버전 1.x용 개발자 가이드

AWS SDK for Java 1.x

AWS SDK for Java 1.x: 버전 1.x용 개발자 가이드

Table of Contents

곧 출시될 end-of-support AWS SDK for Java (v1) 버전을 발표했습니다. AWS SDK for Java V2로 마이그레이션하실 것을 권장합니다. 마이그레이션 날짜, 추가 세부 정보 및 방법에 대한 자세한 내용은 링크된 공지 사항을 참조하세요.

기계 번역으로 제공되는 번역입니다. 제공된 번역과 원본 영어의 내용이 상충하는 경우에는 영어 버전이 우선합니다.

개발자 안내서 - AWS SDK for Java 1.x

AWS SDK for Java에서는 AWS 서비스용 Java API를 제공합니다. SDK를 사용하면 Amazon S3, Amazon EC2, DynamoDB 등에서 작동하는 Java 애플리케이션을 쉽게 빌드할 수 있습니다. Amazon에서는 새 서비스에 대한 지원을 AWS SDK for Java에 정기적으로 추가하고 있습니다. 각 SDK 릴리스에 포함된 지원되는 서비스 및 해당 API 버전 목록은 사용하려는 버전의 릴리스 정보를 참조하십시오.

SDK 버전 2 출시됨

https://github.com/aws/aws-sdk-java-v2/에서 새로운 AWS SDK for Java 2.x를 살펴보세요. 이 버전에는 HTTP 구현에 연결하는 방법 등 오랫동안 기다려 왔던 여러 기능이 포함되어 있습니다. 시작하려면 AWS SDK for Java 2.x 개발자 안내서를 참조하세요.

추가 설명서 및 리소스

이 설명서 외에도, 다음은 AWS SDK for Java 개발자를 위한 귀중한 온라인 리소스입니다.

- AWS SDK for Java API 참조
- Java 개발자 블로그
- Java 개발자 포럼
- GitHub:
  - 설명서 소스
  - 설명서 문제
  - SDK 소스
  - SDK 문제
  - SDK 샘플
  - Gitter 채널
- AWS 코드 샘플 카탈로그은
- @awsforjava(Twitter)
- 출시 정보

Eclipse IDE 지원

Eclipse IDE를 사용하여 코드를 개발하는 경우 AWS Toolkit for Eclipse를 사용하여 AWS SDK for Java를 기존 Eclipse 프로젝트에 추가하거나 새 AWS SDK for Java 프로젝트를 생성할 수 있습니다. 이 도구 키트는 Lambda 함수 생성 및 업로드, Amazon EC2 인스턴스 시작 및 모니터링, IAM 사용자 및 보안 그룹, AWS CloudFormation 템플릿 편집기 등의 관리도 지원합니다.

전체 설명서는 AWS Toolkit for Eclipse사용자 안내서를 참조하세요.

Android용 애플리케이션 개발

Android 개발자인 경우 Amazon Web Services에서는 Android 개발용으로 특화된 Amplify Android(AWS Android용 모바일 SDK)라는 SDK를 발표했습니다.

SDK의 개정 기록 보기

SDK 버전별 변경 내용 및 지원되는 서비스를 비롯하여 AWS SDK for Java의 릴리스 기록을 보려면 SDK의 릴리스 정보를 참조하십시오.

이전 SDK 버전용 Java 참조 설명서 빌드

AWS SDK for Java API 참조에서는 SDK 버전 1.x의 최신 빌드를 다룹니다. 1.x 버전의 이전 빌드를 사용 중인 경우, 사용 중인 버전에 맞는 SDK 참조 설명서를 확인해야 할 수 있습니다.

설명서를 빌드하는 가장 쉬운 방법은 Apache의 Maven 빌드 도구를 사용하는 것입니다. 시스템에 아직 설치하지 않은 경우 먼저 Maven을 다운로드하여 설치하고 나서, 다음 지침에 따라 참조 설명서를 빌드합니다.

1. GitHub SDK 리포지토리의 releases 페이지에서 사용 중인 SDK 버전을 찾아서 선택합니다.
2. `zip`(Windows를 포함한 대부분의 플랫폼) 또는 `tar.gz`(Linux, macOS 또는 Unix) 링크를 선택하여 SDK를 컴퓨터로 다운로드합니다.
3. 로컬 디렉터리에 아카이브의 압축을 풉니다.
4. 명령줄에서 아카이브의 압축을 푼 디렉터리로 이동하여 다음을 입력합니다.

```
mvn javadoc:javadoc
```

5. 빌드가 완료된 후에는 `aws-java-sdk/target/site/apidocs/` 디렉터리에서 생성된 HTML 설명서를 볼 수 있습니다.

시작하기

이 단원에서는 AWS SDK for Java를 설치, 설정 및 사용하는 방법에 대한 정보를 제공합니다.

주제

- AWS 서비스를 사용하기 위한 기본 설정
- 획득 방법 AWS SDK for Java
- 빌드 도구 사용
- 개발을 위한 AWS 임시 자격 증명 및 AWS 리전 설정

AWS 서비스를 사용하기 위한 기본 설정

개요

AWS SDK for Java를 사용하여AWS 서비스에 액세스하는 애플리케이션을 성공적으로 개발하려면 다음 조건이 필요합니다.

- AWS IAM Identity Center에서 사용할 수 있는 AWS 액세스 포털에 로그인할 수 있어야 합니다.
- SDK에 구성된 IAM 역할의 권한은 애플리케이션에 필요한 AWS 서비스 액세스를 허용해야 합니다. PowerUserAccess AWS 관리형 정책과 관련된 권한은 대부분의 개발 요구 사항에 충분합니다.
- 다음 요소가 포함된 개발 환경
  - 다음과 같은 방식으로 설정된 공유 구성 파일:
    - 이 `config` 파일에는 AWS 리전를 지정하는 기본 프로필이 들어 있습니다.
    - `credentials` 파일에는 기본 프로필의 일부인 임시 자격 증명이 들어 있습니다.
  - Java의 적절한 설치.
  - Maven 또는 Gradle과 같은 빌드 자동화 도구.
  - 코드 작업을 위한 텍스트 편집기.
  - (선택 사항이지만 권장됨) IntelliJ IDEA, Eclipse 또는 NetBeans와 같은 IDE(통합 개발 환경).

    IDE를 사용하는 경우 AWS Toolkit를 통합하여 보다 쉽게 AWS 서비스 작업을 할 수 있습니다. AWS Toolkit for IntelliJ 및 AWS Toolkit for Eclipse는 Java 개발에 사용할 수 있는 두 가지 툴킷입니다.

> **Important**
>
> 이 설정 섹션의 지침에서는 사용자 또는 조직이 IAM Identity Center를 사용한다고 가정합니다. 조직에서 IAM Identity Center와 독립적으로 작동하는 외부 ID 공급자를 사용하는 경우 Java용 SDK에 사용할 임시 자격 증명을 얻는 방법을 알아보세요. 이 지침에 따라 `~/.aws/credentials` 파일에 임시 자격 증명을 추가하세요.
> ID 제공자가 임시 자격 증명을 `~/.aws/credentials` 파일에 자동으로 추가하는 경우 SDK 또는 AWS CLI 에 프로필 이름을 제공할 필요가 없도록 프로필 이름을 `[default]`으로 지정해야 합니다.

AWS 액세스 포털에 로그인

AWS 액세스 포털은 IAM Identity Center에 수동으로 로그인하는 웹 위치입니다. URL 형식은 `d-xxxxxxxxxx.awsapps.com/start` 또는 *`your_subdomain`*`.awsapps.com/start`입니다.

AWS액세스 포털에 익숙하지 않은 경우 AWS SDK 및 도구 참조 가이드의 IAM Identity Center 인증 항목 1단계에 있는 계정 액세스 지침을 따르십시오. 2단계에서 설명하는 AWS SDK for Java 1.x는 SDK의 자동 토큰 새로 고침 및 임시 자격 증명 자동 검색을 지원하지 않으므로 2단계를 따르지 마십시오.

공유 구성 파일 설정

공유 구성 파일은 개발 워크스테이션에 있으며 모든 AWS SDK 및 AWS Command Line Interface(CLI)에서 사용하는 기본 설정을 포함합니다. 공유 구성 파일에는 여러 설정이 포함될 수 있지만 이 지침에서는 SDK를 사용하는 데 필요한 기본 요소를 설정합니다.

공유 **`config`** 파일 설정

다음 예제에서는 `config` 파일의 내용을 보여 줍니다.

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

개발 목적으로는 코드를 실행하려는 위치와 가장 가까운 AWS 리전을 사용하세요. `config` 파일에 사용할 지역 코드 목록은 Amazon Web Services 일반 참조 가이드를 참조하세요. 출력 형식에 대한 `json` 설정은 가능한 여러 값 중 하나입니다.

이 섹션의 지침에 따라 `config` 파일을 생성하세요.

SDK용 임시 보안 인증을 설정합니다.

AWS 액세스 포털을 통해 AWS 계정 및 IAM 역할에 액세스한 후에는 SDK가 액세스할 수 있는 임시 자격 증명으로 개발 환경을 구성하세요.

임시 자격 증명으로 로컬 **credentials** 파일을 설정하는 단계

1. 공유 credentials 파일 생성.
2. credentials 파일에 동작하는 임시 보안 인증을 붙여 넣을 때까지 다음의 자리 표시자 텍스트를 붙여 넣습니다.

```
[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. 파일을 저장합니다. 이제 파일 ~/.aws/credentials이 로컬 개발 시스템에 존재해야 합니다. 이 파일에는 이름이 지정된 특정 프로필이 지정되지 않은 경우 Java용 SDK에서 사용하는 [기본] 프로필이 들어 있습니다.
4. AWS 액세스 포털에 로그인합니다.
5. AWS 액세스 포털에서 IAM 역할 자격 증명을 복사하려면 수동 자격 증명 새로 고침 제목 아래의 다음 지침을 따르세요.
   a. 링크된 지침의 4단계에서 개발 요건에 액세스 권한을 부여하는 IAM 역할 이름을 선택합니다. 이 역할은 일반적으로 PowerUserAccess 또는 Developer와 같은 이름을 갖습니다.
   b. 7단계에서 AWS 보안 인증 파일에 수동으로 프로필 추가 옵션을 선택하고 내용을 복사합니다.
6. 복사한 자격 증명을 로컬 credentials 파일에 붙여넣고 붙여넣은 프로필 이름을 모두 제거합니다. 파일은 다음과 같을 것입니다.

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

7. credentials 파일을 저장합니다.

Java용 SDK는 서비스 클라이언트를 생성할 때 이러한 임시 보안 인증에 액세스하여 각 요청에 사용합니다. 5a단계에서 선택한 IAM 역할 설정에 따라 임시 보안 인증의 유효 기간이 결정됩니다. 최대 유효 기간은 12시간입니다.

임시 보안 인증이 만료되면 4~7단계를 반복합니다.

Java 개발 환경을 설치합니다.

AWS SDK for Java에는 J2SE Development Kit 6.0 이상이 필요합니다. http://www.oracle.com/technetwork/java/javase/downloads/에서 최신 Java 소프트웨어를 다운로드할 수 있습니다.

> **Important**
>
> Java 버전 1.6(JS2E 6.0)에서는 SHA256 서명 SSL 인증서에 대한 지원이 기본적으로 제공되지 않으며, 2015년 9월 30일부터는 AWS와의 모든 HTTPS 연결 시 이 인증서가 필요합니다. Java 버전 1.7 이상에는 업데이트된 인증서가 패키지에 포함되어 있으므로 이 문제가 적용되지 않습니다.

JVM 선택

AWS SDK for Java를 사용해 서버 기반 애플리케이션의 성능을 최적화하려면 64비트 버전의 JVM(Java Virtual Machine)을 사용하는 것이 좋습니다. 이 JVM은 실행 시간에 `-Client` 옵션을 지정하더라도 서버 모드로만 실행됩니다.

런타임에서 `-Server` 옵션을 지정하고 32비트 버전의 JVM을 사용할 경우 64비트 JVM에 필적하는 성능이 발휘됩니다.

획득 방법 AWS SDK for Java

필수 조건

AWS SDK for Java를 사용하려면 다음이 있어야 합니다.

- AWS IAM Identity Center에서 사용할 수 있는 AWS 액세스 포털에 로그인할 수 있어야 합니다.
- Java의 적절한 설치.
- 로컬 공유 `credentials` 파일에 설정된 임시 자격 증명.

Java용 SDK를 사용하도록 설정하는 방법에 대한 지침은 the section called “기본 설정” 항목을 참조하세요.

빌드 도구를 사용하여 SDK for Java의 종속성을 관리합니다 (권장).

Java용 SDK의 필수 종속 항목에 액세스하려면 프로젝트에 Apache Maven 또는 Gradle을 사용하는 것이 좋습니다. 이 단원에서는 이러한 도구를 사용하는 방법을 설명합니다.

SDK 다운로드 및 추출(권장하지 않음)

빌드 도구를 사용하여 프로젝트의 SDK에 액세스하는 것이 좋습니다. 하지만 최신 버전 SDK의 미리 빌드된 jar를 다운로드할 수 있습니다.

> **Note**
>
> SDK 이전 버전 다운로드 및 빌드 방법에 대한 자세한 내용은 SDK 이전 버전 설치를 참조하세요.

1. https://.amazonwebservices.com/latest/ .zip에서 SDK를 다운로드하십시오. sdk-for-java aws-java-sdk
2. SDK를 다운로드한 후에는 로컬 디렉터리로 콘텐츠의 압축을 풉니다.

SDK에는 다음 디렉터리가 포함됩니다.

- `documentation`에는 API 설명서가 들어 있습니다(웹에서도 다운로드 가능: AWS SDK for Java API 참조).
- `lib`는 SDK `.jar` 파일을 포함합니다.
- `samples`는 SDK 사용 방법을 보여주는 작업 샘플 코드를 포함합니다.
- `third-party/lib`는 Apache commons logging, AspectJ 및 Spring 프레임워크 같이 SDK에서 사용되는 타사 라이브러리를 포함합니다.

SDK를 사용하려면 `lib` 및 `third-party` 디렉터리의 전체 경로를 빌드 파일의 종속성에 추가하고 코드를 실행할 java `CLASSPATH`에 이러한 종속성을 추가합니다.

소스에서 이전 버전의 SDK를 빌드(권장하지 않음)

전체 SDK의 최신 버전만 다운로드 가능한 jar로 사전 빌드된 형식으로 제공됩니다. 하지만 Apache Maven(오픈 소스)을 사용하여 SDK의 이전 버전을 빌드할 수 있습니다. Maven에서는 한 번에 필요한 모든 종속성을 다운로드하고 SDK를 빌드 및 설치합니다. 설치 지침과 자세한 내용은 http://maven.apache.org/를 참조하세요.

1. SDK GitHub 페이지 () 로 이동하십시오 AWS SDK for Java . GitHub
2. 원하는 SDK 버전 번호에 해당한 태그를 선택합니다. 예를 들어 `1.6.10`입니다.
3. [Download ZIP] 버튼을 클릭하여 선택한 SDK 버전을 다운로드합니다.
4. 개발 시스템의 디렉터리로 파일의 압축을 풉니다. 여러 시스템에서 그래픽 파일 관리자를 사용하여 이 작업을 수행하거나 터미널 창에서 `unzip` 유틸리티를 사용할 수 있습니다.
5. 터미널 창에서 SDK 소스의 압축을 푼 디렉터리로 이동합니다.
6. 다음 명령을 사용하여 SDK를 빌드하고 설치합니다(Maven 필요).

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

   그러면 `.jar` 파일이 `target` 디렉터리로 빌드됩니다.
7. (선택 사항) 다음 명령을 사용하여 API 참조 설명서를 빌드합니다.

   ```
   mvn javadoc:javadoc
   ```

   이 설명서는 `target/site/apidocs/` 디렉터리로 빌드됩니다.

빌드 도구 사용

빌드 도구를 사용하면 Java 프로젝트 개발을 관리하는 데 도움이 됩니다. 여러 빌드 도구를 사용할 수 있지만, 여기서는 두 가지 인기 있는 빌드 도구인 Maven과 Gradle을 사용하여 시작하고 실행하는 방법을 보여줍니다. 이 주제에서는 이러한 빌드 도구를 사용하여 프로젝트에 필요한 Java용 SDK 종속성을 관리하는 방법을 보여줍니다.

주제

- Apache Maven으로 SDK 사용하기
- Gradle로 SDK 사용하기

Apache Maven으로 SDK 사용하기

Apache Maven을 사용하여 AWS SDK for Java 프로젝트를 구성 및 빌드하거나 SDK 자체를 빌드할 수 있습니다.

> **Note**
>
> 이 주제의 지침을 사용하려면 Maven이 설치되어 있어야 합니다. 아직 설치하지 않은 경우 http://maven.apache.org/에서 다운로드하여 설치하십시오.

새 Maven 패키지 생성

기본 Maven 패키지를 생성하려면 터미널(명령줄) 창을 열어서 다음을 실행합니다.

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

org.example.basicapp을 애플리케이션의 전체 패키지 네임스페이스로 바꾸고, myapp을 프로젝트의 이름(이 이름이 프로젝트용 디렉터리의 이름으로 사용됨)으로 바꿉니다.

기본적으로 quickstart 아키타입을 사용하여 프로젝트 템플릿을 생성하는데, 이는 여러 프로젝트의 좋은 출발점입니다. 더 많은 아키타입을 사용할 수 있습니다. 함께 패키지로 제공되는 아키타입 목록을 보려면 Maven 아키타입 페이지를 방문하세요. `-DarchetypeArtifactId` 인수를 `archetype:generate` 명령에 추가하여 사용할 특정 아키타이프를 선택할 수 있습니다. 예:

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

> **Note**
>
> 프로젝트 생성 및 구성에 대한 더 많은 자세한 내용은 Maven 시작 안내서에 나와 있습니다.

SDK를 Maven 종속성으로 구성

AWS SDK for Java를 프로젝트에서 사용하려면 프로젝트의 `pom.xml` 파일에서 종속성으로 선언해야 합니다. 버전 1.9.0부터는 개별 구성 요소 또는 전체 SDK를 가져올 수 있습니다.

개별 SDK 모듈 지정

개별 SDK 모듈을 선택하려면 Maven용 AWS SDK for Java BOM을 사용합니다. 그러면 지정하는 모듈에서 동일 버전의 SDK가 사용되고 모듈이 서로 간에 호환됩니다.

BOM을 사용하려면 `<dependencyManagement>` 섹션을 애플리케이션의 `pom.xml` 파일에 추가하여 `aws-java-sdk-bom`을 종속성으로 추가하고 사용할 SDK의 버전을 지정합니다.

```
<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>
```

Maven Central에서 사용할 수 있는 AWS SDK for Java BOM의 최신 버전을 보려면 https://mvnrepository.com/artifact/com.amazonaws/aws-java-sdk-bom을 참조하십시오. 또한 이 페이지를 사용하여 BOM에 의해 관리되며 프로젝트의 `pom.xml` 파일의 `<dependencies>` 섹션에 포함할 수 있는 모듈(종속성)을 확인할 수도 있습니다.

이제 애플리케이션에서 사용하는 SDK에서 개별 모듈을 선택할 수 있습니다. 이미 BOM에 SDK 버전을 선언했으므로 각 구성 요소마다 버전 번호를 지정할 필요가 없습니다.

```
<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>
```

또한 AWS 코드 샘플 카탈로그 를 참조하여 특정 AWS 서비스에 어떤 종속성을 사용해야 하는지 알아볼 수 있습니다. 특정 서비스 예제의 POM 파일을 참조하십시오. 예를 들어, AWS S3 서비스의 종속성에 관심이 있는 경우 GitHub의 전체 예제를 참조하세요. (/java/example_code/s3 아래의 POM을 확인하십시오).

모든 SDK 모듈 가져오기

전체 SDK를 종속성으로 끌어오려면 BOM 메서드를 사용하지 않고 단순히 다음과 같이 `pom.xml`에서 해당 SDK를 선언하면 됩니다.

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

프로젝트 빌드

프로젝트를 설정했으면 Maven의 `package` 명령을 사용하여 빌드할 수 있습니다.

```
mvn package
```

이렇게 하면 `0jar` 디렉터리에 `target` 파일이 생성됩니다.

Maven을 사용하여 SDK 빌드

Apache Maven을 사용하여 소스에서 SDK를 빌드할 수 있습니다. 이렇게 하려면 GitHub에서 SDK 코드를 다운로드한 다음, 로컬로 압축을 풀고 나서 다음 Maven 명령을 실행합니다.

```
mvn clean install
```

Gradle로 SDK 사용하기

Gradle 프로젝트에 대한 SDK 종속성을 관리하려면 AWS SDK for Java용 Maven BOM을 애플리케이션의 `build.gradle` 파일로 가져옵니다.

> **Note**
>
> 다음 예제에서는 빌드 파일의 *1.12.529*를 AWS SDK for Java의 유효한 버전으로 바꿉니다. Maven 중앙 리포지토리에서 최신 버전을 찾아보세요.

Gradle 4.6 이상을 위한 프로젝트 설정

Gradle 4.6 이후 BOM(Bill Of Material)에 종속성을 선언하면 Gradle의 향상된 POM 지원 기능을 사용하여 BOM 파일을 가져올 수 있습니다.

1. Gradle 5.0 이상을 사용하는 경우 2단계로 건너뜁니다. 그렇지 않으면 `settings.gradle` 파일에서 IMPROVED_POM_SUPPORT 기능을 활성화하세요.

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

2. 애플리케이션의 `build.gradle` 파일에서 종속성 섹션에 BOM을 추가합니다.

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

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

3. 종속성 섹션에 사용할 SDK 모듈을 지정합니다. 예를 들어, 다음은 Amazon Simple Storage Service(Amazon S3)에 대한 종속성을 포함합니다.

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

Gradle에서는 BOM의 정보를 사용하여 올바른 SDK 종속성 버전을 자동으로 확인합니다.

다음은 Amazon S3에 대한 종속성이 포함된 전체 `build.gradle` 파일의 예입니다.

```
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**
>
> 앞의 예에서 Amazon S3의 종속성을 프로젝트에서 사용할 AWS 서비스의 종속성으로 바꿉니다. AWS SDK for Java BOM에서 관리하는 모듈(종속성)은 Maven 중앙 저장소에 나열되어 있습니다.

4.6 이전 Gradle 버전의 프로젝트 설정

4.6 이전의 Gradle 버전에는 기본 BOM 지원이 부족합니다. 프로젝트에 대한 AWS SDK for Java 종속성을 관리하려면 Gradle에서 SDK용 Maven BOM을 가져오도록 Spring의 종속성 관리 플러그인을 사용합니다.

1. 종속성 관리 플러그인을 애플리케이션의 `build.gradle` 파일에 추가.

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

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

2. BOM을 파일의 dependencyManagement 섹션에 추가합니다.

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

3. 종속성 섹션에 사용할 SDK 모듈을 지정합니다. 예를 들어 다음은 Amazon S3에 대한 종속성을 포함합니다.

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

Gradle에서는 BOM의 정보를 사용하여 올바른 SDK 종속성 버전을 자동으로 확인합니다.

다음은 Amazon S3에 대한 종속성이 포함된 전체 `build.gradle` 파일의 예입니다.

```
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**
>
> 앞의 예에서 Amazon S3의 종속성을 프로젝트에서 사용할 AWS 서비스의 종속성으로 바꿉니다. AWS SDK for Java BOM에서 관리하는 모듈(종속성)은 Maven 중앙 저장소에 나열되어 있습니다.

BOM을 사용한 SDK 종속성 지정에 대한 자세한 내용은 Apache Maven으로 SDK 사용하기를 참조하십시오.

개발을 위한 AWS 임시 자격 증명 및 AWS 리전 설정

AWS SDK for Java를 사용하여 지원되는 서비스에 연결하려면 AWS 임시 자격 증명을 제공해야 합니다. AWS SDK 및 CLI는 공급자 체인을 사용하여 시스템 환경 변수 및 사용자 환경 변수, 로컬 AWS 구성 파일 등 다양한 위치에서 AWS 임시 자격 증명을 찾습니다.

이 항목에서는 AWS SDK for Java를 사용한 로컬 애플리케이션 개발을 위한 AWS 임시 자격 증명 설정에 대한 기본 정보를 제공합니다. EC2 인스턴스 내에서 사용할 자격 증명을 설정해야 하거나 개발용으로 Eclipse IDE를 사용하고 있는 경우에는 다음 주제를 참조하십시오.

- EC2 인스턴스를 사용하려면 IAM 역할을 사용하여 Amazon EC2의 AWS 리소스에 대한 액세스 권한 부여에서처럼 IAM 역할을 생성한 다음 해당 역할에 EC2 인스턴스 액세스 권한을 부여합니다.
- AWS Toolkit for Eclipse를 사용하여 Eclipse 내에서 AWS 자격 증명을 설정합니다. 자세한 내용은 AWS Toolkit for Eclipse 사용 설명서에서 AWS 자격 증명 설정을 참조하세요.

임시 자격 증명 설정

AWS SDK for Java에 사용할 임시 자격 증명은 여러 가지 방법으로 설정할 수 있지만, 권장 방법은 다음과 같습니다.

- 로컬 시스템의 다음 위치에 있는 AWS 자격 증명 프로필 파일에서 자격 증명을 설정합니다.
  - Linux, macOS 또는 Unix의 `~/.aws/credentials`의 경우:
  - Windows의 경우 `C:\Users\USERNAME\.aws\credentials`

  임시 자격 증명을 얻는 방법에 대한 지침은 이 안내서의 the section called “SDK용 임시 보안 인증을 설정합니다.”을 참조하세요.
- `AWS_ACCESS_KEY_ID`, `AWS_SECRET_ACCESS_KEY`, 및 `AWS_SESSION_TOKEN` 환경 변수를 설정합니다.

  Linux, macOS 또는 Unix에서 이러한 변수를 설정하려면 를 사용합니다.

  ```
  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
  ```

  Windows에서 이러한 변수를 설정하려면 을 사용합니다.

  ```
  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
  ```
- EC2 인스턴스의 경우 IAM 역할을 지정하고 나서 해당 역할에 EC2 인스턴스 액세스 권한을 부여합니다. 작동 방식에 대한 자세한 내용은 Linux 인스턴스용 Amazon EC2 사용 설명서의 Amazon EC2용 IAM 역할을 참조하세요.

이러한 방법 중 하나를 사용하여 AWS 임시 자격 증명을 설정하고 나면 기본 자격 증명 공급자 체인을 사용하여 이러한 자격 증명이 AWS SDK for Java에서 자동으로 로드됩니다. Java 애플리케이션에서 AWS 자격 증명으로 작업하는 방법에 대한 자세한 내용은 AWS로 작업하기 단원을 참조하세요.

IMDS 자격 증명 갱신

AWS SDK for Java는 자격 증명 만료 시간에 관계없이 백그라운드에서 1분마다 IMDS 자격 증명을 옵트인 갱신합니다. 이렇게 하면 자격 증명을 더 자주 갱신할 수 있으며 IMDS에 도달 실패가 인식된 AWS 가용성에 영향을 미칠 가능성을 줄일 수 있습니다.

```
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();
```

AWS 리전 설정

AWS SDK for Java에서 AWS 서비스에 액세스하는 데 사용될 기본 AWS 리전을 설정해야 합니다. 네트워크 성능을 최대화하려면 본인(또는 본인의 고객)과 지리적으로 가까운 리전을 선택해야 합니다. 각 서비스에 대한 리전 목록은 Amazon Web Services 일반 참조에서 리전 및 엔드포인트를 참조하세요.

> **Note**
>
> 리전을 선택하지 않으면 기본적으로 us-east-1이 사용됩니다.

자격 증명 설정과 비슷한 기술을 사용하여 기본 AWS 리전을 설정할 수 있습니다.

- 로컬 시스템의 다음 위치에 있는 AWS 구성 파일에서 AWS 리전을 설정합니다.
  - Linux, macOS 또는 Unix의 ~/.aws/config
  - Windows의 C:\Users\USERNAME\.aws\config

이 파일에는 다음 형식의 행이 포함되어야 합니다.

+

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

+

your_aws_region을 원하는 AWS 리전(예: “us-east-1”)으로 대체합니다.

- AWS_REGION 환경 변수를 설정합니다.

  Linux, macOS 또는 Unix에서는 을 사용하세요.

  ```
  export AWS_REGION=your_aws_region
  ```

  Windows에서는 을 사용합니다.

  ```
  set AWS_REGION=your_aws_region
  ```

  여기서 your_aws_region은 원하는 AWS 리전 이름입니다.

사용 AWS SDK for Java

이 섹션에서는 SDK와 함께 사용할 수 있는 모든 서비스에 AWS SDK for Java 적용되는 를 사용한 프로그래밍에 대한 중요한 일반 정보를 제공합니다.

서비스별 프로그래밍 정보 및 예제 (Amazon EC2, Amazon S3Amazon SWF, 등) 는 코드 예제를 참조하십시오AWS SDK for Java .

주제

를 사용한 AWS 개발 모범 사례 AWS SDK for Java

다음 모범 사례는 를 사용하여 AWS 응용 프로그램을 개발할 때 발생하는 문제나 문제를 방지하는 데 도움이 될 수 AWS SDK for Java있습니다. 모범 사례를 서비스별로 정리했습니다.

S3

피하십시오 ResetExceptions

스트림을 사용하여 (`AmazonS3`클라이언트를 통하거나`TransferManager`) 객체를 업로드할 때 네트워크 연결 또는 시간 초과 문제가 발생할 수 있습니다. Amazon S3 기본적으로 전송 시작 전에 입력 스트림을 표시한 다음 재시도하기 전에 재설정하여 전송을 재시도하면 전송이 실패했습니다. AWS SDK for Java

스트림이 표시 및 재설정을 지원하지 않는 경우 일시적 실패가 발생하고 재시도가 ResetException활성화되면 SDK에서 a를 발생시킵니다.

모범 사례

표시 및 재설정 작업을 지원하는 스트림을 사용하는 것이 좋습니다.

ResetExceptiona를 방지하는 가장 확실한 방법은 표시 및 재설정 제한의 제한 없이 처리할 AWS SDK for Java 수 있는 File FileInputStreamor를 사용하여 데이터를 제공하는 것입니다.

스트림이 a가 FileInputStream아니지만 마크 및 리셋을 지원하는 경우 의 `setReadLimit` RequestClientOptions방법을 사용하여 마크 제한을 설정할 수 있습니다. 기본값은 128KB입니다. 읽기 제한 값을 스트림 크기보다 1바이트 크게 설정하면 a를 확실하게 방지할 수 있습니다. ResetException

예를 들어 스트림의 최대 예상 크기가 100,000바이트이면 읽기 제한을 100,001(100,000 + 1)바이트로 설정합니다. 표시 및 재설정은 항상 100,000바이트 이하에 대해서만 작동합니다. 이렇게 하면 일부 스트림의 경우 바이트 수가 메모리로 버퍼링될 수도 있습니다.

서비스 클라이언트 생성

에 Amazon Web Services요청하려면 먼저 서비스 클라이언트 객체를 만들어야 합니다. 권장 방법은 서비스 클라이언트 빌더를 사용하는 것입니다.

각 AWS 서비스 인터페이스에는 서비스 API의 각 작업에 대한 메서드가 포함된 서비스 인터페이스가 있습니다. 예를 들어, DynamoDB용 서비스 인터페이스의 이름은 DbClient입니다. AmazonDynamo 각 서비스 인터페이스마다 서비스 인터페이스의 구현을 생성하는 데 사용할 수 있는 해당하는 클라이언트 빌더가 있습니다. 의 클라이언트 빌더 클래스의 이름은 DynamoDB DB입니다. AmazonDynamo ClientBuilder

클라이언트 빌더 가져오기

클라이언트 빌더의 인스턴스를 가져오려면 다음 예에서처럼 정적 팩토리 메서드인 `standard`를 사용합니다.

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

빌더가 있으면 빌더 API에서 여러 유용한 setter를 사용하여 클라이언트의 속성을 사용자 지정할 수 있습니다. 예를 들면 다음과 같이 사용자 지정 리전 및 사용자 지정 자격 증명 공급자를 설정할 수 있습니다.

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

> **Note**
>
> 유용한 `withXXX` 메서드는 `builder` 객체를 반환하므로 보다 읽기 쉽고 편리하도록 메서드 호출을 묶을 수 있습니다. 원하는 속성을 구성한 후에는 `build` 메서드를 호출하여 클라이언트를 생성할 수 있습니다. 클라이언트는 한 번 생성되고 나면 변경할 수 없으며 `setRegion` 또는 `setEndpoint` 호출이 실패합니다.

빌더는 동일한 구성을 사용하여 여러 클라이언트를 생성할 수 있습니다. 애플리케이션을 작성할 때는 빌더가 변경 가능하며 스레드 세이프가 아님에 주의해야 합니다.

다음 코드에서는 빌더가 클라이언트 인스턴스의 팩토리로 사용됩니다.

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

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

빌더는 또한 ClientConfiguration 및 RequestMetricCollector에 대한 유창한 설정자와 2개의 사용자 지정 목록을 제공합니다. RequestHandler

다음은 모든 구성 가능한 속성을 재정의하는 전체 예제입니다.

