DynamoDB se basan en los valores de desempeño aprovisionado que puede definir en sus tablas para que solo se reserve la capacidad que piensa que va a necesitar para la tabla.

El desempeño aprovisionado para una tabla se puede modificar en cualquier momento, por lo que puede ajustar la capacidad si cambian sus necesidades.

Crear una tabla

Use el método del cliente de DynamoDB para crear una nueva tabla de DynamoDB. Debe crear los atributos de la tabla y un esquema de tabla, que se pueden usar para identificar la clave principal de la tabla. También debe proporcionar los valores iniciales de desempeño aprovisionado y el nombre de una tabla. Defina solo los atributos clave de la tabla al crear su tabla de DynamoDB.

> **Note**
>
> Si ya existe una tabla con el nombre elegido, se producirá una excepción AmazonServiceException.

Importaciones

```
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;
```

Creación de una tabla con una clave principal simple

Este código crea una tabla con una clave principal simple ("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);
}
```

Consulte el ejemplo completo en GitHub.

Creación de una tabla con una clave primaria compuesta

Añada otro objeto AttributeDefinition y KeySchemaElement a 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);
```

Consulte el ejemplo completo en GitHub.

Mostrar tablas

Puede mostrar las tablas de una región determinada llamando al método `listTables` del cliente de DynamoDB.

> **Note**
>
> Si la tabla designada no existe para su cuenta y región, se produce una excepción ResourceNotFoundException.

Importaciones

```
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;
        }
```

De forma predeterminada, se devuelve un máximo de 100 tablas para cada llamada: utilice el `getLastEvaluatedTableName` en el objeto ListTablesResult devuelto para obtener la última tabla que se evaluó. Puede utilizar este valor para iniciar la enumeración después del último valor devuelto de la enumeración anterior.

Consulte el ejemplo completo en GitHub.

Describir una tabla (obtener información de ella)

Llame al método `describeTable` del cliente de DynamoDB.

> **Note**
>
> Si la tabla designada no existe para su cuenta y región, se produce una excepción ResourceNotFoundException.

Importaciones

```
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);
}
```

Consulte el ejemplo completo en GitHub.

Modificar (actualizar) una tabla

Puede modificar los valores de desempeño aprovisionado de la tabla en cualquier momento llamando al método `updateTable` del cliente de DynamoDB.

> **Note**
>
> Si la tabla designada no existe para su cuenta y región, se produce una excepción ResourceNotFoundException.

Importaciones

```
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);
}
```

Consulte el ejemplo completo en GitHub.

Eliminar una tabla

Llame al método `deleteTable` del cliente de DynamoDB y pase el nombre de la tabla.

> **Note**
>
> Si la tabla designada no existe para su cuenta y región, se produce una excepción ResourceNotFoundException.

Importaciones

```
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);
}
```

Consulte el ejemplo completo en GitHub.

Más información

- Prácticas recomendadas para trabajar con tablas en la Guía para desarrolladores de Amazon DynamoDB
- Trabajar con tablas en DynamoDB en la Guía para desarrolladores de Amazon DynamoDB

Uso de elementos en DynamoDB

En DynamoDB, un elemento es una colección de atributos, cada uno de los cuales tiene un nombre y un valor. Los valores de los atributos pueden ser escalares, conjuntos o tipos de documentos.

Para obtener más información, consulte Reglas de nomenclatura y tipos de datos en la Guía para desarrolladores de Amazon DynamoDB.

Recuperar (obtener) un elemento de una tabla

Llame al método `getItem` de AmazonDynamoDB y pase un objeto GetItemRequest con el nombre de la tabla y el valor de clave principal del elemento que desee. Este método devuelve un objeto GetItemResult.

Puede utilizar el método `getItem()` del objeto `GetItemResult` para recuperar un mapa de pares de clave (cadena) y valor (AttributeValue) asociados al elemento.

Importaciones

```
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);
```

Consulte el ejemplo completo en GitHub.

Añadir un nuevo elemento a una tabla

Cree un mapa de pares de clave-valor que represente los atributos del elemento. Estos deben incluir valores para los campos de la clave principal de la tabla. Si el elemento identificado por la clave principal ya existe, la solicitud actualiza sus campos.

> **Note**
>
> Si la tabla designada no existe para su cuenta y región, se produce una excepción ResourceNotFoundException.

Importaciones

```
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);
```

Consulte el ejemplo completo en GitHub.

Actualizar un elemento existente en una tabla

Puede actualizar un atributo de un elemento que ya existe en una tabla mediante el método `updateItem` de AmazonDynamoDB, proporcionando el nombre de la tabla, el valor de clave principal y un mapa de los campos que se van a actualizar.

> **Note**
>
> Si la tabla designada no existe para su cuenta y región o si el elemento identificado por la clave principal que ha pasado no existe, se produce una excepción ResourceNotFoundException.

Importaciones

```
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);
```

Consulte el ejemplo completo en GitHub.

Uso de la clase DynamoDBMapper

El AWS SDK for Java proporciona la clase DynamoDBMapper que permite mapear las clases del cliente a las tablas de Amazon DynamoDB. Para utilizar la clase DynamoDBMapper, defina la relación entre los elementos de una tabla de DynamoDB y sus instancias de objeto correspondientes en el código mediante anotaciones (como se muestra en el ejemplo de código siguiente). La clase DynamoDBMapper permite obtener acceso a las tablas, realizar varias operaciones de creación, lectura, actualización y eliminación (CRUD, Create, Read, Update and Delete) y ejecutar consultas.

> **Note**
>
> La clase DynamoDBMapper no permite crear, actualizar o eliminar tablas.

Importaciones

```
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

En el siguiente ejemplo de código Java se muestra cómo añadir contenido a la tabla Music (Música) mediante la clase DynamoDBMapper . Después de agregar el contenido a la tabla, observe que se carga un elemento mediante las claves Partition (Partición) y Sort (Ordenar) . A continuación, se actualiza el elemento Awards (Premios) . Para obtener información sobre la creación de la tabla Música, consulte Crear una tabla en la Guía para desarrolladores de Amazon DynamoDB.

```
        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;
        }
    }
```

Consulte el ejemplo completo en GitHub.

Más información

- Prácticas recomendadas para el uso de elementos en la Guía para desarrolladores de Amazon DynamoDB
- Uso de elementos DynamoDB en la Guía para desarrolladores de Amazon DynamoDB

Ejemplos de Amazon EC2 usando la AWS SDK for Java

En esta sección se proporcionan ejemplos de programación en Amazon EC2 mediante AWS SDK for Java.

Temas

- Tutorial: Inicio de una instancia EC2
- Uso de roles de IAM para conceder acceso a recursos de Amazon EC2 en AWS
- Tutorial: Instancias de spot de Amazon EC2
- Tutorial: Administración avanzada de solicitudes de spot de Amazon EC2

- Administración de instancias de Amazon EC2
- Uso de direcciones IP elásticas en Amazon EC2
- Usar regiones y zonas de disponibilidad
- Uso de pares de claves de Amazon EC2
- Uso de grupos de seguridad en Amazon EC2

Tutorial: Inicio de una instancia EC2

Este tutorial muestra cómo utilizar AWS SDK for Java para iniciar una instancia EC2.

Temas

- Requisitos previos
- Cree un grupo de seguridad de Amazon EC2
- Creación de un par de claves
- Ejecutar una instancia de Amazon EC2

Requisitos previos

Antes de empezar, asegúrese de haber creado una Cuenta de AWS y configurado las credenciales AWS. Para obtener más información, consulte Introducción.

Cree un grupo de seguridad de Amazon EC2

Retirada de EC2-Classic

> **Warning**
>
> Vamos a retirar EC2-Classic el 15 de agosto de 2022. Le recomendamos que migre de EC2-Classic a una VPC. Para obtener más información, consulte Migración de EC2-Classic a una VPC en la Guía del usuario de Amazon EC2 para instancias de Linux o en la Guía del usuario de Amazon EC2 para instancias de Windows. Consulte también la entrada del blog Se va a retirar la red EC2-Classic-Classic: cómo prepararse.

Cree un grupo de seguridad, que funciona como un firewall virtual que controla el tráfico de red de una o varias instancias EC2. De forma predeterminada, Amazon EC2 asocia sus instancias con

un grupo de seguridad que no permite el tráfico entrante. Puede crear un grupo de seguridad que permita a sus instancias EC2 aceptar un tráfico determinado. Por ejemplo, si necesita conectarse a una instancia Linux, debe configurar el grupo de seguridad para permitir el tráfico SSH. Puede crear un grupo de seguridad mediante la consola de Amazon EC2 o AWS SDK for Java.

Puede crear un grupo de seguridad para usarlo en EC2-Classic o en EC2-VPC. Para obtener más información sobre EC2-Classic y EC2-VPC, consulte Plataformas admitidas en la Guía del usuario de Amazon EC2 para instancias de Linux.

Para obtener más información sobre los grupos de seguridad, con la consola de Amazon EC2, consulte Grupos de seguridad de Amazon EC2 en la Guía del usuario de Amazon EC2 para instancias de Linux.

1. Cree e inicialice una instancia de CreateSecurityGroupRequest. Utilice el método withGroupName para establecer el nombre del grupo de seguridad y el método withDescription para establecer la descripción del grupo de seguridad, tal y como se indica a continuación:

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

   El nombre del grupo de seguridad debe ser único dentro de la región de AWS en la que inicializa el cliente de Amazon EC2. Debe utilizar caracteres US-ASCII para el nombre y la descripción del grupo de seguridad.
2. Pase el objeto de la solicitud como un parámetro al método createSecurityGroup. El método devuelve un objeto CreateSecurityGroupResult, de la manera siguiente:

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

   Si intenta crear un grupo de seguridad con el mismo nombre que un grupo de seguridad existente, se produce una excepción `createSecurityGroup`.

De forma predeterminada, un nuevo grupo de seguridad no permite el tráfico entrante a su instancia Amazon EC2. Para permitir el tráfico entrante, debe autorizarlo de forma explícita en el grupo de seguridad. Puede autorizar el tráfico entrante para direcciones IP individuales, para un intervalo de direcciones IP, para un protocolo específico y para puertos TCP/UDP.

1. Cree e inicialice una instancia de IpPermission. Utilice el método withIpv4Ranges para definir el intervalo de direcciones IP para el que se autoriza el tráfico entrante y use el método withIpProtocol para definir el protocolo IP. Utilice los métodos withFromPort y withToPort para especificar el intervalo de puertos para los que se autoriza el tráfico entrante, como se indica a continuación:

```
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);
```

   Todas las condiciones que especifique en el objeto `IpPermission` se deben cumplir para permitir el tráfico entrante.

   Especifique la dirección IP con la notación CIDR. Si especifica el protocolo como TCP/UDP, debe proporcionar un puerto de origen y un puerto de destino. Solo puede autorizar puertos si especifica TCP o UDP.

2. Cree e inicialice una instancia de AuthorizeSecurityGroupIngressRequest. Utilice el método `withGroupName` para especificar el nombre del grupo de seguridad y pase el objeto `IpPermission` que inicializó anteriormente al método withIpPermissions, como se indica a continuación:

```
AuthorizeSecurityGroupIngressRequest authorizeSecurityGroupIngressRequest =
    new AuthorizeSecurityGroupIngressRequest();

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

3. Pase el objeto de la solicitud al método authorizeSecurityGroupIngress, de la siguiente manera:

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

Si llama a `authorizeSecurityGroupIngress` con direcciones IP para las que ya se ha autorizado el tráfico entrante, el método produce una excepción. Cree y inicialice un nuevo objeto `IpPermission` para autorizar el tráfico entrante para diferentes direcciones IP, puertos y protocolos antes de llamar a `AuthorizeSecurityGroupIngress`.

Siempre que llama a los métodos authorizeSecurityGroupIngress o authorizeSecurityGroupEgress, se añade una regla al grupo de seguridad.

Creación de un par de claves

Al conectarse a la instancia, debe especificar un par de claves al lanzar una instancia EC2 y la clave privada del par de claves. Puede crear un par de claves o usar un par de claves existente que haya utilizado al lanzar otras instancias. Para obtener más información, consulte Pares de claves de Amazon EC2 en la Guía del usuario de Amazon EC2 para instancias de Linux.

1. Cree e inicialice una instancia de CreateKeyPairRequest. Utilice el método withKeyName para definir el nombre del par de claves, tal y como se indica a continuación:

```
CreateKeyPairRequest createKeyPairRequest = new CreateKeyPairRequest();

createKeyPairRequest.withKeyName(keyName);
```

> **Important**
>
> Los nombres de pares de claves deben ser únicos. Si intenta crear un par de claves con el mismo nombre de clave que un par de claves existente, se producirá una excepción.

2. Pase el objeto de solicitud al método createKeyPair. El método devuelve una instancia de CreateKeyPairResult, de la manera siguiente:

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

3. Llame al método getKeyPair del objeto resultante para obtener un objeto KeyPair. Llame al método getKeyMaterial del objeto `KeyPair` para obtener la clave privada codificada en PEM descifrada, como se indica a continuación:

```
KeyPair keyPair = new KeyPair();
```

```
keyPair = createKeyPairResult.getKeyPair();

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

Ejecutar una instancia de Amazon EC2

Use el siguiente procedimiento para lanzar una o más instancias EC2 con la misma configuración de la misma Imagen de máquina de Amazon (AMI). Después de crear las instancias EC2, puede comprobar sus estados. Una vez que las instancias EC2 se están ejecutando, puede conectarse a ellas.

1. Cree e inicialice una instancia de RunInstancesRequest. Asegúrese de que la AMI, el par de claves y el grupo de seguridad que especifique existen en la región indicada al crear el objeto de cliente.

```
RunInstancesRequest runInstancesRequest =
   new RunInstancesRequest();

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

withImageId

- Es el ID de la AMI. Para obtener más información sobre cómo buscar una AMI pública proporcionada por Amazon o crear la suya, consulte Imagen de máquina de Amazon (AMI).

withInstanceType

- Se trata de un tipo de instancia que es compatible con la AMI especificada. Para obtener más información, consulte Tipos de instancias en la Guía del usuario de Amazon EC2 para instancias de Linux.

withMinCount

- Se trata del número mínimo de instancias EC2 que se van a lanzar. Si hay más instancias de las que Amazon EC2 puede lanzar en la zona de disponibilidad de destino, Amazon EC2 no lanzará ninguna instancia.

withMaxCount

- Se trata del número máximo de instancias EC2 que se van a lanzar. Si hay más instancias de las que Amazon EC2 puede lanzar en la zona de disponibilidad de destino, Amazon EC2 lanzará el mayor número posible de instancias por encima del `MinCount`. Puede lanzar entre una y el número máximo de instancias permitido para el tipo de instancia. Para obtener más información, consulte ¿Cuántas instancias puedo ejecutar en Amazon EC2? en las preguntas frecuentes de Amazon EC2.

withKeyName

- Se trata del nombre del par de claves de EC2. Si lanza una instancia sin especificar un par de claves, no podrá conectarse a ella. Para obtener más información, consulte la sección Crear un par de claves.

withSecurityGroups

- Uno o varios grupos de seguridad. Para obtener más información, consulte Crear de un grupo de seguridad de Amazon EC2.

2. Lance las instancias pasando el objeto solicitado al método runInstances. El método devuelve un objeto RunInstancesResult, de la manera siguiente:

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

Una vez ejecutada la instancia, puede conectarse a ella usando el par de claves. Para obtener más información, consulte Conexión con su instancia de Linux en la Guía del usuario de Amazon EC2 para instancias de Linux.

Uso de roles de IAM para conceder acceso a recursos de Amazon EC2 en AWS

Todas las solicitudes a Amazon Web Services (AWS) deben firmarse criptográficamente mediante credenciales emitidas por AWS. Puede utilizar roles de IAM como un método sencillo para conceder acceso seguro a los recursos de AWS desde sus instancias de Amazon EC2.

En este tema se proporciona información acerca de cómo utilizar los roles de con aplicaciones del SDK de Java que se ejecutan en Amazon EC2. Para obtener más información sobre los roles de IAM consulte Roles de IAM para Amazon EC2 en la Guía del usuario de Amazon EC2 para instancias de Linux.

Cadena predeterminada de proveedores y perfiles de instancias EC2

Si su aplicación crea un cliente de AWS utilizando el constructor predeterminado, el cliente buscará las credenciales mediante la cadena predeterminada de proveedores de credenciales, en el orden siguiente:

1. En las propiedades del sistema Java: `aws.accessKeyId` y `aws.secretKey`.
2. En las variables de entorno del sistema: `AWS_ACCESS_KEY_ID` y `AWS_SECRET_ACCESS_KEY`.
3. En el archivo de credenciales (la ubicación de este archivo varía en función de la plataforma).
4. Las credenciales entregadas a través del servicio de contenedor de Amazon EC2 si se establece la variable de entorno `AWS_CONTAINER_CREDENTIALS_RELATIVE_URI` y el administrador de seguridad tiene permiso para acceder a la variable.
5. En las credenciales del perfil de la instancia, que residen en los metadatos de la instancia asociadas con la función de IAM para la instancia EC2.
6. Credenciales de Web Identity Token del entorno o contenedor.

El paso credenciales del perfil de la instancia en la cadena predeterminada de proveedores solo está disponible cuando la aplicación se ejecuta en una instancia Amazon EC2, pero es el método más sencillo y más seguro cuando se trabaja con instancias Amazon EC2. También puede pasar una instancia de InstanceProfileCredentialsProvider directamente al constructor del cliente para obtener las credenciales del perfil de la instancia sin recorrer toda la cadena predeterminada de proveedores.

Por ejemplo:

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

Cuando se utiliza este enfoque, el SDK recupera las credenciales temporales de AWS que tienen los mismos permisos que los asociados al rol de IAM asociado a la instancia Amazon EC2 en su perfil de instancia. Aunque estas credenciales son temporales y acaban caducando, `InstanceProfileCredentialsProvider` las actualiza periódicamente para que sigan permitiendo el acceso a AWS.

> **Important**
>
> La actualización automática de las credenciales solo se realiza cuando utiliza el constructor del cliente predeterminado, que crea su propio `InstanceProfileCredentialsProvider` como parte de la cadena predeterminada de proveedores, o cuando pasa una instancia de `InstanceProfileCredentialsProvider` directamente al constructor del cliente. Si utiliza otro método para obtener o pasar credenciales del perfil de la instancia, usted es responsable de comprobar y actualizar las credenciales que hayan caducado.

Si el constructor del cliente no encuentra las credenciales con la cadena de proveedores de credenciales, produce una excepción AmazonClientException.

Tutorial: Uso de roles de IAM para instancias EC2

El siguiente tutorial muestra cómo recuperar un objeto de Amazon S3 mediante un rol de IAM para administrar el acceso.

Creación de un rol de IAM

Cree un rol de IAM que conceda acceso de solo lectura a Amazon S3.

1. Abra la consola de IAM.
2. En el panel de navegación, seleccione Roles y después Create New Role (Crear nuevo rol).
3. Escriba un nombre para la función y, a continuación, seleccione Next Step (Paso siguiente). Recuerde este nombre, ya que lo necesitará cuando lance su instancia Amazon EC2.
4. En la página Seleccionar tipo de rol, en Roles de Servicio de AWS, seleccione Amazon EC2.
5. En la página Definir permisos, bajo Seleccionar plantilla de política, seleccione Acceso de solo lectura de Amazon S3 y después Siguiente paso.
6. En la página Review (Revisar), seleccione Create Role (Crear rol).

Lanzar una instancia EC2 y especificar el rol de IAM

Puede lanzar una instancia de Amazon EC2 con un rol de IAM mediante la consola de Amazon EC2 o el AWS SDK for Java.

- Para lanzar una instancia de Amazon EC2 mediante la consola, siga las instrucciones de Introducción a las instancias de Linux de Amazon EC2 en la Guía del usuario de instancias de Linux.

  Cuando llegue a la página Review Instance Launch (Revisar lanzamiento de instancia), seleccione Edit instance details (Editar detalles de la instancia). En IAM role, elija el rol de IAM que creó anteriormente. Complete el procedimiento siguiendo las instrucciones.

  > **Note**
  >
  > Deberá crear o usar un grupo de seguridad y un par de claves existentes para conectarse a la instancia.

- Para lanzar una instancia de Amazon EC2 con un rol de IAM mediante el AWS SDK for Java, consulte Amazon EC2Ejecución de una instancia de .

Creación de una aplicación

Vamos a compilar la aplicación de ejemplo para que se ejecute en la instancia EC2. En primer lugar, cree el directorio que va a usar para almacenar los archivos del tutorial (por ejemplo, `GetS3ObjectApp`).

A continuación, copie las bibliotecas de AWS SDK for Java en el directorio recién creado. Si ha descargado AWS SDK for Java en su directorio `~/Downloads`, puede copiarlos utilizando los siguientes comandos:

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

Abra un nuevo archivo, llámelo `GetS3Object.java`y añada el siguiente código:

```
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();
  }
}
```

Abra un nuevo archivo, llámelo `build.xml` y añada las líneas siguientes:

```
<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>
```

Compile y ejecute el programa modificado. Tenga en cuenta que no hay credenciales almacenadas en el programa. Por lo tanto, a menos que ya haya especificado las credenciales de AWS, el código producirá una `AmazonServiceException`. Por ejemplo:

```
$ 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
```

Transferir el programa compilado a la instancia EC2

Transfiera el programa a su instancia Amazon EC2 mediante la copia segura ( ), junto con las bibliotecas de AWS SDK for Java. La secuencia de comandos debe ser similar a la siguiente.

```
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**
>
> En función de la distribución Linux que haya utilizado, el nombre de usuario podría ser "ec2-user", "root" o "ubuntu". Para obtener el nombre DNS público de la instancia, abra la consola de EC2 y busque el valor de Public DNS (DNS público) en la pestaña Description (Descripción) (por ejemplo, `ec2-198-51-100-1.compute-1.amazonaws.com`).

En los comandos anteriores:

- `GetS3Object.class` es el programa compilado
- `build.xml` es el archivo ant que se utiliza para compilar y ejecutar el programa
- los directorios `lib` y `third-party` son las carpetas de las bibliotecas de AWS SDK for Java correspondientes.
- El modificador `-r` indica que `scp` debe realizar una copia recursiva de todo el contenido de los directorios `library` y `third-party` en la distribución de AWS SDK for Java.
- El modificador `-p` indica que `scp` deben conservar los permisos de los archivos de código fuente cuando se copien en el destino.

> **Note**
>
> El conmutador `-p` solo funciona en Linux, macOS o Unix. Si va a copiar archivos de Windows, es posible que tenga que corregir los permisos del archivo en su instancia mediante el siguiente comando:

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

Ejecutar el programa de ejemplo en la instancia EC2

Para ejecutar el programa, conéctese a la instancia Amazon EC2. Para obtener más información, consulte Conexión con su instancia de Linux en la Guía del usuario de Amazon EC2 para instancias de Linux.

Si **`ant`** no está disponible en la instancia, instálelo con el siguiente comando:

```
sudo yum install ant
```

A continuación, ejecute el programa mediante `ant` del siguiente modo:

```
ant run
```

El programa escribirá el contenido del objeto de Amazon S3 en la ventana de comandos.

Tutorial: Instancias de spot de Amazon EC2

Información general

Las instancias de spot permiten pujar por capacidad de Amazon Elastic Compute Cloud (Amazon EC2) sin utilizar hasta el 90 % frente al precio de instancia bajo demanda y ejecutar las instancias adquiridas, siempre y cuando su puja sea superior al precio de spot actual. Amazon EC2 cambia periódicamente el precio de spot en función de la oferta y la demanda; los clientes cuyas pujas igualen o superen este precio tendrán acceso a las instancias de spot disponibles. Al igual que las instancias bajo demanda y las instancias reservadas, las instancias de spot proporcionan otra opción para obtener una mayor capacidad de cómputo.

Las instancias de spot pueden reducir de forma significativa los costos de Amazon EC2 del procesamiento por lotes, la investigación científica, el procesamiento de imágenes, la codificación de vídeo, el rastreo web y de datos, el análisis financiero y la realización de pruebas. Además, las instancias de spot le permiten obtener acceso a una gran cantidad de capacidad adicional en aquellas situaciones en las que la necesidad de esa capacidad no es urgente.

Para usar instancias de spot, coloque una solicitud de instancia de spot que especifique el precio máximo que está dispuesto a pagar por hora de instancia; esta es su puja. Si el importe de su puja es mayor que el precio de spot actual, se atenderá su solicitud y sus instancias se ejecutarán hasta que decida terminarlas o hasta que el precio de spot sea mayor que su puja (lo que suceda antes).

Es importante tener en cuenta lo siguiente:

- Con frecuencia pagará por hora un importe inferior al de su puja. Amazon EC2 ajusta el precio de spot periódicamente a medida que llegan las solicitudes y que cambia la oferta disponible. Todo el mundo paga el mismo precio de spot para ese período independientemente de que su puja fuera más alta. Por lo tanto, puede pagar un importe inferior al de su puja, pero nunca pagará un importe superior al de esta.
- Si ejecuta instancias de spot y su puja ya no coincide con el precio de spot actual ni lo supera, se terminarán sus instancias. Esto significa que querrá asegurarse de que sus cargas de trabajo y aplicaciones son lo suficientemente flexibles para aprovechar esta oportunista capacidad.

Las instancias de spot funcionan exactamente igual que otras instancias Amazon EC2 mientras se ejecutan y, al igual que otras instancias Amazon EC2, se pueden terminar cuando ya no las necesita. Si termina su instancia, pagará por las horas parciales empleadas (como lo haría en el caso de las instancias bajo demanda o reservadas). Sin embargo, si el precio de spot es superior al importe de su puja y Amazon EC2 termina su instancia, no se le cobrará por las horas de uso parciales.

Este tutorial muestra cómo utilizar AWS SDK for Java para realizar las siguientes tareas.

- Enviar una solicitud de spot
- Determinar cuándo se atiende la solicitud de spot
- Cancelar la solicitud de spot
- Terminar las instancias asociadas

Requisitos previos

Para utilizar este tutorial, debe tener AWS SDK for Java instalado, así como los requisitos previos de instalación básicos. Para obtener más información, consulte Configuración de AWS SDK for Java.

Paso 1: Configuración de las credenciales

Para empezar a usar este ejemplo de código, debe configurar las credenciales de AWS. Consulte Configuración de credenciales y regiones de AWS para desarrollo para ver instrucciones sobre cómo hacerlo.

> **Note**
>
> Le recomendamos que utilice las credenciales de un usuario de IAM para proporcionar estos valores. Para obtener más información, consulte Inscripción en AWS y creación de un usuario de IAM.

Ahora que ha configurado sus opciones, puede empezar a utilizar el código del ejemplo.

Paso 2: Configuración de un grupo de seguridad

Un grupo de seguridad funciona como un firewall que controla el tráfico permitido de entrada y salida de un grupo de instancias. De forma predeterminada, una instancia se inicia sin ningún grupo de seguridad, lo que significa que se denegará todo el tráfico IP entrante, en cualquier puerto TCP. Por

lo tanto, antes de enviar una solicitud de spot, vamos a configurar un grupo de seguridad que permita el tráfico de red necesario. A efectos de este tutorial, vamos a crear un nuevo grupo de seguridad llamado "GettingStarted" que permita el tráfico Secure Shell (SSH) desde la dirección IP en la que se ejecuta su aplicación. Para configurar un nuevo grupo de seguridad, debe incluir o ejecutar el siguiente ejemplo de código, que configura el grupo de seguridad mediante programación.

Después, creamos un objeto cliente `AmazonEC2` y un objeto `CreateSecurityGroupRequest` con el nombre, "GettingStarted" y una descripción para el grupo de seguridad. A continuación, llamamos a la API `ec2.createSecurityGroup` para crear el grupo.

Para habilitar el acceso al grupo, creamos un objeto `ipPermission` con el intervalo de direcciones IP establecido en la representación CIDR de la subred del equipo local; el sufijo "/10" en la dirección IP indica la subred de la dirección IP especificada. También configuramos el objeto `ipPermission` con el protocolo TCP y el puerto 22 (SSH). El último paso consiste en llamar a `ec2.authorizeSecurityGroupIngress` con el nombre de nuestro grupo de seguridad y el objeto `ipPermission`.

```
// 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());
}
```

Tenga en cuenta que solo necesita ejecutar esta aplicación una vez para crear un nuevo grupo de seguridad.

También puede crear el grupo de seguridad mediante AWS Toolkit for Eclipse. Consulte Administración de grupos de seguridad desde AWS Cost Explorer para obtener más información.

Paso 3: Envío de la solicitud de spot

Para enviar una solicitud de spot, primero es necesario determinar el tipo de instancia, la imagen de máquina de Amazon (AMI) y el precio de puja máximo que desea usar. También debe incluir el grupo de seguridad que hemos configurado anteriormente, de modo que pueda iniciar sesión en la instancia si lo desea.

Hay varios tipos de instancia para elegir; vaya a Tipos de instancias Amazon EC2 para obtener una lista completa. En este tutorial, utilizaremos t1.micro, el tipo de instancia más económica disponible. A continuación, determinaremos el tipo de AMI que desea utilizar. Utilizaremos ami-a9d09ed1, la AMI de Amazon Linux más actualizada disponible cuando escribimos este tutorial. La AMI más reciente

puede cambiar con el tiempo, pero siempre puede determinar la última versión de la AMI siguiendo estos pasos:

1. Abra la consola de Amazon EC2.
2. Elija el botón Launch Instance (Lanzar instancia).
3. La primera ventana muestra las AMI disponibles. El ID de AMI aparece al lado del título de cada AMI. También puede utilizar la API `DescribeImages`, pero el uso de este comando queda fuera del alcance de este tutorial.

Existen muchas formas de pujar por instancias de spot; para obtener una descripción general de los diferentes enfoques, vea el vídeo Bidding for Spot Instances. Sin embargo, para comenzar, describiremos tres estrategias comunes: pujar para garantizar que el costo sea menor que el precio bajo demanda, pujar en función del valor de la computación resultante y pujar con el fin de adquirir capacidad de computación con la mayor rapidez posible.

- Reducir el costo por debajo del precio bajo demanda Tiene una tarea de procesamiento por lotes que tardará en ejecutarse una cantidad determinada de horas o días. Sin embargo, es flexible con respecto a cuándo comienza y finaliza. Desea ver si puede completarla por menos del valor del costo de las instancias bajo demanda. Puede examinar el historial de precios de subasta para tipos de instancia mediante la AWS Management Console o la API de Amazon EC2. Para obtener más información, consulte Historial de precios de instancias de spot. Una vez que haya analizado el historial de precios para su tipo de instancia deseado en una zona de disponibilidad especificada, tendrá dos enfoques alternativos para su puja:
  - Podría pujar en el extremo superior del rango de precios de spot (que aún son inferiores al precio bajo demanda), contando con que lo más probable es que su solicitud de spot puntual se atienda y se ejecute durante un tiempo de computación consecutivo suficiente para completar la tarea.
  - O bien, puede especificar la cantidad que está dispuesto a pagar por las instancias de spot como un porcentaje del precio de la instancia bajo demanda, así como combinar muchas instancias lanzadas a lo largo del tiempo a través de una solicitud persistente. Si se supera el precio especificado, la instancia de spot terminará. (Explicaremos cómo automatizar esta tarea más adelante en este tutorial).
- No pagar un importe superior al valor del resultado Tiene una tarea de procesamiento de datos que ejecutar. Conoce las ventajas de los resultados de la tarea lo suficientemente bien como para saber lo valiosos que son en términos de costos de computación. Una vez que haya analizado el historial de precios de spot para el tipo de instancia, podrá elegir un precio de puja en el que

el costo del tiempo de computación no sea superior al valor de los resultados de la tarea. Puede crear una puja persistente y permitir su ejecución intermitente a medida que el precio de spot fluctúa en torno a su puja o por debajo de esta.

- Adquirir capacidad de computación rápidamente Tiene una necesidad a corto plazo no anticipada de capacidad adicional que no está disponible a través de las instancias bajo demanda. Una vez que haya analizado el historial de precios de spot para el tipo de instancia, podrá pujar por encima del precio histórico más alto para tener mayores probabilidades de que su solicitud se atienda con rapidez y continúe computándose hasta completarse.

Una vez que haya elegido el precio de puja, estará listo para solicitar una instancia de spot. Para los fines de este tutorial, pujaremos por el precio bajo demanda (0,03 USD) para maximizar las posibilidades de que se atienda la puja. Puede determinar los tipos de instancias disponibles y los precios bajo demanda para instancias yendo a la página de precios de Amazon EC2. Cuando una instancia de spot está en ejecución, paga el precio de spot vigente durante el período de tiempo en que se ejecutan las instancias. Amazon EC2 define los precios de las instancias de spot y estos se ajustan gradualmente en función de las tendencias a largo plazo de la oferta y la demanda de capacidad de este tipo de instancia. También puede especificar el importe que está dispuesto a pagar por una instancia de spot como porcentaje del precio de la instancia bajo demanda. Para solicitar una instancia de spot, solo tiene que crear su solicitud con los parámetros que eligió anteriormente. Comencemos creando un objeto `RequestSpotInstanceRequest`. El objeto de la solicitud requiere el número de instancias que desea para comenzar y el precio de puja. Además, necesita establecer `LaunchSpecification` para la solicitud, que incluye el tipo de instancia, el ID de la AMI y el grupo de seguridad que desea utilizar. Una vez rellenada la solicitud, llama al método `requestSpotInstances` en el objeto `AmazonEC2Client`. En el siguiente ejemplo se muestra cómo solicitar una instancia de spot.

```
// 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);
```

Al ejecutarse este código se lanzará una nueva solicitud de instancia de spot. Hay otras opciones que puede usar para configurar las solicitudes de spot. Para obtener más información, consulte Tutorial: Administración avanzada de solicitudes de spot de Amazon EC2 o la clase RequestSpotInstances en la Referencia de la API de AWS SDK for Java.

> **Note**
>
> Se le cobrará por las instancias de spot que se de verdad se lancen, de modo que asegúrese de cancelar cualquier solicitud y terminar las instancias que lance para reducir las tarifas asociadas.

Paso 4: Determinación del estado de la solicitud de spot

A continuación, queremos crear código para esperar hasta que la solicitud de spot alcance el estado "activo" antes de continuar con el último paso. Para determinar el estado de nuestra solicitud de spot, usamos el método describeSpotInstanceRequests para obtener el estado del ID de solicitud de spot que deseamos monitorizar.

El ID de solicitud creado en el paso 2 se inserta en la respuesta a nuestra solicitud `requestSpotInstances`. El siguiente ejemplo muestra cómo obtener los ID de solicitud de la respuesta `requestSpotInstances` y utilizarlos para rellenar una `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());
}
```

Para monitorizar su ID de solicitud, llame al método `describeSpotInstanceRequests` para determinar el estado de la solicitud. A continuación, recorra en bucle la solicitud hasta que deje de tener el estado "abierto". Tenga en cuenta que buscamos un estado distinto de "abierto" en lugar de, por ejemplo, un estado "activo", porque la solicitud podría pasar directamente al estado "cerrado" si surgiera algún problema con los argumentos de la solicitud. El siguiente ejemplo de código proporciona los detalles de cómo realizar esta tarea.

```
// 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);
```

Después de ejecutar este código, su solicitud de instancia de spot se habrá completado o habrá producido un error que se mostrará en la pantalla. En cualquier caso, podemos continuar con el siguiente paso para limpiar todas las solicitudes activas y terminar todas las instancias en ejecución.

Paso 5: Limpieza de las instancias y solicitudes de spot

Por último, tenemos que limpiar nuestras solicitudes e instancias. Esto es importante tanto para cancelar cualquier solicitud pendiente como para terminar cualquier instancia. Las instancias no terminarán con solo cancelarse las solicitudes, lo que significa que se le seguirá cobrando por ellas. Si termina las instancias, es posible que se cancelen las solicitudes de spot, pero hay algunos escenarios (por ejemplo, si usa pujas persistentes), donde terminar las instancias no es suficiente

para evitar que la solicitud vuelva a atenderse. Por lo tanto, se recomienda tanto cancelar cualquier puja activa como terminar cualquier instancia en ejecución.

En el siguiente código se muestra cómo cancelar las solicitudes.

```
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());
}
```

Para terminar todas las instancias pendientes, necesitará el ID de la instancia asociada a la solicitud que las inició. El siguiente ejemplo se basa en el código original de monitorización de instancias en el que se ha añadido una `ArrayList` en la que almacenamos los ID de instancia asociados a la respuesta `describeInstance`.

```
// 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);
```

Usando los ID de instancia almacenados en `ArrayList`, termine todas las instancias en ejecución con el siguiente fragmento de código.

```
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());
}
```

Operación conjunta

Para realizar todas estas operaciones a la vez, ofrecemos un enfoque más orientado a objetos, que combina los pasos mostrados: inicializar el cliente de EC2, enviar la solicitud de spot, determinar el momento en el que las solicitudes de spot ya no tienen el estado abierto y limpiar todas las solicitudes de spot pendientes y las instancias asociadas. Creamos una clase llamada `Requests` que realiza estas acciones.

También creamos una clase `GettingStartedApp`, que tiene un método principal donde realizamos las llamadas a funciones de alto nivel. En concreto, inicializamos el objeto `Requests` descrito anteriormente. Enviamos la solicitud de instancia de spot. A continuación, esperamos a que la solicitud de spot alcance el estado "activo". Por último, limpiamos las solicitudes y las instancias.

El código fuente completo de este ejemplo se puede consultar o descargar en GitHub.

¡Enhorabuena! Ha completado el tutorial de introducción al desarrollo de software de instancias de spot con AWS SDK for Java.

Pasos siguientes

Continúe con el Tutorial: Administración avanzada de solicitudes de spot de Amazon EC2.

Tutorial: Administración avanzada de solicitudes de spot de Amazon EC2

Las instancias de spot de Amazon EC2 le permiten pujar por capacidad de Amazon EC2 sin usar y ejecutar esas instancias mientras su puja supere el precio de spot actual. Amazon EC2 cambia el precio de spot periódicamente en función de la oferta y la demanda. Para más información acerca de las instancias de spot, consulte Instancias de spot en la Guía del usuario de Amazon EC2 para instancias de Linux.

Requisitos previos

Para utilizar este tutorial, debe tener AWS SDK for Java instalado, así como los requisitos previos de instalación básicos. Para obtener más información, consulte Configuración de AWS SDK for Java.

Configuración de las credenciales

Para empezar a usar este ejemplo de código, debe configurar las credenciales de AWS. Consulte Configuración de credenciales y regiones de AWS para desarrollo para ver instrucciones sobre cómo hacerlo.

> **Note**
>
> Le recomendamos que utilice las credenciales de un usuario de IAM para proporcionar estos valores. Para obtener más información, consulte Inscripción en AWS y creación de un usuario de IAM.

Ahora que ha configurado sus opciones, puede empezar a utilizar el código del ejemplo.

Configuración de un grupo de seguridad

Un grupo de seguridad funciona como un firewall que controla el tráfico permitido de entrada y salida de un grupo de instancias. De forma predeterminada, una instancia se inicia sin ningún grupo de seguridad, lo que significa que se denegará todo el tráfico IP entrante, en cualquier puerto TCP. Por lo tanto, antes de enviar una solicitud de spot, vamos a configurar un grupo de seguridad que permita el tráfico de red necesario. A efectos de este tutorial, vamos a crear un nuevo grupo de seguridad llamado "GettingStarted" que permita el tráfico Secure Shell (SSH) desde la dirección IP en la que se ejecuta su aplicación. Para configurar un nuevo grupo de seguridad, debe incluir o ejecutar el siguiente ejemplo de código, que configura el grupo de seguridad mediante programación.

Después, creamos un objeto cliente `AmazonEC2` y un objeto `CreateSecurityGroupRequest` con el nombre, "GettingStarted" y una descripción para el grupo de seguridad. A continuación, llamamos a la API `ec2.createSecurityGroup` para crear el grupo.

Para habilitar el acceso al grupo, creamos un objeto `ipPermission` con el intervalo de direcciones IP establecido en la representación CIDR de la subred del equipo local; el sufijo "/10" en la dirección IP indica la subred de la dirección IP especificada. También configuramos el objeto `ipPermission` con el protocolo TCP y el puerto 22 (SSH). El último paso consiste en llamar a

`ec2 .authorizeSecurityGroupIngress` con el nombre de nuestro grupo de seguridad y el objeto `ipPermission`.

(El siguiente código es el mismo que el que usamos en el primer 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());
}
```

Puede ver este ejemplo de código completo en el ejemplo de código `advanced.CreateSecurityGroupApp.java`. Tenga en cuenta que solo necesita ejecutar esta aplicación una vez para crear un nuevo grupo de seguridad.

> **Note**
>
> También puede crear el grupo de seguridad mediante AWS Toolkit for Eclipse. Consulte Administración de grupos de seguridad desde AWS Cost Explorer en la Guía del usuario de AWS Toolkit for Eclipse para obtener más información.

Opciones detalladas de creación de solicitudes de instancias de spot

Como hemos explicado en el Tutorial: Instancias de spot de Amazon EC2, debe crear la solicitud con un tipo de instancia, una Imagen de máquina de Amazon (AMI) y un precio de puja máximo.

Comencemos creando un objeto `RequestSpotInstanceRequest`. El objeto de la solicitud requiere el número de instancias que desee y el precio de puja. Además, necesitamos establecer `LaunchSpecification` para la solicitud, que incluye el tipo de instancia, el ID de la AMI y el grupo de seguridad que desea utilizar. Una vez rellenada la solicitud, llamamos al método `requestSpotInstances` en el objeto `AmazonEC2Client`. A continuación se incluye un ejemplo de cómo solicitar una instancia de spot.

(El siguiente código es el mismo que el que usamos en el primer 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);
```

Solicitudes persistentes y solicitudes puntuales

Cuando se crea una solicitud de spot, puede especificar varios parámetros opcionales. El primero es si la solicitud es puntual o persistente. De forma predeterminada, es una solicitud puntual. Una solicitud puntual se puede tramitar solo una vez, y una vez que se terminan las instancias solicitadas, la solicitud se cierra. Una solicitud persistente se puede tramitar siempre que no haya ninguna instancia de spot ejecutándose para la misma solicitud. Para especificar el tipo de solicitud, simplemente tiene que establecer el tipo en la solicitud de spot. Esto se puede hacer con el siguiente código.

```
// 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);
```

Limitación de la duración de una solicitud

De forma opcional, puede especificar el tiempo que su solicitud seguirá estando en vigor. Puede especificar una hora de inicio y finalización para este periodo. De forma predeterminada, una solicitud de spot se puede tramitar desde el momento en que se crea hasta que se tramita o se cancela. Sin embargo, puede restringir el periodo de validez en caso de que sea necesario. En el siguiente código se muestra un ejemplo de cómo especificar este periodo.

```
// 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);
```

Agrupación de solicitudes de instancias de spot de Amazon EC2

Tiene la opción de agrupar sus solicitudes de instancias de spot de diferentes maneras. Veamos las ventajas de utilizar grupos de lanzamiento, grupos de zonas de disponibilidad y grupos de ubicación.

Si desea garantizar que las instancias de spot se lancen y terminen a la vez, puede usar un grupo de lanzamiento. Un grupo de lanzamiento es una etiqueta que agrupa un conjunto de pujas. Todas las instancias en un grupo de lanzamiento se inician y se terminan juntas. Tenga en cuenta que si las instancias de un grupo de lanzamiento ya se han tramitado, no hay ninguna garantía de que se tramiten también las nuevas instancias lanzadas con el mismo grupo de lanzamiento. Un ejemplo de cómo configurar un grupo de lanzamiento se muestra en el siguiente ejemplo de código.

```
// 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);
```

Si desea asegurarse de que todas las instancias de una solicitud se lancen en la misma zona de disponibilidad y no tener que preocuparse de cuál de ellas se va a lanzar, puede utilizar grupos de zonas de disponibilidad. Un grupo de zonas de disponibilidad es una etiqueta que agrupa un conjunto de instancias de forma conjunta en la misma zona de disponibilidad. Todas las instancias que comparten un grupo de zonas de disponibilidad y que se tramitan al mismo tiempo se iniciarán en la misma zona de disponibilidad. A continuación se incluye un ejemplo de cómo configurar un grupo de zonas de disponibilidad.

```
// 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);
```

Puede especificar la zona de disponibilidad que desee para sus instancias de spot. El siguiente ejemplo de código muestra cómo definir una zona de disponibilidad.

```
// 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);
```

Por último, puede especificar un grupo de ubicación si utiliza instancias de spot de informática de alto rendimiento (HPC), como instancias de informática en clúster o instancias de GPU de clúster. Los grupos de ubicación ofrecen baja latencia y conectividad de alto ancho de banda entre las instancias. A continuación se incluye un ejemplo de cómo configurar un grupo de ubicación.

```
// 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);
```

Todos los parámetros que se muestran en esta sección son opcionales. También es importante tener en cuenta que la mayoría de estos parámetros, con la excepción de si su puja es puntual o persistente, pueden reducir la probabilidad de que se atienda su puja. Por lo tanto, es importante que utilice estas opciones solo si las necesita. Todos los ejemplos de código anteriores se combinan en un único ejemplo de código mayor, que se puede encontrar en la clase `com.amazonaws.codesamples.advanced.InlineGettingStartedCodeSampleApp.java`.

Cómo conservar una partición raíz después de una interrupción o terminación

Una de las formas más sencillas de administrar la interrupción de sus instancias de spot es garantizar que sus datos se someten comprobación en un volumen de Amazon Elastic Block Store (Amazon Amazon EBS) a un ritmo regular. Mediante la creación de puntos de comprobación de forma periódica, si hay una interrupción, solo perderá los datos creados desde el último punto de comprobación (suponiendo que no se realicen otras acciones no idempotentes entre medias). Para simplificar ese proceso, puede configurar la solicitud de spot para garantizar que la partición raíz no se elimine debido a una interrupción o terminación. Hemos introducido nuevo código en el siguiente ejemplo que muestra cómo permitir este escenario.

En el código añadido, creamos un objeto `BlockDeviceMapping` y establecemos su Amazon Elastic Block Store (Amazon EBS) en un objeto de Amazon EBS que hemos configurado como para que `not` se elimine si la instancia de spot termina. A continuación, añadimos este `BlockDeviceMapping` al ArrayList de asignaciones que incluimos en la especificación de lanzamiento.

```
// 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);
```

En el caso de que desee volver a asociar este volumen a la instancia durante el inicio, también puede utilizar los ajustes de mapeo de dispositivos de bloques. Si la instancia está asociada a un partición distinta de la partición raíz, también puede especificar los volúmenes de Amazon Amazon EBS que desea asociar a la instancia de spot una vez que esta se reanude. Para ello, solo tiene que especificar un ID de snapshot en su `EbsBlockDevice` y un nombre de dispositivo alternativo en sus objetos `BlockDeviceMapping`. Mediante el uso de mapeos de dispositivos de bloques, puede ser más sencillo arrancar su instancia.

El uso de la partición raíz para aplicar un punto de comprobación a su datos críticos es una forma excelente de administrar la posibilidad de que se interrumpan sus instancias. Para conocer otros métodos de administración de posibles interrupciones, vea el vídeo Managing Interruption.

Cómo etiquetar sus solicitudes e instancias de spot

Añadir etiquetas a los recursos de Amazon EC2 puede simplificar la administración de la infraestructura de la nube. Las etiquetas, un tipo de metadatos, se pueden utilizar para crear nombres sencillos, mejorar la capacidad de búsqueda y mejorar la coordinación entre varios usuarios. También puede utilizar las etiquetas para automatizar scripts y partes de sus procesos. Para obtener más información sobre cómo etiquetar recursos de Amazon EC2, consulte Uso de etiquetas en la Guía del usuario de Amazon EC2 para instancias de Linux.

Etiquetado de solicitudes de

Para añadir etiquetas a sus solicitudes de spot, tiene que etiquetarlas después de que se hayan solicitado. El valor devuelto de `requestSpotInstances()` proporciona un objeto RequestSpotInstancesResult que puede utilizar para obtener los ID de las solicitudes de spot que desea etiquetar:

```
// 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());
}
```

Cuando tenga los ID, puede etiquetar las solicitudes añadiendo sus ID a una CreateTagsRequest y llamando al método `createTags()` del cliente de Amazon EC2:

```
// 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());
}
```

Etiquetado de instancias

De forma similar a las solicitudes de spot, solo puede etiquetar una instancia una vez que se haya creado, lo que ocurrirá cuando se haya satisfecho la solicitud de spot (cuando ya no tenga el estado abierto).

Puede comprobar el estado de sus solicitudes llamando al método `describeSpotInstanceRequests()` del cliente de Amazon EC2 con un objeto DescribeSpotInstanceRequestsRequest. El objeto DescribeSpotInstanceRequestsResult devuelto contiene una lista de objetos SpotInstanceRequest que puede usar para consultar el estado de sus solicitudes de spot y obtener sus ID de instancia cuando ya no tengan el estado abierto.

Cuando la solicitud de spot deje de estar abierta, puede recuperar su ID de instancia del objeto `SpotInstanceRequest` llamando a su método `getInstanceId()`.

```
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);
```

Ahora puede etiquetar las instancias que se devuelven:

```
// 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());
}
```

Cancelación de solicitudes de spot y terminación de instancias

Cancelación de una solicitud de spot

Para cancelar una solicitud de instancia de spot, llame a `cancelSpotInstanceRequests` en el cliente de Amazon EC2 con un objeto CancelSpotInstanceRequestsRequest.

```
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());
}
```

Terminación de instancias de spot

Puede terminar las instancias de spot que se estén ejecutando pasando sus ID al método `terminateInstances()` del cliente de Amazon EC2.

```
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());
}
```

Operación conjunta

Para realizar todas estas operaciones a la vez, ofrecemos un enfoque más orientado a objetos, que combina los pasos mostrados en este tutorial en una clase fácil de usar. Creamos una instancia de una clase llamada `Requests` que realiza estas acciones. También creamos una clase `GettingStartedApp`, que tiene un método principal donde realizamos las llamadas a funciones de alto nivel.

El código fuente completo de este ejemplo se puede consultar o descargar en GitHub.

¡Enhorabuena! Ha completado el tutorial de características avanzadas de solicitudes para el desarrollo de instancias de spot con AWS SDK for Java.

Administración de instancias de Amazon EC2

Crear una instancia

Cree una nueva instancia de Amazon EC2 llamando al método `runInstances` de AmazonEC2Client, proporcionando un objeto RunInstancesRequest que contenga la Imagen de máquina de Amazon (AMI) que se va a usar y un tipo de instancia.

Importaciones

```
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();
```

Consulte el ejemplo completo

Iniciar una instancia

Para iniciar una instancia Amazon EC2, llame al método `startInstances` de AmazonEC2Client, proporcionando un objeto StartInstancesRequest que contenga el ID de la instancia que se va a iniciar.

Importaciones

```
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);
```

Consulte el ejemplo completo

Detener una instancia

Para detener una instancia Amazon EC2, llame al método `stopInstances` de AmazonEC2Client, proporcionando un objeto StopInstancesRequest que contenga el ID de la instancia que se va a detener.

Importaciones

```
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);
```

Consulte el ejemplo completo

Reiniciar una instancia

Para reiniciar una instancia Amazon EC2, llame al método `rebootInstances` de AmazonEC2Client, proporcionando un objeto RebootInstancesRequest que contenga el ID de la instancia que se va a reiniciar.

Importaciones

```
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);
```

Consulte el ejemplo completo

Describir instancias

Para enumerar sus instancias, cree un objeto DescribeInstancesRequest y llame al método `describeInstances` de AmazonEC2Client. Se devolverá un objeto DescribeInstancesResult que puede utilizar para mostrar las instancias Amazon EC2 de su cuenta y región.

Las instancias se agrupan por reserva. Cada reserva se corresponde con la llamada a `startInstances` que lanzó la instancia. Para mostrar sus instancias, primero debe llamar al método `getReservations' method, and then call `getInstances` de la clase `DescribeInstancesResult` en cada objeto Reservation devuelto.

Importaciones

```
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;
    }
}
```

Los resultados se paginan; puede obtener más resultados pasando el valor devuelto del método `getNextToken` del objeto resultante al método `setNextToken` del objeto de la solicitud original, usando el mismo objeto de la solicitud en la siguiente llamada a `describeInstances`.

Consulte el ejemplo completo

Monitorizar una instancia

Puede monitorizar distintos aspectos de las instancias Amazon EC2, como el uso de la CPU y la red, la memoria disponible y el espacio en disco restante. Para obtener más información sobre la supervisión de instancias, consulte Supervisión Amazon EC2 en la Guía del usuario de Amazon EC2 para instancias Linux.

Para iniciar la monitorización de una instancia, debe crear un objeto MonitorInstancesRequest con el ID de la instancia que se va a monitorizar y pasarlo al método `monitorInstances` de AmazonEC2Client.

Importaciones

```
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);
```

Consulte el ejemplo completo

Detener la monitorización de instancias

Para detener la monitorización de una instancia, cree un objeto UnmonitorInstancesRequest con el ID de la instancia cuya monitorización se va a detener y pase el objeto al método de AmazonEC2Client.

Importaciones

```
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);
```

Consulte el ejemplo completo

Más información

- RunInstances en la referencia de la API de Amazon EC2
- DescribeInstances en la referencia de la API de Amazon EC2
- StartInstances en la referencia de la API de Amazon EC2
- StopInstances en la referencia de la API de Amazon EC2

- RebootInstances en la referencia de la API de Amazon EC2
- MonitorInstances en la referencia de la API de Amazon EC2
- UnmonitorInstances en la referencia de la API de Amazon EC2

Uso de direcciones IP elásticas en Amazon EC2

Retirada de EC2-Classic

> **Warning**
>
> Vamos a retirar EC2-Classic el 15 de agosto de 2022. Le recomendamos que migre de EC2-Classic a una VPC. Para obtener más información, consulte Migración de EC2-Classic a una VPC en la Guía del usuario de Amazon EC2 para instancias de Linux o en la Guía del usuario de Amazon EC2 para instancias de Windows. Consulte también la entrada del blog Se va a retirar la red EC2-Classic-Classic: cómo prepararse.

Asignación de una dirección IP elástica

Para utilizar una dirección IP elástica, primero debe asignar una a su cuenta y, a continuación, asociarla a su instancia o a una interfaz de red.

Para asignar una dirección IP elástica, llame al método `allocateAddress` del cliente AmazonEC2Client con un objeto AllocateAddressRequest que contenga el tipo de red (EC2 clásico o VPC).

El AllocateAddressResult devuelto contiene un ID de asignación que puede utilizar para asociar la dirección a una instancia, pasando el ID de asignación y el ID de instancia en un objeto AssociateAddressRequest al método `associateAddress` de AmazonEC2Client.

Importaciones

```
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);
```

Consulte el ejemplo completo

Descripción de direcciones IP elásticas

Para listar las direcciones IP elásticas asignadas a su cuenta, llame al método `describeAddresses` de AmazonEC2Client. Este método devuelve un objeto DescribeAddressesResult que puede utilizar para obtener una lista de objetos Address que representan las direcciones IP elásticas de su cuenta.

Importaciones

```
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());
}
```

Consulte el ejemplo completo

Liberación de una dirección IP elástica

Para liberar una dirección IP elástica, llame al método `releaseAddress` de AmazonEC2Client pasando un objeto ReleaseAddressRequest que contenga el ID de asignación de la dirección IP elástica que quiere liberar.