```
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();
```

비동기 클라이언트 생성

AWS SDK for Java 에는 모든 서비스 (제외) 에 대한 비동기 (또는 비동기) 클라이언트와 모든 서비스에 해당하는 비동기 클라이언트 Amazon S3빌더가 있습니다.

기본값을 사용하여 비동기 DynamoDB 클라이언트를 만들려면 ExecutorService

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

동기 (또는 동기화) 클라이언트 빌더가 지원하는 구성 옵션 외에도 비동기 클라이언트를 사용하면 사용자 ExecutorFactory지정을 설정하여 비동기 클라이언트가 사용하는 옵션을 변경할 수 있습니다. `ExecutorService` `ExecutorFactory`는 함수형 인터페이스이므로 Java 8 Lambda 표현식 및 메서드 참조와 상호 운용됩니다.

사용자 지정 executor를 사용하여 비동기 클라이언트를 생성하려면

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

사용: DefaultClient

동기 및 비동기 클라이언트 빌더에는 모두 `defaultClient`라는 또 다른 팩토리 메서드가 있습니다. 이 메서드는 기본 구성을 사용하여 서비스 클라이언트를 생성하며, 기본 공급자 체인을 사용하여 자격 증명 및 AWS 리전을 로드합니다. 자격 증명 또는 리전을 애플리케이션이 실행 중인 환경에서 확인할 수 없으면 `defaultClient` 호출이 실패합니다. AWS 자격 증명 및 지역 결정 방법에 대한 자세한 내용은 자격 증명 및 AWS 리전 선택 작업을 참조하십시오.

기본 서비스 클라이언트를 생성하려면

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

클라이언트 수명 주기

SDK의 서비스 클라이언트는 스레드 세이프이며, 성능을 최대화하려면 서비스 클라이언트를 수명이 긴 객체로 다루어야 합니다. 각 클라이언트마다 고유한 연결 풀 리소스가 있습니다. 더 이상 필요하지 않은 경우 리소스 누수를 방지하기 위해 클라이언트를 명시적으로 종료하십시오.

클라이언트를 명시적으로 종료하려면 `shutdown` 메서드를 호출합니다. `shutdown`을 호출한 후에는 모든 클라이언트 리소스가 해제되고 클라이언트를 사용할 수 없습니다.

클라이언트를 종료하려면

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

에 임시 자격 증명을 제공하십시오. AWS SDK for Java

를 Amazon Web Services요청하려면 에서 서비스를 호출할 때 사용할 AWS 임시 자격 증명을 제공해야 합니다. AWS SDK for Java 다음과 같은 방법으로 추가가 가능합니다.

- 기본 자격 증명 공급자 체인(권장)을 사용합니다.
- 특정 자격 증명 공급자 또는 공급자 체인을 사용합니다(또는 직접 생성).
- 임시 자격 증명 정보를 코드로 직접 제공하세요.

기본 자격 증명 공급자 체인 사용

인수를 제공하지 않고 새 서비스 클라이언트를 초기화하면 Default 클래스로 구현된 기본 자격 증명 공급자 체인을 사용하여 임시 자격 증명을 AWS SDK for Java 찾으려고 시도합니다. AWSCredentialsProviderChain 기본 자격 증명 공급자 체인은 다음 순서대로 자격 증명을 찾습니다.

1. 환경 변수-`AWS_ACCESS_KEY_ID`, `AWS_SECRET_ACCESS_KEY`, 및 `AWS_SESSION_TOKEN`입니다. 에서는 EnvironmentVariableCredentialsProvider클래스를 AWS SDK for Java 사용하여 이러한 자격 증명을 로드합니다.
2. Java 시스템 속성-`aws.accessKeyId`, `aws.secretKey`, 및 `aws.sessionToken`입니다. AWS SDK for Java 는 를 SystemPropertiesCredentialsProvider사용하여 이러한 자격 증명을 로드합니다.
3. 환경 또는 컨테이너의 웹 자격 증명 토큰 자격 증명

4. 기본 자격 증명 프로필 파일은 일반적으로 위치에 있으며 `~/.aws/credentials` (위치는 플랫폼마다 다를 수 있음) 여러 AWS SDK와 에서 공유합니다. AWS CLI AWS SDK for Java 에서는 를 ProfileCredentialsProvider사용하여 이러한 자격 증명을 로드합니다.

   에서 제공하는 `aws configure` 명령을 사용하여 자격 증명 파일을 만들거나 텍스트 편집기로 파일을 편집하여 자격 증명 파일을 만들 수 있습니다. AWS CLI자격 증명 파일 형식에 대한 자세한 내용은 AWS 자격 증명 파일 형식을 참조하세요.

5. Amazon ECS 컨테이너 자격 증명은 환경 변수 `AWS_CONTAINER_CREDENTIALS_RELATIVE_URI`가 설정되면 Amazon ECS에서 로드됩니다. AWS SDK for Java 에서는 를 ContainerCredentialsProvider사용하여 이러한 자격 증명을 로드합니다. 이 값에 대한 IP 주소를 지정할 수 있습니다.

6. 인스턴스 프로필 자격 증명 - EC2 인스턴스에서 사용되며 Amazon EC2 메타데이터 서비스를 통해 전달됩니다. AWS SDK for Java 는 를 InstanceProfileCredentialsProvider사용하여 이러한 자격 증명을 로드합니다. 이 값에 대한 IP 주소를 지정할 수 있습니다.

> **Note**
>
> 인스턴스 프로파일 자격 증명은 `AWS_CONTAINER_CREDENTIALS_RELATIVE_URI`가 설정되지 않은 경우에만 사용됩니다. 자세한 내용은 EC2를 ContainerCredentialsProviderWrapper 참조하십시오.

임시 자격 증명 설정

AWS 임시 자격 증명을 사용하려면 이전 위치 중 하나 이상에 임시 자격 증명을 설정해야 합니다. 자격 증명 설정에 대한 자세한 내용은 다음 주제를 참조하십시오.

- 환경이나 기본 자격 증명 프로필 파일에서 자격 증명을 지정하려면 the section called “임시 자격 증명 설정”을 참조하세요.
- Java 시스템 속성을 설정하려면 공식 Java Tutorials 웹 사이트에서 시스템 속성 자습서를 참조하세요.
- EC2 인스턴스에서 인스턴스 프로필 자격 증명을 설정하고 사용하려면 IAM 역할을 사용하여 리소스에 대한 액세스 권한 부여를 참조하십시오. AWS Amazon EC2

대체 자격 증명 프로필 설정

는 기본적으로 기본 프로필을 AWS SDK for Java 사용하지만 자격 증명 파일에서 가져온 프로필을 사용자 지정하는 방법이 있습니다.

AWS Profile 환경 변수를 사용하여 SDK에서 로드한 프로필을 변경할 수 있습니다.

예를 들어 Linux, macOS, 또는 Unix에서는 다음 명령을 실행하여 프로필을 myProfile로 변경합니다.

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

Windows에서는 다음을 사용합니다.

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

AWS_PROFILE환경 변수를 설정하면 공식적으로 지원되는 모든 AWS SDK 및 도구 (및 포함) 의 AWS CLI 자격 증명 로딩에 영향을 줍니다. AWS Tools for Windows PowerShell Java 애플리케이션용 프로필만 변경하려면 대신에 시스템 속성인 `aws.profile`을 사용할 수 있습니다.

> **Note**
>
> 이 환경 변수는 이 시스템 속성보다 우선합니다.

대체 자격 증명 파일 위치 설정

는 기본 자격 증명 파일 위치에서 AWS 임시 자격 증명을 자동으로 AWS SDK for Java 로드합니다. 하지만 자격 증명 파일의 전체 경로를 사용하여 AWS_CREDENTIAL_PROFILES_FILE 환경 변수를 설정함으로써 이 위치를 지정할 수도 있습니다.

이 기능을 사용하면 자격 증명 파일을 AWS SDK for Java 찾는 위치를 일시적으로 변경할 수 있습니다 (예: 명령줄에서 이 변수를 설정). 또는 사용자 또는 시스템 환경에서 환경 변수를 설정하여 사용자 또는 시스템 수준에서 변경할 수도 있습니다.

기본 자격 증명 파일 위치를 재정의하려면

- AWS_CREDENTIAL_PROFILES_FILE환경 변수를 AWS 자격 증명 파일의 위치로 설정합니다.
  - Linux, macOS 또는 Unix에서는 다음을 사용하세요.

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

- Windows에서는 다음을 사용합니다.

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

`Credentials` 파일 형식

이 안내서의 기본 설정에 있는 지침을 따르면 자격 증명 파일은 다음과 같은 기본 형식을 가져야 합니다.

```
[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>
```

프로필 이름은 대괄호에 지정되며(예: `[default]`), 그 뒤에는 해당 프로필의 구성 가능한 필드가 키-값 페어로 이어집니다. `credentials` 파일에 여러 프로필을 지정할 수 있으며, `aws configure --profile` *`PROFILE_NAME`* 을 사용하여 이러한 프로필을 편집하거나 추가하여 구성할 프로필을 선택할 수 있습니다.

`metadata_service_timeout` 및 `metadata_service_num_attempts`와 같은 추가 필드를 지정할 수 있습니다. CLI에서는 구성할 수 없습니다. 사용하려면 파일을 직접 편집해야 합니다. 구성 파일 및 사용 가능한 필드에 대한 자세한 내용은 사용 AWS Command Line Interface 설명서의 AWS Command Line Interface구성을 참조하십시오.

자격 증명 로드

임시 자격 증명을 설정한 후에는 SDK에서 기본 자격 증명 공급자 체인을 사용하여 로드할 수 있습니다.

이렇게 하려면 다음과 같이 빌더에 자격 증명을 명시적으로 제공하지 않고 AWS 서비스 클라이언트를 인스턴스화합니다.

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

```
                        .build();
```

자격 증명 공급자 또는 공급자 체인 지정

클라이언트 빌드를 사용하여 기본 자격 증명 공급자 체인과 다른 자격 증명 공급자를 지정할 수 있습니다.

인터페이스를 입력으로 받는 클라이언트 빌더에 자격 증명 공급자 또는 공급자 체인의 인스턴스를 제공합니다. AWSCredentialsProvider 다음 예제는 특별히 환경 자격 증명을 사용하는 방법을 보여줍니다.

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

AWS SDK for Java제공된 자격 증명 공급자 및 제공자 체인의 전체 목록은 의 알려진 모든 구현 클래스를 참조하십시오. AWSCredentialsProvider

> **Note**
>
> 이 기법을 사용하면 `AWSCredentialsProvider` 인터페이스를 구현하는 자체 자격 증명 공급자를 사용하거나 클래스를 서브클래싱하여 생성한 자격 증명 공급자 또는 공급자 체인을 제공할 수 있습니다. AWSCredentialsProviderChain

임시 자격 증명을 명시적으로 지정

기본 자격 증명 체인이나 특정/사용자 지정 공급자 또는 공급자 체인이 코드에서 작동하지 않는 경우 명시적으로 제공하는 자격 증명을 설정할 수 있습니다. 를 사용하여 임시 자격 증명을 검색한 경우 이 메서드를 사용하여 AWS STS액세스에 필요한 자격 증명을 지정하십시오. AWS

1. BasicSessionCredentials클래스를 인스턴스화하고 SDK가 연결에 사용할 AWS 액세스 키, AWS 비밀 키, AWS 세션 토큰을 제공합니다.
2. 객체를 AWSStaticCredentialsProvider사용하여 를 생성합니다. `AWSCredentials`
3. `AWSStaticCredentialsProvider`를 사용하여 클라이언트 빌더를 구성하고 클라이언트를 빌드합니다.

다음은 예입니다.

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

추가 정보

- IAM 사용자 가입 AWS 및 생성
- 개발을 위한 AWS 자격 증명 및 지역 설정
- IAM 역할을 사용하여 다음 AWS 리소스에 대한 액세스 권한 부여 Amazon EC2

AWS 리전 선택

지역을 통해 특정 지역에 물리적으로 상주하는 AWS 서비스에 액세스할 수 있습니다. 이는 중복성은 물론, 데이터와 애플리케이션을 고객과 고객의 사용자가 액세스할 위치에 가까운 곳에서 실행 상태로 유지하는 데도 유용할 수 있습니다.

리전에서 서비스 가용성 확인

특정 지역에서 특정 서비스를 사용할 수 AWS 서비스 있는지 확인하려면 사용하려는 지역의 `isServiceSupported` 방법을 사용하세요.

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

지정할 수 있는 리전에 대한 Regions 클래스 설명서를 참조하고, 쿼리할 서비스의 엔드포인트 접두사를 사용합니다. 각 서비스의 엔드포인트 접두사는 서비스 인터페이스에 정의되어 있습니다. 예를 들어 DynamoDB 엔드포인트 접두사는 AmazonDynamoDB에 정의되어 있습니다.

리전 선택

버전 1.4부터 지역 이름을 지정할 수 있으며 AWS SDK for Java, SDK는 적절한 엔드포인트를 자동으로 선택합니다. 엔드포인트를 직접 선택하려면 Choosing a Specific Endpoint 단원을 참조하십시오.

리전을 명시적으로 설정하려면 Regions 열거형을 사용하는 것이 좋습니다. 이 열거형은 공개적으로 사용 가능한 모든 리전의 열거 값입니다. 열거형의 리전을 사용하여 클라이언트를 생성하려면 다음 코드를 사용합니다.

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

사용할 리전이 Regions 열거형에 없으면 리전 이름을 나타내는 문자열을 사용하여 리전을 설정할 수 있습니다.

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

> **Note**
>
> 빌더를 사용하여 클라이언트를 빌드한 후에는 클라이언트가 변경 불가능하며 리전을 변경할 수 없습니다. 동일한 서비스를 AWS 리전 위해 여러 클라이언트를 사용하는 경우 지역당 하나씩 여러 클라이언트를 만들어야 합니다.

특정 엔드포인트 선택

AWS 클라이언트를 만들 때 withEndpointConfiguration 메서드를 호출하여 리전 내의 특정 엔드포인트를 사용하도록 각 클라이언트를 구성할 수 있습니다.

예를 들어 유럽 (아일랜드) 지역을 사용하도록 Amazon S3 클라이언트를 구성하려면 다음 코드를 사용하십시오.

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

현재 지역 목록과 모든 AWS 서비스의 해당 엔드포인트는 지역 및 엔드포인트를 참조하십시오.

환경에서 리전을 자동으로 결정

> **Important**
>
> 이 섹션은 클라이언트 빌더를 사용하여 서비스에 액세스하는 AWS 경우에만 적용됩니다. AWS 클라이언트 생성자를 사용하여 만든 클라이언트는 환경에서 지역을 자동으로 결정하지 않고 대신 기본 SDK 지역 (UseAst1) 을 사용합니다.

Amazon EC2 또는 Lambda에서 실행하는 경우 코드가 실행되는 지역과 동일한 지역을 사용하도록 클라이언트를 구성하는 것이 좋습니다. 이렇게 하면 실행 중인 환경에서 코드가 분리되므로 낮은 지연 시간과 중복성을 위해 손쉽게 애플리케이션을 여러 리전에 배포할 수 있습니다.

코드를 실행 중인 리전을 SDK에서 자동으로 검색하도록 하려면 클라이언트 빌더를 사용해야 합니다.

기본 자격 증명/리전 공급자 체인을 사용하여 환경에서 리전을 결정하려면 클라이언트 빌더의 `defaultClient` 메서드를 사용합니다.

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

이렇게 하면 `standard` 뒤에 `build`를 사용할 때와 똑같이 작동합니다.

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

`withRegion` 메서드를 사용하여 리전을 명시적으로 설정하지 않는 경우 SDK에서는 기본 리전 공급자 체인을 조회하여 사용할 리전을 시도 및 결정합니다.

기본 리전 공급자 체인

다음은 리전 조회 프로세스입니다.

1. 빌더 자체에 대해 `withRegion` 또는 `setRegion`을 사용하여 설정한 명시적 리전을 다른 어떤 것보다 우선합니다.
2. `AWS_REGION` 환경 변수를 확인합니다. 설정한 경우 클라이언트를 구성하는 데 해당 리전이 사용됩니다.

> **Note**
>
> 이 환경 변수는 컨테이너에 Lambda 의해 설정됩니다.

3. SDK는 AWS 공유 구성 파일 (일반적으로 위치~/.aws/config) 을 확인합니다. region 속성이 있으면 SDK가 이 속성을 사용합니다.
   - AWS_CONFIG_FILE 환경 변수는 공유 구성 파일의 위치를 사용자 지정하는 데 사용할 수 있습니다.
   - AWS_PROFILE 환경 변수 또는 aws.profile 시스템 속성을 사용하여 SDK에서 로드하는 프로필을 사용자 지정할 수 있습니다.
4. SDK는 Amazon EC2 인스턴스 메타데이터 서비스를 사용하여 현재 실행 중인 Amazon EC2 인스턴스의 지역을 확인하려고 합니다.
5. 이때까지도 SDK에서 여전히 리전을 찾지 못한 경우 클라이언트 생성이 실패하고 예외가 발생합니다.

AWS 애플리케이션을 개발할 때 일반적인 접근 방식은 공유 구성 파일 (기본 자격 증명 공급자 체인 사용 참조) 을 사용하여 로컬 개발을 위한 지역을 설정하고, 인프라에서 AWS 실행할 때는 기본 지역 공급자 체인을 사용하여 지역을 결정하는 것입니다. 이렇게 하면 클라이언트 생성 작업이 크게 간소화되며 애플리케이션을 이식 가능한 형태로 유지됩니다.

예외 처리

SDK를 사용하여 고품질 애플리케이션을 구축하려면 AWS SDK for Java 예외가 발생하는 방법과 시기를 이해하는 것이 중요합니다. 다음 단원에서는 SDK에서 발생하는 다양한 예외의 경우와 이러한 예외를 적절히 처리하는 방법에 대해 설명합니다.

확인되지 않은 예외가 발생하는 이유

는 다음과 같은 이유로 확인된 예외 대신 런타임 (또는 확인되지 않은) 예외를 AWS SDK for Java 사용합니다.

- 개발자가 중요하지 않은 예외 경우를 강제로 처리하지 않고 (또한 해당 코드를 상세 표시 모드로 설정하지 않고) 처리하고자 하는 오류에 대해서만 세부적으로 제어할 수 있도록 하기 위해
- 대규모 애플리케이션에서 확인된 예외 고유의 확장성 문제를 방지하기 위해

일반적으로 확인된 예외는 소규모 애플리케이션에서 잘 작동하는 편이지만, 애플리케이션이 확장되고 복잡해짐에 따라 문제가 될 수도 있습니다.

확인 및 확인되지 않은 예외의 사용에 대한 자세한 내용은 다음을 참조하십시오.

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

AmazonServiceException (및 서브클래스)

AmazonServiceException를 사용할 때 발생하는 가장 일반적인 예외입니다. AWS SDK for Java이 예외는 AWS 서비스의 오류 응답을 나타냅니다. 예를 들어 존재하지 않는 Amazon EC2 인스턴스를 종료하려고 하면 EC2에서 오류 응답을 반환하고 해당 오류 응답의 모든 세부 정보가 해당 오류 응답에 포함됩니다. `AmazonServiceException` 경우에 따라서는 개발자가 catch 블록을 통해 오류 경우 처리를 세부적으로 제어할 수 있도록 하기 위해 `AmazonServiceException`의 하위 클래스가 발생하기도 합니다.

오류가 `AmazonServiceException` 발생하면 요청이 에 성공적으로 AWS 서비스 전송되었지만 제대로 처리되지 못했다는 것을 알 수 있습니다. 이는 요청의 파라미터 오류 또는 서비스 측의 문제로 인해 발생할 수 있습니다.

`AmazonServiceException`은 다음과 같은 정보를 제공합니다.

- 반환된 HTTP 상태 코드
- 반환된 AWS 오류 코드
- 서비스의 상세 오류 메시지
- AWS 실패한 요청의 요청 ID

`AmazonServiceException`또한 실패한 요청이 발신자의 잘못 (잘못된 값을 가진 요청) 인지 아니면 호출자의 잘못 (내부 서비스 오류) AWS 서비스인지에 대한 정보도 포함됩니다.

AmazonClientException

AmazonClientException요청을 보내려고 시도하거나 응답을 구문 분석하려고 시도하는 동안 Java 클라이언트 코드 내에서 문제가 발생했음을 나타냅니다. AWS AWS일반적으로 `AmazonClientException` A는 a보다 더 심각하며`AmazonServiceException`, 클라이언트가 AWS 서비스에 서비스를 호출하지 못하게 하는 중대한 문제를 나타냅니다. 예를 들어, 클라이언트 중

하나에서 작업을 호출하려고 하면 네트워크 연결을 사용할 수 없는 AmazonClientException 경우가 AWS SDK for Java 발생합니다.

비동기 프로그래밍

동기 또는 비동기 메서드를 사용하여 서비스에 대한 작업을 호출할 수 있습니다. AWS 비동기 메서드는 클라이언트가 서비스로부터 응답을 받을 때까지 스레드의 실행을 차단합니다. 비동기 메서드는 (값을) 즉시 반환하며, 응답을 기다리지 않고 제어 권한을 호출 스레드에 넘겨줍니다.

비동기 메서드는 응답이 제공되기 전에 (값을) 반환하므로 준비되었을 때 응답을 가져올 방법이 필요합니다. 는 미래 객체와 콜백 메서드라는 두 가지 방법을 AWS SDK for Java 제공합니다.

Java Future

의 비동기 메서드는 미래의 비동기 작업 결과를 포함하는 Future 객체를 AWS SDK for Java 반환합니다.

서비스가 아직 응답 객체를 제공하지 않은 경우 Future isDone() 메서드를 호출합니다. 응답이 준비되면 Future get() 메서드를 호출하여 응답 객체를 가져올 수 있습니다. 이 메커니즘을 사용하면 애플리케이션이 계속 다른 작업을 수행하는 동안 비동기 작업의 결과를 정기적으로 폴링할 수 있습니다.

다음은 객체를 보유할 수 있는 함수를 호출하고 객체를 보유할 수 있는 Lambda 함수를 수신하는 비동기 작업의 예입니다. Future InvokeResult InvokeResult 객체는 isDone()이 true가 된 후에만 가져옵니다.

```
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);
    }
}
```

비동기 콜백

SDK를 사용하면 Java Future 객체를 사용하여 비동기 요청의 상태를 모니터링하는 것 외에도 인터페이스를 사용하는 클래스를 구현할 수 있습니다. AsyncHandler AsyncHandler요청 완료 방식에 따라 호출되는 두 가지 메서드 (및) 를 제공합니다. onSuccess onError

콜백 인터페이스 접근 방법의 주요 장점은 Future 객체를 폴링하여 요청 완료 시점을 알아낼 필요가 없다는 점입니다. 대신에 코드에서 다음 활동을 즉시 시작할 수 있으며, SDK에 의존하여 핸들러를 즉시 호출할 수 있습니다.

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

모범 사례

콜백 실행

AsyncHandler의 구현은 비동기 클라이언트가 소유하는 스레드 풀 내부에서 실행됩니다. 단기 실행 코드는 AsyncHandler 구현 내부에 적합합니다. 핸들러 메서드 내부에서 장기 실행 또는 차단 코드를 실행하면 비동기 클라이언트에 사용되는 스레드 풀에 대한 경합이 발생하여 클라이언트에서 요청을 실행할 수 없습니다. 콜백에서 시작해야 하는 장기 실행 작업이 있는 경우 콜백이 애플리케이션에 의해 관리되는 스레드 풀이나 새 스레드에서 작업을 실행하도록 하십시오.

스레드 풀 구성

의 비동기 클라이언트는 대부분의 응용 프로그램에서 작동하는 기본 스레드 풀을 AWS SDK for Java 제공합니다. 사용자 정의를 ExecutorService구현하고 이를 AWS SDK for Java 비동기 클라이언트에 전달하여 스레드 풀 관리 방식을 더 잘 제어할 수 있습니다.

예를 들어 사용자 지정을 사용하여 풀의 스레드 이름 지정 ThreadFactory방식을 제어하거나 스레드 사용에 대한 추가 정보를 기록하는 ExecutorService 구현을 제공할 수 있습니다.

비동기 액세스

SDK의 TransferManager클래스는 작업을 위한 비동기 지원을 제공합니다. Amazon S3`TransferManager`비동기 업로드 및 다운로드를 관리하고, 전송에 대한 자세한 진행 상황 보고를 제공하고, 다양한 이벤트에 대한 콜백을 지원합니다.

AWS SDK for Java 통화 로깅

AWS SDK for Java 는 런타임 시 여러 로깅 시스템 중 하나를 사용할 수 있게 해주는 추상화 계층인 Apache Commons Logging을 통해 계측됩니다.

지원되는 로깅 시스템에는 Java Logging Framework와 Apache Log4j 등이 있습니다. 이 주제에서는 Log4j 사용법을 보여줍니다. 애플리케이션 코드를 변경하지 않고 SDK의 로깅 기능을 사용할 수 있습니다.

Log4j에 대해 자세히 알아보려면 Apache 웹 사이트를 참조하십시오.

> **Note**
>
> 이 항목에서는 Log4j 1.x에 초점을 맞춰 설명합니다. Log4j2는 Apache Commons Logging을 직접 지원하지 않지만, Apache Commons Logging 인터페이스를 사용하는 Log4j2에 로깅 호출을 자동으로 전달하는 어댑터를 제공합니다. 자세한 내용은 Log4j2 설명서의 Commons Logging Bridge을 참조하십시오.

Log4J JAR 다운로드

SDK에서 Log4j를 사용하려면 Apache 웹 사이트에서 Log4j JAR를 다운로드해야 합니다. SDK에는 이 JAR이 포함되지 않습니다. JAR 파일을 classpath의 위치 중 하나로 복사합니다.

Log4j는 구성 파일인 log4j.properties를 사용합니다. 아래에는 예제 구성 파일이 나와 있습니다. 이 구성 파일을 classpath의 디렉터리 중 하나로 복사합니다. Log4j JAR과 log4j.properties 파일이 동일한 디렉터리 안에 있지 않아도 됩니다.

log4j.properties 구성 파일은 로깅 수준, 로깅 출력이 전송될 대상(예를 들면, 파일 또는 콘솔), 출력 형식 같은 속성을 지정합니다. 로깅 수준은 로거가 생성되는 출력의 세부 수준입니다. Log4j는 여러 로깅 계층의 개념을 지원합니다. 로깅 수준은 각 계층마다 독립적으로 설정됩니다. AWS SDK for Java에서는 다음과 같은 두 가지 로깅 계층을 사용할 수 있습니다.

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

Classpath 설정

Log4j JAR 및 log4j.properties 파일 둘 다 classpath에 있어야 합니다. Apache Ant를 사용 중인 경우 Ant 파일의 `path` 요소에서 classpath를 설정합니다. 다음 예제에서는 SDK에 포함된 Amazon S3 예제에 대한 Ant 파일의 경로 요소를 보여줍니다.

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

Eclipse IDE를 사용 중인 경우 메뉴를 열고 프로젝트 | 속성 | Java Build 경로로 이동하여 classpath를 설정할 수 있습니다.

서비스 관련 오류 및 경고

클라이언트 라이브러리의 중요 메시지를 수집할 수 있도록 항상 "com.amazonaws" 로거 계층을 "WARN"로 설정된 상태로 유지하는 것이 좋습니다. 예를 들어 Amazon S3 클라이언트가 애플리케이션이 애플리케이션을 제대로 종료하지 않아 리소스가 유출될 수 있음을 감지하면 S3 클라이언트는 경고 메시지를 통해 로그에 이를 보고합니다. `InputStream` 또한 클라이언트에 요청 또는 응답 처리 문제가 발생하는 경우에도 메시지가 기록됩니다.

다음 log4j.properties 파일에서는 `rootLogger`를 WARN으로 설정하므로 "com.amazonaws"에 있는 모든 로거의 경고 및 오류가 포함됩니다. 또는 com.amazonaws 로거를 WARN으로 명시적으로 설정할 수도 있습니다.

```
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
```

요청/응답 요약 로깅

에 대한 모든 요청은 고유한 AWS 요청 ID를 AWS 서비스 생성하는데, 이는 요청을 처리하는 방식에 문제가 발생할 경우 유용합니다. AWS 서비스 AWS 요청 ID는 실패한 서비스 호출에 대해 SDK의 Exception 객체를 통해 프로그래밍 방식으로 액세스할 수 있으며, “com.amazonaws.request” 로거의 디버그 로그 수준을 통해 보고할 수도 있습니다.

다음 log4j.properties 파일을 사용하면 요청 ID를 포함한 요청 및 응답의 요약을 사용할 수 있습니다. 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
```

다음은 로그 출력의 예입니다.

```
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
```

상세 표시 유선 로깅

어떤 경우에는 보내고 받는 요청과 응답을 정확히 확인하는 것이 유용할 수 있습니다. AWS SDK for Java 대규모 요청 (예: 파일 업로드 대상 Amazon S3) 이나 응답을 기록하면 응용 프로그램 속도가 크게 저하될 수 있으므로 프로덕션 시스템에서는 이 로깅을 활성화하지 않아야 합니다. 이 정보에 정말 액세스해야 하는 경우 Apache HttpClient 4 로거를 통해 일시적으로 활성화할 수 있습니다. `org.apache.http.wire` 로거에서 DEBUG 수준을 활성화하면 모든 요청 및 응답 데이터에 대한 로깅이 활성화됩니다.

다음 log4j.properties 파일은 Apache HttpClient 4에서 풀 와이어 로깅을 활성화합니다. 이 파일은 애플리케이션 성능에 상당한 영향을 미칠 수 있으므로 일시적으로만 켜야 합니다.

```
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
```

지연 시간 지표 로깅

문제를 해결하고 어떤 프로세스가 가장 많은 시간이 걸리는지 또는 서버 또는 클라이언트 측에서 더 긴 지연 시간을 가지는지와 같은 지표를 확인하려면 지연 시간 로거가 유용할 수 있습니다. 이 로거를 활성화하려면 `com.amazonaws.latency` 로거를 DEBUG로 설정합니다.

> **Note**
>
> 이 로거는 SDK 지표가 활성화된 경우에만 사용할 수 있습니다. SDK 지표 패키지에 대한 자세한 내용은 AWS SDK for Java를 위한 지표 활성화를 참조하세요.

```
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
```

다음은 로그 출력의 예입니다.

```
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]
```

클라이언트 구성

를 AWS SDK for Java 사용하면 기본 클라이언트 구성을 변경할 수 있으므로 다음과 같은 경우에 유용합니다.

- 프록시를 통해 인터넷에 연결
- 연결 제한 시간 및 요청 재시도 등 HTTP 전송 설정 변경
- TCP 소켓 버퍼 크기 힌트 지정

프록시 구성

클라이언트 객체를 생성할 때 선택적 ClientConfiguration객체를 전달하여 클라이언트 구성을 사용자 정의할 수 있습니다.

프록시 서버를 통해 인터넷에 연결하는 경우 `ClientConfiguration` 객체를 통해 프록시 서버 설정(프록시 호스트, 포트 및 사용자 이름/암호)을 구성해야 합니다.

HTTP 전송 구성

ClientConfiguration객체를 사용하여 여러 HTTP 전송 옵션을 구성할 수 있습니다. 새 옵션이 추가되는 경우가 있습니다. 검색하거나 설정할 수 있는 전체 옵션 목록을 보려면 AWS SDK for Java API 참조를 참조하십시오.

> **Note**
>
> 각각의 구성 가능한 값마다 기본값이 상수로 정의되어 있습니다. 의 상수 값 목록은 AWS SDK for Java API 참조의 상수 필드 값을 참조하십시오. `ClientConfiguration`

최대 연결 수

를 사용하여 열린 HTTP 연결의 최대 허용 수를 설정할 수 ClientConfiguration있습니다. setMaxConnections메서드.

> **Important**
>
> 연결 충돌과 성능 저하를 방지하기 위해 최대 연결 수를 동시 트랜잭션 수에 맞게 설정합니다. 기본 최대 연결 값은 AWS SDK for Java API 참조의 상수 필드 값을 참조하십시오.

제한 시간 및 오류 처리

HTTP 연결 제한 시간 및 오류 처리와 관련된 옵션을 설정할 수 있습니다.

- 연결 제한 시간

  연결 제한 시간은 HTTP 연결 시 연결 시도를 포기하기 전에 연결이 설정될 때까지 기다리는 시간(밀리초)입니다. 기본값은 10,000ms입니다.

  이 값을 직접 설정하려면 를 사용하십시오 ClientConfiguration. setConnectionTimeout메서드.

- 연결 TTL(Time to Live)

  기본적으로 SDK에서는 최대한 오랫동안 HTTP 연결을 재사용하려고 합니다. 서비스가 불가능한 서버에 연결이 설정된 실패 상황에서는 유한 TTL을 설정하는 것이 애플리케이션을 복구하는 데 도움이 될 수 있습니다. 예를 들어 TTL을 15분으로 설정하면 문제가 발생한 서버에 연결이 설정된 경우라도 15분 내에 새 서버와의 연결이 다시 설정됩니다.

  HTTP 연결 TTL을 설정하려면 ClientConfiguration.setConnectionTTL 메서드를 사용하십시오.

- 최대 오류 재시도 횟수

  재설정 가능한 오류의 기본 최대 재시도 수는 3입니다. 를 사용하여 다른 값을 설정할 수 있습니다. ClientConfiguration setMaxError재시도 방법.

로컬 주소

HTTP 클라이언트가 바인딩할 로컬 주소를 설정하려면 를 사용하십시오ClientConfiguration. setLocalAddress.

TCP 소켓 버퍼 크기 힌트

저수준 TCP 파라미터를 조정하려는 고급 사용자는 객체를 통해 TCP 버퍼 크기 힌트를 추가로 설정할 수 있습니다. ClientConfiguration 대부분의 사용자는 이러한 값을 변경할 필요가 없으며, 이 기능은 고급 사용자용으로 제공됩니다.

애플리케이션용 TCP 버퍼 크기(선택 사항)는 주로 네트워크와 운영 체제 구성 및 기능에 따라 결정됩니다. 예를 들면, 대부분의 현대식 운영 체제는 TCP 버퍼 크기에 대해 자동 튜닝 논리를 제공하는데, 이는 자동 튜닝을 통해 버퍼 크기를 최적화할 수 있을 만큼 TCP 연결이 충분히 오랫동안 열린 상태로 유지되어야 하므로 TCP 연결성능에 큰 영향을 미칠 수 있습니다.

대형 버퍼 크기(예: 2MB)를 사용하면 운영 체제는 원격 서버에서 해당 정보 수신 여부를 확인할 필요가 없으므로 메모리 더 많은 데이터를 버퍼링할 수 있으며, 이는 네트워크 지연 시간이 긴 경우에 특히 유용합니다.

이 옵션은 힌트일 뿐, 운영 체제에서는 인식하지 못할 수도 있습니다. 이 옵션을 사용하는 경우 사용자는 운영 체제의 구성된 제한 및 기본값을 항상 확인해야 합니다. 대부분의 운영 체제에는 최대 TCP 버퍼 크기 제한이 구성되어 있으므로, 최대 TCP 버퍼 크기 제한을 명시적으로 높이지 않는 한 해당 제한을 초과할 수 없습니다.

다음과 같은 여러 리소스는 TCP 버퍼 크기와 운영 체제별 TCP 설정을 구성하는 데 도움이 됩니다.

- 호스트 튜닝

액세스 제어 정책

AWS 액세스 제어 정책을 사용하면 리소스에 대한 세밀한 액세스 제어를 지정할 수 있습니다. AWS 액세스 제어 정책은 다음과 같은 형태의 문 모음으로 이루어집니다.

계정 A에는 조건 D가 적용되는 리소스 C에서 작업 B를 수행할 권한이 있습니다.