Importaciones

```
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);
```

Tras liberar una dirección IP elástica, esta se libera del grupo de direcciones IP de AWS y puede que no esté disponible para usarla más adelante. Asegúrese de actualizar sus registros DNS y los servidores o dispositivos que se comunican con la dirección. Si intenta liberar una dirección IP

elástica que ya ha liberado, obtendrá un error AuthFailure si la dirección ya se ha asignado a otra cuenta de Cuenta de AWS.

Si utiliza EC2-Classic o una VPC predeterminada, al liberar una dirección IP elástica esta se desvincula automáticamente de cualquier instancia a la que esté asociada. Para desvincular una dirección IP elástica sin liberarla, utilice el método `disassociateAddress` de AmazonEC2Client.

Si utiliza una VPC distinta de la predeterminada, debe usar `disassociateAddress` para desvincular la dirección IP elástica antes de intentar liberarla. De lo contrario, Amazon EC2 devuelve un error (InvalidIPAddress.InUse).

Consulte el ejemplo completo

Más información

- Direcciones IP elásticas en la Guía del usuario de Amazon EC2 para instancias de Linux
- AllocateAddress en la referencia a la API Amazon EC2
- DescribeAddresses en la referencia a la API Amazon EC2
- ReleaseAddress en la referencia de la API Amazon EC2

Usar regiones y zonas de disponibilidad

Describir regiones

Para mostrar las regiones disponibles para su cuenta, llame al método `describeRegions` del AmazonEC2Client. Este método devuelve un objeto DescribeRegionsResult. Llame al método `getRegions` del objeto devuelto para obtener una lista de objetos Region que representan cada región.

Importaciones

```
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());
}
```

Consulte el ejemplo completo

Describir zonas de disponibilidad

Para mostrar las zonas de disponibilidad disponibles para su cuenta, llame al método `describeAvailabilityZones` del AmazonEc2Client. Este método devuelve un objeto DescribeAvailabilityZonesResult. Llame al método `getAvailabilityZones` del objeto devuelto para obtener una lista de objetos AvailabilityZone que representan cada zona de disponibilidad.

Importaciones

```
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());
```

```
}
```

Consulte el ejemplo completo

Describir cuentas

Para describir su cuenta, llame al método `describeAccountAttributes` del AmazonEC2Client. Este método devuelve un objeto DescribeAccountAttributesResult. Invoque el método `getAccountAttributes` de este objeto para obtener una lista de objetos AccountAttribute. Puede recorrer en iteración la lista para recuperar un objeto AccountAttribute.

Puede obtener los valores de los atributos de su cuenta invocando el método `getAttributeValues` del objeto AccountAttribute. Este método devuelve una lista de objetos AccountAttributeValue. Puede recorrer en iteración esta segunda lista para mostrar el valor de los atributos (consulte el siguiente ejemplo de código).

Importaciones

```
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();
}
```

Consulte el ejemplo completo en GitHub.

Más información

- Regiones y zonas de disponibilidad en la Guía del usuario de Amazon EC2 para instancias de Linux
- DescribeRegions en la Referencia de la API de Amazon EC2
- DescribeAvailabilityZones en la Referencia de la API de Amazon EC2

Uso de pares de claves de Amazon EC2

Creación de un par de claves

Para crear un par de claves, llame al método `createKeyPair` del AmazonEC2Client con una CreateKeyPairRequest que contenga el nombre de la clave.

Importaciones

```
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);
```

Consulte el ejemplo completo

Descripción de pares de claves

Para mostrar sus pares de claves o para obtener información sobre ellos, llame al método `describeKeyPairs` de AmazonEC2Client. Este método devuelve DescribeKeyPairsResult, que se puede utilizar para obtener acceso a la lista de pares de claves mediante la llamada a su método `getKeyPairs`, que devuelve una lista de objetos KeyPairInfo.

Importaciones

```
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());
}
```

Consulte el ejemplo completo

Eliminación de un par de claves

Para eliminar un par de claves, llame al método `deleteKeyPair` de AmazonEC2Client, pasando un objeto DeleteKeyPairRequest que contenga el nombre del par de claves que desea eliminar.

Importaciones

```
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);
```

Consulte el ejemplo completo

Más información

- Pares de claves Amazon EC2 en la Guía del usuario de Amazon EC2 para instancias de Linux
- CreateKeyPairs en la Referencia de la API de Amazon EC2
- DescribeKeyPairs en la referencia de la API de Amazon EC2
- DeleteKeyPair en la Referencia de la API de Amazon EC2

Uso de grupos de seguridad en Amazon EC2

Creación de un grupo de seguridad

Para crear un grupo de seguridad, llame al método `createSecurityGroup` de AmazonEC2Client con una CreateSecurityGroupRequest que contenga el nombre de la clave.

Importaciones

```
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);
```

Consulte el ejemplo completo

Configuración de un grupo de seguridad

Un grupo de seguridad puede controlar el tráfico de entrada y salida a sus instancias Amazon EC2.

Para añadir reglas de entrada al grupo de seguridad, utilice el método `authorizeSecurityGroupIngress` de AmazonEC2Client, proporcionando el nombre del grupo de seguridad y las reglas de acceso (IpPermission) que desea asignar al grupo dentro de un objeto AuthorizeSecurityGroupIngressRequest. El siguiente ejemplo muestra cómo añadir permisos de IP a un grupo de seguridad.

Importaciones

```
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);
```

Para agregar una regla de salida al grupo de seguridad, proporcione datos similares en un objeto AuthorizeSecurityGroupEgressRequest al método `authorizeSecurityGroupEgress` de AmazonEC2Client.

Consulte el ejemplo completo

Descripción de grupos de seguridad

Para describir los grupos de seguridad o para obtener información sobre ellos, llame al método `describeSecurityGroups` de AmazonEC2Client. Este método devuelve DescribeSecurityGroupsResult, que puede usar para obtener acceso a la lista de grupos de seguridad llamando a su método `getSecurityGroups`, que devuelve una lista de objetos SecurityGroup.

Importaciones

```
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];
```

Consulte el ejemplo completo

Eliminación de un grupo de seguridad

Para eliminar un grupo de seguridad, llame al método `deleteSecurityGroup` de AmazonEC2Client, pasando un DeleteSecurityGroupRequest que contenga el ID del grupo de seguridad que desea eliminar.

Importaciones

```
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);
```

Consulte el ejemplo completo

Más información

- Grupos de seguridad de Amazon EC2 en la Guía del usuario de Amazon EC2 para instancias de Linux
- Autorización del tráfico entrante para sus instancias de Linux en la Guía del usuario de Amazon EC2 para instancias de Linux

- CreateSecurityGroup en la Referencia de la API de Amazon EC2
- DescribeSecurityGroups en la Referencia de la API de Amazon EC2
- DeleteSecurityGroup en la Referencia de la API de Amazon EC2
- AuthorizeSecurityGroupIngress en la Referencia de la API de Amazon EC2

Ejemplos de (IAM) con AWS SDK for Java

En esta sección se proporcionan ejemplos de programación en IAM mediante el AWS SDK for Java.

AWS Identity and Access Management (IAM) permite controlar de forma segura el acceso a los servicios y recursos de AWS de los usuarios. Con IAM puede crear y administrar usuarios y grupos de AWS, así como utilizar permisos para conceder o denegar el acceso de estos a los recursos de AWS. Para obtener instrucciones completas de IAM, consulte la Guía del usuario de IAM.

> **Note**
>
> Los ejemplos incluyen únicamente el código necesario para demostrar cada técnica. El código de ejemplo completo está disponible en GitHub. Desde allí, puede descargar un único archivo de código fuente o clonar el repositorio localmente para obtener todos los ejemplos para compilarlos y ejecutarlos.

Temas

- Administración de las claves de acceso de IAM
- Administración de usuarios de IAM
- Uso de alias de cuenta de IAM
- Uso de políticas de IAM
- Uso de certificados de servidor de IAM

Administración de las claves de acceso de IAM

Creación de una clave de acceso

Para crear una clave de acceso de IAM, llame al método `createAccessKey` de AmazonIdentityManagementClient con un objeto CreateAccessKeyRequest.

`CreateAccessKeyRequest` tiene dos constructores: uno que toma un nombre de usuario y otro sin parámetros. Si utiliza la versión que no toma parámetros, debe definir el nombre de usuario mediante el método setter `withUserName` antes de pasarlo al método `createAccessKey`.

Importaciones

```
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);
```

Consulte el ejemplo completo en GitHub.

Mostrar claves de acceso

Para enumerar las claves de acceso de un determinado usuario, cree un objeto ListAccessKeysRequest que contenga el nombre de usuario cuyas claves desea enumerar y páselo al método `listAccessKeys` de AmazonIdentityManagementClient.

> **Note**
>
> Si no proporciona un nombre de usuario a `listAccessKeys`, intentará enumerar las claves de acceso asociadas a la Cuenta de AWS que firmó la solicitud.

Importaciones

```
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;
    }
}
```

Los resultados de `listAccessKeys` están paginados (con un máximo de 100 registros por llamada). Puede llamar a `getIsTruncated` en el objeto ListAccessKeysResult devuelto para saber si la consulta ha devuelto menos resultados de los que están disponibles. En tal caso, llame a `setMarker` en el objeto `ListAccessKeysRequest` y vuelva a pasarlo a la siguiente invocación de `listAccessKeys`.

Consulte el ejemplo completo en GitHub.

Recuperar el momento en que se usó por última vez una clave de acceso

Para obtener el momento en el que se usó por última vez una clave de acceso, llame al método `getAccessKeyLastUsed` de AmazonIdentityManagementClient con el ID de la clave de acceso (que se puede pasar mediante un objeto GetAccessKeyLastUsedRequest) o directamente a la sobrecarga que toma el ID de clave de acceso.

A continuación, puede utilizar el objeto GetAccessKeyLastUsedResult devuelto para recuperar el momento en que se usó por última vez la clave.

Importaciones

```
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());
```

Consulte el ejemplo completo en GitHub.

Activación o desactivación de claves de acceso

Puede activar o desactivar una clave de acceso creando un objeto UpdateAccessKeyRequest, proporcionando el ID de clave de acceso, el nombre de usuario (de forma opcional) y el estado deseado y, a continuación, pasando el objeto al método `updateAccessKey` de AmazonIdentityManagementClient.

Importaciones

```
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);
```

Consulte el ejemplo completo en GitHub.

Eliminación de una clave de acceso

Para eliminar de forma permanente una clave de acceso, llame al método `deleteKey` de AmazonIdentityManagementClient, proporcionando un objeto DeleteAccessKeyRequest que contenga el ID de clave de acceso y el nombre de usuario.

> **Note**
>
> Una vez eliminada una clave, ya no se puede recuperar ni utilizar. Para desactivar temporalmente una clave de forma que pueda activarse de nuevo más adelante, utilice el método updateAccessKey en su lugar.

Importaciones

```
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);
```

Consulte el ejemplo completo en GitHub.

Más información

- CreateAccessKey en la Referencia de la API de IAM
- ListAccessKeys en la referencia de la API de IAM
- GetAccessKeyLastUsed en la referencia de la API de IAM
- UpdateAccessKey en la Referencia de la API de IAM
- DeleteAccessKey en la Referencia de la API de IAM

Administración de usuarios de IAM

Crear un usuario

Cree un nuevo usuario de IAM proporcionando el nombre del usuario al método `createUser` de AmazonIdentityManagementClient, ya sea directamente o a través de un objeto CreateUserRequest que contenga el nombre de usuario.

Importaciones

```
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);
```

Consulte el ejemplo completo en GitHub.

Mostrar usuarios

Para mostrar los usuarios de IAM de su cuenta, cree un nuevo objeto ListUsersRequest y páselo al método `listUsers` de AmazonIdentityManagementClient. Puede recuperar la lista de usuarios llamando a `getUsers` en el objeto ListUsersResult devuelto.

La lista de usuarios devuelta por `listUsers` está paginada. Puede comprobar que no haya más resultados que recuperar llamando al método `getIsTruncated` del objeto de respuesta. Si devuelve `true`, llame al método `setMarker()` del objeto de solicitud pasando el valor devuelto del método `getMarker()` del objeto de respuesta.

Importaciones

```
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;
    }
}
```

Consulte el ejemplo completo en GitHub.

Actualizar un usuario

Para actualizar un usuario, llame al método `updateUser` de AmazonIdentityManagementClient, que toma un objeto UpdateUserRequest que puede utilizar para cambiar el nombre o la ruta del usuario.

Importaciones

```
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);
```

Consulte el ejemplo completo en GitHub.

Eliminar un usuario

Para eliminar un usuario, llame a la solicitud `deleteUser` de AmazonIdentityManagementClient con un objeto UpdateUserRequest definido con el nombre de usuario que desea eliminar.

Importaciones

```
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;
}
```

Consulte el ejemplo completo en GitHub.

Más información

- Usuarios de IAM en la Guía del usuario de IAM
- Gestión de usuarios de IAM en la Guía del usuario de IAM
- CreateUser en la Referencia de la API de IAM
- ListUsers en la Referencia de la API de IAM
- UpdateUser en la referencia de la API de IAM
- DeleteUser en la referencia de la API de IAM

Uso de alias de cuenta de IAM

Si quiere que la dirección URL de la página de inicio de sesión contenga el nombre de su empresa u otro identificador intuitivo en lugar de su ID de Cuenta de AWS, puede crear un alias para su Cuenta de AWS.

> **Note**
>
> AWS admite exactamente un alias de cuenta por cuenta.

Creación de un alias de cuenta

Para crear un alias de cuenta, llame al método `createAccountAlias` de AmazonIdentityManagementClient con un objeto CreateAccountAliasRequest que contenga el nombre del alias.

Importaciones

```
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);
```

Consulte el ejemplo completo en GitHub.

Mostrar alias de cuenta

Para mostrar sus alias de cuenta, si hay alguno, llame al método `listAccountAliases` de AmazonIdentityManagementClient.

> **Note**
>
> El ListAccountAliasesResult devuelto admite los mismos métodos `getIsTruncated` y `getMarker` que otros métodos list de AWS SDK for Java, pero una Cuenta de AWS solo puede tener un alias de cuenta.

Importaciones.

```
import com.amazonaws.services.identitymanagement.AmazonIdentityManagement;
import com.amazonaws.services.identitymanagement.AmazonIdentityManagementClientBuilder;
import com.amazonaws.services.identitymanagement.model.ListAccountAliasesResult;
```

código

```
final AmazonIdentityManagement iam =
```

```
    AmazonIdentityManagementClientBuilder.defaultClient();

ListAccountAliasesResult response = iam.listAccountAliases();

for (String alias : response.getAccountAliases()) {
    System.out.printf("Retrieved account alias %s", alias);
}
```

Consulte el ejemplo completo en GitHub.

Eliminación de un alias de cuenta

Para mostrar sus alias de cuenta, si hay alguno, llame al método `deleteAccountAlias` de AmazonIdentityManagementClient. Al eliminar un alias de cuenta, debe proporcionar su nombre mediante un objeto DeleteAccountAliasRequest.

Importaciones.

```
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);
```

Consulte el ejemplo completo en GitHub.

Más información

- Su ID de cuenta AWS y su alias en la Guía del usuario de IAM
- CreateAccountAlias en la referencia de la API de IAM
- ListAccountAliases en la referencia de la API de IAM
- DeleteAccountAlias en la referencia de la API de IAM

Uso de políticas de IAM

Creación de una política

Para crear una nueva política, proporcione el nombre de la política y un documento de política en formato JSON en un objeto CreatePolicyRequest al método `createPolicy` de AmazonIdentityManagementClient.

Importaciones

```
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);
```

Los documentos de política de IAM; son cadenas JSON con una sintaxis bien documentada. Este es un ejemplo que proporciona acceso para realizar determinadas solicitudes a 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\"" +
    "    }" +
    "   ]" +
    "}";
```

Consulte el ejemplo completo en GitHub.

Obtención de una política

Para recuperar una política existente, llame al método `getPolicy` de AmazonIdentityManagementClient, proporcionando el ARN de la política en un objeto GetPolicyRequest.

Importaciones

```
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);
```

Consulte el ejemplo completo en GitHub.

Asociar una política de rol

Puede adjuntar una política a un IAM http://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles.html[rol] llamando al `attachRolePolicy` método de AmazonIdentityManagementClient y proporcionándole el nombre del rol y el ARN de la política en una AttachRolePolicyRequest.

Importaciones

```
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);
```

Consulte el ejemplo completo en GitHub.

Mostrar las políticas de rol asociadas

Enumerar las políticas asociadas a un rol llamando al método `listAttachedRolePolicies` de AmazonIdentityManagementClient. Este método toma un objeto ListAttachedRolePoliciesRequest que contiene el nombre del rol para el que se desea mostrar las políticas.

Llame a `getAttachedPolicies` en el objeto ListAttachedRolePoliciesResult devuelto para obtener la lista de políticas asociadas. Los resultados pueden aparecer truncados; si el método `getIsTruncated` del objeto `ListAttachedRolePoliciesResult` devuelve `true`, llame al método `setMarker` del objeto `ListAttachedRolePoliciesRequest` y úselo para llamar a `listAttachedRolePolicies` de nuevo para obtener el siguiente lote de resultados.

Importaciones

```
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());
}
```

Consulte el ejemplo completo en GitHub.

Desvincular una política de rol

Para desvincular una política de un rol, llame al método `detachRolePolicy` de AmazonIdentityManagementClient proporcionando el nombre de rol y el ARN de política en un objeto DetachRolePolicyRequest.

Importaciones

```
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);
```

Consulte el ejemplo completo en GitHub.

Más información

- Descripción general de políticas de IAM en la Guía del usuario de IAM.
- Referencia de políticas de AWS IAM en la Guía del usuario de IAM.
- CreatePolicy en la referencia de la API de IAM
- GetPolicy en la referencia de la API de IAM
- AttachRolePolicy en la referencia de la API de IAM
- ListAttachedRolePolicies en la referencia de la API de IAM
- DetachRolePolicy en la referencia de la API de IAM

Uso de certificados de servidor de IAM

Para habilitar las conexiones HTTPS en su sitio web o aplicación en AWS, necesita un certificado de servidor SSL/TLS. Puede utilizar un certificado de servidor proporcionado por AWS Certificate Manager o uno que haya obtenido de un proveedor externo.

Es recomendable utilizar ACM para aprovisionar, administrar e implementar los certificados de servidor. Con ACM puede solicitar un certificado, implementarlo en los recursos de AWS y dejar que ACM se ocupe de la renovación de los certificados. Los certificados proporcionados por ACM son gratuitos. Para obtener más información acerca de ACM, consulte la Guía del usuario de ACM.

Obtener un certificado de servidor

Puede recuperar un certificado de servidor llamando al método `getServerCertificate` de AmazonIdentityManagementClient, pasando un objeto GetServerCertificateRequest con el nombre del certificado.

Importaciones

```
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);
```

Consulte el ejemplo completo en GitHub.

Mostrar certificados de servidor

Para mostrar sus certificados de servidor, llame al método `listServerCertificates` de AmazonIdentityManagementClient con un objeto ListServerCertificatesRequest. Este método devuelve un objeto ListServerCertificatesResult.

Llame al método `getServerCertificateMetadataList` del objeto `ListServerCertificateResult` devuelto para obtener una lista de objetos ServerCertificateMetadata que puede usar para obtener información de cada certificado.

Los resultados pueden aparecer truncados; si el método `getIsTruncated` del objeto `ListServerCertificateResult` devuelve `true`, llame al método `setMarker` del objeto `ListServerCertificatesRequest` y úselo para llamar a `listServerCertificates` de nuevo para obtener el siguiente lote de resultados.

Importaciones

```
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;
    }
}
```

Consulte el ejemplo completo en GitHub.

Actualizar un certificado de servidor

Puede actualizar el nombre o la ruta de un certificado de servidor llamando al método `updateServerCertificate` de AmazonIdentityManagementClient. Este método toma un objeto UpdateServerCertificateRequest establecido con el nombre actual del certificado de servidor o el nuevo nombre o la nueva ruta que se va a usar.

Importaciones

```
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);
```

Consulte el ejemplo completo en GitHub.

Eliminar un certificado de servidor

Para eliminar un certificado de servidor, llame al método `deleteServerCertificate` de AmazonIdentityManagementClient, con un objeto DeleteServerCertificateRequest que contenga el nombre del certificado.

Importaciones

```
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);
```

Consulte el ejemplo completo en GitHub.

Más información

- Uso de certificados de servidor en la Guía del usuario de IAM
- GetServerCertificate en la referencia de la API de IAM
- ListServerCertificates en la referencia de la API de IAM
- UpdateServerCertificate en la referencia de la API de IAM
- DeleteServerCertificate en la referencia de la API de IAM
- Guía del usuario de ACM

Ejemplos de Lambda usando la AWS SDK for Java

En esta sección se proporcionan ejemplos de programación en Lambda mediante AWS SDK for Java.

> **Note**
>
> Los ejemplos incluyen únicamente el código necesario para demostrar cada técnica. El código de ejemplo completo está disponible en GitHub. Desde allí, puede descargar un único archivo de código fuente o clonar el repositorio localmente para obtener todos los ejemplos para compilarlos y ejecutarlos.

Temas

- Invocar, enumerar y eliminar funciones de Lambda

Invocar, enumerar y eliminar funciones de Lambda

En esta sección se proporcionan ejemplos de programación con el cliente de servicio de Lambda mediante el AWS SDK for Java. Para aprender a crear una función Lambda, consulte Cómo crear funciones AWS Lambda.

Temas

Invocar una función

Puede invocar una función Lambda creando un objeto AWSLambda e invocando su método `invoke`. Cree un objeto InvokeRequest para especificar información adicional, como el nombre de la función y la carga útil que pasar a la función Lambda. Los nombres de funciones aparecen como arn:aws:lambda:us-east-1:555556330391:function:HelloFunction. Puede recuperar el valor viendo la función en la AWS Management Console.

Para pasar datos de carga útil a una función, invoque el método `withPayload` del objeto InvokerEquest y especifique una cadena en formato JSON, como se muestra en el siguiente ejemplo de código.

Importaciones

```
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

En el ejemplo de código siguiente se muestra cómo invocar una función Lambda.

```
        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());
```

Consulte el ejemplo completo en Github.

Mostrar funciones

Construya un objeto AWSLambda e invoque su método `listFunctions`. Este método devuelve un objeto ListFunctionsResult. Puede invocar el método `getFunctions` de este objeto para devolver una lista de objetos FunctionConfiguration. Puede recorrer la lista en iteración para recuperar información sobre las funciones. Por ejemplo, el siguiente ejemplo de código Java muestra cómo obtener el nombre de cada función.

Importaciones

```
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

El siguiente ejemplo de código Java muestra cómo recuperar una lista de nombres de funciones de Lambda.

```
        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);
        }
```

Consulte el ejemplo completo en Github.

Eliminar una función

Construya un objeto AWSLambda e invoque su método `deleteFunction`. Cree un objeto DeleteFunctionRequest y páselo al método `deleteFunction`. Este objeto contiene información como el nombre de la función que se va a eliminar. Los nombres de funciones aparecen como arn:aws:lambda:us-east-1:555556330391:function:HelloFunction. Puede recuperar el valor viendo la función en la AWS Management Console.

Importaciones

```
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

El siguiente código Java muestra cómo eliminar una función Lambda.

```
        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);
        }
```

Consulte el ejemplo completo en Github.

Ejemplos de Amazon Pinpoint usando la AWS SDK for Java

En esta sección se proporcionan ejemplos de programación en Amazon Pinpoint mediante AWS SDK for Java.

> **Note**
>
> Los ejemplos incluyen únicamente el código necesario para demostrar cada técnica. El código de ejemplo completo está disponible en GitHub. Desde allí, puede descargar un único archivo de código fuente o clonar el repositorio localmente para obtener todos los ejemplos para compilarlos y ejecutarlos.

Temas

Creación y eliminación de aplicaciones en Amazon Pinpoint

Una aplicación es un proyecto de Amazon Pinpoint en el que se define el público de una aplicación y se intenta atraer a dicho público con mensajes personalizados. Los ejemplos de esta página demuestran cómo crear una aplicación o cómo eliminar una existente.

Crear una aplicación

Para crear una aplicación en Amazon Pinpoint, proporcione un nombre de aplicación al objeto CreateAppRequest y, a continuación, pase dicho objeto al método `createApp` de AmazonPinpointClient.

Importaciones

```
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);
```

Consulte el ejemplo completo en GitHub.

Eliminar una aplicación

Para eliminar una aplicación, llame a la solicitud `deleteApp` de AmazonPinpointClient con un objeto DeleteAppRequest en el que se indica el nombre de la aplicación que se va a eliminar.

Importaciones

```
import com.amazonaws.services.pinpoint.AmazonPinpoint;
import com.amazonaws.services.pinpoint.AmazonPinpointClientBuilder;
```

Code

```
DeleteAppRequest deleteRequest = new DeleteAppRequest()
  .withApplicationId(appID);

pinpoint.deleteApp(deleteRequest);
```

Consulte el ejemplo completo en GitHub.

Más información

- Apps en la Referencia de la API de Amazon Pinpoint
- App en la Referencia de la API de Amazon Pinpoint

Creación de puntos de conexón en Amazon Pinpoint

Un punto de enlace identifica de forma exclusiva el dispositivo de un usuario al que puede enviar notificaciones de inserción con Amazon Pinpoint. Si la aplicación está habilitada con la compatibilidad con Amazon Pinpoint, registra automáticamente un punto de enlace en Amazon Pinpoint cuando la abre un nuevo usuario. En el siguiente ejemplo se muestra cómo añadir un nuevo punto de enlace mediante programación.

Crear un punto de enlace

Para crear un punto de enlace en Amazon Pinpoint, proporcione los datos correspondientes a este en un objeto EndpointRequest.

Importaciones

```
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);
```

A continuación, cree un objeto UpdateEndpointRequest con ese objeto EndpointRequest. Por último, pase el objeto UpdateEndpointRequest al método `updateEndpoint` de AmazonPinpointClient.

Code

```
UpdateEndpointRequest updateEndpointRequest = new UpdateEndpointRequest()
        .withApplicationId(appId)
        .withEndpointId(endpointId)
        .withEndpointRequest(endpointRequest);

UpdateEndpointResult updateEndpointResponse =
 client.updateEndpoint(updateEndpointRequest);
System.out.println("Update Endpoint Response: " +
 updateEndpointResponse.getMessageBody());
```

Consulte el ejemplo completo en GitHub.

Más información

- Añadir punto de conexión en la Guía para desarrolladores de Amazon Pinpoint
- Endpoint en la referencia de la APP de Amazon Pinpoint

Creación de segmentos en Amazon Pinpoint

Un segmento de usuarios representa un subconjunto de los usuarios basado en ciertas características compartidas, por ejemplo, cuándo abrió el usuario la aplicación por última vez o qué dispositivo utiliza. El siguiente ejemplo muestra cómo definir un segmento de usuarios.

Crear un segmento

Para crear un segmento en Amazon Pinpoint, defina las dimensiones del segmento en un objeto SegmentDimensions.

Importaciones

```
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);
```

A continuación, establezca el objeto SegmentDimensions en una solicitud WriteSegmentRequest que, a su vez, se utiliza para crear un objeto CreateSegmentRequest. Después, pase el objeto CreateSegmentRequest al método `createSegment` de AmazonPinpointClient.

Code

```
WriteSegmentRequest writeSegmentRequest = new WriteSegmentRequest()
        .withName("MySegment").withDimensions(dimensions);

CreateSegmentRequest createSegmentRequest = new CreateSegmentRequest()
        .withApplicationId(appId).withWriteSegmentRequest(writeSegmentRequest);

CreateSegmentResult createSegmentResult = client.createSegment(createSegmentRequest);
```

Consulte el ejemplo completo en GitHub.

Más información

- Segmentos de Amazon Pinpoint en la Guía del usuario de Amazon Pinpoint
- Crear segmentos en la Guía para desarrolladores de Amazon Pinpoint.
- Segmentos en la referencia de la API de Amazon Pinpoint
- Segmento en la Referencia de la API de Amazon Pinpoint

Creación de campañas en Amazon Pinpoint

Use las campañas para ayudar a aumentar la conexión entre la aplicación y los usuarios. Puede crear una campaña que se dirija a un segmento de usuarios concretos, con mensajes personalizados o promociones especiales. Este ejemplo demuestra cómo crear una campaña estándar que envía una notificación de inserción personalizada a un segmento especificado.

Crear una campaña

Antes de crear una campaña, debe definir un valor para Schedule y Message y establecer ambos valores en un objeto WriteCampaignRequest.

Importaciones

```
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);
```

A continuación, cree una campaña en Amazon Pinpoint. Para ello, proporcione el elemento WriteCampaignRequest con la configuración de la campaña a un objeto CreateCampaignRequest. Por último, pase el objeto CreateCampaignRequest al método `createCampaign` de AmazonPinpointClient.

Code

```
CreateCampaignRequest createCampaignRequest = new CreateCampaignRequest()
```

```
        .withApplicationId(appId).withWriteCampaignRequest(request);

CreateCampaignResult result = client.createCampaign(createCampaignRequest);
```

Consulte el ejemplo completo en GitHub.

Más información

- Campañas de Amazon Pinpoint en la Guía del usuario de Amazon Pinpoint
- Creación de campañas en la Guía para desarrolladores de Amazon Pinpoint.
- Campañas en la Referencia de la API de Amazon Pinpoint
- Campaña en la Referencia de la API de Amazon Pinpoint
- Actividades de campaña en la Referencia de la API de Amazon Pinpoint
- Versiones de campaña en la Referencia de la API de Amazon Pinpoint
- Versión de campaña en la Referencia de la API de Amazon Pinpoint

Actualización de canales en Amazon Pinpoint

Un canal define los tipos de plataformas a los que puede entregar mensajes. Este ejemplo muestra cómo utilizar los canales de APNS para enviar un mensaje.

Actualizar un canal

Para habilitar un canal en Amazon Pinpoint, proporcione un ID de aplicación y un objeto de solicitud del tipo de canal que desea actualizar. En este ejemplo se actualiza el canal de APNS, que requiere el objeto APNSChannelRequest. Defina estos valores en UpdateApnsChannelRequest y pase el objeto al método `updateApnsChannel` de AmazonPinpointClient.

Importaciones

```
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);
```

Consulte el ejemplo completo en GitHub.

Más información

- Canales de Amazon Pinpoint en la Guía del usuario de Amazon Pinpoint
- ADM Channel en la referencia de la API de Amazon Pinpoint
- APNs Channel en la referencia de la API de Amazon Pinpoint
- APNs Sandbox Channel en la referencia de la API de Amazon Pinpoint
- APNs VoIP Channel en la referencia de la API de Amazon Pinpoint
- APNs VoIP Sandbox Channel en la referencia de la API de Amazon Pinpoint
- Baidu Channel en la referencia de la API de Amazon Pinpoint
- Email Channel en la referencia de la API de Amazon Pinpoint
- GCM Channel en la referencia de la API de Amazon Pinpoint
- SMS Channel en la referencia de la API de Amazon Pinpoint

Ejemplos de Amazon S3 usando la AWS SDK for Java

En esta sección se proporcionan ejemplos de programación en Amazon S3 mediante AWS SDK for Java.

> **Note**
>
> Los ejemplos incluyen únicamente el código necesario para demostrar cada técnica. El código de ejemplo completo está disponible en GitHub. Desde allí, puede descargar un único

archivo de código fuente o clonar el repositorio localmente para obtener todos los ejemplos para compilarlos y ejecutarlos.

Temas

Creación, enumeración y eliminación de buckets de Amazon S3

Todos los objetos (archivos) de Amazon S3 deben residir en un bucket, que representa una colección (contenedor) de objetos. Cada bucket se designa por medio de una clave (nombre), que debe ser única. Para obtener información detallada acerca de los buckets y su configuración, consulte Uso de buckets de Amazon S3 en la Guía del usuario de Amazon Simple Storage Service.

> **Note**
>
> Práctica recomendada
> Le recomendamos que habilite la regla del ciclo de vida AbortIncompleteMultipartUpload en los buckets de Amazon S3.
> Esta regla le indica a Amazon S3 que anule las cargas multiparte que no se completen en un número especificado de días después de iniciarse. Cuando se supera el plazo establecido, Amazon S3 anula la carga y, a continuación, elimina la carga de datos incompleta.
> Para obtener más información, consulte Configuración de ciclo de vida para un bucket con control de versiones en la Guía del usuario de Amazon S3.

> **Note**
>
> En estos ejemplos de código se presupone que conoce la información que se describe en Uso del AWS SDK for Java y que ha configurado credenciales de AWS predeterminadas

mediante la información de Configuración de credenciales y regiones de AWS para desarrollo.

Crear un bucket

Utilizar el método `createBucket` del cliente AmazonS3. Se devuelve el nuevo bucket. El método `createBucket` producirá una excepción si el bucket ya existe.

> **Note**
>
> Para comprobar si un bucket ya existe antes de intentar crear uno con el mismo nombre, llame al método `doesBucketExist`. Este método devolverá `true` si el bucket existe y `false` en caso contrario.

Importaciones

```
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;
```

Consulte el ejemplo completo en GitHub.

Lista de buckets

Utilizar el método `listBucket` del cliente AmazonS3. Si se ejecuta correctamente, se devuelve una lista de buckets.

Importaciones

```
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());
}
```

Consulte el ejemplo completo en GitHub.

Eliminar un bucket

Antes de eliminar un bucket de Amazon S3, debe asegurarse de que el bucket está vacío o se producirá un error. Si tiene un bucket con control de versiones, también debe eliminar todos los objetos con control de versiones asociados al bucket.

> **Note**
>
> El ejemplo completo incluye cada uno de estos pasos en orden, lo que constituye una solución completa para eliminar un bucket de Amazon S3 y su contenido.

Temas

- Eliminar objetos de un bucket sin control de versiones antes de eliminarlo
- Eliminar objetos de un bucket con control de versiones antes de eliminarlo
- Eliminar un bucket vacío

Eliminar objetos de un bucket sin control de versiones antes de eliminarlo

Utilice el método `listObjects` del cliente AmazonS3 para recuperar la lista de objetos y `deleteObject` para eliminar cada uno de ellos.

Importaciones

```
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;
    }
}
```

Consulte el ejemplo completo en GitHub.

Eliminar objetos de un bucket con control de versiones antes de eliminarlo

Si utiliza un bucket con control de versiones, también tendrá que eliminar todas las versiones almacenadas de los objetos del bucket para poder eliminarlo.

Siguiendo un patrón similar al utilizado para eliminar objetos dentro de un bucket, elimine los objetos con control de versiones utilizando el método `listVersions` del cliente AmazonS3 para mostrar todos los objetos con control de versiones y después `deleteVersion` para eliminar cada uno de ellos.

Importaciones

```
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;
    }
}
```

Consulte el ejemplo completo en GitHub.

Eliminar un bucket vacío

Después de eliminar los objetos de un bucket (incluidos los objetos con control de versiones), puede eliminar el propio bucket mediante el método `deleteBucket` del cliente AmazonS3.

Importaciones

```
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);
```

Consulte el ejemplo completo en GitHub.

Realizar operaciones en objetos de Amazon S3

Un objeto de Amazon S3 representa un archivo o conjunto de datos. Cada objeto debe residir en un bucket.

> **Note**
>
> En estos ejemplos de código se presupone que conoce la información que se describe en Uso del AWS SDK for Java y que ha configurado credenciales de AWS predeterminadas mediante la información de Configuración de credenciales y regiones de AWS para desarrollo.

Temas

- Carga de un objeto

- Lista de objetos
- Descarga de un objeto
- Copiar, mover o cambiar de nombre objetos
- Eliminar un objeto
- Eliminación de varios objetos a la vez

Carga de un objeto

Utilice el método `putObject` del cliente AmazonS3, proporcionando un nombre de bucket, un nombre de clave y el archivo que se va a cargar. El bucket debe existir o se producirá un error.

Importaciones

```
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);
}
```

Consulte el ejemplo completo en GitHub.

Lista de objetos

Para obtener una lista de objetos dentro de un bucket, utilice el método `listObjects` del cliente AmazonS3, proporcionando el nombre de un bucket.

El método `listObjects` devuelve un objeto ObjectListing que proporciona información acerca de los objetos del bucket. Para mostrar los nombres de objeto (claves), utilice el método

`getObjectSummaries` para obtener una lista de objetos S3ObjectSummary, cada uno de los cuales representa un solo objeto del bucket. A continuación, llame a su método `getKey` para recuperar el nombre del objeto.

Importaciones

```
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());
}
```

Consulte el ejemplo completo en GitHub.

Descarga de un objeto

Utilice el método `getObject` del cliente AmazonS3, pasando el nombre del bucket y el objeto que se van a descargar. Si se ejecuta correctamente, el método devuelve un S3Object. El bucket y la clave de objeto especificados deben existir o se producirá un error.

Puede obtener el contenido del objeto llamando a `getObjectContent` en el `S3Object`. Esto devuelve un S3ObjectInputStream que se comporta como un objeto `InputStream` Java estándar.

El siguiente ejemplo descarga un objeto de S3 y guarda su contenido en un archivo (con el mismo nombre que la clave del objeto).

Importaciones

```
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);
}
```

Consulte el ejemplo completo en GitHub.

Copiar, mover o cambiar de nombre objetos

Puede copiar un objeto de un bucket en otro mediante el método `copyObject` del cliente AmazonS3. Este método toma el nombre del bucket desde el que se va a realizar la copia, el objeto destino de la copia y el nombre del bucket de destino.

Importaciones

```
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!");
```

Consulte el ejemplo completo en GitHub.

> **Note**
>
> Puede utilizar `copyObject` con deleteObject para mover o cambiar de nombre un objeto. Para ello, primero copie el objeto en un nuevo nombre (puede utilizar el mismo bucket como origen y destino) y, a continuación, elimine el objeto de su antigua ubicación.

Eliminar un objeto

Utilice el método `deleteObject` del cliente AmazonS3, pasando el nombre del bucket y el objeto que se van a eliminar. El bucket y la clave de objeto especificados deben existir o se producirá un error.

Importaciones

```
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);
}
```

Consulte el ejemplo completo en GitHub.

Eliminación de varios objetos a la vez

Con el método `deleteObjects` del cliente AmazonS3, puede eliminar varios objetos del mismo bucket pasando sus nombres al método link:sdk-for-java/v1/reference/com/amazonaws/services/s3/model/DeleteObjectsRequest.html.

Importaciones

```
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);
}
```

Consulte el ejemplo completo en GitHub.

Administración de permisos de acceso de Amazon S3 para buckets y objetos

Puede utilizar listas de control de acceso (ACL) para los buckets y objetos de Amazon S3 para obtener un control detallado de los recursos de Amazon S3.

> **Note**
>
> En estos ejemplos de código se presupone que conoce la información que se describe en Uso del AWS SDK for Java y que ha configurado credenciales de AWS predeterminadas mediante la información de Configuración de credenciales y regiones de AWS para desarrollo.

Obtener la lista de control de acceso de un bucket

Para obtener la ACL actual de un bucket, llame al método `getBucketAcl` de AmazonS3, pasando el nombre de bucket que se desea consultar. Este método devuelve un objeto AccessControlList. Para obtener cada concesión de acceso en la lista, llame a su método `getGrantsAsList`, que devuelve una lista Java estándar de objetos Grant.

Importaciones

```
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);
}
```

Consulte el ejemplo completo en GitHub.

Establecer la lista de control de acceso de un bucket

Para añadir o modificar permisos de la ACL de un bucket, llame al método `setBucketAcl` de AmazonS3. Este método toma un objeto AccessControlList que contiene una lista de los destinatarios del acceso y los niveles de acceso que se van a establecer.

Importaciones

```
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**
>
> Puede proporcionar directamente el identificador único del destinatario del acceso mediante la clase Grantee o puede usar la clase EmailAddressGrantee para definir el destinatario del acceso por correo electrónico, como hemos hecho aquí.

Consulte el ejemplo completo en GitHub.

Obtener la lista de control de acceso de un objeto

Para obtener la ACL actual de un objeto, llame al método `getObjectAcl` de AmazonS3, pasando el nombre de bucket y el nombre de objeto que se desea consultar. Al igual que `getBucketAcl`, este método devuelve un objeto AccessControlList, que puede utilizar para examinar cada uno de los objetos Grant.

Importaciones

```
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);
}
```

Consulte el ejemplo completo en GitHub.

Establecer la lista de control de acceso de un objeto

Para añadir o modificar permisos para la ACL de un objeto, llame al método `setObjectAcl` de AmazonS3. Este método toma un objeto AccessControlList que contiene una lista de los destinatarios del acceso y los niveles de acceso que se van a establecer.

Importaciones

```
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**
>
> Puede proporcionar directamente el identificador único del destinatario del acceso mediante la clase Grantee o puede usar la clase EmailAddressGrantee para definir el destinatario del acceso por correo electrónico, como hemos hecho aquí.

Consulte el ejemplo completo en GitHub.

Más información

- GET Bucket acl en la referencia de la API de Amazon S3
- PUT Bucket acl en la referencia de la API de Amazon S3
- GET Object acl en la referencia de la API de Amazon S3
- PUT Object acl en la referencia de la API de Amazon S3

Administración del acceso a los buckets de Amazon S3 mediante políticas de buckets

Puede definir, obtener o eliminar una política de bucket para administrar el acceso a los buckets de Amazon S3.

Definir una política de bucket

Puede definir la política de bucket para un determinado bucket de S3:

- Llamando al `setBucketPolicy` de AmazonS3 client y proporcionándole un SetBucketPolicyRequest
- Estableciendo la política directamente mediante la sobrecarga `setBucketPolicy`, que toma un nombre de bucket y el texto de la política (en formato JSON)

Importaciones

```
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);
}
```

Usar la clase Class para generar o validar una política

Cuando proporciona una política de bucket a `setBucketPolicy`, puede hacer lo siguiente:

- Especificar la política directamente como una cadena de texto con formato JSON
- Crear la política con la clase Policy

Si utiliza la clase `Policy`, no tendrá que preocuparse de formatear correctamente la cadena de texto. Para obtener el texto de la política JSON de la clase `Policy`, utilice su método `toJson`.

Importaciones

```
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();
```

La clase `Policy` proporciona también un método `fromJson` que intenta crear una política mediante una cadena JSON que se haya pasado. El método valida la cadena para garantizar que el texto se pueda transformar en una estructura de política válida y dará un error `IllegalArgumentException` si el texto de la política no es válido.

```
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());
}
```

Puede utilizar esta técnica para validar previamente una política leída de un archivo o por otros medios.

Consulte el ejemplo completo en GitHub.

Obtener una política de bucket

Para recuperar la política de un bucket de Amazon S3, llame al método `getBucketPolicy` del AmazonS3 client pasando el nombre del bucket del que se va a obtener la política.

Importaciones

```
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);
}
```

Si el bucket especificado no existe, si no tiene acceso a él o si no tiene una política de bucket, se produce la excepción `AmazonServiceException`.

Consulte el ejemplo completo en GitHub.

Eliminar una política de bucket

Para eliminar una política de bucket, llame al `deleteBucketPolicy` de AmazonS3 client proporcionando el nombre del bucket.

Importaciones

```
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);
```

```
    }
```

Este método se ejecuta correctamente aunque el bucket aún no tenga una política. Si especifica el nombre de un bucket que no existe o si no tiene acceso al bucket, se produce la excepción `AmazonServiceException`.

Consulte el ejemplo completo en GitHub.

Más información

- Información general del lenguaje de la política de acceso en la Guía del usuario de Amazon Simple Storage Service
- Ejemplos de política de bucket en la Guía del usuario de Amazon Simple Storage Service

Uso de TransferManager para operaciones de Amazon S3

Puede utilizar la clase AWS SDK for Java TransferManager para transferir archivos de forma fiable desde el entorno local a Amazon S3 y para copiar objetos de una ubicación de S3 a otra. `TransferManager` puede obtener el progreso de la transferencia y detener o reanudar las cargas y descargas.

> **Note**
>
> Práctica recomendada
> Le recomendamos que habilite la regla del ciclo de vida AbortIncompleteMultipartUpload en los buckets de Amazon S3.
> Esta regla le indica a Amazon S3 que anule las cargas multiparte que no se completen en un número especificado de días después de iniciarse. Cuando se supera el plazo establecido, Amazon S3 anula la carga y, a continuación, elimina la carga de datos incompleta.
> Para obtener más información, consulte Configuración de ciclo de vida para un bucket con control de versiones en la Guía del usuario de Amazon S3.

> **Note**
>
> En estos ejemplos de código se presupone que conoce la información que se describe en Uso del AWS SDK for Java y que ha configurado credenciales de AWS predeterminadas

mediante la información de Configuración de credenciales y regiones de AWS para desarrollo.

Carga de archivos y directorios

TransferManager puede cargar archivos, listas de archivos y directorios en cualquier bucket de Amazon S3 que haya creado previamente.

Temas

- Carga de un solo archivo
- Carga de una lista de archivos
- Carga de un directorio

Carga de un solo archivo

Llame al método `upload` de TransferManager, proporcionando un nombre de bucket de Amazon S3, un nombre de clave (objeto) y un objeto File Java estándar que represente el archivo que se va a cargar.

Importaciones

```
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();
```

El método `upload` regresa inmediatamente, proporcionando el objeto `Upload` que se va a usar para comprobar el estado de la transferencia o esperar a que se complete.

Consulte Esperar a que se complete una transferencia para obtener información sobre cómo usar `waitForCompletion` para completar correctamente una transferencia antes de llamar al método `shutdownNow` de TransferManager. Mientras espera a que se complete la transferencia, puede buscar o atender las actualizaciones sobre su estado y su progreso. Consulte Obtener el estado y el progreso de una transferencia para obtener más información.

Consulte el ejemplo completo en GitHub.

Carga de una lista de archivos

Para cargar varios archivos en una sola operación, llame al método `uploadFileList` de TransferManager, proporcionando lo siguiente:

- Un nombre de bucket de Amazon S3
- Un prefijo de clave para adjuntarlo a los nombres de los objetos creados (la ruta en el bucket en el que se colocan los objetos)
- Un objeto File que represente el directorio relativo desde el que crean las rutas de archivo
- Un objeto List que contenga el conjunto de objetos File que se van a cargar

Importaciones

```
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();
```

Consulte Esperar a que se complete una transferencia para obtener información sobre cómo usar `waitForCompletion` para completar correctamente una transferencia antes de llamar al método `shutdownNow` de TransferManager. Mientras espera a que se complete la transferencia, puede buscar o atender las actualizaciones sobre su estado y su progreso. Consulte Obtener el estado y el progreso de una transferencia para obtener más información.

El objeto MultipleFileUpload devuelto por `uploadFileList` se puede usar para consultar el estado o el progreso de la transferencia. Consulte Sondear el progreso actual de una transferencia y Obtener el progreso de una transferencia con ProgressListener para obtener más información.

También puede usar el método `MultipleFileUpload` de `getSubTransfers` para obtener los distintos objetos `Upload` para cada archivo que se va a transferir. Para obtener más información, consulte Obtener el progreso de las transferencias secundarias.

Consulte el ejemplo completo en GitHub.

Carga de un directorio

Puede utilizar el método `uploadDirectory` de TransferManager para cargar un directorio de archivos completo, con la opción de copiar archivos en subdirectorios recursivamente. Proporciona el nombre de un bucket de Amazon S3, un prefijo de clave de S3, un objeto File que representa el directorio local donde se va a realizar la copia y un valor `boolean` que indica si desea copiar los subdirectorios recursivamente (true o false).

Importaciones

```
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();
```

Consulte Esperar a que se complete una transferencia para obtener información sobre cómo usar `waitForCompletion` para completar correctamente una transferencia antes de llamar al método `shutdownNow` de TransferManager. Mientras espera a que se complete la transferencia, puede buscar o atender las actualizaciones sobre su estado y su progreso. Consulte Obtener el estado y el progreso de una transferencia para obtener más información.

El objeto MultipleFileUpload devuelto por `uploadFileList` se puede usar para consultar el estado o el progreso de la transferencia. Consulte Sondear el progreso actual de una transferencia y Obtener el progreso de una transferencia con ProgressListener para obtener más información.

También puede usar el método `MultipleFileUpload` de `getSubTransfers` para obtener los distintos objetos `Upload` para cada archivo que se va a transferir. Para obtener más información, consulte Obtener el progreso de las transferencias secundarias.

Consulte el ejemplo completo en GitHub.

Descarga de archivos o directorios

Utilice la clase TransferManager para descargar un solo archivo (objeto Amazon S3) o un directorio (el nombre de un bucket de Amazon S3 seguido de un prefijo de objeto) de Amazon S3.

Temas

- Descarga de un solo archivo
- Descarga de un directorio

Descarga de un solo archivo

Utilice el método `download` de TransferManager, proporcionando el nombre del bucket de Amazon S3 que contiene el objeto que desea descargar, la clave (nombre de objeto) y un objeto File que represente el archivo que va a crear en su sistema local.

Importaciones

```
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();
```

Consulte Esperar a que se complete una transferencia para obtener información sobre cómo usar `waitForCompletion` para completar correctamente una transferencia antes de llamar al método `shutdownNow` de TransferManager. Mientras espera a que se complete la transferencia, puede buscar o atender las actualizaciones sobre su estado y su progreso. Consulte Obtener el estado y el progreso de una transferencia para obtener más información.

Consulte el ejemplo completo en GitHub.

Descarga de un directorio

Para descargar un conjunto de archivos que comparten un prefijo de clave común (similar a un directorio en un sistema de archivos) desde Amazon S3, utilice el método `downloadDirectory` de TransferManager. El método toma el nombre del bucket de Amazon S3 que contiene los objetos que desea descargar, el prefijo de objeto compartido por todos los objetos y un objeto File que representa el directorio en el que se van a descargar los archivos en su sistema local. Si el directorio designado aún no existe, se creará.

Importaciones

```
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();
```

Consulte Esperar a que se complete una transferencia para obtener información sobre cómo usar `waitForCompletion` para completar correctamente una transferencia antes de llamar al método `shutdownNow` de TransferManager. Mientras espera a que se complete la transferencia, puede buscar o atender las actualizaciones sobre su estado y su progreso. Consulte Obtener el estado y el progreso de una transferencia para obtener más información.

Consulte el ejemplo completo en GitHub.

Copia de objetos

Para copiar un objeto en un bucket de S3 en otro, utilice el método `copy` de TransferManager.

Importaciones

```
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();
```

Consulte el ejemplo completo en GitHub.

Esperar a que se complete una transferencia

Si la aplicación (o subproceso) se puede bloquear hasta que se complete la transferencia, puede utilizar el método `waitForCompletion` de la interfaz Transfer para aplicar un bloqueo hasta que se complete la transferencia o se produzca una excepción.

```
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);
}
```

Puede obtener el progreso de las transferencias si sondea eventos antes de llamar a `waitForCompletion`, implementar un mecanismo de sondeo en un subproceso distinto o recibir actualizaciones sobre el progreso de forma asíncrona utilizando un ProgressListener.

Consulte el ejemplo completo en GitHub.

Obtener el estado y el progreso de una transferencia

Cada una de las clases devueltas por los métodos `upload*`, `download*` y `copy` de TransferManager devuelve una instancia de una de las siguientes clases, en función de si se trata de una operación en un solo archivo o en varios.

Clase	Devuelta por
Copiar	`copy`
Descargar	`download`
MultipleFileDownload	`downloadDirectory`
Cargar	`upload`
MultipleFileUpload	`uploadFileList` , `uploadDirectory`

Todas estas clases implementan la interfaz Transfer. `Transfer` proporciona métodos útiles para obtener el progreso de una transferencia, detener o reanudar la transferencia y obtener el estado actual o final de la transferencia.

Temas

- Sondear el progreso actual de una transferencia
- Obtener el progreso de una transferencia con ProgressListener
- Obtener el progreso de las transferencias secundarias

Sondear el progreso actual de una transferencia

Este bucle muestra el progreso de una transferencia, examina su progreso actual mientras se ejecuta y, cuando se completa, muestra su estado final.

Importaciones

```
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);
```

Consulte el ejemplo completo en GitHub.

Obtener el progreso de una transferencia con ProgressListener

Puede asociar ProgressListener a cualquier transferencia mediante el método `addProgressListener` de la interfaz Transfer.

Un ProgressListener requiere solo un método `progressChanged`, que toma un objeto ProgressEvent. Puede utilizar el objeto para obtener el total de bytes de la operación llamando a su método `getBytes` y el número de bytes que se han transferido hasta el momento llamando a `getBytesTransferred`.

Importaciones

```
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();
```

Consulte el ejemplo completo en GitHub.

Obtener el progreso de las transferencias secundarias

La clase MultipleFileUpload puede devolver información sobre sus transferencias secundarias llamando a su método `getSubTransfers`. Devuelve una colección no modificable de objetos Upload que proporcionan el estado y el progreso de cada transferencia secundaria.

Importaciones

```
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);
```

Consulte el ejemplo completo en GitHub.

Más información

- Claves de objeto en la Guía del usuario de Amazon Simple Storage Service

Configuración de un bucket de Amazon S3 como un sitio web

Puede configurar un bucket de Amazon S3 para que se comporte como un sitio web. Para ello, debe establecer la configuración de su sitio web.

> **Note**
>
> En estos ejemplos de código se presupone que conoce la información que se describe en Uso del AWS SDK for Java y que ha configurado credenciales de AWS predeterminadas mediante la información de Configuración de credenciales y regiones de AWS para desarrollo.

Establecimiento de la configuración de sitio web de un bucket

Para establecer la configuración de sitio web de un bucket de Amazon S3, llame al método `setWebsiteConfiguration` de AmazonS3 con el nombre del bucket para el que se va a establecer la configuración y un objeto BucketWebsiteConfiguration que contenga la configuración de sitio web del bucket.

Es obligatorio establecer un documento de índice; todos los demás parámetros son opcionales.