위치:

- A는 보안 주체 - 리소스 AWS 계정 중 하나에 대한 액세스 또는 수정을 요청하는 주체입니다. AWS
- B는 작업 - Amazon SQS 대기열에 메시지를 보내거나 Amazon S3 버킷에 객체를 저장하는 등 AWS 리소스에 액세스하거나 수정되는 방식입니다.
- C는 리소스 - 보안 주체가 액세스하려는 AWS 개체 (예: Amazon SQS 대기열 또는 저장된 객체) Amazon S3입니다.

- D는 조건 집합, 즉 주체가 리소스에 액세스할 수 있도록 허용하거나 거부할 시기를 지정하는 선택적 제약 조건입니다. 여러 표현식 조건을 사용할 수 있는 일부 각 서비스에 국한됩니다. 예를 들면, 날짜 조건을 사용하여 특정 시간 이후 또는 이전에만 리소스에 대한 액세스를 허용할 수 있습니다.

Amazon S3 예시

다음 예제는 모든 사용자가 버킷의 모든 객체를 읽을 수 있도록 허용하지만, 해당 버킷에 객체를 업로드할 수 있는 AWS 계정액세스는 버킷 소유자 계정과 함께 특정 2개 (버킷 소유자 계정 포함) 로 제한하는 정책을 보여줍니다.

```
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 예제

정책의 일반적인 용도 중 하나는 Amazon SNS 주제의 메시지를 수신하도록 Amazon SQS 대기열에 권한을 부여하는 것입니다.

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

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

AmazonSQS sqs = AmazonSQSClientBuilder.defaultClient();
```

```
sqs.setQueueAttributes(new SetQueueAttributesRequest(myQueueUrl, queueAttributes));
```

Amazon SNS 예제

일부 서비스는 정책에 사용할 수 있는 추가 조건을 제공합니다. Amazon SNS는 주제 구독 요청의 프로토콜 (예: 이메일, HTTP, HTTPS Amazon SQS) 및 엔드포인트 (예: 이메일 주소, URL, Amazon SQS ARN) 에 따라 SNS 주제 구독을 허용하거나 거부하기 위한 조건을 제공합니다.

```
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()));
```

DNS 이름 조회를 위한 JVM TTL 설정

Java 가상 머신(JVM)은 DNS 이름 조회를 캐시합니다. JVM은 호스트 이름을 IP 주소로 확인하면 지정된 기간 (TTL) 동안 IP 주소를 캐싱합니다. time-to-live

AWS 리소스는 가끔 변경되는 DNS 이름 항목을 사용하므로 TTL 값 5초로 JVM을 구성하는 것이 좋습니다. 이렇게 하면 리소스의 IP 주소가 변경될 때 애플리케이션이 DNS를 다시 쿼리하여 리소스의 새 IP 주소를 수신하고 사용할 수 있습니다.

일부 Java 구성에서는 JVM이 다시 시작될 때까지 DNS 항목을 새로 고치지 않도록 JVM 기본 TTL이 설정되기도 합니다. 따라서 애플리케이션이 실행 중일 때 AWS 리소스의 IP 주소가 변경되면 JVM을 수동으로 다시 시작하고 캐시된 IP 정보를 새로 고칠 때까지 해당 리소스를 사용할 수 없습니다. 이 경우 캐시된 IP 정보를 정기적으로 새로 고치도록 JVM의 TTL을 설정해야 합니다.

JVM TTL을 설정하는 방법

JVM의 TTL을 수정하려면 networkaddress.cache.ttl 보안 속성 값을 설정하고 자바 8의 경우 파일 또는 자바 11 이상의 경우 파일에서 `networkaddress.cache.ttl` 속성을 설정합니다. `$JAVA_HOME/jre/lib/security/java.security` `$JAVA_HOME/conf/security/java.security`

다음은 5초로 설정된 TTL 캐시를 보여 주는 파일의 스니펫입니다. java.security

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

$JAVA_HOME환경 변수로 표시되는 JVM에서 실행되는 모든 응용 프로그램은 이 설정을 사용합니다.

에 대한 메트릭을 활성화합니다. AWS SDK for Java

CloudWatchAmazon의 시각화 및 모니터링을 위해 다음을 측정하는 지표를 생성할 AWS SDK for Java 수 있습니다.

- 액세스 시 애플리케이션의 성능 AWS
- 다음과 함께 사용할 때의 JVM 성능 AWS
- 힙 메모리, 스레드 수 및 열었던 파일 설명자 등 실행 시간 환경 세부 정보

SDK 지표 생성을 활성화하는 방법

SDK가 메트릭을 전송할 수 있도록 하려면 다음 Maven 종속성을 추가해야 합니다. 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>
```

*버전 번호를 Maven Central에서 사용할 수 있는 최신 버전의 SDK로 교체하세요.

AWS SDK for Java 지표는 기본적으로 비활성화되어 있습니다. 로컬 개발 환경에서 활성화하려면 JVM을 시작할 때 AWS 보안 자격 증명 파일을 가리키는 시스템 속성을 포함합니다. 예:

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

SDK가 수집한 데이터포인트를 나중에 분석할 수 있도록 업로드할 수 있도록 자격 증명 파일의 경로를 지정해야 합니다 CloudWatch .

> **Note**
>
> 인스턴스 메타데이터 서비스를 사용하여 AWS Amazon EC2 인스턴스에서 액세스하는 경우 자격 증명 파일을 지정하지 않아도 됩니다. Amazon EC2 이 경우에는 다음 항목만 지정하면 됩니다.
>
> ```
> -Dcom.amazonaws.sdk.enableDefaultMetrics
> ```

에서 캡처한 모든 AWS SDK for Java 지표는 AWSSDK/Java 네임스페이스에 있으며 CloudWatch 기본 리전 (us-east-1) 에 업로드됩니다. 리전을 변경하려면 시스템 속성에서 `cloudwatchRegion` 속성을 사용하여 지정합니다. 예를 들어 CloudWatch 지역을 us-east-1로 설정하려면 다음을 사용하십시오.

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

이 기능을 활성화하면 AWS 에서 서비스를 요청할 때마다 지표 데이터 포인트가 생성되고 통계 요약 대기열에 추가되며 약 1분에 한 번씩 비동기적으로 업로드됩니다. AWS SDK for Java CloudWatch 지

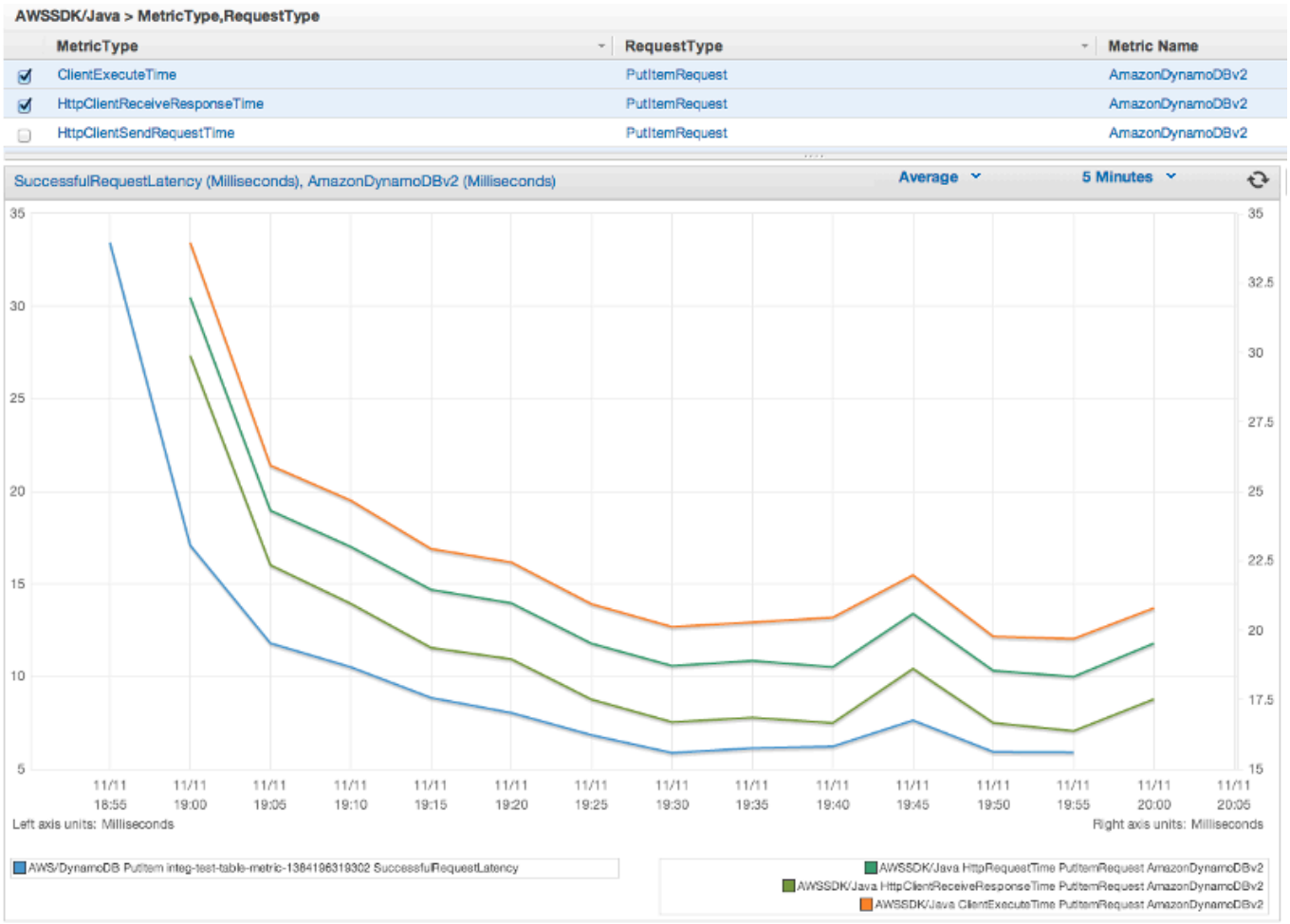
표는 업로드되고 나면 AWS Management Console을 사용하여 시각화할 수 있으며, 메모리 누출, 파일 설명자 누출 등 잠재적인 문제에 대한 경보를 설정할 수 있습니다.

사용 가능한 지표 유형

기본 지표 세트는 세 가지 주요 범주로 나눠집니다.

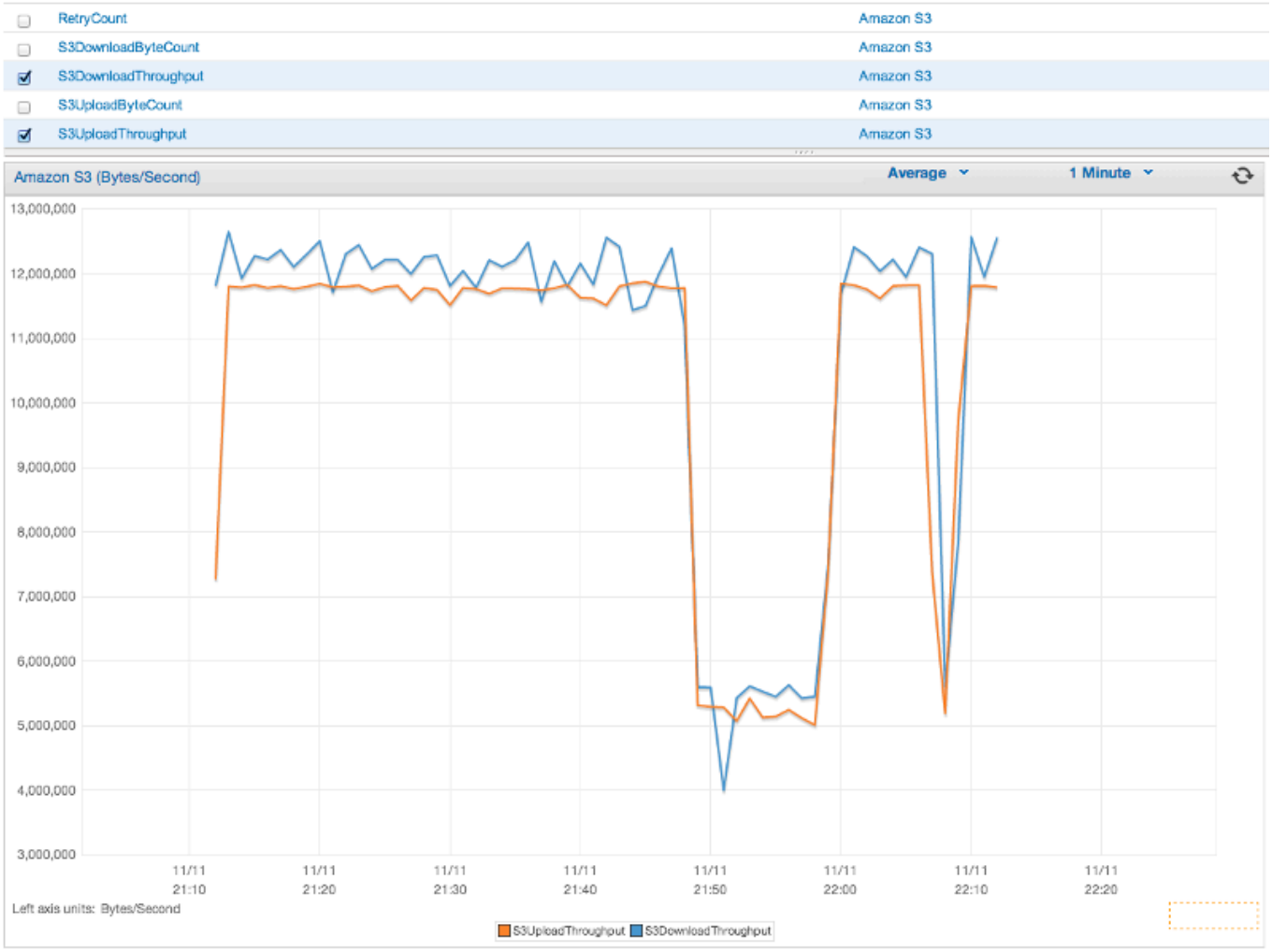
AWS 요청 지표

- HTTP 요청/응답 지연 시간, 요청 수, 예외 및 재시도 등을 포함합니다.



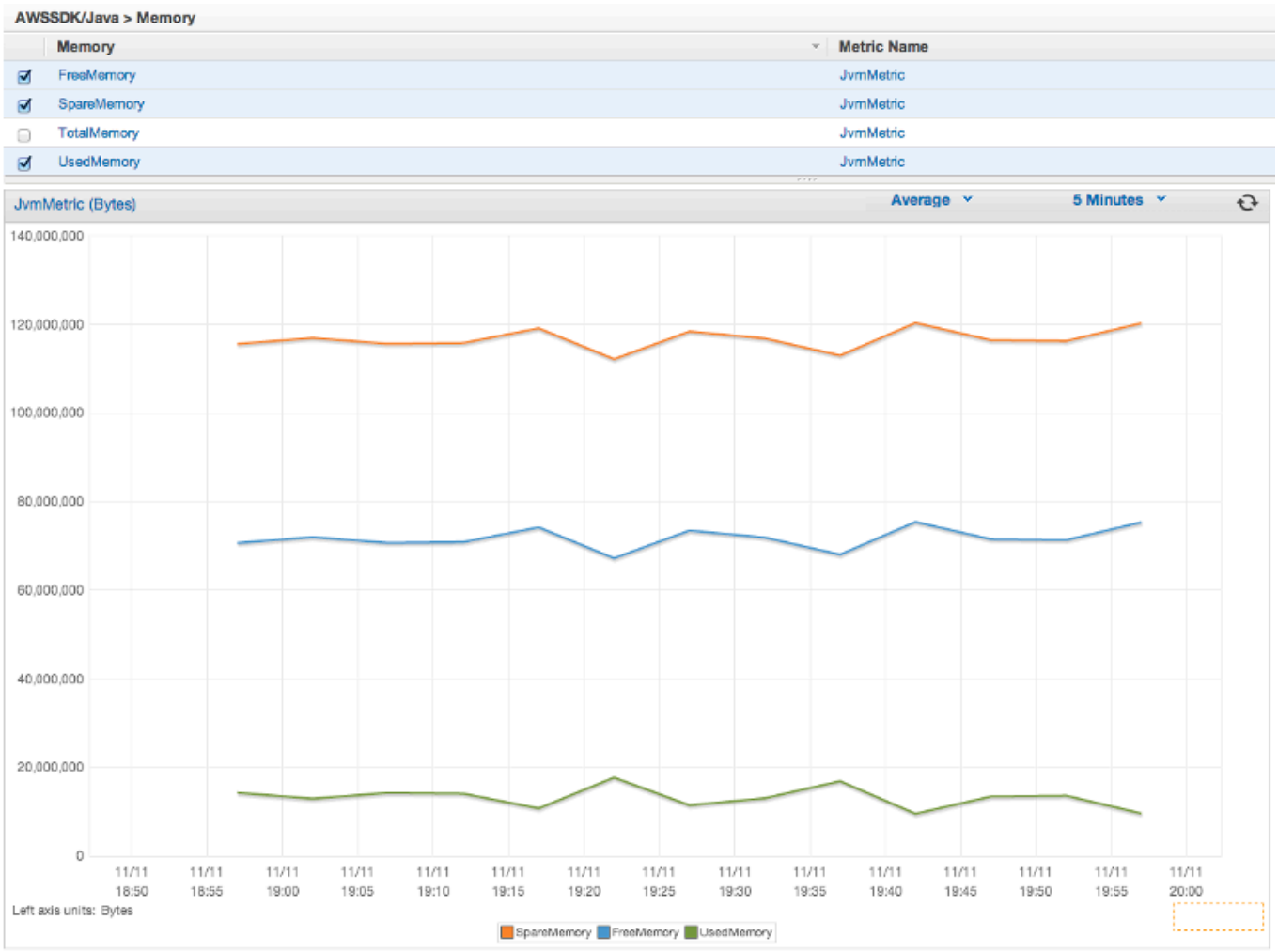
AWS 서비스 메트릭스

- S3 업로드 및 다운로드의 처리량 및 바이트 수와 같은 AWS 서비스특정 데이터를 포함하십시오.



머신 지표

- 힙 메모리, 스레드 수 및 열린 파일 설명자 등 실행 시간 환경을 포함합니다.



머신 지표를 제외하려면 `excludeMachineMetrics`를 시스템 속성에 추가합니다.

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

추가 정보

- 미리 정의된 핵심 지표 유형의 전체 목록은 amazonaws/metrics package summary를 참조하십시오.
- AWS SDK for Java CloudWatch 예제에서 CloudWatch 를 사용하여 작업하는 방법에 대해 알아보십시오. AWS SDK for Java
- 복원력 향상을 위한 조정 블로그 게시물에서 성능 AWS SDK for Java 조정에 대해 자세히 알아보십시오.

AWS SDK for Java 코드 예제

이 단원은 AWS SDK for Java v1를 사용하여 AWS를 프로그래밍하는 방법에 대한 자습서와 예제를 제공합니다.

GitHub의 AWS 설명서 코드 예제 리포지토리에서 이러한 예제의 소스 코드와 다른 코드를 찾습니다.

AWS 설명서 팀이 생성을 고려할 수 있는 새 코드 예제를 제안하려면 새 요청을 생성합니다. 이 팀은 개별 API 호출만 다루는 간단한 코드 조각에 비해 광범위한 시나리오 및 사용 사례를 다루는 코드를 생성하려고 합니다. 지침은 GitHub의 코드 예제 리포지토리에 있는 기여 가이드라인을 참조하세요.

AWS SDK for Java 2.x

AWS는 2018년에 AWS SDK for Java 2.x를 출시했습니다. 이 안내서에는 예제 코드와 함께 최신 Java SDK 사용에 대한 지침이 포함되어 있습니다.

> **Note**
>
> AWS SDK for Java 개발자가 사용할 수 있는 추가 예제와 추가 리소스는 추가 문서 및 리소스를 참조하세요.

AWS SDK for Java를 사용한 CloudWatch 예제

이 단원에서는 AWS SDK for Java를 사용한 CloudWatch 프로그래밍의 예제를 제공합니다.

Amazon CloudWatch는 AWS에서 실행하는 Amazon Web Services(AWS) 리소스와 애플리케이션을 실시간으로 모니터링합니다. CloudWatch를 사용하여 리소스 및 애플리케이션에 대해 측정할 수 있는 변수인 지표를 수집하고 추적할 수 있습니다. CloudWatch 경보는 알림을 보내거나 정의한 규칙을 기준으로 모니터링하는 리소스를 자동으로 변경합니다.

CloudWatch에 대한 자세한 내용은 사용 설명서Amazon CloudWatch를 참조하세요.

> **Note**
>
> 예제에는 각 기술을 보여주는 데 필요한 코드만 포함되어 있습니다. 전체 예제 코드는 GitHub에 있습니다. 이 위치에서 단일 소스 파일을 다운로드하거나 리포지토리를 로컬로 복사하여 모든 예제를 빌드하고 실행할 수 있습니다.

주제

Amazon CloudWatch에서 지표 가져오기

지표 나열

CloudWatch 지표를 나열하려면 ListMetricsRequest를 생성하고 AmazonCloudWatchClient의 `listMetrics` 메서드를 호출합니다. `ListMetricsRequest`를 사용하여 반환된 지표를 네임스페이스, 지표 이름 또는 차원을 기준으로 필터링할 수 있습니다.

> **Note**
>
> AWS 서비스가 게시하는 지표 및 차원 목록은 Amazon CloudWatch 사용 설명서의 {{https---docs-aws-amazon-com-AmazonCloudWatch-latest-monitoring-CW-Support-For-AWS-html} [Amazon CloudWatch 지표 및 차원 참조]에서 확인할 수 있습니다.

가져오기

```
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;
```

코드

```
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;
    }
}
```

지표는 해당 `getMetrics` 메서드를 호출하여 ListMetricsResult에 반환됩니다. 결과를 페이징할 수 있습니다. 다음 결과 배치를 가져오려면 `ListMetricsResult` 객체의 `getNextToken` 메서드에서 반환된 값과 함께 원래 요청 객체에 대해 `setNextToken`을 호출한 후 수정된 요청 객체를 또 다른 `listMetrics` 호출에 다시 전달합니다.

추가 정보

- Amazon CloudWatch API 참조의 ListMetrics.

사용자 지정 지표 데이터 게시

여러 AWS 서비스는 " AWS "으로 시작하는 네임스페이스에 고유의 지표를 게시합니다. 고유의 네임스페이스(" AWS "로 시작하지 않는 경우)를 사용하여 사용자 지정 지표 데이터를 게시할 수도 있습니다.

사용자 지정 지표 데이터 게시

자체 지표 데이터를 게시하려면 PutMetricDataRequest를 사용하여 AmazonCloudWatchClient의 `putMetricData` 메서드를 호출하세요. `PutMetricDataRequest`는 데이터에 사용할 사용자 지정 네임스페이스와, MetricDatum 객체의 데이터 포인트 자체에 대한 정보를 포함해야 합니다.

> **Note**
>
> " AWS "로 시작하는 네임스페이스는 지정할 수 없습니다. " AWS "로 시작하는 네임스페이스는 Amazon Web Services 제품용으로 예약되어 있습니다.

가져오기

```
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;
```

코드

```
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);
```

추가 정보

- Amazon CloudWatch 사용 설명서의 Amazon CloudWatch 지표 사용.
- Amazon CloudWatch 사용 설명서의 AWS 네임스페이스.

- Amazon CloudWatch API 참조의 PutMetricData.

Amazon CloudWatch 경보 작업

경보 만들기

CloudWatch 지표를 기반으로 경보를 생성하려면 경보 조건이 입력된 PutMetricalArmRequest를 사용하여 AmazonCloudWatchClient의 `putMetricAlarm` Client의 메서드를 호출합니다.

가져오기

```
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;
```

코드

```
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);
```

경보 나열

생성한 CloudWatch 경보를 나열하려면 결과에 대한 옵션을 설정하는 데 사용할 수 있는 DescribeAlarmsRequest와 함께 AmazonCloudWatchClient의 `describeAlarms` 메서드를 호출하십시오.

가져오기

```
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;
```

코드

```
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;
    }
}
```

describeAlarms에 의해 반환되는 DescribeAlarmsResult에 대해 getMetricAlarms를 호출하여 경보 목록을 가져올 수 있습니다.

결과를 페이징할 수 있습니다. 다음 결과 배치를 가져오려면 DescribeAlarmsResult 객체의 getNextToken 메서드에서 반환된 값과 함께 원래 요청 객체에 대해 setNextToken을 호출한 후 수정된 요청 객체를 또 다른 describeAlarms 호출에 다시 전달합니다.

> **Note**
>
> AmazonCloudWatchClient의 describeAlarmsForMetric 메서드를 사용하여 특정 지표의 경보를 검색할 수도 있습니다. 이 메서드의 용도는 describeAlarms와 비슷합니다.

경보 삭제

CloudWatch 경보를 삭제하려면 삭제하려는 경보 이름이 하나 이상 포함된 DeleteAlarmsRequest를 사용하여 AmazonCloudWatchClient의 deleteAlarms 메서드를 호출합니다.

가져오기

```
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;
```

코드

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

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

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

추가 정보

- Amazon CloudWatch 사용 설명서의 Amazon CloudWatch 경보 생성
- Amazon CloudWatch API 참조의 PutMetricAlarm

- Amazon CloudWatch API 참조의 DescribeAlarms
- Amazon CloudWatch API 참조의 DeleteAlarms

CloudWatch에서 경보 조치 사용

경보 작업을 사용하면 Amazon EC2 인스턴스 자동 중지, 종료, 재부팅 또는 복구 등의 작업을 수행하는 경보를 생성할 수 있습니다.

> **Note**
>
> 경보를 생성할 때 PutMetricAlarmRequest `setAlarmActions` 메서드를 사용하여 경보 동작을 경보에 추가할 수 있습니다.

경보 작업 활성화

CloudWatch 경보에 대한 경보 작업을 활성화하려면 활성화하려는 경보 이름을 하나 이상 포함하는 EnableAlarmActionsRequest를 사용하여 AmazonCloudWatchClient의 `enableAlarmActions`를 호출하세요.

가져오기

```
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;
```

코드

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

EnableAlarmActionsRequest request = new EnableAlarmActionsRequest()
    .withAlarmNames(alarm);

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

경보 작업 비활성화

CloudWatch 경보에 대한 경보 작업을 비활성화하려면 비활성화하려는 경보 이름을 하나 이상 포함하는 DisableAlarmActionsRequest를 사용하여 AmazonCloudWatchClient의 `disableAlarmActions`를 호출하세요.

가져오기

```
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;
```

코드

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

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

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

추가 정보

- Amazon CloudWatch 사용 설명서의 인스턴스를 중지, 종료, 재부팅 또는 복구하는 경보 생성
- Amazon CloudWatch API 참조의 PutMetricAlarm
- Amazon CloudWatch API 참조의 EnableAlarmActions
- Amazon CloudWatch API 참조의 DisableAlarmActions

CloudWatch로 이벤트 전송

CloudWatch 이벤트는 Amazon EC2 인스턴스, Lambda 함수, Kinesis 스트림, Amazon ECS 작업, Step Functions 상태 시스템, Amazon SNS 주제, Amazon SQS 대기열 또는 기본 제공 대상에 대한 AWS 리소스의 변경 사항을 설명하며 실시간에 가까운 시스템 이벤트 스트림을 제공합니다. 단순 규칙을 사용하여 일치하는 이벤트를 검색하고 하나 이상의 대상 함수 또는 스트림으로 이를 라우팅할 수 있습니다.

이벤트 추가

사용자 지정 CloudWatch 이벤트를 추가하려면 각 이벤트에 대한 세부 정보를 제공하는 PutEventsRequestEntry 객체가 하나 이상 포함된 PutEventsRequest 객체를 사용하여 AmazonCloudWatchEventsClient의 `putEvents` 메서드를 호출하세요. 이벤트 유형 및 소스, 이벤트와 연결된 리소스 등 입력 항목에 대한 여러 파라미터를 지정할 수 있습니다.

> **Note**
>
> `putEvents` 호출당 최대 10개 이벤트를 지정할 수 있습니다.

가져오기

```
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;
```

코드

```
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);
```

규칙 추가

규칙을 생성하거나 업데이트하려면 규칙 이름과 선택적 파라미터(예: PutRuleRequest, 규칙과 연결할 IAM 역할, 규칙 실행 빈도를 설명하는 일정 수식)를 포함한 PutRuleRequest를 사용하여 AmazonCloudWatchEventsClient의 `putRule` 메서드를 호출합니다.

가져오기

```
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;
```

코드

```
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);
```

대상 추가

대상은 규칙이 트리거될 때 호출되는 리소스입니다. 대상의 예로는 Amazon EC2 인스턴스, Lambda 함수, Kinesis 스트림, Amazon ECS 작업, Step Functions 상태 시스템 및 기본 제공 대상이 있습니다.

규칙에 대상을 추가하려면 업데이트할 규칙과 규칙에 추가할 대상 목록이 포함된 PutTargetsRequest를 사용하여 AmazonCloudWatchEventsClient의 `putTargets` 메서드를 호출합니다.

가져오기

```
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;
```

코드

```
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);
```

추가 정보

- Amazon CloudWatch Events 사용 설명서의 PutEvents를 사용하여 이벤트 추가
- Amazon CloudWatch Events 사용 설명서의 규칙에 대한 스케줄 표현식
- Amazon CloudWatch Events 사용 설명서의 CloudWatch 이벤트의 이벤트 유형
- Amazon CloudWatch Events 사용 설명서의 이벤트 및 이벤트 패턴
- Amazon CloudWatch Events API 참조의 PutEvents
- Amazon CloudWatch Events API 참조의 PutTargets
- Amazon CloudWatch Events API 참조의 PutRule

AWS SDK for Java를 사용하는 DynamoDB 예제

이 단원에서는 AWS SDK for Java를 사용한 DynamoDB 프로그래밍의 예제를 제공합니다.

> **Note**
>
> 예제에는 각 기술을 보여주는 데 필요한 코드만 포함되어 있습니다. 전체 예제 코드는 GitHub에 있습니다. 이 위치에서 단일 소스 파일을 다운로드하거나 리포지토리를 로컬로 복사하여 모든 예제를 빌드하고 실행할 수 있습니다.

주제

DynamoDB의 테이블 작업

테이블은 DynamoDB 데이터베이스에 있는 모든 항목의 컨테이너입니다. DynamoDB에 데이터를 추가하거나 삭제하기 전에 먼저 테이블을 만들어야 합니다.

각 테이블마다 다음을 정의해야 합니다.

- 계정 및 리전에 고유한 테이블 이름
- 모든 값이 고유한 기본 키. 테이블의 두 항목에 동일한 기본 키 값을 지정할 수 없습니다.

  기본 키는 단일 파티션(HASH) 키로 이루어진 단순형이거나, 파티션과 정렬(RANGE) 키로 이루어진 복합형일 수 있습니다.

  각 키 값에는 ScalarAttributeType 클래스에 의해 열거되는 관련 데이터 유형이 있습니다. 키 값은 이진(B), 숫자(N) 또는 문자열(S)일 수 있습니다. 자세한 정보는 Amazon DynamoDB 개발자 안내서의 명명 규칙과 데이터 유형을 참조하세요.
- 테이블에 대해 예약된 읽기/쓰기 용량 단위를 정의하는 프로비저닝된 처리량

> **Note**
>
> Amazon DynamoDB요금은 테이블에 대해 설정하는 프로비저닝된 처리량 값을 기준으로 하므로 테이블에 대해 필요한 만큼의 용량만 예약하세요.

언제라도 테이블의 프로비저닝된 처리량을 수정할 수 있으므로 변경이 필요할 경우 용량을 조정할 수 있습니다.

테이블 생성

DynamoDB 클라이언트의 `createTable` 메서드를 사용하여 새 DynamoDB 테이블을 만듭니다. 테이블 속성과 테이블 스키마를 구성해야 하며, 이 두 가지 요소 모두 테이블의 기본 키를 식별하는 데 사용됩니다. 또한 프로비저닝된 초기 처리량 값과 테이블 이름도 지정해야 합니다. DynamoDB 테이블 생성 시 키 테이블 속성만 정의합니다.

> **Note**
>
> 선택한 이름의 테이블이 이미 있는 경우 AmazonServiceException이 발생합니다.

가져오기

```
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;
```

단순형 기본 키를 사용하여 테이블 생성

이 코드는 단순형 기본 키("Name")를 사용하여 테이블을 만듭니다.

코드

```
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);
}
```

GitHub의 전체 예제를 참조하십시오.

복합형 기본 키를 사용하여 테이블 생성

다른 AttributeDefinition 및 KeySchemaElement를 CreateTableRequest에 추가합니다.

코드

```
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);
```

GitHub의 전체 예제를 참조하십시오.

테이블 나열

DynamoDB 클라이언트의 `listTables` 메서드를 호출하여 특정 리전의 테이블을 나열할 수 있습니다.

> **Note**
>
> 계정 및 리전에 대해 이름이 지정된 테이블이 없으면 ResourceNotFoundException이 발생합니다.

가져오기

```
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;
```

코드

```
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;
        }
```

기본적으로 호출당 최대 100개의 테이블이 반환됩니다. 반환된 ListTablesResult 객체에 `getLastEvaluatedTableName`를 사용하여 마지막으로 평가된 테이블을 가져올 수 있습니다. 이 값을 사용하여 이전 목록의 마지막으로 반환된 값 다음에 이어지는 목록을 시작할 수 있습니다.

GitHub의 전체 예제를 참조하십시오.

테이블 설명(테이블에 대한 정보 가져오기)

DynamoDB 클라이언트의 `describeTable` 메서드를 호출합니다.

> **Note**
>
> 계정 및 리전에 대해 이름이 지정된 테이블이 없으면 ResourceNotFoundException이 발생합니다.

가져오기

```
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;
```

코드

```
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);
}
```

GitHub의 전체 예제를 참조하십시오.

테이블 수정(업데이트)

DynamoDB 클라이언트의 `updateTable` 메서드를 호출하여 언제라도 테이블의 프로비저닝된 처리량 값을 수정할 수 있습니다.

> **Note**
>
> 계정 및 리전에 대해 이름이 지정된 테이블이 없으면 ResourceNotFoundException이 발생합니다.

가져오기

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

코드

```
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);
}
```

GitHub의 전체 예제를 참조하십시오.

테이블 삭제

DynamoDB 클라이언트의 `deleteTable` 메서드를 호출하고 테이블의 이름을 이 메서드에 전달합니다.

> **Note**
>
> 계정 및 리전에 대해 이름이 지정된 테이블이 없으면 ResourceNotFoundException이 발생합니다.

가져오기

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

코드

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

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

GitHub의 전체 예제를 참조하십시오.

추가 정보

- Amazon DynamoDB 개발자 안내서의 테이블 작업 가이드라인
- Amazon DynamoDB 개발자 안내서의 DynamoDB의 테이블 다루기

DynamoDB에서의 항목 작업

DynamoDB에서 항목은 각각 이름과 값이 있는 속성의 컬렉션입니다. 속성 값은 스칼라, 세트 또는 문서 유형일 수 있습니다. 자세한 정보는 Amazon DynamoDB 개발자 안내서의 명명 규칙과 데이터 유형을 참조하세요.

테이블에서 항목 검색(가져오기)

AmazonDynamoDB의 `getItem`의 메서드를 호출하여 테이블 이름과 원하는 항목의 기본 키 값이 포함된 GetItemRequest 객체를 이 메서드에 전달합니다. 이 메서드는 GetItemResult 객체를 반환합니다.

반환된 `GetItemResult` 객체의 `getItem()` 메서드를 사용하여 항목과 연결된 키 (문자열) 및 값 (AttributeValue) 쌍의 맵을 가져올 수 있습니다.

가져오기

```
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;
```

코드

```
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);
```

GitHub의 전체 예제를 참조하십시오.

테이블에 새 항목 추가

항목의 속성을 나타내는 키-값 페어의 맵을 생성합니다. 여기에는 테이블의 기본 키 필드 값이 포함되어야 합니다. 기본 키로 식별되는 항목이 이미 존재하면 필드가 요청에 따라 업데이트됩니다.

> **Note**
>
> 계정 및 리전에 대해 이름이 지정된 테이블이 없으면 ResourceNotFoundException이 발생합니다.

가져오기

```
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;
```

코드

```
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);
```

GitHub의 전체 예제를 참조하십시오.

테이블의 기존 항목 업데이트

AmazonDynamoDB의 `updateItem` 메서드를 사용하여 테이블 이름, 기본 키 값 및 업데이트할 필드 맵을 지정함으로써 테이블에 이미 존재하는 항목의 속성을 업데이트할 수 있습니다.

> **Note**
>
> 해당 계정 및 리전에 대해 이름이 지정된 테이블이 없거나 전달한 기본 키로 식별되는 항목이 없으면 ResourceNotFoundException이 발생합니다.

가져오기

```
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;
```

코드

```
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);
```

GitHub의 전체 예제를 참조하십시오.

DynamoDBMapper 클래스 사용

AWS SDK for Java에서는 클라이언트 측 클래스를 Amazon DynamoDB 테이블로 매핑할 수 있는 DynamoDBMapper 클래스를 제공합니다. DynamoDBMapper 클 래스를 사용하려면 주석(다음 코드

예제에 표시됨)을 사용하여 DynamoDB 테이블의 항목과 코드의 해당 객체 인스턴스 간 관계를 정의합니다. DynamoDBMapper 클래스를 통해 테이블에 액세스하여 다양한 생성, 읽기, 업데이트 및 삭제(CRUD) 작업을 수행할 뿐만 아니라 쿼리를 실행할 수도 있습니다.

> **Note**
>
> DynamoDBMapper 클래스는 테이블 생성, 업데이트 또는 삭제를 허용하지 않습니다.

가져오기

```
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;
```

코드

다음 자바 코드 예제는 DynamoDBMapper 클래스를 사용하여 Music 테이블에 콘텐츠를 추가하는 방법을 보여줍니다. 테이블에 콘텐츠가 추가되면 파티션 및 정렬 키를 사용하여 항목이 로드됩니다. 그런 다음 수상 항목이 업데이트됩니다. Music 테이블을 생성하는 방법에 대한 자세한 내용은 Amazon DynamoDB 개발자 안내서의 테이블 생성을 참조하세요.