Importaciones

```
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**
>
> El establecimiento de la configuración de sitio web no modifica los permisos de acceso del bucket. Para que los archivos estén visibles en la web, también deberá definir una política de bucket que permita el acceso de lectura pública a los archivos del bucket. Para obtener más información, consulte Administración del acceso a buckets de Amazon S3 mediante políticas de buckets.

Consulte el ejemplo completo en GitHub.

Obtener la configuración de sitio web de un bucket

Para obtener la configuración de sitio web de un bucket de Amazon S3, llame al método `getWebsiteConfiguration` de AmazonS3 con el nombre del bucket para el que desea recuperar la configuración.

La configuración se devolverá como un objeto BucketWebsiteConfiguration. Si no hay ninguna configuración de sitio web para el bucket, se devolverá `null`.

Importaciones

```
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);
}
```

Consulte el ejemplo completo en GitHub.

Eliminar la configuración de sitio web de un bucket

Para eliminar la configuración de sitio web de un bucket de Amazon S3, llame al método `deleteWebsiteConfiguration` de AmazonS3 con el nombre del bucket del que se va a eliminar la configuración.

Importaciones

```
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);
}
```

Consulte el ejemplo completo en GitHub.

Más información

- Sitio web PUT Bucket en la referencia de la API de Amazon S3
- El sitio web GET Bucket en la referencia de la API de Amazon S3
- Sitio web DELETE Bucket en la referencia de la API de Amazon S3

Usar cifrado del cliente de Amazon S3

El cifrado de los datos con el cliente Amazon S3 para este fin es una forma de proporcionar una capa de protección adicional para la información confidencial que almacena en Amazon S3. Los ejemplos de esta sección demuestran cómo crear y configurar el cliente de cifrado de Amazon S3 para la aplicación.

Si es la primera vez que utiliza la criptografía, consulte Conceptos básicos de criptografía en la Guía para desarrolladores de AWS KMS para obtener información general básica sobre los términos y los algoritmos de criptografía. Para obtener información sobre la compatibilidad con la criptografía en todos los SDK de AWS, consulte Soporte de SDK de AWS para cifrado del cliente para Amazon S3 en la Referencia general de Amazon Web Services.

> **Note**
>
> En estos ejemplos de código se presupone que conoce la información que se describe en Uso del AWS SDK for Java y que ha configurado credenciales de AWS predeterminadas

mediante la información de Configuración de credenciales y regiones de AWS para desarrollo.

Si utiliza la versión 1.11.836 o una anterior del AWS SDK for Java, consulte Migración de clientes de cifrado de Amazon S3 para obtener información sobre la migración de sus aplicaciones a versiones posteriores. Si no puede migrar, consulte este ejemplo completo en GitHub.

De lo contrario, si utiliza la versión 1.11.837 o posterior de AWS SDK for Java, explore los temas de ejemplo que se enumeran a continuación para usar el cifrado del cliente de Amazon S3.

Temas

- Cifrado del cliente de Amazon S3 con claves maestras de cliente
- Cifrado del cliente Amazon S3 con claves administradas por AWS KMS

Cifrado del cliente de Amazon S3 con claves maestras de cliente

Los siguientes ejemplos utilizan la clase AmazonS3EncryptionClientV2Builder para crear un cliente Amazon S3 que tenga habilitado el cifrado del cliente. Una vez habilitado, los objetos que se cargan en Amazon S3 con este cliente se cifrarán. Cualquier objeto que se obtenga de Amazon S3 con este cliente se descifrará automáticamente.

> **Note**
>
> En los siguientes ejemplos se muestra cómo utilizar el cifrado del cliente de Amazon S3 con claves maestras administradas por el cliente. Para obtener información sobre cómo usar el cifrado con las claves administradas de AWS KMS, consulte Cifrado del cliente de Amazon S3 con claves administradas por AWS KMS.

Puede elegir entre dos modos de cifrado al activar el cifrado Amazon S3 del cliente: autenticación estricta o autenticación. En las secciones siguientes se muestran cómo habilitar cada uno de estos tipos. Para saber qué algoritmos utiliza cada modo, consulte la definición de CryptoMode.

Importaciones requeridas

Importe las clases siguientes para estos ejemplos.

Importaciones

```
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;
```

Cifrado de autenticado estricto

El cifrado de autenticado estricto es el modo predeterminado si no se especifica `CryptoMode`.

Para habilitar de manera explícita este modo, especifique el valor `StrictAuthenticatedEncryption` en el método `withCryptoConfiguration`.

> **Note**
>
> Para utilizar el cifrado autenticado del cliente, debe incluir el archivo Bouncy Castle jar más reciente en el classpath de la aplicación.

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");
```

Modo de cifrado autenticado

Al utilizar el modo `AuthenticatedEncryption`, se aplica un algoritmo de encapsulamiento de clave mejorado durante el cifrado. Cuando se descifra en este modo, el algoritmo puede verificar la integridad del objeto descifrado e iniciar una excepción si la comprobación falla. Para más detalles

sobre cómo funciona el cifrado autenticado, consulte la entrada del blog Cifrado autenticado del cliente Amazon S3.

> **Note**
>
> Para utilizar el cifrado autenticado del cliente, debe incluir el archivo Bouncy Castle jar más reciente en el classpath de la aplicación.

Para habilitar este modo, especifique el valor `AuthenticatedEncryption` en el método `withCryptoConfiguration`.

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");
```

Cifrado del cliente Amazon S3 con claves administradas por AWS KMS

Los siguientes ejemplos utilizan la clase AmazonS3EncryptionClientV2Builder para crear un cliente Amazon S3 que tenga habilitado el cifrado del cliente. Una vez configurado, los objetos que se cargan en Amazon S3 con este cliente se cifrarán. Cualquier objeto que se obtenga de Amazon S3 con este cliente, se descifrará automáticamente.

> **Note**
>
> En los siguientes ejemplos se muestra cómo utilizar el cifrado del cliente Amazon S3 con claves administradas por AWS KMS. Para obtener información sobre cómo utilizar el cifrado con sus propias claves, consulte Cifrado del cliente Amazon S3 con claves maestras de cliente.

Puede elegir entre dos modos de cifrado al activar el cifrado Amazon S3 del cliente: autenticación estricta o autenticación. En las secciones siguientes se muestran cómo habilitar cada uno de estos tipos. Para saber qué algoritmos utiliza cada modo, consulte la definición de CryptoMode.

Importaciones requeridas

Importe las clases siguientes para estos ejemplos.

Importaciones

```
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;
```

Cifrado de autenticado estricto

El cifrado de autenticado estricto es el modo predeterminado si no se especifica `CryptoMode`.

Para habilitar de manera explícita este modo, especifique el valor `StrictAuthenticatedEncryption` en el método `withCryptoConfiguration`.

> **Note**
>
> Para utilizar el cifrado autenticado del cliente, debe incluir el archivo Bouncy Castle jar más reciente en el classpath de la aplicación.

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));
```

Llame al método `putObject` en el cliente de cifrado de Amazon S3 para cargar objetos.

Code

```
s3Encryption.putObject(bucket_name, ENCRYPTED_KEY3, "This is the 3rd content to encrypt
 with a key created in the {console}");
```

Para recuperar el objeto, puede utilizar el mismo cliente. Este ejemplo llama al método `getObjectAsString` para recuperar la cadena que se almacenó.

Code

```
System.out.println(s3Encryption.getObjectAsString(bucket_name, ENCRYPTED_KEY3));
```

Modo de cifrado autenticado

Al utilizar el modo `AuthenticatedEncryption`, se aplica un algoritmo de encapsulamiento de clave mejorado durante el cifrado. Cuando se descifra en este modo, el algoritmo puede verificar la integridad del objeto descifrado e iniciar una excepción si la comprobación falla. Para más detalles sobre cómo funciona el cifrado autenticado, consulte la entrada del blog Cifrado autenticado del cliente Amazon S3.

> **Note**
>
> Para utilizar el cifrado autenticado del cliente, debe incluir el archivo Bouncy Castle jar más reciente en el classpath de la aplicación.

Para habilitar este modo, especifique el valor `AuthenticatedEncryption` en el método `withCryptoConfiguration`.

Code

```
AmazonS3EncryptionV2 s3Encryption = AmazonS3EncryptionClientV2Builder.standard()
         .withRegion(Regions.US_WEST_2)
         .withCryptoConfiguration(new
 CryptoConfigurationV2().withCryptoMode((CryptoMode.AuthenticatedEncryption)))
         .withEncryptionMaterialsProvider(new KMSEncryptionMaterialsProvider(keyId))
         .build();
```

Configuración del cliente AWS KMS

El cliente de cifrado Amazon S3 crea un cliente AWS KMS de forma predeterminada, a menos que se especifique uno de forma explícita.

Para establecer la región de este cliente AWS KMS creado automáticamente, defina el `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();
```

Si lo prefiere, puede utilizar su propio cliente AWS KMS para inicializar el cliente de cifrado.

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();
```

Ejemplos de Amazon SQS usando la AWS SDK for Java

En esta sección se proporcionan ejemplos de programación en Amazon SQS mediante AWS SDK for Java.

> **Note**
>
> Los ejemplos incluyen únicamente el código necesario para demostrar cada técnica. El código de ejemplo completo está disponible en GitHub. Desde allí, puede descargar un único archivo de código fuente o clonar el repositorio localmente para obtener todos los ejemplos para compilarlos y ejecutarlos.

Temas

Uso de colas de mensajes de Amazon SQS

Una cola de mensajes es el contenedor lógico utilizado para enviar mensajes de forma fiable en Amazon SQS. Existen dos tipos de colas: estándar y primero en entrar, primero en salir (FIFO). Para obtener más información sobre las colas y las diferencias entre estos tipos, consulte la Guía para desarrolladores de Amazon SQS.

En este tema se describe cómo crear, mostrar, eliminar y obtener la dirección URL de un cola de Amazon SQS mediante AWS SDK for Java.

Creación de una cola

Use el método `createQueue` del cliente AmazonSQS, proporcionando un objeto CreateQueueRequest que describa los parámetros de la cola.

Importaciones

```
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;
    }
}
```

Puede utilizar el formato simplificado `createQueue`, que solo necesita el nombre de una cola, para crear una cola estándar.

```
sqs.createQueue("MyQueue" + new Date().getTime());
```

Consulte el ejemplo completo en GitHub.

Mostrar colas

Para enumerar las colas de Amazon SQS de su cuenta, llame al método `listQueues` del cliente AmazonSQS.

Importaciones

```
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);
}
```

El uso de la sobrecarga `listQueues` sin parámetros devuelve todas las colas. Puede filtrar los resultados devueltos pasando un objeto `ListQueuesRequest`.

Importaciones

```
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);
}
```

Consulte el ejemplo completo en GitHub.

Obtener la URL de una cola

Llame al método `getQueueUrl` del cliente AmazonSQS.

Importaciones

```
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();
```

Consulte el ejemplo completo en GitHub.

Eliminar una cola

Proporcione la URL de la cola al método del cliente AmazonSQS.

Importaciones

```
import com.amazonaws.services.sqs.AmazonSQS;
import com.amazonaws.services.sqs.AmazonSQSClientBuilder;
```

Code

```
AmazonSQS sqs = AmazonSQSClientBuilder.defaultClient();
sqs.deleteQueue(queue_url);
```

Consulte el ejemplo completo en GitHub.

Más información

- Cómo funcionan las colas de Amazon SQS en la Guía para desarrolladores de Amazon SQS
- CreateQueue en la Referencia de la API de Amazon SQS
- GetQueueUrl en la referencia de la API de Amazon SQS
- ListQueues en la Referencia de la API de Amazon SQS
- DeleteQueues en la referencia de la API de Amazon SQS

Envío, recepción y eliminación de mensajes de Amazon SQS

En este tema se describe cómo enviar, recibir y eliminar mensajes de Amazon SQS. Los mensajes se envían siempre a través de una cola de SQS.

Enviar un mensaje

Añada un único mensaje a una cola de Amazon SQS llamando al método `sendMessage` del cliente AmazonSQS. Proporcione un objeto SendMessageRequest que contenga la URL de la cola, el cuerpo del mensaje y el valor de retraso opcional (en segundos).

Importaciones

```
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);
```

Consulte el ejemplo completo en GitHub.

Enviar varios mensajes a la vez

Puede enviar más de un mensaje en una única solicitud. Para enviar varios mensajes, utilice el método `sendMessageBatch` del cliente AmazonSQS, que toma un objeto SendMessageBatchRequest con la URL de la cola y una lista de mensajes (un objeto SendMessageBatchRequestEntry para cada uno) que se van a enviar. También puede definir un valor de retraso opcional para cada mensaje.

Importaciones

```
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);
```

Consulte el ejemplo completo en GitHub.

Recibir mensajes

Recupere todos los mensajes que se encuentran actualmente en la cola llamando al método `receiveMessage` del cliente AmazonSQS, pasando la URL de la cola. Los mensajes se devuelven como una lista de objetos Message.

Importaciones

```
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();
```

Eliminar mensajes después de su recepción

Tras recibir un mensaje y procesar su contenido, elimine el mensaje de la cola enviando el identificador de recepción y la URL de la cola del mensaje al método `deleteMessage` de AmazonSQS.

Code

```
for (Message m : messages) {
    sqs.deleteMessage(queueUrl, m.getReceiptHandle());
}
```

Consulte el ejemplo completo en GitHub.

Más información

- Cómo funcionan las colas de Amazon SQS en la Guía para desarrolladores de Amazon SQS
- SendMessage en la referencia de la API de Amazon SQS
- SendMessageBatch en la referencia de la API de Amazon SQS
- ReceiveMessage en la Referencia de la API de Amazon SQS

- DeleteMessage en la Referencia de la API de Amazon SQS

Habilitar sondeos largos para las colas de mensajes de Amazon SQS

Amazon SQS utiliza el sondeo corto de forma predeterminada; consulta únicamente un subconjunto de los servidores (en función de una distribución aleatoria ponderada) para determinar si hay algún mensaje disponible para su inclusión en la respuesta.

El sondeo largo ayuda a reducir el costo de uso de Amazon SQS al reducir el número de respuestas vacías (cuando no hay ningún mensaje disponible para devolver como respuesta a una solicitud ReceiveMessage enviada a una cola de Amazon SQS) y eliminar falsas respuestas vacías.

> **Note**
>
> Puede definir una frecuencia de sondeo largo de 1-20 segundos.

Habilitar el sondeo largo al crear una cola

Para habilitar el sondeo largo al crear una cola de Amazon SQS, establezca el atributo `ReceiveMessageWaitTimeSeconds` en el objeto CreateQueueRequest antes de llamar al método `createQueue` de la clase AmazonSQS.

Importaciones

```
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;
    }
}
```

Consulte el ejemplo completo en GitHub.

Habilitar el sondeo largo en una cola existente

Además de habilitar el sondeo largo al crear una cola, también puede habilitarlo en una cola existente estableciendo `ReceiveMessageWaitTimeSeconds` en SetQueueAttributesRequest antes de llamar al método `setQueueAttributes` de la clase AmazonSQS.

Importaciones

```
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);
```

Consulte el ejemplo completo en GitHub.

Habilitar el sondeo largo al recibir un mensaje

Puede habilitar el sondeo largo al recibir un mensaje estableciendo el tiempo de espera en segundos en el objeto ReceiveMessageRequest que proporciona al método `receiveMessage` de la clase AmazonSQS.

> **Note**
>
> Debe asegurarse de que el tiempo de espera de la solicitud del cliente de AWS sea mayor que el tiempo del sondeo largo (20 segundos) para que no se agote el tiempo de espera de sus solicitudes `receiveMessage` mientras espera al siguiente evento de sondeo.

Importaciones

```
import com.amazonaws.services.sqs.model.ReceiveMessageRequest;
```

Code

```
ReceiveMessageRequest receive_request = new ReceiveMessageRequest()
        .withQueueUrl(queue_url)
        .withWaitTimeSeconds(20);
sqs.receiveMessage(receive_request);
```

Consulte el ejemplo completo en GitHub.

Más información

- Amazon SQSPolíticas de claves en la Guía para desarrolladores de Amazon SQS
- CreateQueue en la Referencia de la API de Amazon SQS
- ReceiveMessage en la Referencia de la API de Amazon SQS
- SetQueueAttributes en la Referencia de la API de Amazon SQS

Configuración del tiempo de espera de visibilidad en Amazon SQS

Cuando se recibe un mensaje en Amazon SQS, este permanece en la cola hasta que se elimina a fin de garantizar su recepción. Un mensaje que se ha recibido, pero no se ha eliminado, estará disponible en las solicitudes posteriores después de un determinado tiempo de espera de visibilidad para ayudar a evitar que el mensaje se reciba más de una vez antes de que pueda procesarse y eliminarse.

> **Note**
>
> Cuando se utilizan colas estándar, el tiempo de espera de visibilidad no es una garantía de que un mensaje no se reciba dos veces. Si utiliza una cola estándar, asegúrese de que el código pueda tratar aquellas situaciones en las que el mismo mensaje se entrega más de una vez.

Configuración del tiempo de espera de visibilidad de los mensajes para un solo mensaje

Cuando haya recibido un mensaje, puede modificar su tiempo de espera de visibilidad pasando su identificador de recepción en el objeto ChangeMessageVisibilityRequest que pasa al método `changeMessageVisibility` de AmazonSQS class.

Importaciones

```
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);
```

Consulte el ejemplo completo en GitHub.

Configuración del tiempo de espera de visibilidad de los mensajes para varios mensajes a la vez

Para configurar el tiempo de espera de visibilidad para varios mensajes, cree una lista de objetos ChangeMessageVisibilityBatchRequestEntry, cada uno con un ID de cadena y un identificador de recepción únicos. A continuación, pase la lista al método `changeMessageVisibilityBatch` de la clase del cliente de Amazon SQS.

Importaciones

```
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);
```

Consulte el ejemplo completo en GitHub.

Más información

- Tiempo de espera de visibilidad en la Guía para desarrolladores de Amazon SQS
- SetQueueAttributes en la Referencia de la API de Amazon SQS
- GetQueueAttributes en la Referencia de la API de Amazon SQS
- ReceiveMessage en la Referencia de la API de Amazon SQS
- ChangeMessageVisibility en la Referencia de la API de Amazon SQS
- ChangeMessageVisibilityBatch en la Referencia de la API de Amazon SQS

Uso de colas de mensajes fallidos en Amazon SQS

Amazon SQS añade compatibilidad para las colas de mensajes fallidos. Una cola de mensajes fallidos es una cola a la que otras pueden enviar mensajes que no se pueden procesar

correctamente. Puede apartar y aislar estos mensajes en la cola de mensajes fallidos para determinar por qué no se procesaron correctamente.

Creación de una cola de mensajes fallidos

Una cola de mensajes fallidos se crea de la misma forma que una cola normal, pero con las siguientes restricciones:

- Una cola de mensajes fallidos debe ser el mismo tipo de cola (FIFO o estándar) que la cola de origen.
- Una cola de mensajes fallidos se debe crear con la misma cuenta y región de Cuenta de AWS que la cola de origen.

Aquí creamos dos colas de Amazon SQS idénticas, una de los cuales actuará como la cola de mensajes fallidos:

Importaciones

```
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;
    }
}
```

Consulte el ejemplo completo en GitHub.

Designación de una cola de mensajes fallidos para una cola de origen

Para designar una cola de mensajes fallidos, primero debe crear una política de redireccionamiento y, a continuación, configurar la política en los atributos de la cola. Una política de redireccionamiento se especifica en JSON e indica el ARN de la cola de mensajes fallidos y el número máximo de veces que se puede recibir y no procesar el mensaje antes de que se envíe a la cola de mensajes fallidos.

Para establecer la política de redireccionamiento de la cola de origen, llame al método de la clase de `setQueueAttributes` de AmazonSQS con un objeto SetQueueAttributesRequest para el que haya establecido el atributo `RedrivePolicy` con su política de redireccionamiento JSON.

Importaciones

```
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);
```

Consulte el ejemplo completo en GitHub.

Más información

- Uso de colas de cartas muertas de Amazon SQS en la Guía para desarrolladores de Amazon SQS
- SetQueueAttributes en la Referencia de la API de Amazon SQS

Ejemplos de Amazon SWF usando la AWS SDK for Java

Amazon SWF es un servicio de administración de flujos de trabajo que ayuda a los desarrolladores a crear y escalar flujos de trabajo distribuidos que pueden tener pasos paralelos o secuenciales compuestos de actividades, flujos de trabajo secundarios o incluso tareas Lambda.

Existen dos maneras de trabajar con Amazon SWF mediante AWS SDK for Java: utilizando el objeto client de SWF o mediante AWS Flow Framework para Java. AWS Flow Framework es más difícil de configurar inicialmente, ya que hace un uso intensivo de las anotaciones y se basa en otras bibliotecas como AspectJ y Spring Framework. Sin embargo, en el caso de proyectos grandes y complejos, con AWS Flow Framework se ahorrará tiempo de programación. Para obtener más información, consulte la Guía para desarrolladores del AWS Flow Framework para Java.

En esta sección se proporciona ejemplos de programación en Amazon SWF que utilizan directamente el cliente de AWS SDK for Java.

Temas

- Conceptos básicos de SWF
- Creación de una aplicación de Amazon SWF sencilla
- Tareas de Lambda
- Cerrar correctamente los procesos de trabajo de actividad y flujo de trabajo
- Registro de dominios
- Visualización de los dominios

Conceptos básicos de SWF

Estos son los patrones generales para trabajar con Amazon SWF mediante AWS SDK for Java. Se han diseñado principalmente como referencia. Para obtener un tutorial de introducción más completo, consulte Creación de una aplicación de Amazon SWF sencilla.

Dependencias.

Las aplicaciones básicas de Amazon SWF requerirán las siguientes dependencias, que se incluyen con 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**
>
> Los números de versión de estos paquetes serán diferentes en función de la versión del SDK que tenga, pero las versiones que se proporcionan con el SDK se han probado para garantizar su compatibilidad y son las que debe utilizar.

Las aplicaciones de AWS Flow Framework para Java requieren configuración adicional y dependencias adicionales. Consulte la Guía para desarrolladores de AWS Flow Framework para Java para obtener más información acerca de cómo utilizar la plataforma.

Importaciones

En general, puede utilizar las siguientes importaciones para el desarrollo de código:

```
import com.amazonaws.services.simpleworkflow.AmazonSimpleWorkflowClientBuilder;
```

```
import com.amazonaws.services.simpleworkflow.model.*;
```

Sin embargo, es aconsejable que importe solamente las clases que necesite. Lo más probable es que acabe especificando clases concretas en el área de trabajo `com.amazonaws.services.simpleworkflow.model`:

```
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;
```

Si utiliza AWS Flow Framework para Java, importará clases del área de trabajo de `com.amazonaws.services.simpleworkflow.flow`. Por ejemplo:

```
import com.amazonaws.services.simpleworkflow.AmazonSimpleWorkflow;
import com.amazonaws.services.simpleworkflow.flow.ActivityWorker;
```

> **Note**
>
> El AWS Flow Framework para Java tiene requisitos adicionales además de los del AWS SDK for Java básico. Para obtener más información, consulte la Guía para desarrolladores del AWS Flow Framework para Java.

Uso de la clase del cliente SWF

La interfaz básica con Amazon SWF se realiza a través de las clases AmazonSimpleWorkflowClient o AmazonSimpleWorkflowAsyncClient. La diferencia principal entre estas clases es que la clase `\*AsyncClient` devuelve objetos Future para la programación simultánea (asíncrona).

```
AmazonSimpleWorkflowClient swf = AmazonSimpleWorkflowClientBuilder.defaultClient();
```

Creación de una aplicación de Amazon SWF sencilla

Este tema es una introducción a la programación de aplicaciones de Amazon SWF con el AWS SDK for Java, en la que se describen algunos conceptos importantes.

Acerca del ejemplo

El proyecto de ejemplo creará un flujo de trabajo con una única actividad que acepta datos del flujo de trabajo pasados a través de la nube de AWS (siguiendo la tradición de HelloWorld, será el nombre de una persona a la que se saluda) y, a continuación, mostrará un saludo como respuesta.

Aunque puedan parecer muy sencillas, las aplicaciones de Amazon SWF se componen de varias partes que funcionan conjuntamente:

- Un dominio, utilizado como un contenedor lógico para los datos de ejecución del flujo de trabajo.
- Uno o varios flujos de trabajo que representan los componentes de código que definen el orden lógico de ejecución de las actividades del flujo de trabajo y los flujos de trabajo secundarios.
- Un proceso de trabajo de flujo de trabajo, también conocido como decisor, que busca tareas de decisión y actividades de programación o flujos de trabajo secundarios como respuesta.
- Una o varias actividades, cada una de las cuales representa una unidad de trabajo del flujo de trabajo.
- Un proceso de trabajo de actividad que busca tareas de actividad y ejecuta métodos de actividad como respuesta.
- Una o varias listas de tareas, que son colas mantenidas por Amazon SWF utilizadas para emitir solicitudes a los procesos de trabajo del flujo de trabajo o actividad. Las tareas de una lista de tareas dirigidas a los procesos de trabajo de flujo de trabajo se denominan tareas de decisión. Las dirigidas a los procesos de trabajo de actividad se denominan tareas de actividad.
- Un iniciador del flujo de trabajo que inicia la ejecución del flujo de trabajo.

Entre bambalinas, Amazon SWF organiza la operación de estos componentes, coordinando su flujo desde la nube de AWS, pasando datos entre ellos, administrando los tiempos de espera y las notificaciones de latido, y registrando el historial de ejecución del flujo de trabajo.

Requisitos previos

Entorno de desarrollo

El entorno de desarrollo que se utiliza en este tutorial se compone de:

- El valor AWS SDK for Java.
- Apache Maven (3.3.1).

- JDK 1.7 o posterior. Este tutorial se ha desarrollado y probado con JDK 1.8.0.
- Un buen editor de texto de Java (el que prefiera).

> **Note**
>
> Si utiliza un sistema de compilación distinto de Maven, puede crear igualmente un proyecto realizando los pasos correspondientes de su entorno y usar los conceptos que se proporcionan aquí durante el proceso. En la sección AWS SDK for JavaIntroducción se proporciona más información sobre cómo configurar y usar con distintos sistemas de compilación.
> Del mismo modo, pero con más esfuerzo, los pasos que se muestran aquí se pueden implementar con cualquiera de los SDK de AWS admitidos para Amazon SWF.

Todas las dependencias externas necesarias se incluyen con AWS SDK for Java, por lo que no necesita descargar nada más.

Acceso a AWS

Para completar correctamente este tutorial, debe tener acceso al portal de acceso a AWS como se describe en la sección de configuración básica de esta guía.

Las instrucciones describen cómo acceder a las credenciales temporales que se copian y pegan en el archivo de `credentials` compartido local. Las credenciales temporales que pegue deben estar asociadas a un rol de IAM en AWS IAM Identity Center que tenga permisos para acceder a Amazon SWF. Tras pegar las credenciales temporales, el archivo de `credentials` debería tener un aspecto similar al siguiente.

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

Estas credenciales temporales están asociadas al perfil `default`.

Creación de un proyecto de SWF

1. Inicie un nuevo proyecto con Maven:

```
mvn archetype:generate -DartifactId=helloswf \
-DgroupId=aws.example.helloswf -DinteractiveMode=false
```

Se creará un nuevo proyecto con una estructura de proyecto de Maven estándar:

```
helloswf
### pom.xml
### src
    ### main
    #   ### java
    #       ### aws
    #           ### example
    #               ### helloswf
    #                   ### App.java
    ### test
        ### ...
```

Puede omitir o eliminar el directorio `test` y todo su contenido; no lo usaremos en este tutorial. También puede eliminar `App.java`, ya que lo reemplazaremos por nuevas clases.

2. Edite el archivo `pom.xml` del proyecto y añada el módulo aws-java-sdk-simpleworkflow añadiendo una dependencia para él en el bloque `<dependencies>`.

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

3. Asegúrese de que Maven compila su proyecto de manera que sea compatible con JDK 1.7+. Añada lo siguiente a su proyecto (delante o detrás del bloque `<dependencies>`) en `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>
```

Codificación del proyecto

El proyecto de ejemplo consta de cuatro aplicaciones distintas, que examinaremos de una en una:

- HelloTypes.java: contiene el dominio del proyecto y los datos de tipos de actividad y flujo de trabajo, compartidos con los demás componentes. También se encarga de registrar estos tipos con SWF.
- ActivityWorker.java: contiene el proceso de trabajo de actividad, que busca tareas de actividad y ejecuta actividades como respuesta.
- WorkflowWorker.java: contiene el proceso de trabajo de flujo de trabajo (decisor), que busca tareas de decisión y programa nuevas actividades.
- WorkflowStarter.java: contiene el iniciador del flujo de trabajo, que inicia la ejecución de un nuevo flujo de trabajo, que hará que SWF empiece a generar tareas de decisión y flujo de trabajo para los procesos de trabajo.

Pasos comunes para todos los archivos de código fuente

Todos los archivos que crea para alojar sus clases Java tendrán algunas cosas en común. Para ahorrar tiempo, estos pasos estarán implícitos cada vez que añada un nuevo archivo al proyecto:

1. Cree el archivo en el directorio `src/main/java/aws/example/helloswf/` del proyecto.
2. Añada una declaración `package` al principio de cada archivo para declarar su espacio de nombres. El proyecto de ejemplo usa:

   ```
   package aws.example.helloswf;
   ```

3. Añada declaraciones `import` para la clase AmazonSimpleWorkflowClient y para varias clases del espacio de nombres `com.amazonaws.services.simpleworkflow.model`. Para simplificar las cosas, vamos a utilizar:

   ```
   import com.amazonaws.regions.Regions;
   import com.amazonaws.services.simpleworkflow.AmazonSimpleWorkflow;
   ```

```
import com.amazonaws.services.simpleworkflow.AmazonSimpleWorkflowClientBuilder;
import com.amazonaws.services.simpleworkflow.model.*;
```

Registro de un dominio y de tipos de flujo de trabajo y actividad

Comenzaremos creando una nueva clase ejecutable, `HelloTypes.java`. Este archivo contendrá datos compartidos que necesitarán conocer las distintas partes de su flujo de trabajo, como el nombre y la versión de sus tipos de actividad y flujo de trabajo, el nombre de dominio y el nombre de la lista de tareas.

1. Abra su editor de texto y cree el archivo `HelloTypes.java`, añadiendo una declaración del paquete y funciones import de acuerdo con los pasos comunes.
2. Declare la clase `HelloTypes` y proporcione los valores que se van a usar para los tipos de actividad y flujo de trabajo registrados:

   ```
       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";
   ```

   Estos valores se utilizarán en todo el código.
3. Detrás de las declaraciones de cadena, cree una instancia de la clase AmazonSimpleWorkflowClient. Esta es la interfaz básica a los métodos de Amazon SWF proporcionados por AWS SDK for Java.

   ```
   private static final AmazonSimpleWorkflow swf =

    AmazonSimpleWorkflowClientBuilder.standard().withRegion(Regions.DEFAULT_REGION).build();
   ```

   En el fragmento anterior se presupone que las credenciales temporales están asociadas al perfil `default`. Si usa un perfil diferente, modifique el código anterior de la siguiente manera y sustituya *`profile_name por el nombre`* del perfil real.

   ```
   private static final AmazonSimpleWorkflow swf =
           AmazonSimpleWorkflowClientBuilder
                   .standard()
   ```

```
                .withCredentials(new ProfileCredentialsProvider("profile_name"))
                .withRegion(Regions.DEFAULT_REGION)
                .build();
```

4. Añada una nueva función para registrar un dominio de SWF. Un dominio es un contenedor lógico para una serie de tipos de actividad y flujo de trabajo de SWF relacionados. Los componentes de SWF solo pueden comunicarse entre sí si se encuentran en el mismo dominio.

```
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!");
}
```

Cuando registra un dominio, proporciona un nombre (cualquier conjunto de 1-256 caracteres excepto `:`, `/`, `|`, caracteres de control o la cadena literal 'arn') y un periodo de retención, que es el número de días que Amazon SWF conservará los datos del historial de ejecución del flujo de trabajo una vez completada la ejecución del flujo de trabajo. El periodo máximo de retención de ejecución del flujo de trabajo es de 90 días. Consulte RegisterDomainRequest para obtener más información.

Si ya existe un dominio con ese nombre, se produce una excepción DomainAlreadyExistsException. Como no nos interesa si se ha creado o no se ha creado el dominio, podemos omitir esta excepción.

> **Note**
>
> Este código muestra un patrón común cuando se trabaja con métodos de AWS SDK for Java: los datos del método los proporciona una clase del espacio de nombres `simpleworkflow.model`, de la que se crea una instancia y se rellena mediante la ejecución en cadena de métodos `0with*`.

5. Añada una función para registrar un nuevo tipo de actividad. Una actividad representa una unidad de trabajo de su flujo de trabajo.

```
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!");
    }
```

Un tipo de actividad se identifica mediante un nombre y una versión, que se utilizan para identificar de forma inequívoca la actividad de otras actividades que se han registrado en el dominio. Las actividades también contienen una serie de parámetros opcionales, como la lista de tareas predeterminadas para recibir las tareas y los datos de SWF, y una serie de tiempos de espera diferentes que puede utilizar para aplicar restricciones sobre cuánto tiempo pueden tardar las distintas partes de la ejecución de la actividad. Consulte RegisterActivityTypeRequest para obtener más información.

> **Note**
>
> Todos los valores de tiempo de espera se especifican en segundos. Consulte Tipos de tiempo de espera de Amazon SWF para obtener una descripción completa de cómo los tiempos de espera afectan a las ejecuciones del flujo de trabajo.

Si el tipo de actividad que intenta registrar ya existe, se produce una excepción TypeAlreadyExistsException. Añada una función para registrar un nuevo tipo de flujo de trabajo. Un flujo de trabajo, denominado también decisor, representa la lógica de la ejecución del flujo de trabajo.

+

```
    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!");
    }
```

+

Al igual que los tipos de actividad, los tipos de flujos de trabajo se identifican mediante un nombre y una versión, y también tienen tiempos de espera configurables. Consulte RegisterWorkflowTypeRequest para obtener más información.

+

Si el tipo de flujo de trabajo que intenta registrar ya existe, se produce una excepción TypeAlreadyExistsException. Por último, cree la clase ejecutable proporcionando un método `main`, que registrará el dominio, el tipo de actividad y el tipo de flujo de trabajo:

+

```
    registerDomain();
    registerWorkflowType();
    registerActivityType();
```

Puede compilar y ejecutar la aplicación ahora para ejecutar el script de registro o continuar programando los procesos de trabajo de actividad y flujo de trabajo. Una vez que el dominio, el flujo de trabajo y la actividad se hayan registrado, no necesitará ejecutar esto de nuevo: estos tipos persisten hasta que los deja de utilizar.

Implementación del proceso de trabajo de actividad

Una actividad es la unidad básica de trabajo de su flujo de trabajo. Un flujo de trabajo proporciona la lógica, programando las actividades que se van a ejecutar (u otras acciones que se deben llevar a cabo) en respuesta a las tareas de decisión. Un flujo de trabajo típico normalmente se compone de una serie de actividades que se pueden ejecutar de forma síncrona, asíncrona o de ambas formas.

El proceso de trabajo de actividad es la parte del código que busca las tareas de actividad generadas por Amazon SWF en respuesta a las decisiones del flujo de trabajo. Cuando recibe una tarea de

actividad, ejecuta la actividad correspondiente y devuelve una respuesta de éxito/error al flujo de trabajo.

Vamos a implementar un proceso de trabajo de actividad sencillo que se encarga de una sola actividad.

1. Abra su editor de texto y cree el archivo `ActivityWorker.java`, añadiendo una declaración del paquete y funciones import de acuerdo con los pasos comunes.

```
import com.amazonaws.regions.Regions;
import com.amazonaws.services.simpleworkflow.AmazonSimpleWorkflow;
import com.amazonaws.services.simpleworkflow.AmazonSimpleWorkflowClientBuilder;
import com.amazonaws.services.simpleworkflow.model.*;
```

2. Añada la clase `ActivityWorker` al archivo y proporcione un miembro de datos para almacenar el cliente de SWF que usaremos para interactuar con Amazon SWF:

```
    private static final AmazonSimpleWorkflow swf =

 AmazonSimpleWorkflowClientBuilder.standard().withRegion(Regions.DEFAULT_REGION).build();
```

3. Añada el método que usaremos como actividad:

```
private static String sayHello(String input) throws Throwable {
    return "Hello, " + input + "!";
}
```

   La actividad simplemente toma una cadena, la combina en un saludo y devuelve el resultado. Aunque no es muy probable que esta actividad produzca una excepción, es aconsejable que diseñe actividades que puedan producir un error si algo va mal.

4. Añada un método `main` que utilizaremos como el método de sondeo de tareas de actividad. Empezaremos añadiendo código para buscar tareas de actividad en la lista de tareas:

```
        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();
```

La actividad recibe tareas de Amazon SWF llamando al método `pollForActivityTask` del cliente de SWF, especificando el dominio y la lista de tareas que se van a utilizar en el objeto PollForActivityTaskRequest pasado al método.

Una vez que se recibe una tarea, recuperamos un identificador único llamando al método `getTaskToken` de la tarea.

5. A continuación, escribimos código para procesar las tareas que llegan. Añada lo siguiente al método `main`, justo detrás del código que busca la tarea y recupera su token de tarea.

```
    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()));
        }
```

```
    }
```

Si el token de tarea no es `null`, podemos empezar a ejecutar el método de actividad (`sayHello`), facilitándole los datos de entrada que se enviaron con la tarea.

Si la tarea se ejecuta correctamente (no se genera ningún error), el proceso de trabajo responde a SWF llamando al método `respondActivityTaskCompleted` del cliente de SWF con un objeto RespondActivityTaskCompletedRequest que contiene el token de la tarea y los datos de los resultados de la actividad.

Sin embargo, si la tarea ha producido un error, respondemos llamando al método `respondActivityTaskFailed` con un objeto RespondActivityTaskFailedRequest, proporcionándole el token de la tarea e información sobre el error.

> **Note**
>
> Esta actividad no se cerrará correctamente si se cancela. Aunque está fuera del alcance de este tutorial, una implementación alternativa de este proceso de trabajo de actividad se proporciona en el tema complementario Cerrar correctamente los procesos de trabajo de actividad y flujo de trabajo.

Implementación del proceso de trabajo del flujo de trabajo

La lógica del flujo de trabajo reside en una parte del código denominada proceso de trabajo de flujo de trabajo. El proceso de trabajo de flujo de trabajo busca tareas de decisión enviadas por Amazon SWF en el dominio y en la lista de tareas predeterminada en los que se ha registrado este tipo de flujo de trabajo.

Cuando el proceso de trabajo de flujo de trabajo recibe una tarea, toma algún tipo de decisión (normalmente si se va a programar o no una nueva actividad) y realiza la actividad correspondiente (como programar la actividad).

1. Abra su editor de texto y cree el archivo `WorkflowWorker.java`, añadiendo una declaración del paquete y funciones import de acuerdo con los pasos comunes.
2. Añada algunas funciones import adicionales al archivo:

```
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 la clase `WorkflowWorker` y cree una instancia de la clase AmazonSimpleWorkflowClient utilizada para obtener acceso a los métodos de SWF.

```
    private static final AmazonSimpleWorkflow swf =

AmazonSimpleWorkflowClientBuilder.standard().withRegion(Regions.DEFAULT_REGION).build();
```

4. Añada el método `main`. Este método se ejecuta en bucle continuamente en busca de tareas de decisión usando el método `pollForDecisionTask` del cliente de SWF. El objeto PollForDecisionTaskRequest proporciona los detalles.

```
    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();
            }
        }
    }
```

Una vez que se recibe una tarea, llamamos a su método `getTaskToken`, que devuelve una cadena que se puede utilizar para identificar la tarea. Si el token devuelto no es `null`, lo seguimos procesando en el método `executeDecisionTask`, pasándole el token de la tarea y la lista de objetos HistoryEvent enviados con la tarea.

5. Añada el método `executeDecisionTask`, tomando el token de tarea (`String`) y la lista `HistoryEvent`.

```
    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;
```

También configuramos algunos miembros de datos para realizar un seguimiento de cosas como:

- Una lista de objetos Decision usados para registrar los resultados de procesar la tarea
- Una cadena para almacenar la entrada del flujo de trabajo proporcionada por el evento "WorkflowExecutionStarted"
- Un recuento de las actividades programadas y abiertas (en ejecución) para evitar programar la misma actividad cuando ya se ha programado o se está ejecutando en este momento
- Un valor booleano para indicar que la actividad se ha completado
- Una cadena para almacenar los resultados de la actividad, que se devolverán como el resultado del flujo de trabajo

6. A continuación, añada código a `executeDecisionTask` para procesar los objetos `HistoryEvent` que se han enviado a la tarea, en función del tipo de evento notificado por el método `getEventType`.

```
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("]");
```

A efectos de nuestro flujo de trabajo, lo que más nos interesa es:

- El evento "WorkflowExecutionStarted", que indica que ha comenzado la ejecución del flujo de trabajo (lo que normalmente significa que debe ejecutar la primera actividad del flujo de trabajo) y que ofrece la entrada inicial proporcionada en el flujo de trabajo. En este caso, es la parte del nombre del saludo, por lo que se guarda como una cadena para usarla al programar la actividad que se debe ejecutar.
- El evento "ActivityTaskCompleted", que se envía una vez que se haya completado la actividad programada. Los datos del evento también incluyen el valor devuelto de la actividad completada. Como solo tenemos una actividad, usaremos el valor como el resultado de todo el flujo de trabajo.

Los demás tipos de eventos se pueden utilizar si el flujo de trabajo así lo requiere. Consulte la descripción de la clase HistoryEvent para obtener información acerca de cada tipo de evento.

+ NOTA: las cadenas de `switch` se introdujeron en Java 7. Si utiliza una versión anterior de Java, puede utilizar la clase EventType para convertir el valor `String` devuelto por `history_event.getType()` en un valor enum y luego en un valor `String` si es necesario:

```
EventType et = EventType.fromValue(event.getEventType());
```

1. Detrás de la instrucción `switch`, añada más código para responder con una decisión adecuada en función de la tarea que se ha recibido.

```
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);
```

- Si la actividad aún no se ha programado, respondemos con una decisión `ScheduleActivityTask`, que proporciona información en una estructura ScheduleActivityTaskDecisionAttributes sobre la actividad que Amazon SWF debería programar a continuación, incluyendo también los datos que Amazon SWF debe enviar a la actividad.
- Si la actividad se ha completado, consideramos que se ha completado todo el flujo de trabajo y respondemos con una decisión `CompletedWorkflowExecution`, rellenando una estructura CompleteWorkflowExecutionDecisionAttributes para proporcionar información sobre el flujo de trabajo completado. En este caso, devolvemos el resultado de la actividad.

En cualquier caso, la información de la decisión se añade a la lista `Decision` que se declaró encima del método.

2. Complete la tarea de decisión devolviendo la lista de objetos `Decision` recopilados al procesar la tarea. Añada este código al final del método `executeDecisionTask` que hemos escrito:

```
swf.respondDecisionTaskCompleted(
    new RespondDecisionTaskCompletedRequest()
        .withTaskToken(taskToken)
        .withDecisions(decisions));
```

El método `respondDecisionTaskCompleted` del cliente de SWF toma el token de la tarea que identifica la tarea, así como la lista de objetos `Decision`.

Implementación del iniciador del flujo de trabajo

Por último, escribiremos código para iniciar la ejecución del flujo de trabajo.

1. Abra su editor de texto y cree el archivo `WorkflowStarter.java`, añadiendo una declaración del paquete y funciones import de acuerdo con los pasos comunes.
2. Añada la clase `WorkflowStarter`:

```
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() + "'.");
    }
}
```

La clase `WorkflowStarter` consta de un único método, `main`que toma un argumento opcional pasado en la línea de comandos como datos de entrada para el flujo de trabajo.

El método del cliente de SWF, `startWorkflowExecution`, toma un objeto StartWorkflowExecutionRequest como entrada. Aquí, además de especificar el dominio y tipo de flujo de trabajo que se va a ejecutar, proporcionamos:

- Un nombre de ejecución de flujo de trabajo en lenguaje natural
- Los datos de entrada del flujo de trabajo (proporcionados en la línea de comandos en nuestro ejemplo)

- Un valor de tiempo de espera que representa cuánto tiempo, en segundos, debe tardar en ejecutarse todo el flujo de trabajo

El objeto Run que devuelve `startWorkflowExecution` proporciona un ID de ejecución, que es un valor que se puede utilizar para identificar esta ejecución del flujo de trabajo en concreto en el historial de ejecuciones del flujo de trabajo de Amazon SWF.

+ NOTA: el ID de ejecución lo genera Amazon SWF y no es el mismo que el nombre de ejecución del flujo de trabajo que se pasa al iniciar la ejecución del flujo de trabajo.

Compilación del ejemplo

Para crear el proyecto de ejemplo con Maven, vaya al directorio `helloswf` y escriba:

```
mvn package
```

El `helloswf-1.0.jar` resultante se generará en el directorio `target`.

Ejecución del ejemplo

El ejemplo consta de cuatro clases ejecutable distintas, que se ejecutan de forma independiente entre sí.

> **Note**
>
> Si utiliza un sistema Linux, macOS o Unix, puede ejecutarlas todas ellas, una detrás de otra, en una sola ventana del terminal. Si ejecuta Windows, debe abrir dos instancias de línea de comandos adicionales e ir al directorio `helloswf` de cada una de ellas.

Definición del classpath Java

Aunque Maven se encarga de las dependencias por usted, para ejecutar el ejemplo, tendrá que proporcionar la biblioteca del SDK de AWS y sus dependencias en el classpath Java. Puede establecer la variable de entorno `CLASSPATH` en la ubicación de las bibliotecas del SDK de AWS y el directorio `third-party/lib` del SDK, que incluye las dependencias necesarias:

```
export CLASSPATH='target/helloswf-1.0.jar:/path/to/sdk/lib/*:/path/to/sdk/third-party/
lib/*'
```

```
java example.swf.hello.HelloTypes
```

O puede usar la opción `-cp` del comando **`java`** para establecer el classpath mientras se ejecuta cada una de las aplicaciones.

```
java -cp target/helloswf-1.0.jar:/path/to/sdk/lib/*:/path/to/sdk/third-party/lib/* \
  example.swf.hello.HelloTypes
```

Usted decide el método que desea utilizar. Si no ha tenido problemas al compilar el código, pero al intentar ejecutar los ejemplos ha recibido una serie de errores "NoClassDefFound", es probable que el classpath esté establecido de forma incorrecta.

Registro del dominio y de los tipos de flujo de trabajo y actividad

Antes de ejecutar sus procesos de trabajo y el iniciador del flujo de trabajo, tendrá que registrar el dominio y sus tipos de flujo de trabajo y actividad. El código para esto se ha implementado en la sección Registro de un dominio y de tipos de flujo de trabajo y actividad.

Después de la compilación, si ha establecido el CLASSPATH, puede ejecutar el código de registro ejecutando el comando:

```
    echo 'Supply the name of one of the example classes as an argument.'
```

Inicio de los procesos de trabajo de actividad y flujo de trabajo

Ahora que los tipos se han registrado, puede iniciar los procesos de trabajo de actividad y flujo de trabajo. Estos se seguirán ejecutando y buscarán tareas hasta que se cancelen, por lo que deberá ejecutarlos en ventanas de terminal diferentes o, si utiliza Linux, macOS o Unix, puede usar el operador & para hacer que cada uno de ellos genere un proceso distinto cuando se ejecuten.

```
    echo 'If there are arguments to the class, put them in quotes after the class
name.'
    exit 1
```

Si ejecuta estos comandos en ventanas distintas, omita el operador & al final de cada línea.

Inicio de la ejecución del flujo de trabajo

Ahora que los procesos de trabajo de actividad y flujo de trabajo están realizando operaciones de sondeo, puede iniciar la ejecución del flujo de trabajo. Este proceso se ejecutará hasta que el flujo de

trabajo devuelva un estado completado. Debe ejecutarlo en una nueva ventana de terminal (a menos que ejecute sus procesos de trabajo como nuevos procesos generados mediante el operador &).

```
fi
```

> **Note**
>
> Si desea proporcionar sus propios datos de entrada, que se pasarán primero al flujo de trabajo y después a la actividad, añádalos a la línea de comandos. Por ejemplo:
>
> ```
> echo "## Running $className..."
> ```

Una vez que comience la ejecución del flujo de trabajo, debería empezar a ver los resultados enviados por ambos procesos de trabajo y por la propia ejecución del flujo de trabajo. Cuando el flujo de trabajo termine de completarse, el resultado se mostrará en la pantalla.

Código fuente completo de este ejemplo

Puede buscar el código fuente completo de este ejemplo en Github en el repositorio aws-java-developer-guide.

Para obtener más información

- Los procesos de trabajo presentados aquí pueden ocasionar la pérdida de tareas si se cierran mientras se ejecuta un sondeo del flujo de trabajo. Para saber cómo cerrar correctamente los procesos de trabajo, consulte Cerrar correctamente los procesos de trabajo de actividad y flujo de trabajo.
- Para obtener más información sobre Amazon SWF, visite la página principal de Amazon SWF o consulte la Guía para desarrolladores de Amazon SWF.
- Puede utilizar AWS Flow Framework para Java para crear flujos de trabajo más complejos en un estilo Java elegante mediante anotaciones. Para obtener más información, consulte la Guía para desarrolladores de AWS Flow Framework para Java.

Tareas de Lambda

Como alternativa a las actividades de Amazon SWF, o en combinación con ellas, puede utilizar funciones Lambda que representen unidades de trabajo en sus flujos de trabajo y programarlas de manera similar en actividades.

Este tema se centra en cómo implementar tareas Amazon SWF Lambda mediante AWS SDK for Java. Para obtener más información acerca de las tareas de Lambda, consulte Tareas AWS Lambda en la Guía para desarrolladores de Amazon SWF.

Configuración de un rol de IAM de varios servicios para ejecutar su función Lambda

Para que Amazon SWF pueda ejecutar su función Lambda, debe configurar un rol de IAM para conceder a Amazon SWF permiso para ejecutar funciones Lambda en su nombre. Para obtener información completa al respecto, consulte AWS LambdaTareas de .

Necesitará el Nombre de recurso de Amazon (ARN) de este rol de IAM cuando registre un flujo de trabajo que utilice tareas de Lambda.

Crear una función de Lambda

Puede crear funciones Lambda en diferentes lenguajes, incluido Java. Para obtener información completa sobre cómo crear, implementar y utilizar funciones Lambda, consulte la AWS LambdaGuía para desarrolladores de .

> **Note**
>
> Independientemente del lenguaje que use para crear la función Lambda, puede programarla y ejecutarla mediante cualquier flujo de trabajo de Amazon SWF, sea cual sea el lenguaje en el que esté escrito el código del flujo de trabajo. Amazon SWF se encarga de los detalles de la ejecución de la función y de pasar los datos.

A continuación se incluye una función Lambda sencilla que se puede utilizar en lugar de la actividad que se indica en Creación de una aplicación de Amazon SWF sencilla.

- Esta versión está escrita en JavaScript y se puede introducir directamente con la AWS Management Console:

```
exports.handler = function(event, context) {
```

```
    context.succeed("Hello, " + event.who + "!");
};
```

- Esta es la misma función escrita en Java, que también podría implementar y ejecutar en 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**
>
> Para obtener más información acerca de la implementación de funciones Java en Lambda, consulte Creación de un cómo crear un paquete de implementación (Java) en la Guía para desarrolladores de AWS Lambda. Puede consultar también la sección sobre cómo programar modelos de programación para crear funciones Lambda en Java.

Las funciones Lambda toman un evento u objeto de entrada como el primer parámetro y un objeto de contexto como el segundo, que proporciona información sobre la solicitud para ejecutar la función Lambda. Esta función en concreto espera que la entrada esté en formato JSON, con un campo `who` establecido en el nombre usado para crear el saludo.

Registrar un flujo de trabajo para su uso con Lambda

Para que un flujo de trabajo programe una función Lambda, debe proporcionar el nombre del rol de IAM que proporciona Amazon SWF con permiso para invocar funciones Lambda. Puede definir esto durante el registro del flujo de trabajo mediante los métodos `withDefaultLambdaRole` o `setDefaultLambdaRole` de 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) {
```

Programación de una tarea de Lambda

Programar un tarea de Lambda es similar a programar una actividad. Usted proporciona una Decisión con un DecisionType `ScheduleLambdaFunction` y con 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(
```

En el `ScheduleLambdaFuntionDecisionAttributes`, debe proporcionar un nombre, que es el ARN de la función Lambda que se va a llamar, y un ID, que es el nombre que Amazon SWF usará para identificar la función Lambda en los registros del historial.

También puede proporcionar una entrada opcional para la función Lambda y establecer su valor de start to close timeout, que es el número de segundos que la función Lambda se puede ejecutar antes de generar un evento `LambdaFunctionTimedOut`.

> **Note**
>
> Este código utiliza AWSLambdaClient para recuperar el ARN de la función Lambda, dado el nombre de la función. Puede utilizar esta técnica para evitar codificar de forma rígida el ARN completo (que incluye el ID de Cuenta de AWS) en el código.

Controlar eventos de funciones de Lambda en su decisor

Las tareas de Lambda generarán una serie de eventos a partir de los cuales puede emprender acciones cuando se sondeen las tareas de decisión en el proceso de trabajo de flujo de trabajo, correspondientes al ciclo de vida de su tarea de Lambda, con valores de EventType como `LambdaFunctionScheduled`, `LambdaFunctionStarted` y `LambdaFunctionCompleted`. Si la función Lambda produce un error o tarda más tiempo en ejecutarse que el valor de tiempo de espera establecido, recibirá un tipo de evento `LambdaFunctionFailed` o `LambdaFunctionTimedOut`, respectivamente.

```
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;
```

Recibir la salida de su función Lambda

Cuando recibe un `LambdaFunctionCompleted`EventType, you can retrieve your 0 function's return value by first calling `getLambdaFunctionCompletedEventAttributes en el HistoryEvent para obtener un objeto LambdaFunctionCompletedEventAttributes, y luego llamar a su método `getResult` para recuperar la salida de la función Lambda:

```
 LambdaFunctionCompleted:
running_functions--;
```

Código fuente completo de este ejemplo

Puede buscar el código fuente completo :github:<awsdocs/aws-java-developer-guide/tree/master/doc_source/snippets/helloswf_lambda/> de este ejemplo en Github en el repositorio aws-java-developer-guide.

Cerrar correctamente los procesos de trabajo de actividad y flujo de trabajo

En el tema Creación de una aplicación de Amazon SWF sencilla se proporciona una implementación completa de una aplicación de flujo de trabajo sencilla que consta de una solicitud de registro, un proceso de trabajo de actividad y flujo de trabajo y un iniciador de flujo de trabajo.

Las clases de los procesos de trabajo se han diseñado para que se ejecuten continuamente en busca de tareas enviadas por Amazon SWF para ejecutar actividades o devolver decisiones. Una vez que se realiza una solicitud de sondeo, Amazon SWF registra el sondeador e intentará asignarle una tarea.

Si el proceso de trabajo de flujo de trabajo se termina durante un sondeo de larga duración, Amazon SWF puede seguir intentando enviar una tarea al proceso de trabajo terminado, lo que desembocará en una tarea perdida (hasta que se agote el tiempo de espera de la tarea).

Una forma de abordar esta situación es esperar a que todas las solicitudes de sondeo de larga duración finalicen antes de que termine el proceso de trabajo.

En este tema, reescribiremos el proceso de trabajo de actividad de `helloswf`, utilizando enlaces de cierre de Java para cerrar correctamente el proceso de trabajo de actividad.

Este es el código completo:

```
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()));
                }
            }
        }
    }
}
```

En esta versión, el código de sondeo que estaba en la función `main` en la versión original se ha movido a su propio método, `pollAndExecute`.

La función `main` ahora utiliza CountDownLatch junto con un enlace de cierre para hacer que el subproceso espere hasta 60 segundos después de que se solicite su terminación y antes de permitir que se cierre el subproceso.

Registro de dominios

Cada flujo de trabajo y actividad de Amazon SWF requiere un dominio en el que ejecutarse.

1. Cree un nuevo objeto RegisterDomainRequest, proporcionando al menos el nombre de dominio y el periodo de retención de ejecución del flujo de trabajo (estos parámetros son obligatorios).
2. Llame al método AmazonSimpleWorkflowClient.registerDomain con el objeto RegisterDomainRequest.
3. Capture la excepción DomainAlreadyExistsException si el dominio que solicita ya existe (en cuyo caso, no se suele requerir ninguna acción).

El siguiente código muestra este procedimiento:

```
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!");
    }
}
```

Visualización de los dominios

Puede mostrar los dominios de Amazon SWF asociados a su cuenta y región de AWS por tipo de registro.