```
        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;
    }
}
```

GitHub의 전체 예제를 참조하십시오.

추가 정보

- Amazon DynamoDB 개발자 안내서의 항목 작업 가이드라인
- Amazon DynamoDB 개발자 안내서의 DynamoDB의 항목 작업 가이드라인

AWS SDK for Java를 사용하는 Amazon EC2 예제

이 단원에서는 AWS SDK for Java를 사용한 Amazon EC2 프로그래밍의 예제를 제공합니다.

주제

- 자습서: EC2 인스턴스 시작
- IAM 역할을 사용하여 Amazon EC2의 AWS 리소스에 대한 액세스 권한 부여
- 자습서: Amazon EC2 스팟 인스턴스
- 자습서: 고급 Amazon EC2 스팟 요청 관리

- Amazon EC2 인스턴스 관리
- 에서 엘라스틱 IP 주소 사용 Amazon EC2
- 리전 및 가용 영역 사용
- Amazon EC2 키 페어 작업
- Amazon EC2의 보안 그룹 작업

자습서: EC2 인스턴스 시작

이 자습서는 AWS SDK for Java를 사용하여 EC2 인스턴스를 시작하는 방법을 보여줍니다.

주제

- 필수 조건
- Amazon EC2 보안 그룹 생성
- 키 페어 생성
- Amazon EC2 인스턴스 실행

필수 조건

시작하기 전에 AWS 계정을 생성하고 AWS 자격 증명을 설정해야 합니다. 자세한 내용은 시작하기를 참조하십시오.

Amazon EC2 보안 그룹 생성

EC2-Classic은 더 이상 사용되지 않습니다.

> **Warning**
>
> EC2-Classic은 2022년 8월 15일에 사용 중지될 예정입니다. EC2-Classic에서 VPC로 마이그레이션하는 것이 좋습니다. 자세한 내용은 Amazon EC2 사용 설명서 또는 Amazon EC2 사용 설명서의 EC2-Classic에서 VPC로 마이그레이션을 참조하십시오. 또한 블로그 게시물 EC2-Classic-Classic Networking은 사용 중지됩니다. 준비 방법은 다음과 같습니다를 참조하세요.

하나 이상의 EC2 인스턴스에 대한 네트워크 트래픽을 제어하는 가상 방화벽 역할을 하는 보안 그룹을 생성합니다. 기본적으로 인바운드 트래픽을 허용하지 않는 보안 그룹과 인스턴스를 Amazon EC2 연결합니다. EC2 인스턴스가 특정 트래픽을 받아들이도록 허용하는 보안 그룹을 생성할 수 있습니다. 예를

들어 Linux 인스턴스에 연결해야 하는 경우 SSH 트래픽을 허용하도록 보안 그룹을 구성해야 합니다. Amazon EC2 콘솔이나 를 사용하여 보안 그룹을 생성할 수 있습니다. AWS SDK for Java

EC2-Classic 또는 EC2-VPC에서 사용할 보안 그룹을 생성합니다. EC2-Classic 및 EC2-VPC 관련 자세한 내용은 Linux 인스턴스용 Amazon EC2 사용 설명서의 지원 플랫폼을 참조하십시오.

Amazon EC2 콘솔을 사용하여 보안 그룹을 생성하는 방법에 대한 자세한 내용은 Linux 인스턴스용 사용 Amazon EC2 설명서의 Amazon EC2 보안 그룹을 참조하십시오.

1. CreateSecurityGroupRequest인스턴스 생성 및 초기화 다음과 같이 withGroupName메서드를 사용하여 보안 그룹 이름을 설정하고 withDescription 메서드를 사용하여 보안 그룹 설명을 설정합니다.

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

   보안 그룹 이름은 클라이언트를 초기화하는 AWS 지역 내에서 고유해야 합니다. Amazon EC2 보안 그룹 이름과 설명에는 US-ASCII 문자를 사용해야 합니다.
2. 요청 객체를 파라미터로 createSecurityGroup메서드에 전달합니다. 메서드는 다음과 같이 CreateSecurityGroupResult객체를 반환합니다.

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

   기존 보안 그룹과 동일한 이름의 보안 그룹을 생성하려고 하는 경우 `createSecurityGroup`에서 예외가 발생합니다.

기본적으로 새 보안 그룹은 Amazon EC2 인스턴스로의 인바운드 트래픽을 허용하지 않습니다. 인바운드 트래픽을 허용하려면 보안 그룹 수신을 명시적으로 승인해야 합니다. 개별 IP 주소, IP 주소 범위, 특정 프로토콜 및 TCP/UDP 포트에 대해 수신을 승인할 수 있습니다.

1. 인스턴스 생성 및 초기화. IpPermission withIPv4Ranges 메서드를 사용하여 수신을 승인할 IP 주소 범위를 설정하고 이 방법을 사용하여 IP 프로토콜을 설정합니다. withIpProtocol 다음과 같이 withFromPort및 withToPort메서드를 사용하여 수신을 승인할 포트 범위를 지정합니다.

   ```
   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);
```

수신이 허용되려면 IpPermission 객체에 지정하는 모든 조건이 충족되어야 합니다.

CIDR 표기법을 사용하여 IP 주소를 지정합니다. TCP/UDP를 프로토콜로 지정할 경우 소스 포트와 대상 포트를 지정해야 합니다. TCP나 UDP를 지정하는 경우에만 포트를 승인할 수 있습니다.

2. 인스턴스를 생성하고 초기화합니다. AuthorizeSecurityGroupIngressRequest withGroupName메서드를 사용하여 보안 그룹 이름을 지정하고 이전에 초기화한 IpPermission 객체를 다음과 같이 withIpPermissions메서드에 전달합니다.

```
AuthorizeSecurityGroupIngressRequest authorizeSecurityGroupIngressRequest =
    new AuthorizeSecurityGroupIngressRequest();

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

3. 다음과 같이 요청 객체를 authorizeSecurityGroupIngress 메서드에 전달합니다.

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

수신이 이미 승인된 IP 주소를 사용하여 authorizeSecurityGroupIngress를 호출하는 경우 이 메서드에서 예외가 발생합니다. AuthorizeSecurityGroupIngress를 호출하기 전에 다른 IP, 포트 및 프로토콜에 대한 수신을 승인할 새 IpPermission 객체를 생성하고 초기화합니다.

authorizeSecurityGroup인그레스 또는 authorizeSecurityGroup이그레스 메서드를 호출할 때마다 보안 그룹에 규칙이 추가됩니다.

키 페어 생성

EC2 인스턴스를 시작할 때 키 페어를 지정한 다음 해당 인스턴스에 연결할 때 키 페어의 프라이빗 키를 지정해야 합니다. 키 페어를 생성하거나 다른 인스턴스를 시작할 때 사용한 기존 키 페어를 사용할 수 있습니다. 자세한 내용은 Linux 인스턴스용 사용 설명서의 Amazon EC2 키 페어를 참조하세요.

1. CreateKeyPairRequest 인스턴스를 생성하고 초기화합니다. 다음과 같이 withKeyName 메서드를 사용하여 키 페어 이름을 설정합니다.

```
CreateKeyPairRequest createKeyPairRequest = new CreateKeyPairRequest();

createKeyPairRequest.withKeyName(keyName);
```

> **Important**
>
> 키 페어 이름은 고유해야 합니다. 키 이름이 기존 키 페어와 동일한 키 페어를 생성하려 할 경우 예외가 발생합니다.

2. 요청 객체를 createKeyPair 메서드로 전달합니다. 그러면 메서드는 다음과 같이 CreateKeyPairResult 인스턴스를 반환합니다.

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

3. 결과 객체의 getKeyPair 메서드를 호출하여 KeyPair 객체를 가져옵니다. 다음과 같이 `KeyPair` 객체의 getKeyMaterial 메서드를 호출하여 암호화되지 않은 PEM 인코딩된 프라이빗 키를 가져옵니다.

```
KeyPair keyPair = new KeyPair();

keyPair = createKeyPairResult.getKeyPair();

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

Amazon EC2 인스턴스 실행

다음 절차를 사용하여 동일한 Amazon 머신 이미지(AMI)에서 동일하게 구성된 하나 이상의 EC2 인스턴스를 시작합니다. EC2 인스턴스를 생성한 후 EC2 인스턴스의 상태를 확인할 수 있습니다. EC2 인스턴스를 실행한 후에는 해당 인스턴스에 연결할 수 있습니다.

1. RunInstancesRequest 인스턴스를 생성하고 초기화합니다. 지정하는 AMI, 키 페어 및 보안 그룹이 클라이언트 객체를 생성할 때 지정한 리전에 존재하는지 확인합니다.

```
RunInstancesRequest runInstancesRequest =
   new RunInstancesRequest();

runInstancesRequest.withImageId("ami-a9d09ed1")
```

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

withImageId

- AMI의 ID. Amazon에서 제공하는 퍼블릭 AMI 확인 방법을 알아보거나 사용자 자신의 표현식을 생성하려면 Amazon 머신 이미지(AMI)를 참조하십시오.

withInstanceType

- 지정한 AMI와 호환되는 인스턴스 유형. 자세한 내용은 Linux 인스턴스용 Amazon EC2 사용 설명서의 인스턴스 유형을 참조하세요.

withMinCount

- 시작할 최소 EC2 인스턴스 수. 이 값이 Amazon EC2가 대상 가용 영역에서 시작할 수 있는 인스턴스 수보다 많으면 Amazon EC2는 인스턴스를 시작하지 않습니다.

withMaxCount

- 시작할 최대 EC2 인스턴스 수. 이 값이 Amazon EC2가 대상 가용 영역에서 시작할 수 있는 인스턴스 수보다 많으면 Amazon EC2는 `MinCount`보다 많은 최대 가능 인스턴스 수를 시작합니다. 1과 인스턴스 유형에 대해 허용된 최대 인스턴스 수 사이에서 시작할 수 있습니다. 자세한 내용은 Amazon EC2 일반적인 FAQ에서 Amazon EC2에서 실행 가능한 인스턴스 수를 참조하세요.

withKeyName

- EC2 키 페어의 이름. 키 페어를 지정하지 않고 인스턴스를 시작하면 해당 인스턴스에 연결할 수 없습니다. 자세한 내용은 키 페어 생성 단원을 참조하십시오.

withSecurityGroups

- 하나 이상의 보안 그룹. 자세한 내용은 Amazon EC2 보안 그룹 생성을 참조하세요.

2. 요청 객체를 runInstances 메서드에 전달하여 인스턴스를 시작합니다. 그러면 이 메서드는 다음과 같이 RunInstancesResult 객체를 반환합니다.

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

인스턴스를 실행하고 나면 키 페어를 사용하여 해당 인스턴스에 연결할 수 있습니다. 자세한 정보는 Linux 인스턴스용 Amazon EC2 사용 설명서에서 Linux 인스턴스에 연결을 참조하세요.

IAM 역할을 사용하여 Amazon EC2의 AWS 리소스에 대한 액세스 권한 부여

Amazon Web Services(AWS)에 대한 모든 요청은 AWS가 발급한 자격 증명을 사용해 암호화 서명되어야 합니다. IAM 역할을 사용하여 Amazon EC2 인스턴스에서 AWS 리소스에 대한 보안 액세스 권한을 편리하게 부여할 수 있습니다.

이 주제에서는 Amazon EC2에서 실행 중인 Java SDK 애플리케이션과 함께 IAM 역할을 사용하는 방법에 대해 설명합니다. IAM 인스턴스에 대한 자세한 내용은 Linux 인스턴스용 Amazon EC2 사용 설명서에서 Amazon EC2의 IAM 역할을 참조하세요.

기본 공급자 체인 및 EC2 인스턴스 프로파일

애플리케이션이 기본 생성자를 사용하여 AWS 클라이언트를 생성하는 경우 클라이언트는 다음 순서대로 기본 자격 증명 공급자 체인을 사용하여 자격 증명을 검색합니다.

1. Java 시스템 속성 `aws.accessKeyId`와 `aws.secretKey`에서
2. 시스템 환경 변수 `AWS_ACCESS_KEY_ID` 및 `AWS_SECRET_ACCESS_KEY`에서
3. 기본 자격 증명 파일(이 파일의 위치는 플랫폼마다 다름)에서
4. `AWS_CONTAINER_CREDENTIALS_RELATIVE_URI` 환경 변수가 설정되어 있고 보안 관리자가 변수에 액세스할 수 있는 권한이 있는 경우 Amazon EC2 컨테이너 서비스를 통해 제공되는 자격 증명.
5. 인스턴스 프로파일 자격 증명(EC2 인스턴스의 IAM 역할과 연결된 인스턴스 메타데이터 안에 존재함)에서
6. 환경 또는 컨테이너의 웹 자격 증명 토큰 자격 증명

기본 공급자 체인의 인스턴스 프로파일 자격 증명 단계는 Amazon EC2 인스턴스에서 애플리케이션을 실행 중일 때만 사용할 수 있지만, Amazon EC2 인스턴스 작업 시 최상의 용이성과 최대 보안을 제공합니다. 또한 InstanceProfileCredentialsProvider 인스턴스를 클라이언트 생성자로 직접 전달하여 전체 기본 공급자 체인을 따라 진행하지 않고도 인스턴스 프로파일 자격 증명을 가져올 수 있습니다.

예:

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

이 접근 방법을 사용할 경우 SDK는 인스턴스 프로파일의 Amazon EC2 인스턴스에 대한 IAM 역할과 연결된 자격 증명과 동일한 권한을 가진 임시 AWS 자격 증명을 가져옵니다. 이러한 자격 증명은 임시

용이므로 기간이 경과되면 만료되지만, `InstanceProfileCredentialsProvider`는 가져온 자격 증명을 통해 계속 AWS에 액세스할 수 있도록 자격 증명을 정기적으로 새로 고칩니다.

> **Important**
>
> 자격 증명 자동 새로 고침은 기본 공급자 체인의 일부로 자체 `InstanceProfileCredentialsProvider`를 생성하는 기본 클라이언트 생성자를 사용하거나 `InstanceProfileCredentialsProvider` 인스턴스를 클라이언트 생성자에 직접 전달하는 경우에만 발생합니다. 다른 메서드를 사용하여 인스턴스 프로파일 자격 증명을 가져오거나 전달하는 경우 개발자 스스로 만료된 자격 증명을 확인하고 새로 고쳐야 합니다.

클라이언트 생성자가 자격 증명 공급자 체인을 사용하여 자격 증명을 찾을 수 없는 경우 AmazonClientException이 발생합니다.

연습: EC2 인스턴스에서 IAM 역할 사용

다음 연습에서는 IAM 역할을 사용하여 액세스를 관리하기 위해 Amazon S3 에서 객체를 가져오는 방법을 보여줍니다.

IAM 역할 생성

Amazon S3에 대한 읽기 전용 액세스를 부여하는 IAM 역할을 생성합니다.

1. IAM 콘솔(IAM console)을 엽니다.
2. 탐색 창에서 [Roles]를 클릭한 다음 [Create New Role]을 선택합니다.
3. 역할 이름을 입력한 다음 [Next Step]을 클릭합니다. 이 이름은 Amazon EC2 인스턴스를 시작할 때 필요하므로 꼭 기억해 두세요.
4. AWS 서비스 역할의 역할 유형 선택 페이지에서 Amazon EC2을 선택합니다.
5. 정책 템플릿 선택의 권한 설정 페이지에서 Amazon S3 읽기 전용 액세스를 선택한 후 다음 단계를 선택합니다.
6. [Review] 페이지에서 [Create Role]을 선택합니다.

EC2 인스턴스 시작과 IAM 역할 지정

Amazon EC2 콘솔이나 AWS SDK for Java를 사용하여 IAM 역할로 Amazon EC2 인스턴스를 시작할 수 있습니다.

- 콘솔을 사용하여 Amazon EC2 인스턴스를 시작하려면 Amazon EC2 Linux 인스턴스용 사용 설명서에서 Amazon EC2 Linux 인스턴스 시작하기에 나와 있는 지침을 따르세요.

  Review Instance Launch(인스턴스 시작 검토) 페이지가 표시되면 Edit instance details(인스턴스 세부 정보 편집)를 선택합니다. IAM 역할에서 이전에 생성한 IAM 역할을 선택합니다. 안내에 따라 절차를 완료합니다.

  > **Note**
  >
  > 보안 그룹을 생성하거나 기존 보안 그룹 및 키 페어를 사용하여 인스턴스에 연결해야 합니다.

- AWS SDK for Java를 사용하여 IAM 역할로 Amazon EC2 인스턴스를 시작하려면 Amazon EC2 인스턴스 실행 단원을 참조하세요.

애플리케이션 생성

EC2 인스턴스에서 실행할 샘플 애플리케이션을 빌드해 보겠습니다. 먼저 자습서 파일(예: `GetS3ObjectApp`)을 저장하는 데 사용할 수 있는 디렉터리를 생성합니다.

다음에는 AWS SDK for Java 라이브러리를 새로 생성한 디렉터리로 복사합니다. AWS SDK for Java를 `~/Downloads` 디렉터리로 다운로드한 경우 다음 명령을 사용하여 복사할 수 있습니다.

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

새 파일을 열어서 이름을 `GetS3Object.java`로 지정한 후 다음 코드를 추가합니다.

```
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();
  }
}
```

새 파일을 열어서 이름을 build.xml로 지정한 후 다음 행을 추가합니다.

```
<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>
```

수정된 프로그램을 빌드하고 실행합니다. 이 프로그램에는 자격 증명이 저장되어 있지 않습니다. 그러므로 이미 AWS 자격 증명을 지정한 경우가 아니면 코드는 AmazonServiceException를 발생시킵니다. 예:

```
$ 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
```

컴파일된 프로그램을 EC2 인스턴스로 전송

AWS SDK for Java 라이브러리와 함께 보안 복사( )를 사용하여 프로그램을 Amazon EC2 인스턴스로 전송합니다. 명령 순서는 다음과 유사합니다.

```
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**
>
> 사용한 Linux 배포판에 따라 사용자 이름은 "ec2-user", "root" 또는 "ubuntu"입니다. 인스턴스의 퍼블릭 DNS 이름을 가져오려면 EC2 콘솔을 열고 설명 탭에서 퍼블릭 DNS 값(예:`ec2-198-51-100-1.compute-1.amazonaws.com`) 을 찾으세요.

앞의 명령에서:

- `GetS3Object.class`는 컴파일된 프로그램입니다.
- `build.xml`는 프로그램을 빌드하고 실행하는 데 사용되는 ant 파일입니다.
- `lib` 및 `third-party` 디렉터리는 AWS SDK for Java의 해당 라이브러리 폴더입니다.
- `-r` 스위치는 `scp`가 AWS SDK for Java 배포판의 `library` 및 `third-party` 디렉터리에 들어 있는 모든 내용을 재귀적으로 복사하도록 지정합니다.
- `-p` 스위치는 `scp`가 소스 파일을 대상으로 복사할 때 소스 파일의 권한을 유지하도록 지정합니다.

> **Note**
>
> `-p` 스위치는 Linux, macOS 또는 Unix에서만 작동합니다. Windows에서 파일을 복사하려는 경우에는 다음 명령을 사용하여 인스턴스에서 파일 권한을 수정해야 할 수도 있습니다.

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

EC2 인스턴스에서 샘플 프로그램 실행

프로그램을 실행하려면 Amazon EC2 인스턴스에 연결합니다. 자세한 정보는 Linux 인스턴스용 Amazon EC2 사용 설명서에서 Linux 인스턴스에 연결을 참조하세요.

인스턴스에서 **`ant`** 를 사용할 수 없는 경우 다음 명령을 사용하여 설치합니다.

```
sudo yum install ant
```

그러고 나서, 다음과 같이 `ant`를 사용하여 프로그램을 실행합니다.

```
ant run
```

프로그램이 Amazon S3 객체의 내용을 명령 창에 씁니다.

자습서: Amazon EC2 스팟 인스턴스

개요

스폿 인스턴스를 사용하면 온디맨드 인스턴스 가격 대비 최대 90%까지 미사용 Amazon Elastic Compute Cloud(Amazon EC2) 용량을 입찰할 수 있으며 입찰 가격이 현재 스팟 가격을 초과하는 한 획득된 인스턴스를 실행할 수 있습니다. Amazon EC2는 공급과 수요를 기반으로 스팟 가격을 정기적으로 변경하며 입찰 가격이 스팟 가격과 일치하거나 초과하는 고객은 사용 가능한 스팟 인스턴스에 대한 액세스 권한을 얻습니다. 온디맨드 인스턴스 및 예약 인스턴스와 마찬가지로 스팟 인스턴스는 추가 컴퓨팅 파워를 얻기 위한 또 다른 옵션을 제공합니다.

스팟 인스턴스를 통해 배치 처리, 과학 연구, 이미지 처리, 동영상 인코딩, 데이터 및 웹 크롤링, 재무 분석, 테스트에 대한 Amazon EC2 비용을 크게 절감할 수 있습니다. 뿐만 아니라, 스팟 인스턴스를 사용하면 해당 용량이 긴급하게 필요하지 않은 상황에서 대량의 추가 용량에 액세스할 수 있습니다.

스팟 인스턴스를 사용하려면 인스턴스 시간당 지불하려는 최고 가격을 지정하는 스팟 인스턴스 요청을 배치해야 하는데, 이것이 바로 입찰입니다. 입찰 가격이 현재 스팟 가격을 초과하는 경우 요청이 이행되며 사용자가 인스턴스를 종료하거나 스팟 가격이 입찰 가격보다 높아질 때까지(둘 중에 더 빠른 것에 따라) 인스턴스가 실행됩니다.

다음 사항에 유의해야 합니다.

- 대개 입찰 가격보다 시간당 더 적은 금액을 지불합니다. Amazon EC2에서는 요청이 들어오고 사용 가능한 공급이 변화함에 따라 정기적으로 스팟 가격을 조정합니다. 입찰 가격이 더 높은지 여부와 상관없이 모든 사람이 해당 기간 동안 동일한 스팟 가격을 지불합니다. 따라서 자신의 입찰 가격보다 더 적게 지불할 수는 있어도 그보다 더 지불하는 경우는 있을 수 없습니다.
- 스팟 인스턴스를 실행 중이고 입찰 가격이 현재 스팟 가격을 충족 또는 초과하지 않는 경우 인스턴스는 종료됩니다. 이는 사용자가 워크로드와 애플리케이션이 이러한 편의적 용량을 활용할 수 있을 만큼 유연한지 확인하고자 한다는 것을 의미합니다.

스팟 인스턴스는 실행 중에 다른 Amazon EC2 인스턴스와 똑같이 기능을 수행하며, 다른 Amazon EC2 인스턴스와 마찬가지로 더 이상 필요하지 않을 때 종료할 수 있습니다. 인스턴스를 종료할 경우 부분적인 사용 시간에 대해 요금을 지불합니다(온디맨드 또는 예약 인스턴스의 경우와 마찬가지로). 그러나 스팟 가격이 입찰 가격을 초과하여 Amazon EC2에서 인스턴스를 종료하는 경우 부분적인 사용 시간에 대해서는 요금이 청구되지 않습니다.

이 자습서에서는 AWS SDK for Java를 사용하여 다음을 수행하는 방법을 보여줍니다.

- 스팟 요청 제출
- 스팟 요청이 이행되는 시점 결정
- 스팟 요청 취소
- 연결된 인스턴스 종료

필수 조건

이 자습서를 사용하려면 AWS SDK for Java가 설치되어 있고 기본 설치 사전 요구 사항이 충족되어야 합니다. 자세한 내용은 AWS SDK for Java 설정 단원을 참조하세요.

1단계: 자격 증명 설정

이 코드 샘플 사용을 시작하려면 AWS 자격 증명을 설정해야 합니다. 작업 방법에 대한 지침은 개발을 위한 AWS 자격 증명 및 리전 설정을 참조하세요.

> **Note**
>
> IAM 사용자 자격 증명을 사용하여 이러한 값을 제공하는 것이 좋습니다. 자세한 내용은 AWS에 가입 및 IAM 사용자 생성을 참조하세요.

이제 설정을 구성했으니 예제의 코드를 사용할 수 있습니다.

2단계: 보안 그룹 설정

보안 그룹은 인스턴스 그룹과 송수신이 허용되는 트래픽을 제어하는 방화벽 역할을 합니다. 기본적으로 인스턴스는 보안 그룹 없이 시작되므로 TCP 포트로 들어오는 모든 IP 트래픽이 거부됩니다. 따라서 스팟 요청을 제출하기 전에 필요한 네트워크 트래픽을 허용하는 보안 그룹을 설정하겠습니다. 이 자습서의 목적상, 애플리케이션을 실행 중인 IP 주소에서 Secure Shell(SSH) 트래픽을 허용하는 "GettingStarted"라는 새 보안 그룹을 생성하겠습니다. 새 보안 그룹을 설정하려면 보안 그룹을 프로그래밍 방식으로 설정하는 다음 코드 샘플을 포함하거나 실행해야 합니다.

`AmazonEC2` 클라이언트 객체를 생성한 후에는 "GettingStarted" 이름과 이 보안 그룹에 대한 설명을 사용하여 `CreateSecurityGroupRequest` 객체를 생성합니다. 그 다음에는 `ec2.createSecurityGroup` API를 호출하여 그룹을 생성합니다.

그룹에 대한 액세스를 활성화하기 위해 IP 주소 범위가 로컬 컴퓨터의 CIDR 표현으로 설정된 `ipPermission` 객체를 생성합니다. IP 주소의 "/10" 접미사는 지정된 IP 주소용 서브넷을 나타냅니다.

또한 TCP 프로토콜과 포트 22(SSH)를 사용하여 ipPermission 객체를 구성합니다. 마지막 단계는 보안 그룹의 이름과 ec2.authorizeSecurityGroupIngress 객체를 사용하여 ipPermission를 호출하는 것입니다.

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

이 애플리케이션을 한 번만 실행하면 새 보안 그룹이 생성됩니다.

또한 AWS Toolkit for Eclipse를 사용하여 보안 그룹을 생성할 수도 있습니다. 자세한 내용은 AWS Cost Explorer의 보안 그룹 관리를 참조하세요.

3단계: 스팟 요청 제출

스팟 요청을 제출하려면 먼저 사용하고자 하는 인스턴스 유형, Amazon 머신 이미지(AMI) 및 최대 입찰 가격을 결정해야 합니다. 또한 원하는 경우 인스턴스에 로그인할 수 있도록 이전에 구성했던 보안 그룹도 포함해야 합니다.

여러 가지 인스턴스 유형 중에 선택할 수 있습니다. 전체 목록은 Amazon EC2 인스턴스 유형을 참조하세요. 이 자습서에서는 사용 가능한 가장 저렴한 인스턴스 유형인 t1.micro를 사용하겠습니다. 다음에는 사용할 AMI 유형을 결정하겠습니다. ami-a9d09ed1을 사용하겠습니다. 이 유형은 이 자습서 작성 당시 사용 가능한 최신 Amazon Linux AMI입니다. 최신 AMI는 시간이 경과됨에 따라 변경될 수도 있지만, 언제든지 다음 단계를 수행하여 최신 버전의 AMI를 확인할 수 있습니다.

1. Amazon EC2 콘솔을 엽니다.
2. 인스턴스 실행 버튼을 선택합니다.
3. 첫 번째 Window는 사용 가능한 AMI를 표시합니다. AMI ID는 각 AMI 제목 옆에 표시됩니다. 또는 `DescribeImages` API를 사용할 수도 있지만, 이 명령의 활용은 이 자습서의 범위를 벗어납니다.

스팟 인스턴스에 대한 입찰 방법에는 여러 가지가 있습니다. 다양한 방법에 대한 간략한 설명을 보려면 Bidding for Spot Instances 비디오를 시청하세요. 그러나 시작하기 위해서는 세 가지 일반적인 전략, 즉 비용이 온디맨드 요금보다 적은 입찰, 그 결과 얻는 계산 값에 따른 입찰, 가능한 한 빨리 컴퓨팅 용량을 얻기 위한 입찰에 대해 설명이 필요합니다.

- 비용을 온디맨드 이하로 낮추기 실행하는 데 몇 시간 또는 몇 일이 걸릴 배치 처리 작업이 있습니다. 그러나 시작 및 완료 시점은 유연하게 선택할 수 있습니다. 그 작업을 온디맨드 인스턴스보다 적은

비용으로 완료할 수 있는지 확인할 수 있습니다. AWS Management Console 또는 Amazon EC2 API를 사용하여 인스턴스 유형에 대한 스팟 가격 기록을 살펴보세요. 자세한 내용은 스팟 가격 내역 보기를 참조하세요. 특정 가용 영역에서 원하는 인스턴스 유형의 가격 내역을 분석하였다면 입찰에 사용할 수 있는 다른 방법이 두 가지 있습니다.

  - 일회적인 스팟 요청이 이행되고 해당 작업을 완료하기에 충분한 연속적 컴퓨팅 시간 동안 실행될 가능성이 높을 것이라는 기대로 스팟 가격 범위의 상단(여전히 온디맨드 가격보다는 낮음)에서 입찰할 수 있습니다.
  - 또는 온디맨드 인스턴스 가격의 %로 지정하여 스팟 인스턴스에 지불할 의향이 있는 금액을 지정할 수 있습니다. 그리고 지속적인 요청을 통해 시간이 지남에 따라 시작된 많은 인스턴스를 결합할 계획입니다. 지정된 가격이 초과하면 스팟 인스턴스가 종료됩니다. (이 자습서 후반부에서 이 작업을 자동화하는 방법에 대해 설명할 것입니다.)
- 결과로 얻은 가치보다 더 많은 요금 지불 금지 실행할 데이터 처리 작업이 있습니다. 작업 당사자는 그 작업의 결과가 지니는 가치를 충분히 이해하여 컴퓨팅 비용 측면에서 그 결과의 가치가 얼마나 되는지 알 수 있습니다. 해당 인스턴스 유형의 스팟 가격 내역을 분석한 후 컴퓨팅 시간의 비용이 해당 작업의 결과가 지니는 가치보다 더 많지 않은 입찰 가격을 선택합니다. 영구 입찰을 생성하여 스팟 가격이 입찰 가격까지 또는 그 이하로 변동하는 과정에서 간헐적으로 실행되도록 합니다.
- 컴퓨팅 용량을 신속하게 획득 온디맨드 인스턴스를 통해서는 사용할 수 없는 추가 용량이 예기치 않게 단기간 동안 필요한 경우가 있습니다. 이때는 해당 인스턴스 유형의 스팟 가격 내역을 분석한 후 과거의 최고 입찰 가격보다 높게 입찰함으로써 요청이 신속하게 이행되고 완료될 때까지 컴퓨팅이 지속될 수 있는 가능성을 높입니다.

입찰 가격을 선택하고 나면 스팟 인스턴스를 요청할 준비가 된 것입니다. 본 자습서의 목적상 온디맨드 가격(0.03 USD)으로 입찰하여 입찰이 이행될 가능성을 최대화할 것입니다. Amazon EC2 요금 페이지를 참조하여 사용 가능한 인스턴스의 유형과 인스턴스의 온디맨드 요금을 결정할 수 있습니다. 스팟 인스턴스가 실행되는 동안 인스턴스가 실행되는 기간에 대한 스팟 가격만 지불합니다. 스팟 인스턴스 가격은 Amazon EC2에서 정하고, 스팟 인스턴스 용량의 장기적인 수요 공급 추세에 따라 점진적으로 조정됩니다. 또한 스팟 인스턴스에 대해 지불하려는 금액을 온디맨드 인스턴스 가격의 %로 지정할 수 있습니다. 스팟 인스턴스를 요청하려면 이전에 선택한 매개 변수를 사용하여 요청을 작성하면 됩니다. `RequestSpotInstanceRequest` 객체 생성부터 시작하겠습니다. 요청 객체에는 시작하고자 하는 인스턴스 수와 입찰 가격이 필요합니다. 뿐만 아니라, 요청에 대한 `LaunchSpecification`을 설정해야 하며, 여기에는 사용할 인스턴스 유형, AMI ID 및 보안 그룹이 포함됩니다. 요청을 작성했으면 `requestSpotInstances` 객체에 대해 `AmazonEC2Client` 메서드를 호출합니다. 다음 예제에서는 스팟 인스턴스를 요청하는 방법을 보여줍니다.

```
// 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);
```

이 코드를 실행하면 새로운 스팟 인스턴스 요청이 시작됩니다. 스팟 요청을 구성할 때 사용할 수 있는 다른 옵션도 있습니다. 자세한 내용은 자습서: 고급 Amazon EC2 스팟 요청 관리 또는 AWS SDK for Java API 참조의 RequestSpotInstances 클래스를 참조하세요.

> **Note**
>
> 실제로 시작되는 모든 스팟 인스턴스에 대해서는 요금이 부과되므로 모든 요청을 취소하고 시작하는 모든 인스턴스를 종료하여 관련된 모든 요금을 줄여야 합니다.

4단계: 스팟 요청 상태 확인

다음에는 마지막 단계로 이동하기 전에 스팟 요청이 "활성" 상태가 될 때까지 기다렸다가 코드를 생성하려고 합니다. 스팟 요청의 상태를 확인하기 위해 describeSpotInstanceRequests 메서드를 통해 모니터링할 스팟 요청 ID의 상태를 폴링합니다.

2단계에서 생성한 요청 ID가 requestSpotInstances 요청에 대한 응답에 포함됩니다. 다음 예제 코드에서는 requestSpotInstances 응답에서 요청 ID를 수집하고 이러한 요청 ID를 사용하여 ArrayList를 작성하는 방법을 보여줍니다.

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

요청 ID를 모니터링하려면 describeSpotInstanceRequests 메서드를 호출하여 요청의 상태를 확인합니다. 그런 다음 요청이 "열림" 이외의 상태가 될 때까지 반복합니다. 요청 인쇄에 문제가 있는 경우 요청이 곧바로 "닫힘" 상태가 될 수 있으므로 "열림" 이외의 상태(즉, "활성" 상태)를 모니터링합니다. 다음 코드 예제에서는 이 작업을 수행하는 방법을 자세히 보여줍니다.

```
// 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);
```

이 코드를 실행하고 나면 스팟 인스턴스 요청이 완료되거나, 오류로 인해 실패하고 오류가 화면으로 출력됩니다. 어느 경우든, 다음 단계로 이동하여 활성 요청을 모두 정리하고 실행 중인 인스턴스를 모두 종료할 수 있습니다.

5단계:: 스팟 요청과 인스턴스 정리

마지막으로 스팟 요청과 인스턴스를 정리해야 합니다. 대기 중인 요청을 모두 취소하고 인스턴스를 모두 종료하는 것이 중요합니다. 요청을 취소하는 것만으로는 인스턴스가 중지되지 않는데, 이는 인스턴스에 대해 계속해서 비용이 부과됨을 뜻합니다. 인스턴스를 종료하면 스팟 요청이 취소될 수 있지만, 인스턴스 종료만으로는 재이행되고 있는 요청을 중지할 수 없는 경우 영구에는 입찰을 사용하는 것과

같은 시나리오도 가능합니다. 따라서 활성화 상태의 입찰을 모두 취소하고 실행 중인 인스턴스를 모두 종료하는 것이 가장 좋습니다.

다음 코드는 요청을 취소하는 방법을 보여줍니다.

```
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());
}
```

대기 중인 인스턴스를 모두 종료하려면 요청과 연결되어 있으며 요청을 시작했던 인스턴스 ID가 필요합니다. 다음 코드 예제는 인스턴스를 모니터링하는 원래 코드를 사용하고 `ArrayList` 응답과 연결된 인스턴스 ID를 저장했던 `describeInstance`를 추가합니다.

```
// 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);
```

ArrayList에 저장된 인스턴스 ID와 다음 코드 조각을 사용하여 실행 중인 인스턴스를 종료합니다.

```
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());
    }
```

모두 종합

지금까지의 작업을 모두 종합하여, 앞에서 살펴봤던 단계(EC2 클라이언트 초기화, 스팟 요청 제출, 스팟 요청이 더 이상 열림 상태가 아닌 경우 확인, 모든 활성 스팟 요청 및 연결된 인스턴스 정리)를 결합한 보다 객체 지향적인 방법을 제공합니다. 이러한 작업을 수행하는 `Requests` 클래스를 생성하겠습니다.

또한 상위 수준의 함수 호출을 수행하는 기본 메서드가 있는 `GettingStartedApp` 클래스도 생성합니다. 특히 이전에 설명한 `Requests` 객체를 초기화하겠습니다. 스팟 인스턴스 요청을 제출합니다. 그런 다음 스팟 요청이 "활성" 상태가 될 때까지 기다립니다. 마지막으로 요청과 인스턴스를 정리합니다.

이 예제의 전체 소스 코드는 GitHub에서 확인하거나 다운로드할 수 있습니다.

축하합니다! AWS SDK for Java를 사용한 스팟 인스턴스 소프트웨어 개발 관련 시작하기 자습서를 완료했습니다.

다음 단계

자습서: 고급 Amazon EC2 스팟 요청 관리로 이동하세요.

자습서: 고급 Amazon EC2 스팟 요청 관리

Amazon EC2 스팟 인스턴스를 통해 미사용 Amazon EC2 용량에 입찰하고 입찰가가 현재 스팟 가격을 초과하는 경우 해당 인스턴스를 실행할 수 있습니다. Amazon EC2는 공급 및 수요에 기초하여 스팟 가격을 정기적으로 변경합니다. 스팟 인스턴스에 대한 자세한 내용은 Linux 인스턴스용 Amazon EC2 사용 설명서의 Spot Instances를 참조하세요.

필수 조건

이 자습서를 사용하려면 AWS SDK for Java가 설치되어 있고 기본 설치 사전 요구 사항이 충족되어야 합니다. 자세한 내용은 AWS SDK for Java 설정을 참조하세요.

자격 증명 설정

이 코드 샘플 사용을 시작하려면 AWS 자격 증명을 설정해야 합니다. 작업 방법에 대한 지침은 개발을 위한 AWS 자격 증명 및 리전 설정을 참조하세요.

> **Note**
>
> IAM 사용자 자격 증명을 사용하여 이러한 값을 제공하는 것이 좋습니다. 자세한 내용은 AWS에 가입 및 IAM 사용자 생성을 참조하세요.

이제 설정을 구성했으니 예제의 코드를 사용할 수 있습니다.

보안 그룹 설정

보안 그룹은 인스턴스 그룹과 송수신이 허용되는 트래픽을 제어하는 방화벽 역할을 합니다. 기본적으로 인스턴스는 보안 그룹 없이 시작되므로 TCP 포트로 들어오는 모든 IP 트래픽이 거부됩니다. 따라서 스팟 요청을 제출하기 전에 필요한 네트워크 트래픽을 허용하는 보안 그룹을 설정하겠습니다. 이 자습서의 목적상, 애플리케이션을 실행 중인 IP 주소에서 Secure Shell(SSH) 트래픽을 허용하는 "GettingStarted"라는 새 보안 그룹을 생성하겠습니다. 새 보안 그룹을 설정하려면 보안 그룹을 프로그래밍 방식으로 설정하는 다음 코드 샘플을 포함하거나 실행해야 합니다.

`AmazonEC2` 클라이언트 객체를 생성한 후에는 "GettingStarted" 이름과 이 보안 그룹에 대한 설명을 사용하여 `CreateSecurityGroupRequest` 객체를 생성합니다. 그 다음에는 `ec2.createSecurityGroup` API를 호출하여 그룹을 생성합니다.

그룹에 대한 액세스를 활성화하기 위해 IP 주소 범위가 로컬 컴퓨터의 CIDR 표현으로 설정된 `ipPermission` 객체를 생성합니다. IP 주소의 "/10" 접미사는 지정된 IP 주소용 서브넷을 나타냅니다. 또한 TCP 프로토콜과 포트 22(SSH)를 사용하여 `ipPermission` 객체를 구성합니다. 마지막 단계는 보안 그룹의 이름과 `ec2 .authorizeSecurityGroupIngress` 객체를 사용하여 `ipPermission`를 호출하는 것입니다.

(다음 코드는 첫 번째 자습서에 사용된 것과 동일한 코드입니다.)

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

advanced.CreateSecurityGroupApp.java 코드 샘플에서 이 코드 샘플의 전체 내용을 볼 수 있습니다. 이 애플리케이션을 한 번만 실행하면 새 보안 그룹이 생성됩니다.

> **Note**
>
> 또한 AWS Toolkit for Eclipse를 사용하여 보안 그룹을 생성할 수도 있습니다. 자세한 내용은 AWS Toolkit for Eclipse 사용 설명서의 AWS Cost Explorer에서 보안 그룹 관리를 참조하세요.

세부 스팟 인스턴스 요청 생성 옵션

자습서: Amazon EC2 스팟 인스턴스에서 설명한 대로 인스턴스 유형, Amazon 머신 이미지(AMI) 및 최대 입찰 가격을 사용하여 요청을 작성해야 합니다.

RequestSpotInstanceRequest 객체 생성부터 시작하겠습니다. 요청 객체에는 원하는 인스턴스 수와 입찰 가격이 필요합니다. 뿐만 아니라, 요청에 대한 LaunchSpecification을 설정해야 하며, 여기에는 사용할 인스턴스 유형, AMI ID 및 보안 그룹이 포함됩니다. 요청을 작성한 후에는 requestSpotInstances 객체에 대해 AmazonEC2Client 메서드를 호출합니다. 다음 예는 스팟 인스턴스를 요청하는 방법을 보여줍니다.

(다음 코드는 첫 번째 자습서에 사용된 것과 동일한 코드입니다.)

```
// 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);
```

영구 요청 vs. 일회성 요청

스팟 요청을 작성할 때 여러 선택적 매개 변수를 지정할 수 있습니다. 첫 번째는 일회성 요청인지 영구 요청인지 여부를 지정합니다. 기본적으로 요청은 일회성 요청입니다. 일회성 요청은 한 번만 이행될 수 있으며, 요청한 인스턴스가 종료된 후에는 요청이 종결됩니다. 영구 요청은 동일 요청에 대해 실행 중인 스팟 인스턴스가 없을 때마다 이행해야 할 요청으로 간주됩니다. 이 요청 유형을 지정하려면 스팟 요청에 대해 이 유형을 설정하기만 하면 됩니다. 다음 코드를 사용하여 설정할 수 있습니다.

```
// 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);
```

요청 기간 제한

필요에 따라 요청이 유효 상태로 유지될 시간 길이를 지정할 수도 있습니다. 이 기간의 시작 및 종료 시간을 둘 다 지정할 수 있습니다. 기본적으로 스팟 요청은 생성된 시점부터 이행되거나 요청자가 취소할 때까지 이행해야 할 요청으로 간주됩니다. 하지만 필요한 경우 유효 기간을 제한할 수도 있습니다. 이 기간을 지정하는 방법의 예는 다음 코드에 나와 있습니다.

```
// 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);
```

Amazon EC2 스팟 인스턴스 요청 그룹화

여러 가지 방식으로 스팟 인스턴스 요청을 그룹화할 수 있습니다. 시작 그룹, 가용 영역 그룹 및 배치 그룹을 사용할 때의 이점에 대해 살펴보겠습니다.

스팟 인스턴스가 모두 함께 시작 및 종료되도록 하려는 경우 시작 그룹을 활용하면 됩니다. 시작 그룹은 입찰 세트를 함께 그룹화하는 레이블입니다. 시작 그룹에 있는 모든 인스턴스가 함께 시작되고 종료됩니다. 시작 그룹에 포함된 인스턴스가 이미 이행된 경우 동일한 시작 그룹과 함께 시작된 새 인스턴스도 이행될지는 장담할 수 없습니다. 시작 그룹을 설정하는 방법의 예는 다음 코드 예제에 나와 있습니다.

```
// 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);
```

인스턴스를 구분하지 않고 요청 안에 포함된 모든 인스턴스를 동일한 가용 영역에서 시작되도록 하려는 경우 가용 영역 그룹을 활용할 수 있습니다. 가용 영역 그룹은 동일한 가용 영역 안에서 인스턴스 세트를 함께 그룹화하는 레이블입니다. 가용 영역 그룹을 공유하고 동시에 이행되는 모든 인스턴스는 동일한 가용 영역에서 시작됩니다. 다음 예는 가용 영역 그룹을 설정하는 방법을 보여줍니다.

```
// 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);
```

스팟 인스턴스에 사용할 가용 영역을 지정할 수 있습니다. 다음 코드 예제는 가용 영역을 설정하는 방법을 보여줍니다.

```
// 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);
```

마지막으로 클러스터 컴퓨팅 인스턴스나 클러스터 GPU 인스턴스 같은 HPC(고성능 컴퓨팅) 스팟 인스턴스를 사용하려는 경우 배치 그룹을 지정할 수 있습니다. 배치 그룹은 인스턴스 간에 있어 낮은 지연 시간과 높은 대역폭 연결성을 제공합니다. 다음 예는 배치 그룹을 설정하는 방법을 보여줍니다.

```
// 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);
```

이 단원에 표시된 모든 매개 변수는 선택적 매개 변수입니다. 또한 입찰이 일회성인지 지속적인지를 제외하고 대부분의 매개 변수가 입찰 이행 가능성을 줄일 수 있다는 점을 인식하는 것도 중요합니다. 그러므로 꼭 필요한 경우에만 이러한 옵션을 활용해야 합니다. 앞에 제시된 코드 예제는 모두 하나의 긴 코드 샘플로 결합되어 있으며 `com.amazonaws.codesamples.advanced.InlineGettingStartedCodeSampleApp.java` 클래스에서 확인할 수 있습니다.

중단 또는 종료 후 루트 파티션을 지속하는 방법

스팟 인스턴스 중단을 관리하는 가장 쉬운 방법 중 하나는 데이터가 정기적으로 Amazon Elastic Block Store(Amazon Amazon EBS) 볼륨으로 체크포인트되도록 하는 것입니다. 정기적으로 검사하여 중단이 있는 경우 마지막 체크포인트 이후에 생성된 데이터만 손실됩니다(그 사이에 다른 비 idempotent 작업이 수행되지 않는다고 가정할 때). 이 프로세스를 보다 쉽게 처리하기 위해 루트 파티션이 중단 또는 종료 시 삭제되지 않도록 스팟 요청을 구성할 수 있습니다. 다음 예제에는 이 시나리오를 활성화하는 방법을 보여주는 새 코드를 삽입했습니다.

추가된 코드에서는 `BlockDeviceMapping` 객체를 생성하고 관련 Amazon Elastic Block Store(Amazon EBS)를 스팟 인스턴스가 종료되는 경우 삭제되지 않도록 not로 구성된 Amazon EBS 객체로 설정합니다. 그런 다음 이 `BlockDeviceMapping`을 시작 사양에 포함할 매핑 ArrayList에 추가합니다.

```
// 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);
```

시작 시 인스턴스에 이 볼륨을 다시 연결하려 한다고 가정할 때, 블록 디바이스 매핑 설정도 사용할 수 있습니다. 또는 루트가 아닌 파티션을 연결한 경우 스팟 인스턴스가 재개된 후 연결하려는 Amazon Amazon EBS 볼륨을 지정할 수 있습니다. 이렇게 하려면 `EbsBlockDevice` 객체에서 스냅샷 ID를 지정하고 `BlockDeviceMapping` 객체에서 대체 장치 이름을 지정하면 됩니다. 블록 디바이스 매핑을 활용하면 인스턴스를 좀 더 쉽게 부트스트래핑할 수 있습니다.

루트 파티션을 사용하여 중요 데이터를 검사하는 것은 인스턴스 중단 가능성을 관리하는 좋은 방법입니다. 중단 가능성 관리에 대한 그 밖의 방법은 Managing Interruption 비디오를 참조하십시오.

스팟 요청 및 인스턴스에 태그를 지정하는 방법

Amazon EC2 리소스에 태그를 추가하면 클라우드 인프라 관리를 간소화할 수 있습니다. 메타데이터 형태의 태그를 사용해 사용자 친화적인 이름을 만들고, 검색 능력을 강화하며, 여러 사용자 간의 조정을 개선할 수 있습니다. 태그를 사용하여 프로세스의 부분과 스크립트를 자동화할 수도 있습니다. Amazon EC2 리소스 태깅에 대해 더 읽어보려면 Linux 인스턴스용 Amazon EC2 사용 설명서의 태그 사용을 참조하세요.

요청 태그 지정

스팟 요청에 태그를 추가하려면 요청된 이후에 태그를 지정해야 합니다. `requestSpotInstances()`의 반환 값이 태그 지정용 스팟 요청 ID를 가져오는 데 사용할 수 있는 RequestSpotInstancesResult 객체를 제공합니다.

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

ID를 얻고 나면 이 ID를 CreateTagsRequest에 추가하고 Amazon EC2 클라이언트의 `createTags()` 메서드를 호출하여 요청에 태그를 지정할 수 있습니다.

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

인스턴스 태그 지정

스팟 요청 자체와 마찬가지로, 인스턴스에도 생성된 후에만 태그를 지정할 수 있습니다. 즉, 태그 지정은 스팟 요청이 충족되고 나서만(더 이상 열림 상태가 아닌 경우)만 발생합니다.

DescribeSpotInstanceRequestsRequest 객체를 사용하여 Amazon EC2 클라이언트의 `describeSpotInstanceRequests()` 메서드를 호출하여 요청 상태를 확인할 수 있습니다. 반환된

DescribeSpotInstanceRequestsResult 객체에는 스팟 요청의 상태를 쿼리하고 더 이상 열린 상태가 아닌 경우 해당 인스턴스 ID를 가져오는 데 사용할 수 있는 SpotInstanceRequest 객체 목록이 포함되어 있습니다.

스팟 요청이 더 이상 열려 있지 않으면 `getInstanceId()` 메서드를 호출하여 `SpotInstanceRequest` 객체에서 인스턴스 ID를 가져올 수 있습니다.

```
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);
```

이제 반환된 인스턴스에 태그를 지정할 수 있습니다.

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

스팟 요청 취소 및 인스턴스 종료

스팟 요청 취소

스팟 인스턴스 요청을 취소하려면 CancelSpotInstanceRequestsRequest 객체를 사용하여 Amazon EC2 클라이언트에서 `cancelSpotInstanceRequests`를 호출하세요.

```
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());
}
```

스팟 인스턴스 종료

ID를 Amazon EC2 client의 `terminateInstances()` 메서드에 전달하여 실행 중인 스팟 인스턴스를 종료할 수 있습니다.

```
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());
}
```

모두 종합

지금까지의 작업을 모두 종합하여, 이 자습서에 표시된 단계를 사용하기 쉬운 단일 클래스로 결합하는 보다 객체 지향적인 방법을 제공하려고 합니다. 이러한 작업을 수행하는 `Requests` 클래스를 인스턴스화하겠습니다. 또한 상위 수준의 함수 호출을 수행하는 기본 메서드가 있는 `GettingStartedApp` 클래스도 생성합니다.

이 예제의 전체 소스 코드는 GitHub에서 확인하거나 다운로드할 수 있습니다.

축하합니다! AWS SDK for Java를 사용한 스팟 인스턴스 소프트웨어 개발 관련 고급 요청 기능 자습서를 완료했습니다.

Amazon EC2 인스턴스 관리

인스턴스 생성

AmazonEC2Client의 `runInstances` 메서드를 호출하여 새 Amazon EC2 인스턴스를 생성하고, 사용할 Amazon 머신 이미지(AMI)와 인스턴스 유형이 포함된 RunInstancesRequest를 제공합니다.

가져오기

```
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;
```

코드

```
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();
```

전체 예제를 참조하십시오.

인스턴스 시작

Amazon EC2 인스턴스를 시작하려면 AmazonEC2Client의 `startInstances` 메서드를 호출하고 시작할 인스턴스의 ID가 포함된 StartInstancesRequest를 이 메서드에 제공합니다.

가져오기

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

코드

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

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

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

전체 예제를 참조하십시오.

인스턴스 중지

Amazon EC2 인스턴스를 중지하려면 AmazonEC2Client의 `stopInstances` 메서드를 호출하고 중지할 인스턴스의 ID가 포함된 StopInstancesRequest를 이 메서드에 제공합니다.

가져오기

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

코드

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

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

ec2.stopInstances(request);
```

전체 예제를 참조하십시오.

인스턴스 재부팅

Amazon EC2 인스턴스를 재부팅하려면 AmazonEC2Client의 `rebootInstances` 메서드를 호출하고 재부팅할 인스턴스의 ID가 포함된 RebootInstancesRequest를 이 메서드에 제공합니다.

가져오기

```
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;
```

코드

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

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

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

전체 예제를 참조하십시오.

인스턴스 설명

인스턴스를 나열하려면 DescribeInstancesRequest를 생성하고 AmazonEC2Client의 `describeInstances` 메서드를 호출합니다. 그러면 계정과 지역의 Amazon EC2 인스턴스를 나열하는 데 사용할 수 있는 DescribeInstancesResult 객체가 반환됩니다.

인스턴스는 예약별로 그룹화됩니다. 각 예약은 인스턴스를 시작하는 `startInstances` 호출에 해당합니다. 인스턴스를 나열하려면 먼저 반환된 각 Reservation 객체에서 `DescribeInstancesResult` 클래스의 `getReservations' method, and then call `getInstances`를 호출해야 합니다.

가져오기

```
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;
```

코드

```
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;
    }
}
```

결과가 페이징됩니다. 결과 객체의 `getNextToken` 메서드에서 반환된 값을 원래 요청 객체의 `setNextToken` 메서드로 전달하고 다음 번 `describeInstances` 호출에서 동일한 요청 객체를 사용함으로써 추가 결과를 가져올 수 있습니다.

전체 예제를 참조하십시오.

인스턴스 모니터링

CPU와 네트워크 사용률, 사용 가능한 메모리 및 남은 디스크 공간 등 Amazon EC2 인스턴스의 다양한 측면을 모니터링할 수 있습니다. 인스턴스 모니터링에 대한 자세한 내용은 Linux 인스턴스용 Amazon EC2 사용 설명서의 Amazon EC2 모니터링을 참조하세요.

인스턴스 모니터링을 시작하려면 모니터링할 인스턴스의 ID로 MonitorInstancesRequest를 생성하고 AmazonEC2Client의 `monitorInstances` 메서드에 전달해야 합니다.

가져오기

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

코드

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

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

ec2.monitorInstances(request);
```

전체 예제를 참조하십시오.

인스턴스 모니터링 중지

인스턴스 모니터링을 중지하려면 모니터링을 중지할 인스턴스의 ID로 UnmonitorInstancesRequest를 생성하고 AmazonEC2Client의 `unmonitorInstances` 메서드에 전달합니다.

가져오기

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

코드

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

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

ec2.unmonitorInstances(request);
```

전체 예제를 참조하십시오.

추가 정보

- Amazon EC2 API 참조에서 RunInstances
- Amazon EC2 API 참조에서 DescribeInstances
- Amazon EC2 API 참조에서 StartInstances
- Amazon EC2 API 참조에서 StopInstances
- Amazon EC2 API 참조에서 RebootInstances

- Amazon EC2 API 참조에서 MonitorInstances
- Amazon EC2 API 참조에서 UnmonitorInstances

에서 엘라스틱 IP 주소 사용 Amazon EC2

EC2-Classic은 더 이상 사용되지 않습니다.

> **Warning**
>
> EC2-Classic은 2022년 8월 15일에 사용 중지될 예정입니다. EC2-Classic에서 VPC로 마이그레이션하는 것이 좋습니다. 자세한 내용은 Amazon EC2 사용 설명서 또는 Amazon EC2 사용 설명서의 EC2-Classic에서 VPC로 마이그레이션을 참조하십시오. 또 블로그 게시물 EC2-Classic 네트워킹은 사용 중지됩니다 - 준비 방법은 다음과 같습니다를 참조하세요.

탄력적 IP 주소 할당

탄력적 IP 주소를 사용하려면 먼저 계정에 주소를 할당한 후 인스턴스 또는 네트워크 인터페이스와 연결합니다.

엘라스틱 IP 주소를 할당하려면 네트워크 유형 (클래식 EC2 또는 VPC) 을 포함하는 AllocateAddressRequest객체를 사용하여 AmazonEC2Client의 `allocateAddress` 메서드를 호출합니다.

반환된 AllocateAddressResult항목에는 할당 ID와 인스턴스 ID를 Amazon EC2Client의 메서드에 전달하여 주소를 인스턴스와 연결하는 데 사용할 수 있는 할당 ID가 포함되어 있습니다. AssociateAddressRequest`associateAddress`

가져오기

```
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;
```

코드

```
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);
```

전체 예제를 참조하세요.

탄력적 IP 주소 설명

계정에 지정된 탄력적 IP 주소를 나열하려면 AmazonEC2Client의 `describeAddresses` 메서드를 호출합니다. 계정의 엘라스틱 IP 주소를 나타내는 주소 객체 목록을 가져오는 데 사용할 수 있는 코드를 반환합니다. DescribeAddressesResult

가져오기

```
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;
```

코드

```
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());
}
```

전체 예제를 참조하세요.

탄력적 IP 주소 해제

엘라스틱 IP 주소를 릴리스하려면 AmazonEC2Client의 `releaseAddress` 메서드를 호출하여 릴리스하려는 엘라스틱 IP 주소의 할당 ID가 ReleaseAddressRequest포함된 메서드를 전달하십시오.

가져오기

```
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;
```

코드

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

ReleaseAddressRequest request = new ReleaseAddressRequest()
    .withAllocationId(alloc_id);

ReleaseAddressResult response = ec2.releaseAddress(request);
```

엘라스틱 IP 주소를 해제하면 AWS IP 주소 풀로 릴리스되며 이후에는 사용할 수 없게 될 수 있습니다. 해당 주소와 통신하는 모든 서버 또는 장치와 DNS 레코드를 업데이트해야 합니다. 이미 릴리스한 엘라스틱 IP 주소를 릴리스하려고 하면 해당 주소가 이미 다른 엘라스틱 IP 주소에 할당되어 있으면 AuthFailure오류가 발생합니다. AWS 계정

EC2-Classic 또는 기본 VPC를 사용하려는 경우에는 탄력적 IP 주소를 릴리스하면 연결되어 있는 인스턴스에서 연결 해제됩니다. 탄력적 IP 주소를 릴리스하지 않고 연결을 해제하려면 AmazonEC2Client의 `disassociateAddress` 메서드를 사용합니다.

기본이 아닌 VPC를 사용하는 경우, 릴리스하기 전에 반드시 `disassociateAddress`를 사용해 탄력적 IP 주소를 연결 해제합니다. 그렇지 않으면 오류를 Amazon EC2 반환합니다 (잘못된 IP 주소). InUse).

전체 예제를 참조하세요.

추가 정보

- Linux 인스턴스용 Amazon EC2 사용 설명서의 엘라스틱 IP 주소
- AllocateAddress Amazon EC2 API 레퍼런스에서
- DescribeAddresses Amazon EC2 API 레퍼런스에서
- ReleaseAddress Amazon EC2 API 레퍼런스에서

리전 및 가용 영역 사용

리전 설명

계정에 사용할 수 있는 리전을 나열하려면 AmazonEC2Client의 `describeRegions` 메서드를 호출합니다. 이 메서드는 DescribeRegionsResult를 반환합니다. 반환된 객체의 `getRegions` 메서드를 호출하여 각 리전을 나타내는 Region 객체의 목록을 가져옵니다.

가져오기

```
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;
```

코드

```
DescribeRegionsResult regions_response = ec2.describeRegions();

for(Region region : regions_response.getRegions()) {
    System.out.printf(
        "Found region %s " +
        "with endpoint %s",
        region.getRegionName(),
        region.getEndpoint());
```

```
}
```

전체 예제를 참조하십시오.

가용 영역 설명

계정에 사용할 수 있는 각 가용 영역을 나열하려면 AmazonEC2Client의 `describeAvailabilityZones` 메서드를 호출합니다. 이 메서드는 DescribeAvailabilityZonesResult를 반환합니다. `getAvailabilityZones` 메서드를 호출하여 각 가용 영역을 나타내는 AvailabilityZone 객체의 목록을 가져옵니다.

가져오기

```
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;
```

코드

```
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());
}
```

전체 예제를 참조하십시오.

계정 설명

계정을 설명하려면 AmazonEC2Client의 `describeAccountAttributes` 메서드를 호출합니다. 이 메서드는 DescribeAccountAttributesResult 객체를 반환합니다. 이 객체 `getAccountAttributes` 메

서드를 호출하여 AccountAttribute 객체 목록을 확보합니다. 목록을 반복하여 AccountAttribute 객체를 검색할 수 있습니다.

AccountAttribute 객체의 `getAttributeValues` 메서드를 호출하여 사용자의 속성 값을 확보할 수 있습니다. 이 메서드는 AccountAttributeValue 객체 목록을 반환합니다. 이 두 번째 목록을 반복하여 속성 값을 표시할 수 있습니다(다음 코드 예제 참조).

가져오기

```
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;
```

코드

```
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();
}
```

GitHub의 전체 예제를 참조하십시오.

추가 정보

- Linux 인스턴스용 Amazon EC2 사용 설명서의 리전 및 가용 영역
- Amazon EC2 API 참조에서 DescribeRegions
- Amazon EC2 API 참조에서 DescribeAvailabilityZones

Amazon EC2 키 페어 작업

키 페어 만들기

키 페어를 생성하려면 키 이름이 포함된 CreateKeyPairRequest를 사용하여 Amazon EC2Client의 `createKeyPair` 메서드를 호출하세요.

가져오기

```
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;
```

코드

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

CreateKeyPairRequest request = new CreateKeyPairRequest()
    .withKeyName(key_name);

CreateKeyPairResult response = ec2.createKeyPair(request);
```

전체 예제를 참조하십시오.

키 페어 설명

키 페어를 나열하거나 키 페어에 대한 정보를 가져오려면 AmazonEC2Client의 `describeKeyPairs` 메서드를 호출합니다. 이 메서드는 DescribeKeyPairsResult를 반환하는데, 여기서 `getKeyPairs` 메서드를 호출하여 키 페어 목록에 액세스할 수 있습니다. 그러면 KeyPairInfo 객체 목록이 반환됩니다.

가져오기

```
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;
```

코드

```
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());
}
```

전체 예제를 참조하십시오.

키 페어 삭제

키 페어를 삭제하려면 AmazonEC2Client의 `deleteKeyPair` 메서드를 호출하고 삭제할 키 페어 이름이 포함된 DeleteKeyPairRequest에 이 메서드를 전달합니다.

가져오기

```
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;
```

코드

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

DeleteKeyPairRequest request = new DeleteKeyPairRequest()
    .withKeyName(key_name);

DeleteKeyPairResult response = ec2.deleteKeyPair(request);
```

전체 예제를 참조하십시오.

추가 정보

- Linux 인스턴스용 Amazon EC2 사용 설명서의 Amazon EC2 키 페어
- Amazon EC2 API 참조에서 CreateKeyPair
- Amazon EC2 API 참조에서 DescribeKeyPairs
- Amazon EC2 API 참조에서 DeleteKeyPair

Amazon EC2의 보안 그룹 작업

보안 그룹 생성

보안 그룹을 생성하려면 키 이름이 포함된 CreateSecurityGroupRequest를 사용하여 AmazonEC2Client의 `createSecurityGroup` 메서드를 호출하세요.

가져오기

```
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;
```

코드

```
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);
```

전체 예제를 참조하십시오.

보안 그룹 구성

보안 그룹은 Amazon EC2 인스턴스에 대한 인바운드(수신) 및 아웃바운드(송신) 트래픽을 모두 제어할 수 있습니다.

보안 그룹에 인그레스 규칙을 추가하려면 AmazonEC2Client의 `authorizeSecurityGroupIngress` 메서드를 사용하여 AuthorizeSecurityGroupIngressRequest 객체 내에 보안 그룹의 이름과 보안 그룹에 할당하려는 액세스 규칙(IpPermission)을 입력합니다. 다음 예제에서는 IP 권한을 보안 그룹에 추가하는 방법을 보여줍니다.

가져오기

```
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;
```

코드

```
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);
```

이그레스 규칙을 보안 그룹에 추가하려면 AuthorizeSecurityGroupEgressRequest의 유사한 데이터를 AmazonEC2Client의 `authorizeSecurityGroupEgress` 메서드에 제공합니다.

전체 예제를 참조하십시오.

보안 그룹 설명

보안 그룹을 설명하거나 보안 그룹에 대한 정보를 가져오려면 AmazonEC2Client의 `describeSecurityGroups` 메서드를 호출합니다. 이 메서드는 DescribeSecurityGroupsResult를 반환하는데, 이를 사용하여 `getSecurityGroups` 메서드를 호출하여 보안 그룹 목록에 액세스할 수 있습니다. 그러면 SecurityGroup 객체 목록이 반환됩니다.

가져오기

```
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;
```

코드

```
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];
```

전체 예제를 참조하십시오.

보안 그룹 삭제

보안 그룹을 삭제하려면 AmazonEC2Client의 `deleteSecurityGroup` 메서드를 호출하여 삭제할 보안 그룹의 ID가 포함된 DeleteSecurityGroupRequest를 이 메서드에 전달합니다.