1. Cree un objeto ListDomainsRequest y especifique el estado de registro de los dominios en los que está interesado (obligatorio).
2. Llame a AmazonSimpleWorkflowClient.listDomains con el objeto ListDomainRequest. Los resultados se proporcionan en un objeto DomainInfos.
3. Llame a getDomainInfos devuelto en el objeto devuelto para obtener una lista de objetos DomainInfo.
4. Llame a getName en cada objeto DomainInfo para obtener su nombre.

El siguiente código muestra este procedimiento:

```
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());
    }
}
```

Ejemplos de código incluidos con el SDK

AWS SDK for Java incluye ejemplos de código que muestran muchas de las características del SDK en programas compilables y ejecutables. Puede estudiar o modificar estos ejemplos para implementar sus propias soluciones de AWS utilizando el AWS SDK for Java.

Cómo obtener los ejemplos

Los ejemplos de código de AWS SDK for Java se proporcionan en el directorio samples del SDK. Si ha descargado e instalado el SDK usando la información de Configuración del AWS SDK for Java, ya tiene los ejemplos en su sistema.

También puede consultar los últimos ejemplos en el repositorio de GitHub de AWS SDK for Java en el directorio src/samples.

Compilación y ejecución de los ejemplos mediante la línea de comandos

Los ejemplos incluyen scripts de compilación Ant para que pueda compilarlos y ejecutarlos fácilmente desde la línea de comandos. Cada ejemplo contiene también un archivo README en formato HTML que incluye información específica de cada ejemplo.

> **Note**
>
> Si examina el código de ejemplo en GitHub, haga clic en el botón Raw de la pantalla de código fuente cuando consulte el archivo README.html del ejemplo. En modo "raw", el HTML se mostrará de acuerdo con los requisitos del navegador.

Requisitos previos

Antes de ejecutar alguno de los ejemplos de AWS SDK for Java, necesitará configurar las credenciales de AWS en el entorno o con la AWS CLI, tal y como se especifica en Configuración de credenciales y regiones de AWS para desarrollo. Los ejemplos utilizan la cadena predeterminada de proveedores de credenciales siempre que sea posible. Por lo tanto, configurando las credenciales de esta forma, puede evitar el procedimiento arriesgado de insertar las credenciales de AWS en los archivos del directorio de código fuente (en el que puede activarlas sin querer y compartirlas públicamente).

Ejecución de los ejemplos

1. Vaya al directorio que contiene el código del ejemplo. Por ejemplo, si está en el directorio raíz de la descarga del SDK de AWS y desea ejecutar el ejemplo `AwsConsoleApp`, escriba:

```
cd samples/AwsConsoleApp
```

2. Compile y ejecute el ejemplo con Ant. El destino de la compilación predeterminado realiza ambas opciones, por lo que solo puede introducir:

```
ant
```

El ejemplo muestra información en la salida estándar:

```
===========================================

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.
```

Compilación y ejecución de los ejemplos mediante el IDE de Eclipse

Si utiliza AWS Toolkit for Eclipse, también puede iniciar un nuevo proyecto en Eclipse basado en AWS SDK for Java o añadir el SDK a un proyecto de Java existente.

Requisitos previos

Una vez instalado AWS Toolkit for Eclipse, le recomendamos que configure el conjunto de herramientas con sus credenciales de seguridad. Puede hacer esto en cualquier momento eligiendo Preferencias en el menú Ventana de Eclipse y eligiendo la sección Toolkit de AWS.

Ejecución de los ejemplos

1. Abra Eclipse.
2. Crear un nuevo proyecto Java de AWS En Eclipse, en el menú File (Archivo), elija New (Nuevo) y haga clic en Project (Proyecto). Se abre el asistente New Project (Nuevo proyecto).
3. Expanda la categoría AWS y, a continuación, elija Proyecto Java de AWS.
4. Elija Next (Siguiente). Se muestra la página de configuración del proyecto.
5. Introduzca el nombre en el cuadro Project Name (Nombre del proyecto). El grupo de ejemplos de AWS SDK for Java muestra los disponibles en el SDK, descritos anteriormente.
6. Seleccione los ejemplos que desea incluir en su proyecto seleccionando cada casilla de verificación.
7. Introduzca las credenciales de AWS Si ya ha configurado AWS Toolkit for Eclipse con sus credenciales, estas se rellenan automáticamente.
8. Elija Finalizar. El proyecto se crea y se añade a Project Explorer (Explorador de proyectos).
9. Elija el archivo `.java` del ejemplo que desea ejecutar. Por ejemplo, en el caso del ejemplo de Amazon S3, elija `S3Sample.java`.
10. Elija Run (Ejecutar) en el menú Run (Ejecutar).
11. Haga clic con el botón derecho en el proyecto en Project Explorer (Explorador de proyectos), seleccione Build Path (Ruta de compilación) y, a continuación, seleccione Add Libraries (Añadir bibliotecas).
12. Elija SDK de Java AWS, seleccione Siguiente y, a continuación, siga las instrucciones que aparecen en pantalla.

Seguridad para el AWS SDK for Java

La seguridad en la nube de Amazon Web Services (AWS) es la máxima prioridad. Como cliente de AWS, se beneficia de una arquitectura de red y un centro de datos que se han diseñado para satisfacer los requisitos de seguridad de las organizaciones más exigentes. La seguridad es una responsabilidad compartida entre AWS usted y usted. En el modelo de responsabilidad compartida, se habla de “seguridad de la nube” y “seguridad en la nube”:

Seguridad de la nube: AWS se encarga de proteger la infraestructura en la que se ejecutan todos los servicios que se ofrecen en la AWS nube y de proporcionarle servicios que pueda utilizar de forma segura. Nuestra responsabilidad en materia de seguridad es nuestra máxima prioridad AWS, y auditores externos comprueban y verifican periódicamente la eficacia de nuestra seguridad como parte de los programas de AWS conformidad.

Seguridad en la nube: su responsabilidad viene determinada por el AWS servicio que utilice y otros factores, como la confidencialidad de sus datos, los requisitos de su organización y las leyes y reglamentos aplicables.

Este AWS producto o servicio sigue el modelo de responsabilidad compartida a través de los servicios específicos de Amazon Web Services (AWS) a los que da soporte. Para obtener información sobre la seguridad de los AWS servicios, consulte la página de documentación sobre la seguridad del AWS servicio y AWS los servicios que se encuentran dentro del ámbito de aplicación de AWS las medidas de conformidad establecidas por el programa de conformidad.

Temas

Protección de datos en AWS SDK for Java 1.x

El modelo de responsabilidad compartida se aplica a la protección de datos en este AWS producto o servicio. Como se describe en este modelo, AWS es responsable de proteger la infraestructura global en la que se ejecuta toda la AWS nube. Usted es responsable de mantener el control sobre el contenido alojado en esta infraestructura. Este contenido incluye la configuración de seguridad y las tareas de administración de los servicios de AWS que usted utiliza. Para obtener más información sobre la privacidad de datos, consulte las Preguntas frecuentes sobre la privacidad de datos. Para obtener información sobre la protección de datos en Europa, consulte la entrada del blog sobre el modelo de responsabilidad AWS compartida y el RGPD en el blog AWS de seguridad.

Para proteger los datos, le recomendamos que proteja Cuenta de AWS las credenciales y configure cuentas de usuario individuales con AWS Identity and Access Management (IAM). De esta manera, cada usuario recibe únicamente los permisos necesarios para cumplir con sus obligaciones laborales. También recomendamos proteger sus datos de las siguientes maneras:

- Utilice autenticación multifactor (MFA) en cada cuenta.
- Utilice SSL/TLS para comunicarse con los recursos. AWS
- Configure la API y el registro de actividad de los usuarios con. AWS CloudTrail
- Utilice soluciones de AWS cifrado, con todos los controles de seguridad predeterminados en AWS los servicios.
- Utilice servicios de seguridad administrados avanzados, como Amazon Macie, que lo ayuden a detectar y proteger los datos personales almacenados en Amazon S3.
- Si necesita módulos criptográficos validados por FIPS 140-2 para acceder a AWS través de una interfaz de línea de comandos o una API, utilice un punto final FIPS. Para obtener más información acerca de los puntos de conexión de FIPS disponibles, consulte Estándar de procesamiento de la información federal (FIPS) 140-2.

Le recomendamos encarecidamente que nunca introduzca información de identificación confidencial, como, por ejemplo, números de cuenta de sus clientes, en los campos de formato libre, como el campo Nombre. Esto incluye cuando trabaja con este AWS producto o servicio u otros AWS servicios mediante la consola, la API o los SDK. AWS CLI AWS Todos los datos que introduzcas en este AWS producto o servicio o en otros servicios podrían recopilarse para incluirlos en los registros de diagnóstico. Cuando le proporcione una URL a un servidor externo, no incluya información sobre las credenciales en la URL para validar la solicitud en ese servidor.

AWS SDK for Java soporte para TLS

La siguiente información se aplica únicamente a la implementación de SSL de Java (la implementación de SSL predeterminada en el AWS SDK for Java). Si está usando una implementación SSL distinta, consulte su implementación SSL específica para saber cómo aplicar versiones de TLS.

Cómo verificar la versión de TLS

Consulte la documentación del proveedor de su máquina virtual Java (JVM) para determinar qué versiones de TLS son compatibles con su plataforma. En el caso de algunas JVM, el siguiente código mostrará qué versiones de SSL son compatibles.

```
System.out.println(Arrays.toString(SSLContext.getDefault().getSupportedSSLParameters().getProto
```

Para ver el protocolo de enlace SSL en acción y qué versión de TLS se utiliza, puede utilizar la propiedad del sistema javax.net.debug.

```
java app.jar -Djavax.net.debug=ssl
```

> **Note**
>
> TLS 1.3 no es compatible con las versiones 1.9.5 a 1.10.31 del SDK para Java. Para obtener más información, consulte la siguiente entrada del blog: https://aws.amazon.com/blogs/developer/tls-1-3 - incompatibility-with-aws-sdk - for-java-versions -1-9-5-to-1-10-31/

Aplicación de una versión mínima de TLS

El SDK siempre prefiere la última versión de TLS compatible con la plataforma y el servicio. Si desea aplicar una versión mínima específica de TLS, consulte la documentación de su JVM. Para las JVM basadas en OpenJDK, puede utilizar la propiedad del sistema `jdk.tls.client.protocols`.

```
java app.jar -Djdk.tls.client.protocols=PROTOCOLS
```

Consulte la documentación de su JVM para conocer los valores admitidos de los PROTOCOLOS.

Identity and Access Management

AWS Identity and Access Management (IAM) es una herramienta Servicio de AWS que ayuda al administrador a controlar de forma segura el acceso a los AWS recursos. Los administradores de IAM controlan quién puede autenticarse (iniciar sesión) y quién puede autorizarse (tener permisos) para usar los recursos. AWS La IAM es una Servicio de AWS opción que puede utilizar sin coste adicional.

Temas

Público

La forma de usar AWS Identity and Access Management (IAM) varía según el trabajo en el que se realice. AWS

Usuario del servicio: si Servicios de AWS solía hacer su trabajo, el administrador le proporcionará las credenciales y los permisos que necesita. A medida que vaya utilizando más AWS funciones para realizar su trabajo, es posible que necesite permisos adicionales. Entender cómo se administra el acceso puede ayudarlo a solicitar los permisos correctos al administrador. Si no puede acceder a una función de AWS, consulte Solución de problemas de AWS identidad y acceso o consulte la guía del usuario de la Servicio de AWS que está utilizando.

Administrador de servicios: si está a cargo de AWS los recursos de su empresa, probablemente tenga acceso total a ellos AWS. Su trabajo consiste en determinar a qué AWS funciones y recursos deben acceder los usuarios del servicio. Luego, debe enviar solicitudes a su administrador de IAM para cambiar los permisos de los usuarios de su servicio. Revise la información de esta página para conocer los conceptos básicos de IAM. Para obtener más información sobre cómo su empresa puede utilizar la IAM AWS, consulte la guía del usuario del Servicio de AWS que está utilizando.

Administrador de IAM: si es un administrador de IAM, es posible que quiera conocer más detalles sobre cómo escribir políticas para administrar el acceso a AWS. Para ver ejemplos de políticas AWS

basadas en la identidad que puede utilizar en IAM, consulte la guía del usuario de la Servicio de AWS que está utilizando.

Autenticación con identidades

La autenticación es la forma de iniciar sesión AWS con sus credenciales de identidad. Debe estar autenticado (con quien haya iniciado sesión AWS) como usuario de IAM o asumiendo una función de IAM. Usuario raíz de la cuenta de AWS

Puede iniciar sesión AWS como una identidad federada mediante las credenciales proporcionadas a través de una fuente de identidad. AWS IAM Identity Center Los usuarios (Centro de identidades de IAM), la autenticación de inicio de sesión único de su empresa y sus credenciales de Google o Facebook son ejemplos de identidades federadas. Al iniciar sesión como una identidad federada, su administrador habrá configurado previamente la federación de identidades mediante roles de IAM. Cuando accedes AWS mediante la federación, estás asumiendo un rol de forma indirecta.

Según el tipo de usuario que sea, puede iniciar sesión en el portal AWS Management Console o en el de AWS acceso. Para obtener más información sobre cómo iniciar sesión AWS, consulte Cómo iniciar sesión Cuenta de AWS en su Guía del AWS Sign-In usuario.

Si accede AWS mediante programación, AWS proporciona un kit de desarrollo de software (SDK) y una interfaz de línea de comandos (CLI) para firmar criptográficamente sus solicitudes con sus credenciales. Si no utilizas AWS herramientas, debes firmar las solicitudes tú mismo. Para obtener más información sobre cómo usar el método recomendado para firmar las solicitudes usted mismo, consulte Firmar las solicitudes de la AWS API en la Guía del usuario de IAM.

Independientemente del método de autenticación que use, es posible que deba proporcionar información de seguridad adicional. Por ejemplo, le AWS recomienda que utilice la autenticación multifactor (MFA) para aumentar la seguridad de su cuenta. Para más información, consulte Autenticación multifactor en la Guía del usuario de AWS IAM Identity Center y Uso de la autenticación multifactor (MFA) en AWS en la Guía del usuario de IAM.

Cuenta de AWS usuario root

Al crear una Cuenta de AWS, comienza con una identidad de inicio de sesión que tiene acceso completo a todos Servicios de AWS los recursos de la cuenta. Esta identidad se denomina usuario Cuenta de AWS raíz y se accede a ella iniciando sesión con la dirección de correo electrónico y la contraseña que utilizaste para crear la cuenta. Recomendamos encarecidamente que no utilice el usuario raíz para sus tareas diarias. Proteja las credenciales del usuario raíz y utilícelas solo para las tareas que solo el usuario raíz pueda realizar. Para ver la lista completa de las tareas que requieren

que inicie sesión como usuario raíz, consulte Tareas que requieren credenciales de usuario raíz en la Guía del usuario de IAM.

Identidad federada

Como práctica recomendada, exija a los usuarios humanos, incluidos los que requieren acceso de administrador, que utilicen la federación con un proveedor de identidades para acceder Servicios de AWS mediante credenciales temporales.

Una identidad federada es un usuario del directorio de usuarios de su empresa, un proveedor de identidades web AWS Directory Service, el directorio del Centro de Identidad o cualquier usuario al que acceda Servicios de AWS mediante las credenciales proporcionadas a través de una fuente de identidad. Cuando las identidades federadas acceden Cuentas de AWS, asumen funciones y las funciones proporcionan credenciales temporales.

Para una administración de acceso centralizada, le recomendamos que utilice AWS IAM Identity Center. Puede crear usuarios y grupos en el Centro de identidades de IAM, o puede conectarse y sincronizarse con un conjunto de usuarios y grupos de su propia fuente de identidad para usarlos en todas sus Cuentas de AWS aplicaciones. Para más información, consulte ¿Qué es IAM Identity Center? en la Guía del usuario de AWS IAM Identity Center .

Usuarios y grupos de IAM

Un usuario de IAM es una identidad propia Cuenta de AWS que tiene permisos específicos para una sola persona o aplicación. Siempre que sea posible, recomendamos emplear credenciales temporales, en lugar de crear usuarios de IAM que tengan credenciales de larga duración como contraseñas y claves de acceso. No obstante, si tiene casos de uso específicos que requieran credenciales de larga duración con usuarios de IAM, recomendamos rotar las claves de acceso. Para más información, consulte Rotar las claves de acceso periódicamente para casos de uso que requieran credenciales de larga duración en la Guía del usuario de IAM.

Un grupo de IAM es una identidad que especifica un conjunto de usuarios de IAM. No puede iniciar sesión como grupo. Puede usar los grupos para especificar permisos para varios usuarios a la vez. Los grupos facilitan la administración de los permisos de grandes conjuntos de usuarios. Por ejemplo, podría tener un grupo cuyo nombre fuese IAMAdmins y conceder permisos a dicho grupo para administrar los recursos de IAM.

Los usuarios son diferentes de los roles. Un usuario se asocia exclusivamente a una persona o aplicación, pero la intención es que cualquier usuario pueda asumir un rol que necesite. Los usuarios tienen credenciales permanentes a largo plazo y los roles proporcionan credenciales temporales.

Para más información, consulte Cuándo crear un usuario de IAM (en lugar de un rol) en la Guía del usuario de IAM.

Roles de IAM

Un rol de IAM es una identidad dentro de usted Cuenta de AWS que tiene permisos específicos. Es similar a un usuario de IAM, pero no está asociado a una determinada persona. Puede asumir temporalmente una función de IAM en el AWS Management Console cambiando de función. Puede asumir un rol llamando a una operación de AWS API AWS CLI o utilizando una URL personalizada. Para más información sobre los métodos para el uso de roles, consulte Uso de roles de IAM en la Guía del usuario de IAM.

Los roles de IAM con credenciales temporales son útiles en las siguientes situaciones:

- Acceso de usuario federado: para asignar permisos a una identidad federada, puede crear un rol y definir sus permisos. Cuando se autentica una identidad federada, se asocia la identidad al rol y se le conceden los permisos define el rol. Para obtener información acerca de roles para federación, consulte Creación de un rol para un proveedor de identidades de terceros en la Guía del usuario de IAM. Si utiliza el IAM Identity Center, debe configurar un conjunto de permisos. El Centro de identidades de IAM correlaciona el conjunto de permisos con un rol en IAM para controlar a qué pueden acceder sus identidades después de autenticarse. Para obtener información acerca de los conjuntos de permisos, consulte Conjuntos de permisos en la Guía del usuario de AWS IAM Identity Center .
- Permisos de usuario de IAM temporales: un usuario de IAM puede asumir un rol de IAM para recibir temporalmente permisos distintos que le permitan realizar una tarea concreta.
- Acceso entre cuentas: puede utilizar un rol de IAM para permitir que alguien (una entidad principal de confianza) de otra cuenta acceda a los recursos de la cuenta. Los roles son la forma principal de conceder acceso entre cuentas. Sin embargo, con algunas Servicios de AWS, puedes adjuntar una política directamente a un recurso (en lugar de usar un rol como proxy). Para obtener información acerca de la diferencia entre los roles y las políticas basadas en recursos para el acceso entre cuentas, consulte Cómo los roles de IAM difieren de las políticas basadas en recursos en la Guía del usuario de IAM.
- Acceso entre servicios: algunos Servicios de AWS utilizan funciones en otros Servicios de AWS. Por ejemplo, cuando realiza una llamada en un servicio, es común que ese servicio ejecute aplicaciones en Amazon EC2 o almacene objetos en Amazon S3. Es posible que un servicio haga esto usando los permisos de la entidad principal, usando un rol de servicio o usando un rol vinculado al servicio.

  - Sesiones de acceso directo (FAS): cuando utilizas un usuario o un rol de IAM para realizar acciones en ellas AWS, se te considera director. Cuando utiliza algunos servicios, es posible que realice una acción que desencadene otra acción en un servicio diferente. El FAS utiliza los permisos del principal que llama Servicio de AWS y los solicita Servicio de AWS para realizar solicitudes a los servicios descendentes. Las solicitudes de FAS solo se realizan cuando un servicio recibe una solicitud que requiere interacciones con otros Servicios de AWS recursos para completarse. En este caso, debe tener permisos para realizar ambas acciones. Para obtener información sobre las políticas a la hora de realizar solicitudes de FAS, consulte Reenviar sesiones de acceso.
  - Rol de servicio: un rol de servicio es un rol de IAM que adopta un servicio para realizar acciones en su nombre. Un administrador de IAM puede crear, modificar y eliminar un rol de servicio desde IAM. Para más información, consulte Creación de un rol para delegar permisos a un Servicio de AWS en la Guía del usuario de IAM.
  - Función vinculada al servicio: una función vinculada a un servicio es un tipo de función de servicio que está vinculada a un. Servicio de AWS El servicio puede asumir el rol para realizar una acción en su nombre. Los roles vinculados al servicio aparecen en usted Cuenta de AWS y son propiedad del servicio. Un administrador de IAM puede ver, pero no editar, los permisos de los roles vinculados a servicios.
- Aplicaciones que se ejecutan en Amazon EC2: puede usar un rol de IAM para administrar las credenciales temporales de las aplicaciones que se ejecutan en una instancia EC2 y realizan AWS CLI solicitudes a la API. AWS Es preferible hacerlo de este modo a almacenar claves de acceso en la instancia EC2. Para asignar una AWS función a una instancia EC2 y ponerla a disposición de todas sus aplicaciones, debe crear un perfil de instancia adjunto a la instancia. Un perfil de instancia contiene el rol y permite a los programas que se ejecutan en la instancia EC2 obtener credenciales temporales. Para más información, consulte Uso de un rol de IAM para conceder permisos a aplicaciones que se ejecutan en instancias Amazon EC2 en la Guía del usuario de IAM.

Para obtener información sobre el uso de los roles de IAM, consulte Cuándo crear un rol de IAM (en lugar de un usuario) en la Guía del usuario de IAM.

Administración de acceso mediante políticas

El acceso se controla AWS creando políticas y adjuntándolas a AWS identidades o recursos. Una política es un objeto AWS que, cuando se asocia a una identidad o un recurso, define sus permisos. AWS evalúa estas políticas cuando un director (usuario, usuario raíz o sesión de rol) realiza una

solicitud. Los permisos en las políticas determinan si la solicitud se permite o se deniega. La mayoría de las políticas se almacenan AWS como documentos JSON. Para obtener más información sobre la estructura y el contenido de los documentos de política JSON, consulte Información general de políticas JSON en la Guía del usuario de IAM.

Los administradores pueden usar las políticas de AWS JSON para especificar quién tiene acceso a qué. Es decir, qué entidad principal puede realizar acciones en qué recursos y en qué condiciones.

De forma predeterminada, los usuarios y los roles no tienen permisos. Para conceder permiso a los usuarios para realizar acciones en los recursos que necesiten, un administrador de IAM puede crear políticas de IAM. A continuación, el administrador puede agregar las políticas de IAM a los roles y los usuarios pueden asumir esos roles.

Las políticas de IAM definen permisos para una acción independientemente del método que se utilice para realizar la operación. Por ejemplo, suponga que dispone de una política que permite la acción `iam:GetRole`. Un usuario con esa política puede obtener información sobre el rol de la API AWS Management Console AWS CLI, la o la AWS API.

Políticas basadas en identidades

Las políticas basadas en identidad son documentos de políticas de permisos JSON que puede asociar a una identidad, como un usuario de IAM, un grupo de usuarios o un rol. Estas políticas controlan qué acciones pueden realizar los usuarios y los roles, en qué recursos y en qué condiciones. Para obtener más información sobre cómo crear una política basada en identidad, consulte Creación de políticas de IAM en la Guía del usuario de IAM.

Las políticas basadas en identidades pueden clasificarse además como políticas insertadas o políticas administradas. Las políticas insertadas se integran directamente en un único usuario, grupo o rol. Las políticas administradas son políticas independientes que puede adjuntar a varios usuarios, grupos y roles de su Cuenta de AWS empresa. Las políticas administradas incluyen políticas AWS administradas y políticas administradas por el cliente. Para más información sobre cómo elegir una política administrada o una política insertada, consulte Elegir entre políticas administradas y políticas insertadas en la Guía del usuario de IAM.

Políticas basadas en recursos

Las políticas basadas en recursos son documentos de política JSON que se asocian a un recurso. Ejemplos de políticas basadas en recursos son las políticas de confianza de roles de IAM y las políticas de bucket de Amazon S3. En los servicios que admiten políticas basadas en recursos, los

administradores de servicios pueden utilizarlos para controlar el acceso a un recurso específico. Para el recurso al que se asocia la política, la política define qué acciones puede realizar una entidad principal especificada en ese recurso y en qué condiciones. Debe especificar una entidad principal en una política en función de recursos. Los principales pueden incluir cuentas, usuarios, roles, usuarios federados o. Servicios de AWS

Las políticas basadas en recursos son políticas insertadas que se encuentran en ese servicio. No puedes usar políticas AWS gestionadas de IAM en una política basada en recursos.

Listas de control de acceso (ACL)

Las listas de control de acceso (ACL) controlan qué entidades principales (miembros de cuentas, usuarios o roles) tienen permisos para acceder a un recurso. Las ACL son similares a las políticas basadas en recursos, aunque no utilizan el formato de documento de políticas JSON.

Amazon S3 y Amazon VPC son ejemplos de servicios que admiten las ACL. AWS WAF Para obtener más información sobre las ACL, consulte Información general de Lista de control de acceso (ACL) en la Guía para desarrolladores de Amazon Simple Storage Service.

Otros tipos de políticas

AWS admite tipos de políticas adicionales y menos comunes. Estos tipos de políticas pueden establecer el máximo de permisos que los tipos de políticas más frecuentes le conceden.

- Límites de permisos: un límite de permisos es una característica avanzada que le permite establecer los permisos máximos que una política basada en identidad puede conceder a una entidad de IAM (usuario o rol de IAM). Puede establecer un límite de permisos para una entidad. Los permisos resultantes son la intersección de las políticas basadas en la identidad de la entidad y los límites de permisos. Las políticas basadas en recursos que especifique el usuario o rol en el campo `Principal` no estarán restringidas por el límite de permisos. Una denegación explícita en cualquiera de estas políticas anulará el permiso. Para obtener más información sobre los límites de los permisos, consulte Límites de permisos para las entidades de IAM en la Guía del usuario de IAM.
- Políticas de control de servicios (SCP): las SCP son políticas de JSON que especifican los permisos máximos para una organización o unidad organizativa (OU). AWS Organizations AWS Organizations es un servicio para agrupar y gestionar de forma centralizada varios de los Cuentas de AWS que son propiedad de su empresa. Si habilita todas las características en una organización, entonces podrá aplicar políticas de control de servicio (SCP) a una o a todas sus cuentas. El SCP limita los permisos de las entidades en las cuentas de los miembros, incluidas las

de cada una. Usuario raíz de la cuenta de AWS Para más información sobre Organizations y las SCP, consulte Funcionamiento de las SCP en la Guía del usuario de AWS Organizations .

- Políticas de sesión: las políticas de sesión son políticas avanzadas que se pasan como parámetro cuando se crea una sesión temporal mediante programación para un rol o un usuario federado. Los permisos de la sesión resultantes son la intersección de las políticas basadas en identidades del rol y las políticas de la sesión. Los permisos también pueden proceder de una política en función de recursos. Una denegación explícita en cualquiera de estas políticas anulará el permiso. Para más información, consulte Políticas de sesión en la Guía del usuario de IAM.

Varios tipos de políticas

Cuando se aplican varios tipos de políticas a una solicitud, los permisos resultantes son más complicados de entender. Para saber cómo AWS determinar si se debe permitir una solicitud cuando se trata de varios tipos de políticas, consulte la lógica de evaluación de políticas en la Guía del usuario de IAM.