가져오기

```
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;
```

코드

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

DeleteSecurityGroupRequest request = new DeleteSecurityGroupRequest()
    .withGroupId(group_id);

DeleteSecurityGroupResult response = ec2.deleteSecurityGroup(request);
```

전체 예제를 참조하십시오.

추가 정보

- Linux 인스턴스용 Amazon EC2 사용 설명서의 Amazon EC2 보안 그룹
- Linux 인스턴스용 Amazon EC2 사용 설명서에서 Linux 인스턴스의 인바운드 트래픽 승인
- Amazon EC2 API 참조에서 CreateSecurityGroup
- Amazon EC2 API 참조에서 DescribeSecurityGroups
- Amazon EC2 API 참조에서 DeleteSecurityGroup
- Amazon EC2 API 참조에서 AuthorizeSecurityGroupIngress

AWS SDK for Java를 사용하는 IAM 예제

이 단원에서는 AWS SDK for Java를 사용한 IAM 프로그래밍의 예제를 제공합니다.

AWS Identity and Access Management(IAM)를 사용하면 사용자의 AWS 서비스 및 리소스에 대한 액세스를 안전하게 제어할 수 있습니다. IAM을 사용하여 AWS 사용자 및 그룹을 만들고 관리하며 AWS 리소스에 대한 액세스를 허용 및 거부할 수 있습니다. IAM에 대한 전체 설명서는 IAM 사용 설명서를 참조하세요.

> **Note**
>
> 예제에는 각 기술을 보여주는 데 필요한 코드만 포함되어 있습니다. 전체 예제 코드는 GitHub에 있습니다. 이 위치에서 단일 소스 파일을 다운로드하거나 리포지토리를 로컬로 복사하여 모든 예제를 빌드하고 실행할 수 있습니다.

주제

IAM 액세스 키 관리

액세스 키 생성

IAM 액세스 키를 생성하려면 CreateAccessKeyRequest 객체를 사용하여 AmazonIdentityManagementClient `createAccessKey` 메서드를 호출하세요.

`CreateAccessKeyRequest`에는 사용자 이름을 사용하는 생성자와 파라미터를 사용하지 않는 생성자, 이렇게 두 가지 생성자가 있습니다. 파라미터를 사용하지 않는 버전을 사용하는 경우 `withUserName` 설정자 메서드를 사용하여 사용자 이름을 설정하고 나서 이 이름을 `createAccessKey` 메서드에 전달해야 합니다.

가져오기

```
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;
```

코드

```
final AmazonIdentityManagement iam =
    AmazonIdentityManagementClientBuilder.defaultClient();

CreateAccessKeyRequest request = new CreateAccessKeyRequest()
    .withUserName(user);

CreateAccessKeyResult response = iam.createAccessKey(request);
```

GitHub의 전체 예제를 참조하십시오.

액세스 키 나열

해당 사용자의 액세스 키를 나열하려면 키를 나열할 사용자 이름이 포함된 ListAccessKeysRequest 객체를 생성하여 AmazonIdentityManagementClient의 `listAccessKeys` 메서드에 전달합니다.

> **Note**
>
> 사용자 이름을 `listAccessKeys`에 제공하지 않으면 이 메서드는 요청에 서명한 AWS 계정과 연결된 액세스 키를 나열하려고 합니다.

가져오기

```
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;
```

코드

```
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;
    }
}
```

`listAccessKeys`의 결과가 페이징됩니다(호출당 기본 최대 100개 레코드). 반환된 ListAccessKeysResult 객체를 `getIsTruncated`를 호출하여 쿼리에서 반환된 결과 수가 사용 가능한 것보다 적은지 확인할 수 있습니다. 거의 반환되지 않으면 `ListAccessKeysRequest`에 대해 `setMarker`를 호출하여 `listAccessKeys`의 다음 번 호출로 다시 전달할 수 있습니다.

GitHub의 전체 예제를 참조하십시오.

액세스 키의 마지막 사용 시간 가져오기

액세스 키가 마지막으로 사용된 시간을 알아보려면 액세스 키의 ID (GetAccessKeyLastUsedRequest 객체를 사용하여 전달하거나 액세스 키 ID를 직접 가져오는 오버로드에 직접 전달할 수 있음)를 사용하여 AmazonIdentityManagementClient의 `getAccessKeyLastUsed` 메서드를 호출합니다.

그런 다음 반환된 GetAccessKeyLastUsedResult 객체를 사용하여 키의 마지막 사용 시간을 가져올 수 있습니다.

가져오기

```
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;
```

코드

```
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());
```

GitHub의 전체 예제를 참조하십시오.

액세스 키 활성화 또는 비활성화

UpdateAccessKeyRequest 객체를 생성하고 액세스 키 ID, 사용자 이름(선택 사항) 및 원하는 상태를 제공한 후, 해당 요청 객체를 AmazonIdentityManagementClient의 `updateAccessKey` 메서드로 전달함으로써 액세스 키를 활성화하거나 비활성화할 수 있습니다.

가져오기

```
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;
```

코드

```
final AmazonIdentityManagement iam =
    AmazonIdentityManagementClientBuilder.defaultClient();

UpdateAccessKeyRequest request = new UpdateAccessKeyRequest()
    .withAccessKeyId(access_id)
    .withUserName(username)
    .withStatus(status);

UpdateAccessKeyResult response = iam.updateAccessKey(request);
```

GitHub의 전체 예제를 참조하십시오.

액세스 키 삭제

액세스 키를 영구적으로 삭제하려면 AmazonIdentityManagementClient의 `deleteKey` 메서드를 호출하여 액세스 키의 ID와 사용자 이름이 포함된 DeleteAccessKeyRequest를 이 메서드에 제공합니다.

> **Note**
>
> 키는 삭제하고 나면 더 이상 가져오거나 사용할 수 없습니다. 나중에 다시 활성화할 수 있도록 키를 일시적으로 비활성화하려면 대신에 updateAccessKey 메서드를 사용합니다.

가져오기

```
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;
```

코드

```
final AmazonIdentityManagement iam =
    AmazonIdentityManagementClientBuilder.defaultClient();

DeleteAccessKeyRequest request = new DeleteAccessKeyRequest()
    .withAccessKeyId(access_key)
    .withUserName(username);

DeleteAccessKeyResult response = iam.deleteAccessKey(request);
```

GitHub의 전체 예제를 참조하십시오.

추가 정보

- IAM API 참조에서 CreateAccessKey
- IAM API 참조에서 ListAccessKeys
- IAM API 참조에서 GetAccessKeyLastUsed
- IAM API 참조에서 UpdateAccessKey
- IAM API 참조에서 DeleteAccessKey

IAM 사용자 관리

사용자 생성

사용자 이름이 포함된 CreateUserRequest 객체 AmazonIdentityManagementClient의 `createUser` 메서드에 사용자 이름을 직접 제공하거나 사용자 이름이 포함된 CreateUserRequest 객체를 사용하여 새 IAM 사용자를 생성합니다.

가져오기

```
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;
```

코드

```
final AmazonIdentityManagement iam =
    AmazonIdentityManagementClientBuilder.defaultClient();

CreateUserRequest request = new CreateUserRequest()
    .withUserName(username);

CreateUserResult response = iam.createUser(request);
```

GitHub의 전체 예제를 참조하십시오.

사용자 표시

계정의 IAM 사용자를 나열하려면 새 ListUsersRequest를 생성하여 AmazonIdentityManagementClient의 `listUsers` 메서드에 전달합니다. 반환된 ListUsersResult 객체의 `getUsers`를 호출하여 사용자 목록을 검색할 수 있습니다.

`listUsers`에서 반환된 사용자 목록이 페이징됩니다. 응답 객체의 `getIsTruncated` 메서드를 호출하여 가져올 결과가 더 있는지 확인할 수 있습니다. `true`를 반환하면 요청 객체의 `setMarker()` 메서드를 호출하고 응답 객체의 `getMarker()` 메서드의 반환 값을 이 메서드에 전달합니다.

가져오기

```
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;
```

코드

```
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;
    }
}
```

GitHub의 전체 예제를 참조하십시오.

사용자 업데이트

사용자를 업데이트하려면 AmazonIdentityManagementClient 객체의 `updateUser` 메서드를 호출합니다. 이 메서드는 사용자 이름 또는 경로를 변경하는 데 사용할 수 있는 UpdateUserRequest 객체를 사용합니다.

가져오기

```
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;
```

코드

```
final AmazonIdentityManagement iam =
    AmazonIdentityManagementClientBuilder.defaultClient();

UpdateUserRequest request = new UpdateUserRequest()
    .withUserName(cur_name)
    .withNewUserName(new_name);

UpdateUserResult response = iam.updateUser(request);
```

GitHub의 전체 예제를 참조하십시오.

사용자 삭제

사용자를 삭제하려면 삭제할 사용자 이름으로 설정된 UpdateUserRequest 객체를 사용하여 AmazonIdentityManagementClient의 `deleteUser` 요청을 호출합니다.

가져오기

```
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;
```

코드

```
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;
}
```

GitHub의 전체 예제를 참조하십시오.

추가 정보

- IAM 사용 설명서의 IAM 사용자
- IAM 사용 설명서의 IAM 사용자 관리
- IAM API 참조에서 CreateUser
- IAM API 참조에서 ListUsers
- IAM API 참조에서 UpdateUser
- IAM API 참조에서 DeleteUser

IAM 계정 별칭 사용

AWS 계정 ID 대신 회사 이름이나 기타 친숙한 식별자를 로그인 페이지의 URL에 포함하려는 경우 AWS 계정 ID의 별칭을 만들 수 있습니다.

> **Note**
>
> AWS에서는 계정당 정확히 계정 별칭 하나를 지원합니다.

계정 별칭 생성

계정 별칭을 생성하려면 별칭 이름이 포함된 CreateAccountAliasRequest 객체를 사용하여 AmazonIdentixtyManagementClient의 `createAccountAlias` 메서드를 호출합니다.

가져오기

```
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;
```

코드

```
final AmazonIdentityManagement iam =
    AmazonIdentityManagementClientBuilder.defaultClient();

CreateAccountAliasRequest request = new CreateAccountAliasRequest()
    .withAccountAlias(alias);
```

```
CreateAccountAliasResult response = iam.createAccountAlias(request);
```

GitHub의 전체 예제를 참조하십시오.

계정 별칭 나열

계정 별칭을 나열하려면 AmazonIdentityManagementClient의 `listAccountAliases` 메서드를 호출합니다.

> **Note**
>
> 반환된 ListAccountAliasesResult는 다른 AWS SDK for Java 목록 메서드와 동일한 `getIsTruncated` 및 `getMarker` 메서드를 지원하지만, AWS 계정에는 계정 별칭이 하나만 있을 수 있습니다.

가져오기

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

code

```
final AmazonIdentityManagement iam =
    AmazonIdentityManagementClientBuilder.defaultClient();

ListAccountAliasesResult response = iam.listAccountAliases();

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

GitHub의 전체 예제를 참조하십시오.

계정 별칭 삭제

계정 별칭을 삭제하려면 AmazonIdentityManagementClient의 `deleteAccountAlias` 메서드를 호출합니다. 계정 별칭을 삭제하려는 경우 DeleteAccountAliasRequest 객체를 사용하여 별칭 이름을 지정해야 합니다.

가져오기

```
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;
```

코드

```
final AmazonIdentityManagement iam =
    AmazonIdentityManagementClientBuilder.defaultClient();

DeleteAccountAliasRequest request = new DeleteAccountAliasRequest()
    .withAccountAlias(alias);

DeleteAccountAliasResult response = iam.deleteAccountAlias(request);
```

GitHub의 전체 예제를 참조하십시오.

추가 정보

- IAM 사용 설명서의 AWS 계정 ID 및 별칭
- IAM API 참조에서 CreateAccountAlias
- IAM API 참조에서 ListAccountAliases
- IAM API 참조에서 DeleteAccountAlias

IAM 정책 작업

정책 만들기

새 정책을 생성하려면 CreatePolicyRequest의 정책 이름과 JSON 형식으로 된 정책 문서를 AmazonIdentityManagementClient의 `createPolicy` 메서드에 제공합니다.

가져오기

```
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;
```

코드

```
final AmazonIdentityManagement iam =
    AmazonIdentityManagementClientBuilder.defaultClient();

CreatePolicyRequest request = new CreatePolicyRequest()
    .withPolicyName(policy_name)
    .withPolicyDocument(POLICY_DOCUMENT);

CreatePolicyResult response = iam.createPolicy(request);
```

IAM 정책 문서는 올바르게 문서화된 구문으로 된 JSON 문자열입니다. 다음은 DynamoDB에 대한 특정 요청을 작성하기 위한 액세스 권한을 제공하는 예입니다.

```
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\"" +
    "    }" +
    "   ]" +
    "}";
```

GitHub의 전체 예제를 참조하십시오.

정책 가져오기

기존 정책을 검색하려면 GetPolicyRequest 객체 내에 정책의 ARN을 제공하여 AmazonIdentityManagementClient의 `getPolicy` 메서드를 호출하세요.

가져오기

```
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;
```

코드

```
final AmazonIdentityManagement iam =
    AmazonIdentityManagementClientBuilder.defaultClient();

GetPolicyRequest request = new GetPolicyRequest()
    .withPolicyArn(policy_arn);

GetPolicyResult response = iam.getPolicy(request);
```

GitHub의 전체 예제를 참조하십시오.

역할 정책 연결

AmazonIdentityManagementClient의 `attachRolePolicy` 메서드를 호출하고 AttachRolePolicyRequest에 역할 이름 및 정책 ARN을 제공하여 IAMhttp://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles.html[role]에 정책을 연결할 수 있습니다.

가져오기

```
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;
```

코드

```
final AmazonIdentityManagement iam =
    AmazonIdentityManagementClientBuilder.defaultClient();
```

```
AttachRolePolicyRequest attach_request =
    new AttachRolePolicyRequest()
        .withRoleName(role_name)
        .withPolicyArn(POLICY_ARN);

iam.attachRolePolicy(attach_request);
```

GitHub의 전체 예제를 참조하십시오.

연결된 역할 정책 나열

AmazonIdentityManagementClient의 `listAttachedRolePolicies` 메서드를 호출하여 역할의 연결된 정책을 나열합니다. 이 메서드는 정책을 나열할 역할 이름을 포함하는 ListAttachedRolePoliciesRequest 객체를 사용합니다.

반환된 ListAttacheDrolePoliciesResult 객체에 `getAttachedPolicies`를 호출하여 연결된 정책 목록을 가져옵니다. 결과가 잘릴 수도 있습니다. `ListAttachedRolePoliciesResult` 객체의 `getIsTruncated` 메서드가 `true`를 반환하는 경우 `ListAttachedRolePoliciesRequest` 객체의 `setMarker` 메서드를 호출하고 이 메서드를 사용하여 `listAttachedRolePolicies`를 다시 호출함으로써 다음 결과 배치를 가져옵니다.

가져오기

```
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;
```

코드

```
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());
}
```

GitHub의 전체 예제를 참조하십시오.

역할 정책 분리

역할에서 정책을 분리하려면 AmazonIdentityManagementClient의 `detachRolePolicy` 메서드를 호출하여 DetachRolePolicyRequest의 역할 이름과 정책 ARN을 이 메서드에 지정합니다.

가져오기

```
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;
```

코드

```
final AmazonIdentityManagement iam =
    AmazonIdentityManagementClientBuilder.defaultClient();

DetachRolePolicyRequest request = new DetachRolePolicyRequest()
    .withRoleName(role_name)
    .withPolicyArn(policy_arn);
```

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

GitHub의 전체 예제를 참조하십시오.

추가 정보

- IAM 사용 설명서의 IAM 정책 개요.
- IAM 사용 설명서의 AWS IAM 정책 참조.
- IAM API 참조에 CreatePolicy
- IAM API 참조에 GetPolicy
- IAM API 참조에 AttachRolePolicy
- IAM API 참조에 ListAttachedRolePolicies
- IAM API 참조에 DetachRolePolicy

IAM 서버 인증서 작업

AWS에서 웹 사이트나 애플리케이션에 대한 HTTPS 연결을 활성화하려면 SSL/TLS 서버 인증서가 필요합니다. AWS Certificate Manager에서 제공하거나 외부 공급자에게서 얻은 서버 인증서를 사용할 수 있습니다.

서버 인증서를 프로비저닝, 관리 및 배포할 때 ACM를 사용하는 것이 좋습니다. ACM을 사용하면 인증서를 요청하여 AWS 리소스에 배포할 수 있고, ACM을 통해 인증서 갱신을 처리할 수 있습니다. ACM에서 제공하는 인증서는 무료입니다. ACM에 대한 자세한 내용은 ACM 사용 설명서를 참조하세요.

서버 인증서 가져오기

AmazonIdentityManagementClient의 `getServerCertificate` 메서드를 호출하고 인증서 이름이 포함된 GetServerCertificateRequest를 이 메서드에 전달하여 서버 인증서를 검색할 수 있습니다.

가져오기

```
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;
```

코드

```
final AmazonIdentityManagement iam =
    AmazonIdentityManagementClientBuilder.defaultClient();

GetServerCertificateRequest request = new GetServerCertificateRequest()
            .withServerCertificateName(cert_name);

GetServerCertificateResult response = iam.getServerCertificate(request);
```

GitHub의 전체 예제를 참조하십시오.

서버 인증서 목록 조회

서버 인증서를 나열하려면 ListServerCertificatesRequest를 사용하여 AmazonIdentityManagementClient의 `listServerCertificates` 메서드를 호출하세요. 이 메서드는 ListServerCertificatesResult를 반환합니다.

반환된 `ListServerCertificateResult` 객체의 `getServerCertificateMetadataList` 메서드를 호출하여 각 인증서에 대한 정보를 가져오는 데 사용할 수 있는 ServerCertificateMetadata 객체의 목록을 가져옵니다.

결과가 잘릴 수도 있습니다. `ListServerCertificateResult` 객체의 `getIsTruncated` 메서드가 `true`를 반환하는 경우 `ListServerCertificatesRequest` 객체의 `setMarker` 메서드를 호출하고 이 메서드를 사용하여 `listServerCertificates`를 다시 호출함으로써 다음 결과 배치를 가져옵니다.

가져오기

```
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;
```

코드

```
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;
    }
}
```

GitHub의 전체 예제를 참조하십시오.

서버 인증서 업데이트

AmazonIdentityManagementClient의 `updateServerCertificate` 메서드를 호출하여 서버 인증서의 이름이나 경로를 업데이트할 수 있습니다. 이 메서드는 서버 인증서의 현재 이름 및 사용할 새 이름이나 새 경로로 설정된 UpdateServerCertificateRequest 객체를 사용합니다.

가져오기

```
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;
```

코드

```
final AmazonIdentityManagement iam =
    AmazonIdentityManagementClientBuilder.defaultClient();

UpdateServerCertificateRequest request =
    new UpdateServerCertificateRequest()
        .withServerCertificateName(cur_name)
```

```
        .withNewServerCertificateName(new_name);

UpdateServerCertificateResult response =
    iam.updateServerCertificate(request);
```

GitHub의 전체 예제를 참조하십시오.

서버 인증서 삭제

서버 인증서를 삭제하려면 인증서 이름이 포함된 DeleteServerCertificateRequest와 함께 AmazonIdentityManagementClient의 `deleteServerCertificate` 메서드를 호출하세요.

가져오기

```
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;
```

코드

```
final AmazonIdentityManagement iam =
    AmazonIdentityManagementClientBuilder.defaultClient();

DeleteServerCertificateRequest request =
    new DeleteServerCertificateRequest()
        .withServerCertificateName(cert_name);

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

GitHub의 전체 예제를 참조하십시오.

추가 정보

- IAM 사용 설명서의 서버 인증서 사용
- IAM API 참조에서 GetServerCertificate
- IAM API 참조에서 ListServerCertificates
- IAM API 참조에서 UpdateServerCertificate
- IAM API 참조에서 DeleteServerCertificate

- ACM 사용 설명서

AWS SDK for Java를 사용하는 Lambda 예제

이 단원에서는 AWS SDK for Java를 사용한 Lambda 프로그래밍의 예제를 제공합니다.

> **Note**
>
> 예제에는 각 기술을 보여주는 데 필요한 코드만 포함되어 있습니다. 전체 예제 코드는 GitHub에 있습니다. 이 위치에서 단일 소스 파일을 다운로드하거나 리포지토리를 로컬로 복사하여 모든 예제를 빌드하고 실행할 수 있습니다.

주제

- Lambda 함수 호출, 나열 및 삭제

Lambda 함수 호출, 나열 및 삭제

이 단원에서는 AWS SDK for Java를 사용하여 Lambda 서비스 클라이언트를 통해 프로그래밍하는 예제를 제공합니다. Lambda 함수 생성 방법에 대한 자세한 내용은 AWS Lambda 함수 생성 방법을 참조하세요.

주제

- 함수 호출.
- 함수 나열
- 함수 삭제

함수 호출.

AWSLambda 객체를 만들고 객체의 `invoke` 메서드를 호출하여 Lambda 함수를 호출할 수 있습니다. InvokeRequest 객체를 만들어 Lambda 함수에 전달할 함수 이름 및 페이로드와 같은 추가 정보를 지정합니다. 함수 이름은 arn:aws:lambda:us-east-1:555556330391:function:HelloFunction과 같이 나타납니다. AWS Management Console에서 함수를 확인해 값을 검색할 수 있습니다.

함수에 페이로드 데이터를 전달하려면 InvokeRequest 객체의 `withPayload` 메서드를 호출하고 다음 코드 예제와 같이 JSON 형식으로 문자열을 지정합니다.

가져오기

```
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;
```

코드

다음 코드 예제는 Lambda 함수를 호출하는 방법을 보여줍니다.

```
        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());
```

GitHub의 전체 예제를 참조하십시오.

함수 나열

AWSLambda 객체를 빌드하고 객체의 `listFunctions` 메서드를 호출합니다. 이 메서드는 ListFunctionsResult 객체를 반환합니다. 이 객체의 `getFunctions` 메서드를 호출하여 FunctionConfiguration 객체 목록을 반환할 수 있습니다. 목록을 반복하여 함수에 대한 정보를 검색할 수 있습니다. 예를 들어 다음 Java 코드 예제는 각 함수 이름을 가져오는 방법을 보여줍니다.

가져오기

```
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;
```

코드

다음 Java 코드 예제는 Lambda 함수 이름 목록을 검색하는 방법을 보여 줍니다.

```
        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);
        }
```

GitHub의 전체 예제를 참조하십시오.

함수 삭제

AWSLambda 객체를 빌드하고 객체의 deleteFunction 메서드를 호출합니다. DeleteFunctionRequest 객체를 만들고 deleteFunction 메서드에 전달합니다. 이 개체에는 삭제할 함수의 이름과 같은 정보가 포함되어 있습니다. 함수 이름은 arn:aws:lambda:us-east-1:555556330391:function:HelloFunction과 같이 나타납니다. AWS Management Console에서 함수를 확인해 값을 검색할 수 있습니다.

가져오기

```
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;
```

코드

다음 Java 코드는 Lambda 함수를 삭제하는 방법을 설명합니다.

```
        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);
        }
```

GitHub의 전체 예제를 참조하십시오.

AWS SDK for Java를 사용하는 Amazon Pinpoint 예제

이 단원에서는 AWS SDK for Java를 사용한 Amazon Pinpoint 프로그래밍의 예제를 제공합니다.

> **Note**
>
> 예제에는 각 기술을 보여주는 데 필요한 코드만 포함되어 있습니다. 전체 예제 코드는 GitHub에 있습니다. 이 위치에서 단일 소스 파일을 다운로드하거나 리포지토리를 로컬로 복사하여 모든 예제를 빌드하고 실행할 수 있습니다.

주제

- Amazon Pinpoint에서 앱 생성 및 삭제
- Amazon Pinpoint에 엔드포인트 생성
- Amazon Pinpoint에서 세그먼트 생성
- Amazon Pinpoint에서 캠페인 생성
- Amazon Pinpoint에서 채널 업데이트

Amazon Pinpoint에서 앱 생성 및 삭제

앱이란 특정 애플리케이션의 대상 사용자를 정의하는 Amazon Pinpoint 프로젝트를 말하며, 이러한 대상 사용자를 맞춤형 메시지와 연계할 수 있습니다. 이 페이지의 예제에서는 새 앱을 생성하거나 기존 앱을 삭제하는 방법을 보여 줍니다.

앱 생성

CreateAppRequest 객체에 앱 이름을 제공한 다음 해당 객체를 AmazonPinpointClient의 `createApp` 메서드에 전달하여 Amazon Pinpoint에서 새 앱을 생성합니다.

가져오기

```
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;
```

코드

```
CreateApplicationRequest appRequest = new CreateApplicationRequest()
  .withName(appName);

CreateAppRequest request = new CreateAppRequest();
request.withCreateApplicationRequest(appRequest);
CreateAppResult result = pinpoint.createApp(request);
```

GitHub의 전체 예제를 참조하십시오.

앱 삭제

앱을 삭제하려면 삭제할 사용자 이름으로 설정된 DeleteAppRequest 객체를 사용하여 AmazonPinpointClient의 `deleteApp` 요청을 호출합니다.

가져오기

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

코드

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

pinpoint.deleteApp(deleteRequest);
```

GitHub의 전체 예제를 참조하십시오.

추가 정보

- Amazon Pinpoint API 참조에서 Apps
- Amazon Pinpoint API 참조에서 App

Amazon Pinpoint에 엔드포인트 생성

엔드포인트는 Amazon Pinpoint를 사용하여 푸시 알림이 전송될 수 있는 사용자 디바이스를 고유하게 식별합니다. Amazon Pinpoint를 지원하는 앱이라면 새 사용자가 앱을 열 때 자동으로 Amazon Pinpoint에 엔드포인트를 등록합니다. 다음 예제는 새 엔드포인트를 프로그래밍 방식으로 추가하는 방법을 보여 줍니다.

엔드포인트 생성

EndpointRequest 객체에 엔드포인트 데이터를 제공하여 Amazon Pinpoint에서 새 엔드포인트를 생성합니다.

가져오기

```
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;
```

코드

```
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);
```

그런 다음 그 EndpointRequest 객체로 UpdateEndpointRequest 객체를 생성합니다. 마지막으로 AmazonPinpointClient의 `updateEndpoint` 메서드에 UpdateEndpointRequest 객체를 전달합니다.

코드

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

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

GitHub의 전체 예제를 참조하십시오.

추가 정보

- Amazon Pinpoint 개발자 안내서의 엔드포인트 추가
- Amazon Pinpoint API 참조에서 엔드포인트

Amazon Pinpoint에서 세그먼트 생성

사용자 세그먼트는 사용자가 앱을 마지막으로 연 시점 또는 사용한 디바이스 등 공유 특성을 기준으로 나눈 사용자 하위 집합을 나타냅니다. 다음 예제에서는 사용자 세그먼트를 정의하는 방법을 보여 줍니다.

세그먼트 생성

SegmentDimensions 객체에 세그먼트의 치수를 정의하여 Amazon Pinpoint에서 새 세그먼트를 생성합니다.

가져오기

```
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;
```

코드

```
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);
```

다음으로 WriteSegmentRequest에서 SegmentDimensions 객체를 설정합니다. 이 객체는 다시 CreateSegmentRequest 객체를 만드는 데 사용됩니다. 마지막으로 CreateSegmentRequest 객체를 AmazonPinpointClient의 `createSegment` 메서드에 전달합니다.

코드

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

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

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

GitHub의 전체 예제를 참조하십시오.

추가 정보

- Amazon Pinpoint 사용 설명서의 Amazon Pinpoint 세그먼트
- Amazon Pinpoint 개발자 안내서의 세그먼트 생성
- Amazon Pinpoint API 참조에서 세그먼트

- Amazon Pinpoint API 참조에서 세그먼트

Amazon Pinpoint에서 캠페인 생성

캠페인을 사용하여 앱과 사용자 간의 관계를 강화할 수 있습니다. 캠페인을 만들고 맞춤형 메시지나 특별 프로모션으로 특정한 사용자 세그먼트에 접근해 보십시오. 이 예제에서는 사용자 지정 푸시 알림을 지정된 세그먼트로 보내는 새로운 표준 캠페인을 만드는 방법을 보여 줍니다.

캠페인 생성

새 캠페인을 만들려면 먼저 일정과 메시지를 정의하고 WriteCampaignRequest 객체에 이러한 값을 설정해야 합니다.

가져오기

```
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;
```

코드

```
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);
```

그런 다음 캠페인 구성과 함께 WriteCampaignRequest를 CreateCampaignRequest 객체에 제공하여 Amazon Pinpoint에서 새 캠페인을 생성합니다. 마지막으로 CreateCampaignRequest 객체를 AmazonPinpointClient의 `createCampaign` 메서드에 전달합니다.

코드

```
CreateCampaignRequest createCampaignRequest = new CreateCampaignRequest()
        .withApplicationId(appId).withWriteCampaignRequest(request);

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

GitHub의 전체 예제를 참조하십시오.

추가 정보

- Amazon Pinpoint 사용 설명서의 Amazon Pinpoint캠페인
- Amazon Pinpoint 개발자 안내서의 캠페인 생성
- Amazon Pinpoint API 참조에서 캠페인
- Amazon Pinpoint API 참조에서 캠페인
- Amazon Pinpoint API 참조에서 캠페인 활동
- Amazon Pinpoint API 참조에서 캠페인 버전
- Amazon Pinpoint API 참조에서 캠페인 버전

Amazon Pinpoint에서 채널 업데이트

사용자가 메시지를 전송할 수 있는 플랫폼의 유형은 채널에 따라 정의됩니다. 이 예제에서는 APN 채널을 사용하여 메시지를 보내는 방법을 보여 줍니다.

채널 업데이트

앱 ID와 업데이트할 채널 유형의 요청 객체를 입력하여 Amazon Pinpoint에서 채널을 활성화합니다. 이 예제에서는 APNSChannelRequest 객체를 요구하는 APN 채널을 업데이트합

니다. UpdateApnsChannelRequest에 이를 설정하고 해당 객체를 AmazonPinpointClient의 `updateApnsChannel` 메서드로 전달합니다.

가져오기

```
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;
```

코드

```
APNSChannelRequest request = new APNSChannelRequest()
  .withEnabled(enabled);

UpdateApnsChannelRequest updateRequest = new UpdateApnsChannelRequest()
  .withAPNSChannelRequest(request)
  .withApplicationId(appId);
UpdateApnsChannelResult result = client.updateApnsChannel(updateRequest);
```

GitHub의 전체 예제를 참조하십시오.

추가 정보

- Amazon Pinpoint 사용 설명서에서 Amazon Pinpoint 채널
- Amazon Pinpoint API 참조에서 ADM 채널
- Amazon Pinpoint API 참조에서 APNs 채널
- Amazon Pinpoint API 참조에서 APNs 샌드박스 채널
- Amazon Pinpoint API 참조에서 APNs VoIP 채널
- Amazon Pinpoint API 참조에서 APNs VoIP 샌드박스 채널
- Amazon Pinpoint API 참조에서 Baidu 채널
- Amazon Pinpoint API 참조에서 이메일 채널
- Amazon Pinpoint API 참조에서 GCM 채널

- Amazon Pinpoint API 참조에서 SMS 채널

AWS SDK for Java를 사용하는 Amazon S3 예제

이 단원에서는 AWS SDK for Java를 사용한 Amazon S3 프로그래밍의 예제를 제공합니다.

> **Note**
>
> 예제에는 각 기술을 보여주는 데 필요한 코드만 포함되어 있습니다. 전체 예제 코드는 GitHub에 있습니다. 이 위치에서 단일 소스 파일을 다운로드하거나 리포지토리를 로컬로 복사하여 모든 예제를 빌드하고 실행할 수 있습니다.

주제

- Amazon S3 버킷 생성, 나열, 삭제
- Amazon S3 객체에 대한 작업 수행
- 버킷 및 객체에 대한 Amazon S3 액세스 권한 관리
- 버킷 정책을 사용한 Amazon S3 버킷 액세스 관리
- Amazon S3 운영을 위한 TransferManager 사용
- Amazon S3 버킷을 웹 사이트로 구성
- Amazon S3 클라이언트 측 암호화 사용

Amazon S3 버킷 생성, 나열, 삭제

Amazon S3의 모든 객체(파일)은 객체의 모음(컨테이너)을 나타내는 버킷에 상주해야 합니다. 각 버킷은 키(이름)로 인식되며, 각 키는 고유해야 합니다. 버킷 및 구성에 대한 자세한 내용은 Amazon Simple Storage Service 사용 설명서의 Amazon S3 버킷 사용을 참조하세요.

> **Note**
>
> 모범 사례
> Amazon S3 버킷에서 AbortIncompleteMultipartUpload 수명 주기 규칙을 활성화하는 것이 좋습니다.
> 이 규칙은 시작된 후 지정된 일수 내에 완료되지 않은 멀티파트 업로드를 중단하도록 Amazon S3에 지시합니다. 설정된 시간 제한을 초과하면 Amazon S3가 업로드를 중단한 후 완료되지 않은 업로드 데이터를 삭제합니다.

자세한 내용은 Amazon S3 사용 설명서의 버전 관리가 포함된 버킷의 수명 주기 구성을 참조하세요.

> **Note**
>
> 이 코드 예제에서는 사용자가 AWS SDK for Java 사용의 내용을 이해하고 개발을 위한 AWS 자격 증명 및 리전 설정의 정보를 사용하여 기본 AWS 자격 증명을 구성했다고 가정합니다.

버킷 만들기

AmazonS3 클라이언트의 `createBucket` 메서드를 사용하세요. 새 버킷이 반환됩니다. 해당 버킷이 이미 존재하는 경우에는 `createBucket` 메서드에서 예외가 발생합니다.

> **Note**
>
> 동일한 이름의 버킷을 생성하기 전에 버킷이 이미 존재하는지 여부를 확인하려면 `doesBucketExist` 메서드를 호출합니다. 이 메서드는 버킷이 존재하는 경우 `true`를 반환하고, 그렇지 않으면 `false`를 반환합니다.

가져오기

```
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;
```

코드

```
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;
```

GitHub의 전체 예제를 참조하십시오.

버킷 목록 생성

AmazonS3 클라이언트의 `listBucket` 메서드를 사용하세요. 성공하면 버킷이 반환됩니다.

가져오기

```
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;
```

코드

```
List<Bucket> buckets = s3.listBuckets();
System.out.println("Your {S3} buckets are:");
for (Bucket b : buckets) {
    System.out.println("* " + b.getName());
}
```

GitHub의 전체 예제를 참조하십시오.

버킷 삭제

Amazon S3 버킷을 삭제하려면 먼저 버킷이 비어 있는지 확인해야 합니다. 비어 있지 않으면 오류가 발생합니다. 버전 지정된 버킷이 있으면 버킷과 연결된 버전 지정된 객체도 모두 삭제해야 합니다.

> **Note**
>
> 전체 예제에는 Amazon S3 버킷 및 관련 콘텐츠를 삭제하기 위한 전체 솔루션을 제공하는 다음의 각 단계가 순서대로 포함됩니다.

주제

삭제하기 전에 버전이 지정되지 않은 버킷에서 객체 제거

AmazonS3 클라이언트의 `listObjects` 메서드를 사용하여 객체 목록을 가져오고 `deleteObject`를 사용하여 각 객체를 삭제합니다.

가져오기

```
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;
```

코드

```
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;
    }
}
```

GitHub의 전체 예제를 참조하십시오.

삭제하기 전에 버전 지정된 버킷에서 객체 제거

버전 지정된 버킷을 사용 중인 경우 버킷을 삭제하려면 먼저 버킷에 있는 저장된 객체 버전도 모두 제거해야 합니다.

버킷에 있는 객체를 제거할 때와 비슷한 패턴을 사용하여, AmazonS3 클라이언트의 `listVersions` 메서드를 사용하여 버전 지정된 객체를 나열한 후 `deleteVersion`을 사용하여 각 객체를 삭제함으로써 버전 지정된 객체를 제거합니다.

가져오기

```
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;
```

코드

```
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;
    }
}
```

GitHub의 전체 예제를 참조하십시오.

빈 버킷 삭제

버킷에서 객체(버전 지정된 모든 객체 포함)를 제거하고 나면 클라이언트의 AmazonS3의 `deleteBucket` 메서드를 사용하여 버킷 자체를 삭제할 수 있습니다.

가져오기

```
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;
```

코드

```
System.out.println(" OK, bucket ready to delete!");
s3.deleteBucket(bucket_name);
```

GitHub의 전체 예제를 참조하십시오.

Amazon S3 객체에 대한 작업 수행

Amazon S3 객체는 데이터 모음 또는 파일을 나타냅니다. 각 객체는 버킷 안에 상주해야 합니다.

> **Note**
>
> 이 코드 예제에서는 사용자가 AWS SDK for Java 사용의 내용을 이해하고 개발을 위한 AWS 자격 증명 및 리전 설정의 정보를 사용하여 기본 AWS 자격 증명을 구성했다고 가정합니다.

주제

- 객체 업로드
- 객체 목록 생성
- 객체 다운로드
- 객체 복사, 이동 또는 이름 바꾸기
- 객체 삭제
- 여러 객체를 한 번에 삭제

객체 업로드

AmazonS3 클라이언트의 `putObject` 메서드를 사용하여 버킷 이름, 키 이름, 업로드할 파일을 지정합니다. 버킷 이름이 존재해야 합니다. 그렇지 않으면 오류가 발생합니다.

가져오기

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
```

코드

```
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);
}
```

GitHub의 전체 예제를 참조하십시오.