¿Cómo Servicios de AWS trabajar con IAM

Para obtener una visión general de cómo Servicios de AWS trabajar con la mayoría de las funciones de IAM, consulte AWS los servicios que funcionan con IAM en la Guía del usuario de IAM.

Para obtener información sobre cómo utilizar una función específica Servicio de AWS con IAM, consulte la sección de seguridad de la guía del usuario del servicio correspondiente.

Solución de problemas de AWS identidad y acceso

Utilice la siguiente información como ayuda para diagnosticar y solucionar los problemas habituales que pueden surgir al trabajar con un AWS IAM.

Temas

- No estoy autorizado a realizar ninguna acción en AWS
- No estoy autorizado a realizar tareas como: PassRole
- Quiero permitir que personas ajenas a mí accedan Cuenta de AWS a mis AWS recursos

No estoy autorizado a realizar ninguna acción en AWS

Si recibe un error que indica que no tiene autorización para realizar una acción, las políticas se deben actualizar para permitirle realizar la acción.

En el siguiente ejemplo, el error se produce cuando el usuario de IAM `mateojackson` intenta utilizar la consola para consultar los detalles acerca de un recurso ficticio *`my-example-widget`*, pero no tiene los permisos ficticios `awes:`*`GetWidget`*.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 awes:GetWidget on resource: my-example-widget
```

En este caso, la política del usuario `mateojackson` debe actualizarse para permitir el acceso al recurso *`my-example-widget`* mediante la acción `awes:`*`GetWidget`*.

Si necesita ayuda, póngase en contacto con su AWS administrador. El administrador es la persona que le proporcionó las credenciales de inicio de sesión.

No estoy autorizado a realizar tareas como: PassRole

Si recibe un error que indica que no tiene autorización para realizar la acción `iam:PassRole`, las políticas deben actualizarse a fin de permitirle pasar un rol a AWS.

Algunas Servicios de AWS permiten transferir una función existente a ese servicio en lugar de crear una nueva función de servicio o una función vinculada a un servicio. Para ello, debe tener permisos para transferir el rol al servicio.

En el siguiente ejemplo, el error se produce cuando un usuario de IAM denominado `marymajor` intenta utilizar la consola para realizar una acción en AWS. Sin embargo, la acción requiere que el servicio cuente con permisos que otorguen un rol de servicio. Mary no tiene permisos para transferir el rol al servicio.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

En este caso, las políticas de Mary se deben actualizar para permitirle realizar la acción `iam:PassRole`.

Si necesita ayuda, póngase en contacto con su administrador. AWS El administrador es la persona que le proporcionó las credenciales de inicio de sesión.

Quiero permitir que personas ajenas a mí accedan Cuenta de AWS a mis AWS recursos

Puede crear un rol que los usuarios de otras cuentas o las personas externas a la organización puedan utilizar para acceder a sus recursos. Puede especificar una persona de confianza para que

asuma el rol. En el caso de los servicios que admitan las políticas basadas en recursos o las listas de control de acceso (ACL), puede utilizar dichas políticas para conceder a las personas acceso a sus recursos.

Para más información, consulte lo siguiente:

- Para saber si AWS es compatible con estas funciones, consulte¿Cómo Servicios de AWS trabajar con IAM.
- Para obtener información sobre cómo proporcionar acceso a los recursos de su Cuentas de AWS propiedad, consulte Proporcionar acceso a un usuario de IAM en otro usuario de su propiedad Cuenta de AWS en la Guía del usuario de IAM.
- Para obtener información sobre cómo proporcionar acceso a tus recursos a terceros Cuentas de AWS, consulta Cómo proporcionar acceso a recursos que Cuentas de AWS son propiedad de terceros en la Guía del usuario de IAM.
- Para obtener información sobre cómo proporcionar acceso mediante la federación de identidades, consulte Proporcionar acceso a usuarios autenticados externamente (federación de identidades) en la Guía del usuario de IAM.
- Para obtener información sobre la diferencia entre los roles y las políticas basadas en recursos para el acceso entre cuentas, consulte Cómo los roles de IAM difieren de las políticas basadas en recursos en la Guía del usuario de IAM.

Validación de la conformidad de este AWS producto o servicio

Para saber si un programa de cumplimiento Servicio de AWS está dentro del ámbito de aplicación de programas de cumplimiento específicos, consulte Servicios de AWS Alcance por programa de de cumplimiento y elija el programa de cumplimiento que le interese. Para obtener información general, consulte Programas de AWS cumplimiento > Programas AWS .

Puede descargar informes de auditoría de terceros utilizando AWS Artifact. Para obtener más información, consulte Descarga de informes en AWS Artifact .

Su responsabilidad de cumplimiento al Servicios de AWS utilizarlos viene determinada por la confidencialidad de sus datos, los objetivos de cumplimiento de su empresa y las leyes y reglamentos aplicables. AWS proporciona los siguientes recursos para ayudar con el cumplimiento:

- Guías de inicio rápido sobre seguridad y cumplimiento: estas guías de implementación analizan las consideraciones arquitectónicas y proporcionan los pasos para implementar entornos básicos centrados en AWS la seguridad y el cumplimiento.
- Diseño de arquitectura para garantizar la seguridad y el cumplimiento de la HIPAA en Amazon Web Services: este documento técnico describe cómo las empresas pueden crear aplicaciones aptas para AWS la HIPAA.

> **Note**
>
> No Servicios de AWS todas cumplen con los requisitos de la HIPAA. Para más información, consulte la Referencia de servicios compatibles con HIPAA.

- AWS Recursos de de cumplimiento: esta colección de libros de trabajo y guías puede aplicarse a su industria y ubicación.
- AWS Guías de cumplimiento para clientes: comprenda el modelo de responsabilidad compartida desde el punto de vista del cumplimiento. Las guías resumen las mejores prácticas para garantizar la seguridad Servicios de AWS y orientan los controles de seguridad en varios marcos (incluidos el Instituto Nacional de Estándares y Tecnología (NIST), el Consejo de Normas de Seguridad del Sector de Tarjetas de Pago (PCI) y la Organización Internacional de Normalización (ISO)).
- Evaluación de los recursos con reglas en la guía para AWS Config desarrolladores: el AWS Config servicio evalúa en qué medida las configuraciones de los recursos cumplen con las prácticas internas, las directrices del sector y las normas.
- AWS Security Hub— Esto Servicio de AWS proporciona una visión completa del estado de su seguridad interior AWS. Security Hub utiliza controles de seguridad para evaluar sus recursos de AWS y comprobar su cumplimiento con los estándares y las prácticas recomendadas del sector de la seguridad. Para obtener una lista de los servicios y controles compatibles, consulte la Referencia de controles de Security Hub.
- AWS Audit Manager— Esto le Servicio de AWS ayuda a auditar continuamente su AWS consumo para simplificar la gestión del riesgo y el cumplimiento de las normativas y los estándares del sector.

Este AWS producto o servicio sigue el modelo de responsabilidad compartida a través de los servicios específicos de Amazon Web Services (AWS) a los que da soporte. Para obtener información sobre la seguridad de los AWS servicios, consulte la página de documentación sobre la seguridad del AWS servicio y AWS los servicios que se encuentran dentro del ámbito de aplicación de AWS las medidas de conformidad establecidas por el programa de conformidad.

Resiliencia de este AWS producto o servicio

La infraestructura AWS global se basa en Regiones de AWS zonas de disponibilidad.

Regiones de AWS proporcionan varias zonas de disponibilidad aisladas y separadas físicamente, que están conectadas mediante redes de baja latencia, alto rendimiento y alta redundancia.

Con las zonas de disponibilidad, puede diseñar y utilizar aplicaciones y bases de datos que realizan una conmutación por error automática entre las zonas sin interrupciones. Las zonas de disponibilidad tienen una mayor disponibilidad, tolerancia a errores y escalabilidad que las infraestructuras tradicionales de uno o varios centros de datos.

Para obtener más información sobre AWS las regiones y las zonas de disponibilidad, consulte Infraestructura global.AWS

Este AWS producto o servicio sigue el modelo de responsabilidad compartida a través de los servicios específicos de Amazon Web Services (AWS) a los que da soporte. Para obtener información sobre la seguridad de los AWS servicios, consulte la página de documentación sobre la seguridad del AWS servicio y AWS los servicios que se encuentran dentro del ámbito de aplicación de AWS las medidas de conformidad establecidas por el programa de conformidad.

Seguridad de la infraestructura para este AWS producto o servicio

Este AWS producto o servicio utiliza servicios gestionados y, por lo tanto, está protegido por la seguridad de la red AWS global. Para obtener información sobre los servicios AWS de seguridad y cómo se AWS protege la infraestructura, consulte Seguridad AWS en la nube. Para diseñar su AWS entorno utilizando las mejores prácticas de seguridad de la infraestructura, consulte Protección de infraestructuras en un marco de buena AWS arquitectura basado en el pilar de la seguridad.

Utiliza las llamadas a la API AWS publicadas para acceder a este AWS producto o servicio a través de la red. Los clientes deben admitir lo siguiente:

- Seguridad de la capa de transporte (TLS). Exigimos TLS 1.2 y recomendamos TLS 1.3.
- Conjuntos de cifrado con confidencialidad directa total (PFS) como DHE (Ephemeral Diffie-Hellman) o ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). La mayoría de los sistemas modernos como Java 7 y posteriores son compatibles con estos modos.

Además, las solicitudes deben estar firmadas mediante un ID de clave de acceso y una clave de acceso secreta que esté asociada a una entidad principal de IAM. También puede utilizar AWS

Security Token Service (AWS STS) para generar credenciales de seguridad temporales para firmar solicitudes.

Este AWS producto o servicio sigue el modelo de responsabilidad compartida a través de los servicios específicos de Amazon Web Services (AWS) a los que da soporte. Para obtener información sobre la seguridad de los AWS servicios, consulte la página de documentación sobre la seguridad del AWS servicio y AWS los servicios que se encuentran dentro del ámbito de aplicación de AWS las medidas de conformidad establecidas por el programa de conformidad.

Amazon S3 Migración de clientes de cifrado

En este tema se muestra cómo migrar las aplicaciones de la versión 1 (V1) del cliente de cifrado Amazon Simple Storage Service (Amazon S3) a la versión 2 (V2) y cómo garantizar la disponibilidad de las aplicaciones durante todo el proceso de migración.

Requisitos previos

Amazon S3 El cifrado del lado del cliente requiere lo siguiente:

- Java 8 o una versión posterior instalada en el entorno de la aplicación. AWS SDK for Java Funciona con el kit de desarrollo Java SE de Oracle y con distribuciones del Open Java Development Kit (OpenJDK) Amazon Corretto, como Red Hat OpenJDK y JDK. AdoptOpen
- El paquete criptográfico Bouncy Castle. Puede colocar el archivo .jar de Bouncy Castle en la ruta de clases del entorno de su aplicación o añadir una dependencia del ArtifactID `bcprov-ext-jdk15on` (con el GroupID de `org.bouncycastle`) a su archivo Maven `pom.xml`.

Información general sobre la migración

Esta migración se produce en dos fases:

1. Actualizar los clientes existentes para leer nuevos formatos. Actualice la aplicación para que utilice la versión 1.11.837 o posterior y vuelva a implementar la aplicación. AWS SDK for Java Esto permite a los clientes del servicio de cifrado Amazon S3 del lado del cliente de su aplicación descifrar los objetos creados por los clientes del servicio V2. Si su aplicación usa varios AWS SDK, debe actualizar cada SDK por separado.
2. Migue los clientes de cifrado y descifrado a la versión V2. Una vez que todos sus clientes de cifrado V1 puedan leer los formatos de cifrado V2, actualice los Amazon S3 clientes de cifrado y descifrado del lado del cliente en el código de la aplicación para usar sus equivalentes en V2.

Actualizar los clientes existentes para leer nuevos formatos

El cliente de cifrado V2 utiliza algoritmos de cifrado que las versiones anteriores no admiten. AWS SDK for Java

El primer paso de la migración consiste en actualizar los clientes de cifrado de la versión 1 para que utilicen la versión 1.11.837 o posterior del AWS SDK for Java. (Le recomendamos que actualice a la versión más reciente, que encontrará en la versión 1.x de la Referencia de la API de Java). Para ello, actualice la dependencia en la configuración de su proyecto. Una vez actualizada la configuración del proyecto, reconstruya el proyecto y vuelva a implementarlo.

Cuando haya completado estos pasos, los clientes de cifrado V1 de su aplicación podrán leer los objetos escritos por los clientes de cifrado V2.

Actualizar la dependencia en la configuración de su proyecto.

Modificar el archivo de configuración del proyecto (por ejemplo, pom.xml o build.gradle) para usar la versión 1.11.837 o posterior de AWS SDK for Java. A continuación, reconstruya su proyecto y vuelva a implementarlo.

Completar este paso antes de implementar el nuevo código de aplicación ayuda a garantizar que las operaciones de cifrado y descifrado permanezcan consistentes en toda la flota durante el proceso de migración.

Ejemplo de uso de Maven

Fragmento de un archivo pom.xml:

```
<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>
```

Ejemplo de uso de Gradle

Fragmento de un archivo build.gradle:

```
dependencies {
  implementation platform('com.amazonaws:aws-java-sdk-bom:1.11.837')
  implementation 'com.amazonaws:aws-java-sdk-s3'
}
```

Migrar clientes de cifrado y descifrado a la versión V2

Una vez que tu proyecto se haya actualizado con la última versión del SDK, puede modificar el código de la aplicación para usar el cliente V2. Para ello, primero actualice el código para usar el nuevo generador de clientes de servicios. A continuación, proporcione los materiales de cifrado mediante un método del generador al que se le haya cambiado el nombre y configure el cliente de servicio según sea necesario.

Estos fragmentos de código muestran cómo utilizar el cifrado del lado del cliente con los AWS SDK for Java clientes de cifrado V1 y V2 y proporcionan comparaciones entre ellos.

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();
```

En el ejemplo anterior se establece el `cryptoMode` como `AuthenticatedEncryption`. Esta es una configuración que permite a un cliente de cifrado V2 leer objetos escritos por un cliente de cifrado V1. Si su cliente no necesita leer objetos escritos por un cliente V1, le recomendamos que utilice la configuración predeterminada `StrictAuthenticatedEncryption`.

Crear un cliente de cifrado V2

El cliente de cifrado V2 se puede crear llamando a EncryptionClientAmazonS3 v2.encryptionBuilder ().

Puede sustituir todos sus clientes de cifrado V1 existentes por clientes de cifrado V2. Un cliente de cifrado V2 siempre podrá leer cualquier objeto que haya escrito un cliente de cifrado V1 siempre y cuando usted lo permita configurando el cliente de cifrado V2 para que utilice `. AuthenticatedEncryption `cryptoMode`

La creación de un nuevo cliente de cifrado V2 es muy similar a la creación de un cliente de cifrado V1. Sin embargo, hay algunas diferencias:

- Utilizará un objeto `CryptoConfigurationV2` para configurar el cliente en lugar de un objeto `CryptoConfiguration`. Este parámetro es obligatorio.
- La configuración `cryptoMode` predeterminada para el cliente de cifrado V2 es `StrictAuthenticatedEncryption`. Para el cliente de cifrado V1 es`EncryptionOnly`.
- Se ha cambiado el nombre del método withEncryptionMaterials() del generador del cliente de cifrado a withEncryptionMaterialsProvider (). Se trata simplemente de un cambio estético que refleja con mayor precisión el tipo de argumento. Debe utilizar el nuevo método al configurar el cliente de servicio.

> **Note**
>
> Al descifrar con AES-GCM, lea todo el objeto hasta el final antes de empezar a utilizar los datos descifrados. Esto se hace para verificar que el objeto no se ha modificado desde que se cifró.

Utilizar proveedores de materiales de cifrado

Puede seguir utilizando los mismos proveedores de materiales de cifrado y los mismos objetos de materiales de cifrado que ya utiliza con el cliente de cifrado V1. Estas clases son responsables de

proporcionar las claves que el cliente de cifrado utiliza para proteger sus datos. Se pueden usar indistintamente con el cliente de cifrado V2 y V1.

Configurar el cliente de cifrado V2

El cliente de cifrado V2 se configura con un objeto `CryptoConfigurationV2`. Este objeto se puede crear llamando a su constructor predeterminado y, a continuación, modificando sus propiedades según sea necesario a partir de los valores predeterminados.

Los valores predeterminados para `CryptoConfigurationV2` son:

- `cryptoMode` = `CryptoMode.StrictAuthenticatedEncryption`
- `storageMode` = `CryptoStorageMode.ObjectMetadata`
- `secureRandom` = instancia de `SecureRandom`
- `rangeGetMode` = `CryptoRangeGetMode.DISABLED`
- `unsafeUndecryptableObjectPassthrough` = `false`

Tenga en cuenta que no EncryptionOnlyes compatible con `cryptoMode` el cliente de cifrado V2. El cliente de cifrado V2 siempre cifra el contenido mediante un cifrado autenticado y protege las claves de cifrado de contenido (CEK) mediante objetos `KeyWrap` V2.

En el siguiente ejemplo, se muestra cómo especificar la configuración criptográfica en la versión 1 y cómo crear una instancia de un objeto de la versión 2 para CryptoConfigurationpasarlo al generador de clientes de cifrado de la versión 2.

V1

```
CryptoConfiguration cryptoConfiguration = new CryptoConfiguration()
        .withCryptoMode(CryptoMode.StrictAuthenticatedEncryption);
```

V2

```
CryptoConfigurationV2 cryptoConfiguration = new CryptoConfigurationV2()
        .withCryptoMode(CryptoMode.StrictAuthenticatedEncryption);
```

Ejemplos adicionales

Los ejemplos siguientes muestran cómo abordar casos prácticos específicos relacionados con la migración de la V1 a la V2.

Configurar un cliente de servicio para leer los objetos creados por el cliente de cifrado V1

Para leer objetos que se escribieron anteriormente con un cliente de cifrado V1, defina `cryptoMode` como `AuthenticatedEncryption`. El siguiente fragmento de código muestra cómo crear un objeto de configuración con esta configuración.

```
CryptoConfigurationV2 cryptoConfiguration = new CryptoConfigurationV2()
        .withCryptoMode(CryptoMode.AuthenticatedEncryption);
```

Configurar un cliente de servicio para obtener rangos de bytes de objetos

Para poder `get` un rango de bytes de un objeto S3 cifrado, habilite el nuevo ajuste de configuración `rangeGetMode`. Esta configuración está deshabilitada en el cliente de cifrado V2 de forma predeterminada. Tenga en cuenta que, aunque esté activado, un `get` con rango solo funciona en objetos que se hayan cifrado mediante algoritmos compatibles con la configuración `cryptoMode` del cliente. Para obtener más información, consulta la referencia CryptoRangeGetModede la AWS SDK for Java API.

Si planea utilizar el Amazon S3 TransferManager para realizar descargas multiparte de Amazon S3 objetos cifrados mediante el cliente de cifrado V2, primero debe habilitar la `rangeGetMode` configuración en el cliente de cifrado V2.

El siguiente fragmento de código muestra cómo configurar el cliente V2 para efectuar un `get` con rango.

```
// 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);
```

Clave OpenPGP para AWS SDK for Java

Todos los artefactos de Maven disponibles públicamente para AWS SDK for Java están firmados con el estándar OpenPGP. La clave pública que necesita para verificar la firma de un artefacto está disponible en la siguiente sección.

Clave actual

En la siguiente tabla se muestra la información clave de OpenPGP para las versiones actuales del SDK para Java 1x y del SDK para Java 2.x.

ID de clave	0xAC107B386692DADD
Tipo	RSA
Tamaño	4096/4096
Creado	30-06-2016
Expires	2024-10-08
ID de usuario	SDK de AWS y herramientas < aws-dr-tools@amazon.com >
Huella digital clave	FEB9 209F 2F2F 3F46 6484 1E55 AC10 7B38 6692 DADD

Para copiar la siguiente clave pública de OpenPGP para el SDK para Java en el portapapeles, seleccione el icono “Copiar” en la esquina superior derecha.

```
-----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-----
```

Historial de documentos

En este tema se describen los cambios importantes en la Guía para desarrolladores de AWS SDK for Java a lo largo de su historia.

Esta documentación se creó el: 6 de diciembre de 2023

12 de enero de 2024

Añadir banner que anuncia el fin del soporte para AWS SDK for Java v1.x.

6 de diciembre de 2023

- Proporcionar la clave OpenPGP actual.

14 de marzo de 2023

- Se ha actualizado la guía para implementar las prácticas recomendadas de IAM. Para obtener más información, consulte prácticas recomendadas de seguridad en IAM.

28 de julio de 2022

- La plataforma EC2-Classic se retirará el 15 de agosto de 2022.

22 de marzo de 2018

- Se eliminó el ejemplo de administración de sesiones de Tomcat en DynamoDB porque dicha herramienta ya no es compatible.

2 de noviembre de 2017

- Se han añadido ejemplos de criptografía para el cliente de cifrado Amazon S3, incluyendo nuevos temas: Utilizar el cifrado del cliente Amazon S3 y el cifrado del cliente Amazon S3 con claves gestionadas por KMS AWS y el cifrado del cliente Amazon S3 con claves maestras de cliente.

14 de abril de 2017

- Se han realizado varias actualizaciones en la sección Ejemplos de Amazon S3 utilizando el AWS SDK for Java, incluyendo nuevos temas: Administrar los permisos de acceso a Amazon S3 para buckets y objetos y Configurar un bucket Amazon S3 como sitio web.

04 de abril de 2017

- Un nuevo tema, Habilitación de métricas para el AWS SDK for Java, describe cómo generar métricas de aplicación y de rendimiento de AWS SDK for Java.

03 de abril de 2017

- Se han añadido nuevos ejemplos de CloudWatch a la sección Ejemplos de CloudWatch utilizando el AWS SDK for Java: Obtener métricas de CloudWatch, Publicar datos de métricas personalizadas, Trabajar con alarmas de CloudWatch, Utilizar acciones de alarma en CloudWatch y Enviar eventos a CloudWatch

27 de marzo de 2017

- Añadidos más ejemplos de Amazon EC2 a la sección Ejemplos de uso de la Amazon EC2: Administrar instancias Amazon EC2, Utilizar direcciones IP elásticas en Amazon EC2, Utilizar regiones y zonas de disponibilidad, Trabajar con pares de claves Amazon EC2 y Trabajar con grupos de seguridad en Amazon EC2.

21 de marzo de 2017

- Se ha añadido un nuevo conjunto de ejemplos de IAM a la sección Ejemplos de IAM utilizando el AWS SDK for Java: Administrar claves de acceso IAM, Administrar usuarios de IAM, Utilizar alias de cuentas de IAM, Trabajar con políticas de IAM y Trabajar con certificados de servidor IAM

13 de marzo de 2017

- Añadidos tres nuevos temas a la sección Amazon SQS: Activación del Sondeo Largo para las Colas de Mensajes Amazon SQS, Configuración del Tiempo de Espera de Visibilidad en Amazon SQS, y Utilización de las colas de mensajes fallidos en Amazon SQS.

26 de enero de 2017

- Se ha añadido un nuevo tema Amazon S3, Uso de TransferManager para las operaciones Amazon S3, y un nuevo tema Prácticas recomendadas de desarrollo de AWS con el AWS SDK for Java en la sección Uso del AWS SDK for Java.

16 de enero de 2017

- Se añadió un nuevo tema de Amazon S3, Administración del acceso a buckets de Amazon S3 mediante políticas de buckets, y dos nuevos temas de Amazon SQS, Trabajar con colas de mensajes de Amazon SQS y Envío, recepción y eliminación de mensajes de Amazon SQS.

16 de diciembre de 2016

- Se añadieron nuevos temas de ejemplo de DynamoDB: Uso de tablas en DynamoDB y Trabajar con elementos en DynamoDB.

26 de septiembre de 2016

- Los temas de la sección Avanzada se movieron a Uso del AWS SDK for Java, ya que son fundamentales para usar el SDK.

25 de agosto de 2016

- Se añadió un nuevo tema, Creación de clientes de servicio, a Uso del AWS SDK for Java, que muestra cómo usar creadores de clientes para simplificar la creación de clientes de Servicio de AWS.

  La sección Ejemplos de código de AWS SDK for Java se actualizó con nuevos ejemplos de S3 respaldados por un repositorio en GitHub que contiene el código de ejemplo completo.

02 de mayo de 2016

- Se añadió un nuevo tema, Programación asíncrona, a la sección Uso del AWS SDK for Java, que describe cómo trabajar con métodos de cliente asíncronos que devuelven objetos `Future` o que toman un objeto `AsyncHandler`.

26 de abril de 2016

- Se eliminó el tema Requisitos de certificados de SSL, ya que no es relevante. El soporte para los certificados firmados por SHA-1 se retiró en 2015 y se eliminó el sitio que alojaba los scripts de prueba.

14 de marzo de 2016

- Se añadió un nuevo tema a la sección de Amazon SWF, Tareas Lambda, que describe cómo implementar un flujo de trabajo de Amazon SWF que llama a funciones Lambda como tareas como una alternativa a usar actividades de Amazon SWF tradicionales.

04 de marzo de 2016

- La sección Ejemplos de Amazon SWF usando el AWS SDK for Java se actualizó con nuevo contenido:
  - Conceptos básicos de Amazon SWF: ofrece información básica acerca de cómo incluir SWF en sus proyectos.
  - Creación de una aplicación de Amazon SWF sencilla: un nuevo tutorial que proporciona instrucciones paso a paso para los desarrolladores de Java que no tienen experiencia con Amazon SWF.
  - Cerrar correctamente los procesos de trabajo de actividad y flujo de trabajo: describe cómo cerrar correctamente las clases de procesos de trabajo de Amazon SWF mediante clases de simultaneidad de Java.

23 de febrero de 2016

- La fuente para la Guía para el desarrollador de AWS SDK for Java se ha trasladado a aws-java-developer-guide.

28 de diciembre de 2015

- Configuración del TTL de JVM para búsquedas de nombres DNS se ha movido de la sección Avanzada a Uso de AWS SDK for Java, y se ha vuelto a escribir para mayor claridad.

  Uso del SDK con Apache Maven se actualizó con información sobre cómo incluir la lista de materiales (BOM) del SDK en un proyecto.

04 de agosto de 2015

- Requisitos de certificados SSL es un nuevo tema de la sección Introducción que describe la migración de AWS a certificados firmados con SHA256 para las conexiones SSL y cómo corregir los entornos de Java anteriores a la versión 1.6 para usar estos certificados, que son necesarios para el acceso de AWS desde el 30 de septiembre de 2015.

  > **Note**
  >
  > Java 1.7+ ya es capaz de trabajar con certificados firmados con SHA256.

14 de mayo de 2014

- El material de Introducción y Primeros pasos se ha revisado en su totalidad para adaptarlo a la nueva estructura de las guías y ahora incluye instrucciones sobre cómo Configurar credenciales y regiones de AWS para desarrollo.

  La descripción de ejemplos de código se ha movido a su propio tema en la sección Documentación y recursos adicionales.

  La información sobre cómo consultar el historial de revisiones del SDK se ha movido a la introducción.

9 de mayo de 2014

- La estructura general de la documentación de AWS SDK for Java se ha simplificado, y los temas Introducción y Documentos y recursos adicionales se han actualizado.

  Se han añadido nuevos temas:

  - Trabajo con de credenciales de AWS: describe las distintas maneras en las que puede especificar credenciales para su uso con AWS SDK for Java.
  - Uso de roles de IAM para conceder acceso a los recursos de AWS en Amazon EC2: proporciona información sobre cómo especificar credenciales de forma segura para las aplicaciones que se ejecutan en instancias EC2.

9 de septiembre de 2013

- Este tema, Historial de documentos, hace un seguimiento de los cambios en la Guía para desarrolladores de AWS SDK for Java. Su uso previsto es acompañar el historial de notas de la versión.