객체 목록 생성

버킷 내에서 객체 목록을 가져오려면 AmazonS3 클라이언트의 `listObjects` 메서드를 사용하여 버킷 이름을 지정합니다.

`listObjects` 메서드는 버킷의 객체에 대한 정보를 제공하는 ObjectListing 객체를 반환합니다. 객체 이름(키)을 나열하려면 `getObjectSummaries` 메서드를 사용하여 S3ObjectSummary 객체 목록을 가져옵니다. 이때 각 객체는 버킷에 있는 단일 객체를 나타냅니다. 그런 다음 `getKey` 메서드를 호출하여 객체의 이름을 가져옵니다.

가져오기

```
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;
```

코드

```
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());
}
```

GitHub의 전체 예제를 참조하십시오.

객체 다운로드

AmazonS3 클라이언트의 `getObject` 메서드를 사용하여 다운로드할 버킷의 이름 및 객체의 이름을 전달합니다. 성공하면 이 메서드는 S3Object를 반환합니다. 지정한 버킷 및 객체 키가 존재해야 합니다. 그렇지 않으면 오류가 발생합니다.

`S3Object`에 대해 `getObjectContent`를 호출하여 객체의 내용을 가져올 수 있습니다. 그러면 표준 자바 `InputStream` 객체처럼 동작하는 S3ObjectInputStream이 반환됩니다.

다음 예제는 S3에서 객체를 다운로드하고 이 객체의 내용을 (객체 키와 동일한 이름을 사용하여) 파일에 저장합니다.

가져오기

```
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;
```

코드

```
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);
}
```

GitHub의 전체 예제를 참조하십시오.

객체 복사, 이동 또는 이름 바꾸기

AmazonS3 클라이언트의 `copyObject` 메서드를 사용하여 한 버킷에서 다른 버킷으로 객체를 복사할 수 있습니다. 복사할 버킷의 이름, 복사할 객체 그리고 대상 버킷 이름을 가져옵니다.

가져오기

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.regions.Regions;
```

코드

```
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!");
```

GitHub의 전체 예제를 참조하십시오.

> **Note**
>
> copyObject와 deleteObject를 함께 사용하면 먼저 객체를 새 이름으로 복사한 다음(동일한 버킷을 소스와 대상으로 모두 사용 가능) 이전 위치에서 해당 객체를 삭제하는 방식으로 객체를 이동하거나 이름을 바꿀 수 있습니다.

객체 삭제

AmazonS3 클라이언트의 deleteObject 메서드를 사용하여 삭제할 버킷 및 객체의 이름을 전달합니다. 지정한 버킷 및 객체 키가 존재해야 합니다. 그렇지 않으면 오류가 발생합니다.

가져오기

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.regions.Regions;
```

코드

```
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);
}
```

GitHub의 전체 예제를 참조하십시오.

여러 객체를 한 번에 삭제

Amazon S3 클라이언트의 deleteObjects 메서드를 사용하면 link:sdk-for-java/v1/reference/com/amazonaws/services/s3/model/DeleteObjectsRequest.html 메서드에 해당 이름을 전달하여 동일한 버킷에서 여러 객체를 삭제할 수 있습니다.

가져오기

```
import com.amazonaws.AmazonServiceException;
```

```
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
```

코드

```
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);
}
```

GitHub의 전체 예제를 참조하십시오.

버킷 및 객체에 대한 Amazon S3 액세스 권한 관리

Amazon S3 리소스를 세부적으로 제어하려는 경우 Amazon S3 버킷 및 객체에 대한 ACL(액세스 제어 목록)을 사용할 수 있습니다.

> **Note**
>
> 이 코드 예제에서는 사용자가 AWS SDK for Java 사용의 내용을 이해하고 개발을 위한 AWS 자격 증명 및 리전 설정의 정보를 사용하여 기본 AWS 자격 증명을 구성했다고 가정합니다.

버킷에 대한 액세스 제어 목록 가져오기

버킷에 대한 현재 ACL(액세스 제어 목록)을 가져오려면 AmazonS3의 `getBucketAcl` 메서드를 호출하고 쿼리할 버킷 이름을 메서드에 전달합니다. 이 메서드는 AccessControlList 객체를 반환합니다. 각 액세스 권한 부여(grant) 목록을 가져오려면 `getGrantsAsList` 메서드를 호출합니다. 그러면 Grant 객체의 표준 Java 목록이 반환됩니다.

가져오기

```
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;
```

코드

```
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);
}
```

GitHub의 전체 예제를 참조하십시오.

버킷에 대한 액세스 제어 목록 설정

버킷에 대한 ACL(액세스 제어 목록)에 권한을 추가하거나 권한을 수정하려면 AmazonS3의 `setBucketAcl` 메서드를 호출합니다. 이 메서드는 설정할 액세스 수준과 피부여자 목록을 포함하는 AccessControlList 객체를 사용합니다.

가져오기

```
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;
```

코드

```
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**
>
> Grantee 클래스를 사용하여 직접 피부여자의 고유 식별자를 제공하거나, 여기서처럼 EmailAddressGrantee 클래스를 사용하여 이메일을 통해 피부여자를 설정할 수 있습니다.

GitHub의 전체 예제를 참조하십시오.

객체에 대한 액세스 제어 목록 가져오기

객체에 대한 현재 ACL(액세스 제어 목록)을 가져오려면 AmazonS3의 `getObjectAcl` 메서드를 호출하고 쿼리할 버킷 이름과 객체 이름을 이 메서드에 전달합니다. `getBucketAcl`과 마찬가지로 이 메서드는 각 Grant 검사하는 데 사용할 수 있는 AccessControlList 객체를 반환합니다.

가져오기

```
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;
```

코드

```
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);
}
```

GitHub의 전체 예제를 참조하십시오.

객체에 대한 액세스 제어 목록 설정

객체에 대한 ACL(액세스 제어 목록)에 권한을 추가하거나 권한을 수정하려면 AmazonS3의 `setObjectAcl` 메서드를 호출합니다. 이 메서드는 설정할 액세스 수준과 피부여자 목록을 포함하는 AccessControlList 객체를 사용합니다.

가져오기

```
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;
```

코드

```
    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**
>
> Grantee 클래스를 사용하여 직접 피부여자의 고유 식별자를 제공하거나, 여기서처럼 EmailAddressGrantee 클래스를 사용하여 이메일을 통해 피부여자를 설정할 수 있습니다.

GitHub의 전체 예제를 참조하십시오.

추가 정보

- Amazon S3 API 참조에서 GET 버킷 ACL(액세스 제어 목록)
- Amazon S3 API 참조에서 PUT 버킷 ACL(액세스 제어 목록)
- Amazon S3 API 참조에서 GET 객체 ACL(액세스 제어 목록)
- Amazon S3 API 레퍼런스의 PUT 객체 ACL(액세스 제어 목록)

버킷 정책을 사용한 Amazon S3 버킷 액세스 관리

버킷 정책을 설정하거나 가져오거나 삭제하여 Amazon S3 버킷에 대한 액세스를 관리할 수 있습니다.

버킷 정책 설정

다음과 같은 방법으로 특정 S3 버킷에 대한 버킷 정책을 설정할 수 있습니다.

- AmazonS3 클라이언트의 `setBucketPolicy`를 호출하고 이 메서드에 SetBucketPolicyRequest 지정
- 버킷 이름과 정책 텍스트(JSON 형식)를 사용하는 `setBucketPolicy` 오버로드를 사용하여 정책을 직접 설정

가져오기

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.auth.policy.Policy;
import com.amazonaws.auth.policy.Principal;
```

코드

```
    s3.setBucketPolicy(bucket_name, policy_text);
} catch (AmazonServiceException e) {
```

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

Policy 클래스를 사용하여 정책 생성 또는 검사

`setBucketPolicy`에 버킷 정책을 제공하려는 경우 다음을 수행합니다.

- 정책을 JSON 형식 텍스트 문자열로 직접 지정
- Policy 클래스를 사용하여 정책 빌드

`Policy` 클래스를 사용하면 텍스트 문자열에 대한 올바른 형식 지정에 대해 걱정할 필요가 없습니다. `Policy` 클래스에서 JSON 정책 텍스트를 가져오려면 `toJson` 메서드를 사용합니다.

가져오기

```
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;
```

코드

```
        new Statement(Statement.Effect.Allow)
                .withPrincipals(Principal.AllUsers)
                .withActions(S3Actions.GetObject)
                .withResources(new Resource(
                        "{region-arn}s3:::" + bucket_name + "/*")));
return bucket_policy.toJson();
```

`Policy` 클래스는 전달된 JSON 문자열을 사용하여 정책을 빌드할 수 있는 `fromJson` 메서드도 제공합니다. 이 메서드는 해당 텍스트를 유효한 정책 구조로 변환할 수 있는지 검사하며, 정책 텍스트가 유효하지 않은 경우 실패하고 `IllegalArgumentException`가 발생합니다.

```
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());
}
```

이 기술을 사용하여 파일이나 기타 수단에서 읽어온 정책을 사전 검사할 수 있습니다.

GitHub의 전체 예제를 참조하십시오.

버킷 정책 가져오기

Amazon S3 버킷에 대한 정책을 가져오려면 AmazonS3 클라이언트의 `getBucketPolicy` 메서드를 호출하고 정책을 가져올 버킷의 이름을 이 메서드에 전달합니다.

가져오기

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
```

코드

```
  try {
      BucketPolicy bucket_policy = s3.getBucketPolicy(bucket_name);
      policy_text = bucket_policy.getPolicyText();
  } catch (AmazonServiceException e) {
      System.err.println(e.getErrorMessage());
      System.exit(1);
  }
```

이름이 지정된 버킷이 없거나, 해당 버킷에 대한 액세스 권한이 없거나, 버킷 정책이 없는 경우 `AmazonServiceException`이 발생합니다.

GitHub의 전체 예제를 참조하십시오.

버킷 정책 삭제

버킷 정책을 삭제하려면 AmazonS3 클라이언트의 `deleteBucketPolicy`를 호출하고 이 메서드에 버킷 이름을 제공합니다.

가져오기

```
import com.amazonaws.AmazonServiceException;
```

```
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
```

코드

```
try {
    s3.deleteBucketPolicy(bucket_name);
} catch (AmazonServiceException e) {
    System.err.println(e.getErrorMessage());
    System.exit(1);
}
```

버킷에 정책이 아직 없더라도 이 메서드는 성공합니다. 존재하지 않는 버킷 이름을 지정하거나 해당 버킷에 대한 액세스 권한이 없는 경우 `AmazonServiceException`이 발생합니다.

GitHub의 전체 예제를 참조하십시오.

추가 정보

- Amazon Simple Storage Service 사용 설명서에서 액세스 정책 언어 개요
- Amazon Simple Storage Service 사용 설명서에서 버킷 정책 예제

Amazon S3 운영을 위한 TransferManager 사용

AWS SDK for Java TransferManager 클래스를 사용하여 로컬 환경에서 Amazon S3로 파일을 안정적으로 전송하고 한 S3 위치에서 다른 위치로 객체를 복사할 수 있습니다. `TransferManager`는 전송 진행 상황을 가져오고 업로드 및 다운로드를 일시 중지 또는 재개할 수 있습니다.

> **Note**
>
> 모범 사례
> Amazon S3 버킷에서 AbortIncompleteMultipartUpload 수명 주기 규칙을 활성화하는 것이 좋습니다.
> 이 규칙은 시작된 후 지정된 일수 내에 완료되지 않은 멀티파트 업로드를 중단하도록 Amazon S3에 지시합니다. 설정된 시간 제한을 초과하면 Amazon S3가 업로드를 중단한 후 완료되지 않은 업로드 데이터를 삭제합니다.
> 자세한 내용은 Amazon S3 사용 설명서의 버전 관리가 포함된 버킷의 수명 주기 구성을 참조하세요.

> **Note**
>
> 이 코드 예제에서는 사용자가 AWS SDK for Java 사용의 내용을 이해하고 개발을 위한 AWS 자격 증명 및 리전 설정의 정보를 사용하여 기본 AWS 자격 증명을 구성했다고 가정합니다.

파일 및 디렉터리 업로드

TransferManager는 이전에 생성한 모든 Amazon S3 버킷에 파일, 파일 목록 및 디렉터리를 업로드할 수 있습니다.

주제

- 단일 파일 업로드
- 파일 목록 업로드
- 디렉터리 업로드

단일 파일 업로드

TransferManager의 `upload` 메서드를 호출하여 Amazon S3 버킷 이름, 키(객체) 이름, 업로드할 파일을 나타내는 표준 Java File 객체를 입력합니다.

가져오기

```
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;
```

코드

```
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();
```

upload 메서드는 (값을) 즉시 반환하며, 전송 상태를 확인하거나 완료될 때까지 대기하는 데 사용할 Upload 객체를 제공합니다.

TransferManager의 shutdownNow 메서드를 호출하기 전에 전송을 성공적으로 완료하는 데 waitForCompletion를 사용하는 방법에 대한 자세한 내용은 전송이 완료될 때까지 대기를 참조하세요. 전송이 완료될 때까지 대기하는 동안 상태 및 진행 상황에 대한 업데이트를 폴링하거나 수신 대기할 수 있습니다. 자세한 내용은 전송 상태 및 진행 상황 가져오기를 참조하십시오.

GitHub의 전체 예제를 참조하십시오.

파일 목록 업로드

여러 파일을 한 번에 업로드하려면 다음을 지정하여 TransferManager uploadFileList 메서드를 호출합니다.

- Amazon S3 버킷 이름
- 키 접두사 - 생성된 객체의 이름(객체를 넣을 버킷 내 경로) 앞에 붙습니다.
- File 객체 - 파일 경로를 생성할 상대 디렉터리를 나타냅니다.
- List 객체 - 업로드할 File 객체 세트를 포함합니다.

가져오기

```
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;
```

코드

```
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();
```

TransferManager의 `shutdownNow` 메서드를 호출하기 전에 전송을 성공적으로 완료하는 데 `waitForCompletion`를 사용하는 방법에 대한 자세한 내용은 전송이 완료될 때까지 대기를 참조하세요. 전송이 완료될 때까지 대기하는 동안 상태 및 진행 상황에 대한 업데이트를 폴링하거나 수신 대기할 수 있습니다. 자세한 내용은 전송 상태 및 진행 상황 가져오기를 참조하십시오.

`uploadFileList`에서 반환한 MultipleFileUpload 객체를 사용하여 전송 상태 또는 진행 상황을 쿼리할 수 있습니다. 자세한 내용은 현재 전송 진행 상황 폴링 및 ProgressListener를 사용하여 전송 진행 상황 가져오기를 참조하십시오.

또한 `MultipleFileUpload's getSubTransfers` 메서드를 사용하여 전송될 각 파일의 `Upload` 객체를 가져올 수도 있습니다. 자세한 내용은 하위 전송 진행 상황 가져오기를 참조하십시오.

GitHub의 전체 예제를 참조하십시오.

디렉터리 업로드

TransferManager의 `uploadDirectory` 메서드와 하위 디렉터리의 파일을 재귀적으로 복사하는 옵션을 사용하여 전체 파일 디렉터리를 업로드할 수 있습니다. Amazon S3 버킷 이름, S3 키 접두사, File 객

체(복사할 로컬 디렉터리를 나타냄) 및 boolean 값(하위 디렉터리를 재귀적으로 복사할지 여부를 나타내는 값으로, true 또는 false)을 제공합니다.

가져오기

```
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;
```

코드

```
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();
```

TransferManager의 shutdownNow 메서드를 호출하기 전에 전송을 성공적으로 완료하는 데 waitForCompletion를 사용하는 방법에 대한 자세한 내용은 전송이 완료될 때까지 대기를 참조하세요. 전송이 완료될 때까지 대기하는 동안 상태 및 진행 상황에 대한 업데이트를 폴링하거나 수신 대기할 수 있습니다. 자세한 내용은 전송 상태 및 진행 상황 가져오기를 참조하십시오.

uploadFileList에서 반환한 MultipleFileUpload 객체를 사용하여 전송 상태 또는 진행 상황을 쿼리할 수 있습니다. 자세한 내용은 현재 전송 진행 상황 폴링 및 ProgressListener를 사용하여 전송 진행 상황 가져오기를 참조하십시오.

또한 MultipleFileUpload's getSubTransfers 메서드를 사용하여 전송될 각 파일의 Upload 객체를 가져올 수도 있습니다. 자세한 내용은 하위 전송 진행 상황 가져오기를 참조하십시오.

GitHub의 전체 예제를 참조하십시오.

파일 또는 디렉터리 다운로드

TransferManager 클래스를 사용하여 Amazon S3에서 단일 파일(Amazon S3 객체) 또는 디렉터리(Amazon S3 버킷 이름 뒤에 객체 접두사 지정)를 다운로드할 수 있습니다.

주제

- 단일 파일 다운로드
- 디렉터리 다운로드

단일 파일 다운로드

TransferManager의 `download` 메서드를 사용하여 다운로드할 객체를 포함하는 Amazon S3 버킷 이름, 키(객체) 이름 및 로컬 시스템에 생성할 파일을 나타내는 File 객체를 지정합니다.

가져오기

```
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;
```

코드

```
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();
```

TransferManager의 shutdownNow 메서드를 호출하기 전에 전송을 성공적으로 완료하는 데 waitForCompletion를 사용하는 방법에 대한 자세한 내용은 전송이 완료될 때까지 대기를 참조하세요. 전송이 완료될 때까지 대기하는 동안 상태 및 진행 상황에 대한 업데이트를 폴링하거나 수신 대기할 수 있습니다. 자세한 내용은 전송 상태 및 진행 상황 가져오기를 참조하십시오.

GitHub의 전체 예제를 참조하십시오.

디렉터리 다운로드

Amazon S3에서 공통 키 접두사(파일 시스템의 디렉터리와 유사)를 공유하는 파일 세트를 다운로드하려면 TransferManager downloadDirectory 메서드를 사용합니다. 이 메서드는 다운로드할 객체를 포함하는 Amazon S3 버킷 이름, 모든 객체에서 공유하는 객체 접두사, 그리고 파일을 로컬 시스템으로 다운로드할 디렉터리를 나타내는 File 객체를 사용합니다. 이름이 지정된 디렉터리가 아직 없으면 생성됩니다.

가져오기

```
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;
```

코드

```
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();
```

TransferManager의 shutdownNow 메서드를 호출하기 전에 전송을 성공적으로 완료하는 데 waitForCompletion를 사용하는 방법에 대한 자세한 내용은 전송이 완료될 때까지 대기를 참조하세요. 전송이 완료될 때까지 대기하는 동안 상태 및 진행 상황에 대한 업데이트를 폴링하거나 수신 대기할 수 있습니다. 자세한 내용은 전송 상태 및 진행 상황 가져오기를 참조하십시오.

GitHub의 전체 예제를 참조하십시오.

객체 복사

한 S3 버킷에서 다른 버킷으로 객체를 복사하려면 TransferManager copy 메서드를 사용합니다.

가져오기

```
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;
```

코드

```
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();
```

GitHub의 전체 예제를 참조하십시오.

전송 완료 대기

전송이 완료될 때까지 애플리케이션(또는 스레드)을 차단할 수 있는 경우 Transfer 인터페이스의 `waitForCompletion` 메서드를 사용하여 전송이 완료되거나 예외가 발생할 때까지 차단할 수 있습니다.

```
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);
}
```

`waitForCompletion`를 호출하기 전에 이벤트를 폴링하거나, 별도의 스레드에서 폴링 메커니즘을 구현하거나, ProgressListener를 사용하여 비동기적으로 진행 상황 업데이트를 수신하면 전송 진행 상황을 확인할 수 있습니다.

GitHub의 전체 예제를 참조하십시오.

전송 상태 및 진행 상황 가져오기

TransferManager `upload*`, `download*` 및 `copy` 메서드에 의해 반환되는 각각의 클래스는 단일 파일 작업인지 다중 파일 작업인지에 따라 다음 클래스 중 하나의 인스턴스를 반환합니다.

클래스	반환 메서드
복사	`copy`
다운로드	`download`
MultipleFileDownload	`downloadDirectory`
업로드	`upload`
MultipleFileUpload	`uploadFileList` , `uploadDirectory`

이들 클래스는 모두 Transfer 인터페이스를 구현합니다. Transfer는 전송 진행 상황을 가져오거나, 전송을 일시 중지 또는 재개하거나, 전송의 현재 또는 최종 상태를 가져올 수 있는 유용한 메서드를 제공합니다.

주제

- 현재 전송 진행 상황 폴링
- ProgressListener를 사용하여 전송 진행 상황 가져오기
- 하위 전송 진행 상황 가져오기

현재 전송 진행 상황 폴링

이 루프는 전송 진행 상황을 출력하며, 실행 중에 현재 진행 상황을 검사하고, 완료되었을 때 최종 상태를 출력합니다.

가져오기

```
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;
```

코드

```
// 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);
```

GitHub의 전체 예제를 참조하십시오.

ProgressListener를 사용하여 전송 진행 상황 가져오기

전송 인터페이스의 addProgressListener 메서드를 사용하여 ProgressListener를 모든 전송에 연결할 수 있습니다.

ProgressListener에는 progressChanged라는 메서드 하나만 필요하며, 이 메서드는 ProgressEvent 객체를 사용합니다. 이 객체를 사용하면 getBytes 메서드를 호출하여 총 작업 바이트 수와 getBytesTransferred를 호출하여 지금까지 전송된 바이트 수를 가져올 수 있습니다.

가져오기

```
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;
```

코드

```
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();
```

GitHub의 전체 예제를 참조하십시오.

하위 전송 진행 상황 가져오기

MultipleFileUpload 클래스는 getSubTransfers 메서드를 호출하여 하위 전송에 대한 정보를 반환할 수 있습니다. 각 하위 전송의 개별 전송 상태 및 진행 상황을 제공하는 수정 불가능한 업로드 컬렉션 객체를 반환합니다.

가져오기

```
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;
```

코드

```
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);
```

GitHub의 전체 예제를 참조하십시오.

추가 정보

- Amazon Simple Storage Service 사용 설명서의 객체 키

Amazon S3 버킷을 웹 사이트로 구성

웹 사이트로 작동하도록 Amazon S3 버킷을 구성할 수 있습니다. 이렇게 하려면 웹 사이트 구성을 설정해야 합니다.

> **Note**
>
> 이 코드 예제에서는 사용자가 AWS SDK for Java 사용의 내용을 이해하고 개발을 위한 AWS 자격 증명 및 리전 설정의 정보를 사용하여 기본 AWS 자격 증명을 구성했다고 가정합니다.

버킷의 웹 사이트 구성 설정

Amazon S3버킷의 웹 사이트 구성을 설정하려면 구성을 설정할 버킷 이름과 버킷의 웹 사이트 구성이 포함된 BucketWebsiteConfiguration 객체를 사용하여 AmazonS3의 `setWebsiteConfiguration` 메서드를 호출합니다.

인덱스 문서 설정은 필수이며, 그 밖의 다른 파라미터는 선택적 파라미터입니다.

가져오기

```
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;
```

코드

```
    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**
>
> 웹 사이트 구성을 설정해도 버킷의 액세스 권한을 수정되지 않습니다. 또한 파일이 웹에 표시되도록 하려면 버킷 내 파일에 대한 퍼블릭 읽기 액세스 권한을 허용하는 버킷 정책을 설정해야 합니다. 자세한 내용은 버킷 정책을 사용하여 Amazon S3 버킷에 대한 액세스 관리 단원을 참조하세요.

GitHub의 전체 예제를 참조하십시오.

버킷의 웹 사이트 구성 가져오기

Amazon S3 버킷의 웹 사이트 구성을 가져오려면 구성을 가져올 버킷의 이름과 함께 AmazonS3의 `getWebsiteConfiguration` 메서드를 호출합니다.

구성이 BucketWebsiteConfiguration 객체로 반환됩니다. 버킷에 대한 웹 사이트 구성이 없으면 `null`이 반환됩니다.

가져오기

```
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;
```

코드

```
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);
}
```

GitHub의 전체 예제를 참조하십시오.

버킷의 웹 사이트 구성 삭제

Amazon S3 버킷의 웹 사이트 구성을 삭제하려면 구성을 삭제할 버킷의 이름과 함께 AmazonS3의 `deleteWebsiteConfiguration` 메서드를 호출합니다.

가져오기

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
```

코드

```
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);
}
```

GitHub의 전체 예제를 참조하십시오.

추가 정보

- Amazon S3 API 참조에서 버킷 웹사이트 추가

- Amazon S3 API 참조에서 버킷 웹사이트 가져오기
- Amazon S3 API 참조에서 버킷 웹사이트 삭제

Amazon S3 클라이언트 측 암호화 사용

Amazon S3 암호화 클라이언트로 데이터를 암호화하는 것은 Amazon S3에 저장된 민감한 정보를 위해 보호 계층을 하나 더 추가하는 방법입니다. 이 단원의 예제에서는 애플리케이션용 Amazon S3 암호화 클라이언트를 만들고 구성하는 방법을 보여 줍니다.

암호화를 처음 사용하는 경우 AWS KMS 개발자 안내서의 암호화 기본 사항에서 암호화 용어 및 알고리즘에 대한 기본 개요를 참조하세요. 모든 AWS SDK의 암호화 지원에 대한 자세한 내용은 Amazon Web Services 일반 참조의 Amazon S3 클라이언트 측 암호화에 대한 AWS SDK 지원을 참조하세요.

> **Note**
>
> 이 코드 예제에서는 개발자가 AWS SDK for Java 사용의 내용을 이해하고 개발을 위한 AWS 자격 증명 및 리전 설정의 정보를 사용하여 기본 AWS 자격 증명을 구성했다고 가정합니다.

AWS SDK for Java의 버전 1.11.836 또는 그 이전 버전을 사용하는 경우 애플리케이션을 최신 버전으로 마이그레이션하는 방법에 대한 자세한 내용은 Amazon S3 암호화 클라이언트 마이그레이션을 참조하세요. 마이그레이션할 수 없는 경우 GitHub에서 이 전체 예제를 참조하세요.

또는 AWS SDK for Java의 버전 1.11.837 이상을 사용하는 경우 아래 나열된 예제 항목을 살펴보고 Amazon S3 클라이언트 측 암호화를 사용하세요.

주제

- 클라이언트 마스터 키를 사용한 Amazon S3 클라이언트 측 암호화
- AWS KMS 관리형 키를 사용한 Amazon S3 클라이언트 측 암호화

클라이언트 마스터 키를 사용한 Amazon S3 클라이언트 측 암호화

다음 예제에서는 AmazonS3EncryptionClientV2Builder 클래스를 사용하여 클라이언트 측 암호화가 활성화된 Amazon S3 클라이언트를 만듭니다. 이렇게 설정하면 이 클라이언트로 Amazon S3에 업로드하는 모든 객체가 암호화됩니다. 또한 이 클라이언트를 사용하여 Amazon S3에서 가져오는 모든 객체는 자동으로 암호 해독됩니다.

> **Note**
>
> 다음 예제에서는 고객 관리형 클라이언트 마스터 키로 Amazon S3 클라이언트 측 암호화를 사용하는 방법을 보여 줍니다. AWS KMS 관리형 키로 암호화를 사용하는 방법은 AWS KMS 관리형 키를 사용한 Amazon S3 클라이언트 측 암호화를 참조하세요.

클라이언트 측 Amazon S3 암호화를 활성화할 때는 엄격히 인증된 암호화 또는 암호화 등 두 가지 암호화 모드 중에서 선택할 수 있습니다. 다음 단원에서는 각각의 유형을 활성화하는 방법에 대해 설명합니다. 각 모드에서 사용하는 알고리즘은 CryptoMode 정의를 참조하십시오.

필수 가져오기

이 예제에 사용할 다음 클래스를 가져옵니다.

가져오기

```
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;
```

엄격히 인증된 암호화

어떠한 `CryptoMode`도 지정되지 않은 경우 엄격한 인증 암호화가 기본 모드입니다.

이 모드를 명시적으로 활성화하려면 `withCryptoConfiguration` 메서드에 `StrictAuthenticatedEncryption` 값을 지정합니다.

> **Note**
>
> 클라이언트 측 인증된 암호화를 사용하려면 애플리케이션의 클래스 경로에 최신 Bouncy Castle jar 파일을 포함시켜야 합니다.

코드

```
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");
```

인증된 암호화 모드

`AuthenticatedEncryption` 모드를 사용하여 암호화하면 향상된 키 래핑 알고리즘이 적용됩니다. 이 모드에서 암호를 해독할 때는 이 알고리즘이 해독된 객체의 무결성을 확인하고, 확인에 실패하면 예외가 발생합니다. 인증된 암호화의 작동 방식에 대한 자세한 내용은 Amazon S3 클라이언트 측 인증된 암호화 블로그 게시물을 참조하세요.

> **Note**
>
> 클라이언트 측 인증된 암호화를 사용하려면 애플리케이션의 클래스 경로에 최신 Bouncy Castle jar 파일을 포함시켜야 합니다.

이 모드를 활성화하려면 `withCryptoConfiguration` 메서드에 `AuthenticatedEncryption` 값을 지정합니다.

코드

```
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");
```

AWS KMS 관리형 키를 사용한 Amazon S3 클라이언트 측 암호화

다음 예제에서는 AmazonS3EncryptionClientV2Builder 클래스를 사용하여 클라이언트 측 암호화가 활성화된 Amazon S3 클라이언트를 만듭니다. 이렇게 구성하면 이 클라이언트로 Amazon S3에 업로드하는 모든 객체가 암호화됩니다. 또한 이 클라이언트를 사용하여 Amazon S3에서 가져오는 모든 객체는 자동으로 해독됩니다.

> **Note**
>
> 다음 예제에서는 AWS KMS 관리형 키로 Amazon S3 클라이언트 측 암호화를 사용하는 방법을 보여 줍니다. 고객 자체 키로 암호화를 사용하는 방법은 클라이언트 마스터 키로 Amazon S3 클라이언트 측 암호화를 참조하십시오.

클라이언트 측 Amazon S3 암호화를 활성화할 때는 엄격히 인증된 암호화 또는 암호화 등 두 가지 암호화 모드 중에서 선택할 수 있습니다. 다음 단원에서는 각각의 유형을 활성화하는 방법에 대해 설명합니다. 각 모드에서 사용하는 알고리즘은 CryptoMode 정의를 참조하십시오.

필수적 가져오기

이 예제에 사용할 다음 클래스를 가져옵니다.

가져오기

```
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;
```

엄격히 인증된 암호화

어떠한 `CryptoMode`도 지정되지 않은 경우 엄격한 인증 암호화가 기본 모드입니다.

이 모드를 활성화하려면 `withCryptoConfiguration` 메서드에 `StrictAuthenticatedEncryption` 값을 지정합니다.

> **Note**
>
> 클라이언트 측 인증된 암호화를 사용하려면 애플리케이션의 클래스 경로에 최신 Bouncy Castle jar 파일을 포함시켜야 합니다.

코드

```
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));
```

Amazon S3 암호화 클라이언트에서 `putObject` 메서드를 호출하여 객체를 업로드합니다.

코드

```
s3Encryption.putObject(bucket_name, ENCRYPTED_KEY3, "This is the 3rd content to encrypt
 with a key created in the {console}");
```

같은 클라이언트를 사용하여 객체를 검색할 수 있습니다. 이 예제에서는 `getObjectAsString` 메서드를 호출하여 저장된 문자열을 검색합니다.

코드

```
System.out.println(s3Encryption.getObjectAsString(bucket_name, ENCRYPTED_KEY3));
```

인증된 암호화 모드

`AuthenticatedEncryption` 모드를 사용하여 암호화하면 향상된 키 래핑 알고리즘이 적용됩니다. 이 모드에서 암호를 해독할 때는 이 알고리즘이 해독된 객체의 무결성을 확인하고, 확인에 실패하면 예

외가 발생합니다. 인증된 암호화의 작동 방식에 대한 자세한 내용은 Amazon S3클라이언트 측 인증된 암호화 블로그 게시물을 참조하십시오.

> **Note**
>
> 클라이언트 측 인증된 암호화를 사용하려면 애플리케이션의 클래스 경로에 최신 Bouncy Castle jar 파일을 포함시켜야 합니다.

이 모드를 활성화하려면 `withCryptoConfiguration` 메서드에 `AuthenticatedEncryption` 값을 지정합니다.

코드

```
AmazonS3EncryptionV2 s3Encryption = AmazonS3EncryptionClientV2Builder.standard()
         .withRegion(Regions.US_WEST_2)
         .withCryptoConfiguration(new
 CryptoConfigurationV2().withCryptoMode((CryptoMode.AuthenticatedEncryption)))
         .withEncryptionMaterialsProvider(new KMSEncryptionMaterialsProvider(keyId))
         .build();
```

AWS KMS 클라이언트 구성

Amazon S3 암호화 클라이언트는 명시적으로 지정하지 않는 한 기본적으로 AWS KMS 클라이언트를 만듭니다.

자동으로 생성되는 AWS KMS 클라이언트의 지역을 설정하려면 `awsKmsRegion`를 설정합니다.

코드

```
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();
```

또는 고유한 AWS KMS 클라이언트를 사용하여 암호화 클라이언트를 초기화할 수 있습니다.

코드

```
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();
```

AWS SDK for Java를 사용하는 Amazon SQS 예제

이 단원에서는 AWS SDK for Java를 사용한 Amazon SQS 프로그래밍의 예제를 제공합니다.

> **Note**
>
> 예제에는 각 기술을 보여주는 데 필요한 코드만 포함되어 있습니다. 전체 예제 코드는 GitHub에 있습니다. 이 위치에서 단일 소스 파일을 다운로드하거나 리포지토리를 로컬로 복사하여 모든 예제를 빌드하고 실행할 수 있습니다.

주제

- Amazon SQS 메시지 대기열 사용
- Amazon SQS 메시지 전송, 수신 및 삭제
- Amazon SQS 메시지 대기열에 대한 긴 폴링 활성화
- Amazon SQS의 가시성 제한 시간 설정
- Amazon SQS에서 배달 못한 편지 대기열 사용

Amazon SQS 메시지 대기열 사용

메시지 대기열은 Amazon SQS에서 메시지를 안정적으로 전송하는 데 사용되는 논리적 컨테이너입니다. 표준과 선입선출(FIFO), 이렇게 두 가지 유형의 대기열이 있습니다. 대기열과 이러한 유형 간의 차이에 대해 자세히 알아보려면 Amazon SQS 개발자 안내서를 참조하세요.

이 주제에서는 AWS SDK for Java를 사용하여 Amazon SQS 대기열의 URL을 생성, 나열, 삭제 및 가져오는 방법을 설명합니다.

대기열 생성

AmazonSQS 클라이언트의 `createQueue` 메서드를 사용하여 대기열 파라미터를 설명하는 CreateQueueRequest 객체를 지정합니다.

가져오기

```
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;
```

코드

```
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;
    }
}
```

`createQueue`의 간소화된 형식을 사용할 수 있습니다. 이 형식에서는 표준 대기열을 생성하는 데 대기열 이름만 있으면 됩니다.

```
sqs.createQueue("MyQueue" + new Date().getTime());
```

GitHub의 전체 예제를 참조하십시오.

대기열 나열

계정에 대한 Amazon SQS 대기열을 나열하려면 AmazonSQS 클라이언트의 `listQueues` 메서드를 호출합니다.

가져오기

```
import com.amazonaws.services.sqs.AmazonSQS;
import com.amazonaws.services.sqs.AmazonSQSClientBuilder;
import com.amazonaws.services.sqs.model.ListQueuesResult;
```

코드

```
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);
}
```

파라미터 없이 `listQueues` 오버로드를 사용하면 모든 대기열이 반환됩니다. `ListQueuesRequest` 객체를 전달하여 반환된 결과를 필터링할 수 있습니다.

가져오기

```
import com.amazonaws.services.sqs.AmazonSQS;
import com.amazonaws.services.sqs.AmazonSQSClientBuilder;
import com.amazonaws.services.sqs.model.ListQueuesRequest;
```

코드

```
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);
}
```

GitHub의 전체 예제를 참조하십시오.

대기열의 URL 가져오기

AmazonSQS 클라이언트의 `getQueueUrl` 메서드를 호출합니다.

가져오기

```
import com.amazonaws.services.sqs.AmazonSQS;
import com.amazonaws.services.sqs.AmazonSQSClientBuilder;
```

코드

```
AmazonSQS sqs = AmazonSQSClientBuilder.defaultClient();
String queue_url = sqs.getQueueUrl(QUEUE_NAME).getQueueUrl();
```

GitHub의 전체 예제를 참조하십시오.

대기열 삭제

대기열의 URL을 AmazonSQS 클라이언트의 `deleteQueue` 메서드에 제공합니다.

가져오기

```
import com.amazonaws.services.sqs.AmazonSQS;
import com.amazonaws.services.sqs.AmazonSQSClientBuilder;
```

코드

```
AmazonSQS sqs = AmazonSQSClientBuilder.defaultClient();
sqs.deleteQueue(queue_url);
```

GitHub의 전체 예제를 참조하십시오.

추가 정보

- Amazon SQS 개발자 안내서의 Amazon SQS 대기열 작동 방식
- Amazon SQS API 참조에서 CreateQueue
- Amazon SQS API 참조에서 GetQueueUrl
- Amazon SQS API 참조에서 ListQueues
- Amazon SQS API 참조에서 DeleteQueues

Amazon SQS 메시지 전송, 수신 및 삭제

이 주제에서는 Amazon SQS 메시지를 전송, 수신 및 삭제하는 방법을 설명합니다. 메시지는 항상 SQS 대기열을 사용하여 전달됩니다.

메시지 보내기

AmazonSQS 클라이언트의 sendMessage 메서드를 호출하여 Amazon SQS 대기열에 단일 메시지를 추가합니다. 대기열의 URL, 메시지 본문 및 선택적 지연 값(초 단위)이 포함된 SendMessageRequest 객체를 제공합니다.

가져오기

```
import com.amazonaws.services.sqs.AmazonSQS;
import com.amazonaws.services.sqs.AmazonSQSClientBuilder;
import com.amazonaws.services.sqs.model.SendMessageRequest;
```

코드

```
SendMessageRequest send_msg_request = new SendMessageRequest()
        .withQueueUrl(queueUrl)
        .withMessageBody("hello world")
        .withDelaySeconds(5);
sqs.sendMessage(send_msg_request);
```

GitHub의 전체 예제를 참조하십시오.

한 번에 여러 메시지 전송

단일 요청으로 두 개 이상의 메시지를 전송할 수 있습니다. 여러 메시지를 보내려면 AmazonSQS 클라이언트의 sendMessageBatch 메서드를 사용하십시오. 이 메서드는 대기열 URL과 메시지 목록(각각 SendMessageBatchRequestEntry)이 포함된 SendMessageBatchRequest를 받아 전송합니다. 메시지마다 지연 값(선택 사항)을 설정할 수도 있습니다.

가져오기

```
import com.amazonaws.services.sqs.model.SendMessageBatchRequest;
import com.amazonaws.services.sqs.model.SendMessageBatchRequestEntry;
```

코드

```
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);
```

GitHub의 전체 예제를 참조하십시오.

메시지 수신

AmazonSQS 클라이언트의 `receiveMessage` 메서드를 호출하여 대기열의 URL을 이 메서드에 전달함으로써 현재 대기열에 있는 메시지를 모두 가져옵니다. 메시지는 Message 객체의 목록으로 반환됩니다.

가져오기

```
import com.amazonaws.services.sqs.AmazonSQSClientBuilder;
import com.amazonaws.services.sqs.model.AmazonSQSException;
import com.amazonaws.services.sqs.model.SendMessageBatchRequest;
```

코드

```
List<Message> messages = sqs.receiveMessage(queueUrl).getMessages();
```

수신 후 메시지 삭제

메시지를 수신하고 메시지의 내용을 처리한 후에는 메시지의 수신 핸들과 대기열 URL을 AmazonSQS 클라이언트의 `deleteMessage` 메서드로 전송하여 대기열에서 메시지를 삭제합니다.

코드

```
for (Message m : messages) {
    sqs.deleteMessage(queueUrl, m.getReceiptHandle());
}
```

GitHub의 전체 예제를 참조하십시오.

추가 정보

- Amazon SQS 개발자 안내서의 Amazon SQS 큐 작동 방식
- Amazon SQS API 참조에서 SendMessage
- Amazon SQS API 참조에서 SendMessageBatch

- Amazon SQS API 참조에서 ReceiveMessage
- Amazon SQS API 참조에서 DeleteMessage

Amazon SQS 메시지 대기열에 대한 긴 폴링 활성화

Amazon SQS는 기본적으로 짧은 폴링을 사용하여 서버의 하위 세트(가중치 기반 무작위 배포 기반)만을 쿼리하여 응답에 포함할 메시지를 사용할지 여부를 결정합니다.

긴 폴링은 Amazon SQS 대기열로 전송된 ReceiveMessage 요청에 대한 응답으로 반환할 수 있는 메시지가 없을 때 빈 응답 수를 줄이고 거짓 빈 응답을 제거하여 Amazon SQS 사용 시 드는 비용을 줄여 줍니다.

> **Note**
>
> 긴 폴링 빈도는 1–20초로 설정할 수 있습니다.

대기열 생성 시 긴 폴링 활성화

Amazon SQS 대기열을 생성할 때 긴 폴링을 활성화하려면 AmazonSQS 클래스의 `createQueue` 메서드를 호출하기 전에 CreateQueueRequest 객체에 `ReceiveMessageWaitTimeSeconds` 속성을 설정하세요.

가져오기

```
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;
```

코드

```
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;
    }
}
```

GitHub의 전체 예제를 참조하세요.

기존 대기열에 대해 긴 폴링 활성화

대기열을 생성할 때 긴 폴링을 활성화하는 것 외에도, AmazonSQS 클래스의 `setQueueAttributes` 메서드를 호출하기 전에 SetQueueAttributesRequest에 `ReceiveMessageWaitTimeSeconds`를 설정하여 기존 대기열에서 긴 폴링을 활성화할 수도 있습니다.

가져오기

```
import com.amazonaws.services.sqs.model.SetQueueAttributesRequest;
```

코드

```
SetQueueAttributesRequest set_attrs_request = new SetQueueAttributesRequest()
        .withQueueUrl(queue_url)
        .addAttributesEntry("ReceiveMessageWaitTimeSeconds", "20");
sqs.setQueueAttributes(set_attrs_request);
```

GitHub의 전체 예제를 참조하세요.

메시지 수신 시 긴 폴링 활성화

AmazonSQS 클래스의 `receiveMessage` 메서드에 제공하는 ReceiveMessageRequest에서 대기 시간을 초 단위로 설정하여 메시지 수신 시 긴 폴링을 활성화할 수 있습니다.

> **Note**
>
> AWS 클라이언트의 요청 제한 시간이 최대 긴 폴링 시간(20초)보다 길어야만 다음 폴링 이벤트까지 기다리는 동안 `receiveMessage` 요청의 제한 시간이 초과되지 않습니다.

가져오기

```
import com.amazonaws.services.sqs.model.ReceiveMessageRequest;
```

코드

```
ReceiveMessageRequest receive_request = new ReceiveMessageRequest()
        .withQueueUrl(queue_url)
        .withWaitTimeSeconds(20);
sqs.receiveMessage(receive_request);
```

GitHub의 전체 예제를 참조하세요.

추가 정보

- Amazon SQS 개발자 안내서의 Amazon SQS키 정책
- Amazon SQS API 참조에서 CreateQueue
- Amazon SQS API 참조에서 ReceiveMessage
- Amazon SQS API 참조에서 SetQueueAttributes

Amazon SQS의 가시성 제한 시간 설정

메시지가 Amazon SQS에 수신되면 수신 여부를 확인하기 위해 삭제될 때까지 대기열에서 유지됩니다. 수신되었지만 삭제되지 않은 메시지는 메시지가 처리 및 삭제되기 전에 두 번 이상 수신되지 않도록 하기 위해 지정된 제한 시간 초과 이후에는 후속 요청에서 제공됩니다.

> **Note**
>
> 표준 대기열을 사용 중인 경우 제한 시간 초과를 설정해도 메시지가 두 번 이상 수신되지 않는다고 장담할 수 없습니다. 표준 대기열을 사용 중인 경우 동일 메시지가 두 번 이상 전달된 경우를 코드에서 처리할 수 있도록 해야 합니다.

단일 메시지에 대한 메시지 제한 시간 초과 설정

메시지를 수신한 후에는 AmazonSQS 클래스의 `changeMessageVisibility` 메서드에 전달하는 ChangeMessageVisibilityRequest에 해당 수신 핸들을 전달하여 가시성 제한 시간을 수정할 수 있습니다.

가져오기

```
import com.amazonaws.services.sqs.AmazonSQS;
import com.amazonaws.services.sqs.AmazonSQSClientBuilder;
```

코드

```
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);
```

GitHub의 전체 예제를 참조하십시오.

한 번에 여러 메시지에 대한 메시지 제한 시간 초과 설정

한 번에 여러 메시지에 대한 메시지 제한 시간 초과를 설정하려면 ChangeMessageVisibilityBatchRequestEntry 객체 목록을 생성합니다. 이때 각 객체에는 고유 ID 문자열과 수신 핸들이 들어 있습니다. 다음에는 이 목록을 Amazon SQS 클라이언트 클래스의 `changeMessageVisibilityBatch` 메서드로 전달합니다.

가져오기

```
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;
```

코드

```
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);
```

GitHub의 전체 예제를 참조하십시오.

추가 정보

- Amazon SQS개발자 안내서의 가시성 제한 시간
- Amazon SQS API 참조에서 SetQueueAttributes
- Amazon SQS API 참조에서 GetQueueAttributes
- Amazon SQS API 참조에서 ReceiveMessage
- Amazon SQS API 참조에서 ChangeMessageVisibility
- Amazon SQS API 참조에서 ChangeMessageVisibilityBatch

Amazon SQS에서 배달 못한 편지 대기열 사용

Amazon SQS는 배달 못한 편지 대기열을 지원합니다. 배달 못한 편지 대기열은 다른(소스) 대기열에서 성공적으로 처리할 수 없는 메시지를 보낼 수 있는 대기열입니다. 배달 못한 편지 대기열에서 이 메시지를 구분하고 격리하여 처리에 실패한 이유를 확인할 수 있습니다.

배달 못한 편지 대기열 생성

배달 못한 편지 대기열은 일반적인 대기열과 동일한 방식으로 생성되지만, 다음과 같은 제한이 수반됩니다.

- 배달 못한 편지 대기열은 소스 대기열과 동일한 유형의 대기열(FIFO 또는 표준)이어야 합니다.
- 데드레터 대기열은 소스 대기열과 동일한 AWS 계정 및 리전을 사용하여 생성해야 합니다.

두 가지 동일한 Amazon SQS 대기열을 생성하는 방법. 이 대기열 중 하나는 배달 못한 편지 대기열로 사용됩니다.

가져오기

```
import com.amazonaws.services.sqs.AmazonSQS;
import com.amazonaws.services.sqs.AmazonSQSClientBuilder;
import com.amazonaws.services.sqs.model.AmazonSQSException;
```

코드

```
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;
    }
}
```

GitHub의 전체 예제를 참조하십시오.

배달 못한 편지 대기열 하나를 소스 대기열로 지정

배달 못한 편지 대기열을 지정하려면 먼저 리드라이브 정책을 생성하고 나서 대기열의 속성에서 해당 정책을 설정해야 합니다. 리드라이브 정책은 JSON에 명시되어 있으며, 배달 못한 편지 대기열의 ARN과 배달 못한 편지 대기열로 전송되기 전에 메시지를 수신하여 처리하지 못한 최대 횟수를 지정합니다.

소스 대기열에 대한 재구동 정책을 설정하려면 JSON 재구동 정책으로 `RedrivePolicy` 속성을 설정한 SetQueueAttributesRequest 객체를 사용하여 AmazonSQS 클래스의 `setQueueAttributes` 메서드를 호출하세요.

가져오기

```
import com.amazonaws.services.sqs.model.GetQueueAttributesRequest;
import com.amazonaws.services.sqs.model.GetQueueAttributesResult;
import com.amazonaws.services.sqs.model.SetQueueAttributesRequest;
```

코드

```
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);
```

GitHub의 전체 예제를 참조하십시오.

추가 정보

- Amazon SQS 개발자 안내서의 Amazon SQS 데드레터 대기열 사용
- Amazon SQS API 참조에서 SetQueueAttributes

AWS SDK for Java를 사용하는 Amazon SWF 예제

Amazon SWF는 개발자가 활동, 하위 워크플로 또는 Lambda 작업으로 구성된 병렬 또는 순차 단계를 가질 수 있는 분산 워크플로를 구축하고 확장하는 데 도움이 되는 워크플로 관리 서비스입니다.

AWS SDK for Java를 사용하여 Amazon SWF로 작업하는 방법에는 SWF 클라이언트 객체를 사용하는 방법과 Java용 AWS Flow Framework를 사용하는 방법, 이렇게 두 가지가 있습니다. Java용 AWS Flow Framework는 주석을 많이 사용하고 AspectJ 및 Spring Framework 같은 추가 라이브러리에 의존하므로 초기 구성이 더 어렵습니다. 하지만 규모가 크거나 복잡한 프로젝트의 경우 Java용 AWS Flow Framework를 사용하면 코딩 시간을 절약할 수 있습니다. 자세한 내용은 Java용 AWS Flow Framework 개발자 안내서를 참조하세요.

이 단원에서는 AWS SDK for Java 클라이언트를 직접 사용하여 Amazon SWF를 프로그래밍하는 예를 제공합니다.

주제

SWF 기본 사항

이러한 기본 사항은 AWS SDK for Java를 사용한 Amazon SWF 작업의 일반적인 패턴입니다. 기본적으로는 참고용입니다. 자세한 소개 자습서는 간단한 Amazon SWF 애플리케이션 빌드 단원을 참조하세요.

종속성

기본 Amazon SWF 애플리케이션에는 다음 종속성이 필요하며, 이러한 종속성은 AWS SDK for Java에 포함되어 있습니다.

- 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**
>
> 이러한 패키지의 버전 번호는 보유한 SDK 버전에 따라 다르지만, SDK와 함께 제공된 버전의 경우 호환성 테스트를 거쳤으므로 이러한 버전을 사용해야 합니다.

Java용 AWS Flow Framework 애플리케이션에는 추가 설정 및 추가 종속성이 필요합니다. 이 프레임워크 사용에 대한 자세한 내용은 Java용 AWS Flow Framework 개발자 안내서를 참조하세요.

가져오기

일반적으로 코드 개발용으로 다음 가져오기를 사용할 수 있습니다.

```
import com.amazonaws.services.simpleworkflow.AmazonSimpleWorkflowClientBuilder;
import com.amazonaws.services.simpleworkflow.model.*;
```

하지만 필요한 클래스만 가져오는 것이 바람직합니다. 대체로는 `com.amazonaws.services.simpleworkflow.model` Workspace에서 특정 클래스를 지정할 가능성이 높습니다.

```
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;
```

Java용 AWS Flow Framework를 사용 중인 경우 `com.amazonaws.services.simpleworkflow.flow` 작업공간에서 클래스를 가져옵니다. 예:

```
import com.amazonaws.services.simpleworkflow.AmazonSimpleWorkflow;
import com.amazonaws.services.simpleworkflow.flow.ActivityWorker;
```

> **Note**
>
> Java용 AWS Flow Framework에는 기본 AWS SDK for Java의 요구 사항 외에도 추가적인 요구 사항이 있습니다. 자세한 내용은 Java용 AWS Flow Framework 개발자 안내서를 참조하세요.

SWF 클라이언트 클래스 사용

Amazon SWF에 대한 기본 인터페이스는 AmazonSimpleWorkflowClient 또는 AmazonSimpleWorkflowAsyncClient 클래스를 사용하는 것입니다. 두 클래스 간의 주된 차이점은 *AsyncClient 클래스의 경우 동시(비동기) 프로그래밍을 위해 Future 객체를 반환한다는 것입니다.

```
AmazonSimpleWorkflowClient swf = AmazonSimpleWorkflowClientBuilder.defaultClient();
```

간단한 Amazon SWF 애플리케이션 구축

이 항목에서는 AWS SDK for Java를 사용하여 Amazon SWF 애플리케이션을 프로그래밍하는 방법을 소개하고 그 과정에서 몇 가지 중요한 개념을 설명합니다.

예제 소개

예제 프로젝트에서는 AWS 클라우드를 통해 전달된 워크플로 데이터(HelloWorld 식으로는 인사할 사람의 이름임)를 수락한 다음 응답으로 인사말을 출력하는 단일 활동으로 워크플로를 생성합니다.

이 동작을 수행하는 Amazon SWF 애플리케이션은 표면적으로는 매우 간단해 보이지만, 함께 작동하는 여러 부분으로 이루어집니다.

- 도메인 - 워크플로 실행 데이터의 논리적 컨테이너로 사용됩니다.
- 하나 이상의 워크플로 - 워크플로의 활동 및 하위 워크플로의 논리적 실행 순서를 정의하는 코드 구성 요소를 나타냅니다.
- 워크플로 작업자 - 결정자라고도 하며, 결정 작업에 대해 폴링을 수행하거나 응답으로 수행되는 활동이나 하위 워크플로를 예약합니다.
- 하나 이상의 활동 - 각 활동은 워크플로 내 작업 단위를 나타냅니다.
- 활동 작업자 - 활동 작업을 폴링하고 응답으로 활동 메서드를 실행합니다.
- 하나 이상의 작업 목록 - Amazon SWF에서 유지 관리되며 워크플로 및 활동 작업자에 대한 요청을 발행하는 데 사용되는 대기열입니다. 워크플로 작업자에 대한 작업 목록의 작업을 결정 작업이라고 합니다. 활동 작업자에 대한 작업 목록의 작업을 활동 작업이라고 합니다.
- 워크플로 시작자 - 워크플로 실행을 시작합니다.

보이지는 않지만 Amazon SWF는 뒤에서 이러한 구성 요소의 작동을 조정하고, AWS 클라우드로부터의 흐름을 조정하고, 구성 요소 간에 데이터를 전달하고, 시간 초과 및 하트비트 알림을 처리하고, 워크플로 실행 기록을 기록합니다.

필수 조건

개발 환경

이 자습서에 사용되는 개발 환경은 다음으로 구성됩니다.

- 이 AWS SDK for Java.
- Apache Maven(3.3.1)
- JDK 1.7 이상. 이 자습서는 JDK 1.8.0을 사용해 개발 및 테스트하였습니다.
- 적합한 Java 텍스트 편집기(개발자 선택)

> **Note**
>
> Maven과 다른 빌드 시스템을 사용하는 경우에도 환경에 적합한 단계를 사용하여 프로젝트를 생성하고 여기에 제공된 개념을 사용하여 따라갈 수 있습니다. 다양한 빌드 시스템에서 AWS SDK for Java 사용 및 구성에 대한 자세한 내용은 시작하기에 나와 있습니다.
> 마찬가지로, Amazon SWF를 지원하는 AWS SDK를 사용하여 여기에 표시된 단계를 구현할 수도 있습니다. 단, 이 경우에는 좀 더 많은 노력이 요구됩니다.

필요한 외부 종속성이 모두 AWS SDK for Java와 함께 포함되어 있으므로 추가로 다운로드할 필요가 없습니다.

AWS 액세스

이 자습서를 성공적으로 마치려면 이 가이드의 기본 설정 단원에 설명된 대로 AWS 액세스 포털에 액세스할 수 있어야 합니다.

지침에서는 로컬 공유 `credentials` 파일에 복사하여 붙여넣는 임시 자격 증명에 액세스하는 방법을 설명합니다. 붙여넣는 임시 자격 증명은 Amazon SWF에 액세스할 권한이 있는 AWS IAM Identity Center의 IAM 역할과 연결되어야 합니다. 임시 자격 증명을 붙여넣으면 `credentials` 파일은 다음과 비슷하게 보입니다.

```
[default]
aws_access_key_id=AKIAIOSFODNN7EXAMPLE
aws_secret_access_key=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
aws_session_token=IQoJb3JpZ2luX2IQoJb3JpZ2luX2IQoJb3JpZ2luX2IQoJb3JpZ2luX2IQoJb3JpZVERYLONGSTRI
```

이러한 임시 자격 증명은 default 프로필과 연결되어 있습니다.

SWF 프로젝트 생성

1. Maven에서 새 프로젝트를 시작합니다.

```
mvn archetype:generate -DartifactId=helloswf \
-DgroupId=aws.example.helloswf -DinteractiveMode=false
```

그러면 표준 maven 프로젝트 구조의 새 프로젝트가 생성됩니다.

```
helloswf
### pom.xml
### src
    ### main
    #   ### java
    #       ### aws
    #           ### example
    #               ### helloswf
    #                   ### App.java
    ### test
        ### ...
```

test 디렉터리와 그 안에 포함된 항목은 이 자습서에서 사용되지 않으므로 모두 무시하거나 삭제할 수 있습니다. 또한 App.java도 새 클래스로 대체될 것이므로 삭제할 수 있습니다.

2. 프로젝트 pom.xml 파일을 편집하고 <dependencies> 블록 내에 해당 모듈에 대한 종속성을 추가하여 aws-java-sdk-simpleworkflow 모듈을 추가합니다.

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

3. Maven에서 JDK 1.7+가 지원되는 프로젝트가 빌드되는지 확인합니다. pom.xml에서 프로젝트(<dependencies> 블록 앞 또는 뒤)에 다음을 추가합니다.

```
<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>
```

프로젝트를 코딩합니다.

예제 프로젝트는 네 가지 개별 애플리케이션으로 구성되며, 각 애플리케이션을 하나씩 살펴보겠습니다.

- HelloTypes.java 다른 구성 요소와 공유되는 프로젝트의 도메인, 활동 및 워크플로 유형 데이터가 들어 있습니다. 또한 SWF에 이러한 유형을 등록하는 작업도 처리합니다.
- ActivityWorker.java 활동 작업을 폴링하며 응답으로 활동을 실행하는 활동 작업자가 들어 있습니다.
- WorkflowWorker.java 결정 작업을 폴링하거나 새 활동을 예약하는 워크플로 작업자(결정자)가 들어 있습니다.
- WorkflowStarter.java 새 워크플로 실행을 시작하는 워크플로 시작자가 들어 있으며, SWF에서 작업자가 소비할 결정 및 워크플로 작업 생성을 시작합니다.

모든 소스 파일에 적용되는 공통 단계

Java 클래스를 포함할 용도로 생성하는 모든 파일에는 몇 가지 공통점이 있습니다. 시간 관계상 별도로 언급되지 않더라도 프로젝트에 새 파일을 추가할 때마다 항상 이러한 단계의 수행됩니다.

1. 프로젝트의 `src/main/java/aws/example/helloswf/` 디렉터리에서 파일을 생성합니다.
2. `package` 선언을 각 파일의 시작 부분에 추가하여 네임스페이스를 선언합니다. 이 예제 프로젝트에는 다음이 사용됩니다.

   ```
   package aws.example.helloswf;
   ```

3. AmazonSimpleWorkflowClient 클래스와 `com.amazonaws.services.simpleworkflow.model` 네임스페이스의 여러 클래스에 대한 `import` 선언을 추가합니다. 간단히 말해서, 다음을 사용하겠습니다.

```
import com.amazonaws.regions.Regions;
import com.amazonaws.services.simpleworkflow.AmazonSimpleWorkflow;
import com.amazonaws.services.simpleworkflow.AmazonSimpleWorkflowClientBuilder;
import com.amazonaws.services.simpleworkflow.model.*;
```

도메인, 워크플로 및 활동 유형 등록

새로운 실행 가능한 클래스인 `HelloTypes.java`를 생성하는 것부터 시작하겠습니다. 이 파일에는 활동 및 워크플로 유형의 이름과 버전, 도메인 이름과 작업 목록 이름 등 워크플로의 다양한 부분에서 알고 있어야 하는 공유 데이터가 포함됩니다.

1. 텍스트 편집기를 열고 `HelloTypes.java` 파일을 생성하여 공통 단계에 따라 패키지 선언과 가져오기를 추가합니다.
2. `HelloTypes` 클래스를 선언하고 등록된 활동 및 워크플로 유형에 사용할 값을 해당 클래스에 제공합니다.

   ```
       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";
   ```

   이러한 값은 코드 전체에서 사용됩니다.
3. String 선언 뒤에 AmazonSimpleWorkflowClient 클래스의 인스턴스를 생성합니다. 이 인스턴스는 AWS SDK for Java에서 제공하는 Amazon SWF 메서드의 기본 인터페이스입니다.

   ```
   private static final AmazonSimpleWorkflow swf =

    AmazonSimpleWorkflowClientBuilder.standard().withRegion(Regions.DEFAULT_REGION).build();
   ```

   이전 코드 조각에서는 임시 자격 증명이 `default` 프로필과 연결되어 있다고 가정합니다. 다른 프로필을 사용하는 경우 위 코드를 다음과 같이 수정하고 *`profile_name`*을 실제 프로필 이름으로 바꾸십시오.

   ```
   private static final AmazonSimpleWorkflow swf =
           AmazonSimpleWorkflowClientBuilder
   ```

```
                .standard()
                .withCredentials(new ProfileCredentialsProvider("profile_name"))
                .withRegion(Regions.DEFAULT_REGION)
                .build();
```

4. 새 함수를 추가하여 SWF 도메인을 등록합니다. 도메인은 관련 SWF 활동 및 워크플로 유형을 포함할 논리적 컨테이너입니다. SWF 구성 요소는 동일한 도메인 안에 있는 경우에만 서로 통신할 수 있습니다.

```
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!");
}
```

도메인을 등록하는 경우 도메인에 이름(`:`, `/`, `|`, 제어 문자 또는 리터럴 문자열 '`arn'을 제외한 1~256자로 구성된 모든 문자 집합)과 보존 기간(Amazon SWF에서 워크플로 실행이 완료된 후 워크플로의 실행 내역 데이터를 보관하는 일수)을 지정합니다. 최대 워크플로 실행 보존 기간은 90일입니다. 자세한 내용은 RegisterDomainRequest를 참조하십시오.

해당 이름의 도메인이 이미 있으면 DomainAlreadyExistsException이 발생합니다. 도메인이 생성된 경우에는 관련이 없으므로 이 예외를 무시해도 됩니다.

> **Note**
>
> 이 코드는 AWS SDK for Java 메서드 작업 시 일반적인 패턴을 보여 줍니다. 메서드의 데이터는 `simpleworkflow.model` 네임스페이스의 클래스에 의해 제공되며, 체인 연결 가능한 `0with*` 메서드를 사용하여 인스턴스화하고 채울 수 있습니다.

5. 함수를 추가하여 새 활동 유형을 등록합니다. 활동은 워크플로의 작업 단위를 나타냅니다.

```
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!");
    }
```

활동 유형은 이름과 버전으로 식별되는데, 이러한 항목은 해당 활동을 등록된 도메인 내 다른 활동으로부터 고유하게 식별하는 데 사용됩니다. 활동에는 활동 실행의 다양한 부분을 수행하는 데 걸리는 시간에 대한 제약 조건을 설정할 때 사용할 수 있는 다양한 제한 시간이나 SWF에서 작업 및 데이터를 받는 데 사용되는 기본 작업 목록과 같은 선택적 파라미터가 포함됩니다. 자세한 내용은 RegisterActivityTypeRequest를 참조하십시오.

> **Note**
>
> 모든 제한 시간 값은 초로 지정합니다. 제한 시간이 워크플로 실행에 어떤 영향을 미치는지에 대한 전체 설명은 Amazon SWF제한 시간 유형을 참조하세요.

등록하려고 하는 활동 유형이 이미 있는 경우 TypeAlreadyExistsException이 발생합니다. 함수를 추가하여 새 워크플로 유형을 등록합니다. 결정자라고도 하는 워크플로는 워크플로 실행 논리를 나타냅니다.

+

```
    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!");
    }
```

+

활동 유형과 마찬가지로, 워크플로 유형은 이름 및 버전에 의해 식별되며 구성 가능한 제한 시간도 지정됩니다. 자세한 내용은 RegisterWorkflowTypeRequest를 참조하십시오.

+

등록하려고 하는 워크플로 유형이 이미 있는 경우 TypeAlreadyExistsException이 발생합니다. 마지막으로 `main` 메서드를 제공하여 클래스를 실행 가능하도록 설정합니다. 이렇게 하면 도메인, 활동 유형 및 워크플로 유형이 차례로 등록됩니다.

+

```
    registerDomain();
    registerWorkflowType();
    registerActivityType();
```

이제 애플리케이션을 빌드 및 실행하여 등록 스크립트를 실행하거나 활동 및 워크플로 작업자 코딩을 계속 진행할 수 있습니다. 도메인, 워크플로 및 활동을 등록한 후에는 다시 실행할 필요가 없습니다. 이러한 유형은 직접 사용을 중단할 때까지 유지됩니다.

활동 작업자 구현

활동은 워크플로의 기본 작업 단위를 입니다. 워크플로는 논리를 제공하며 결정 작업에 대한 응답으로 실행할 활동(또는 수행할 다른 작업)을 예약합니다. 일반적인 워크플로는 대체로 동기적, 비동기적 또는 동기/비동기의 결합 형태로 실행할 수 있는 하나 이상의 활동으로 구성됩니다.

활동 작업자는 워크플로 결정에 대한 응답으로 Amazon SWF에서 생성하는 활동 작업을 폴링하는 간략한 코드입니다. 활동 작업자는 활동 작업을 수신하면 해당 활동을 실행하고 성공/실패 응답을 다시 해당 워크플로로 반환합니다.

단일 활동을 유도하는 단순 활동 작업자를 구현해 보겠습니다.

1. 텍스트 편집기를 열고 `ActivityWorker.java` 파일을 생성하여 공통 단계에 따라 패키지 선언과 가져오기를 추가합니다.

   ```
   import com.amazonaws.regions.Regions;
   import com.amazonaws.services.simpleworkflow.AmazonSimpleWorkflow;
   import com.amazonaws.services.simpleworkflow.AmazonSimpleWorkflowClientBuilder;
   import com.amazonaws.services.simpleworkflow.model.*;
   ```

2. ActivityWorker 클래스를 파일에 추가하고 Amazon SWF와 상호 작용하는 데 사용할 SWF 클라이언트를 포함하는 데이터 멤버를 제공합니다.

```
    private static final AmazonSimpleWorkflow swf =

 AmazonSimpleWorkflowClientBuilder.standard().withRegion(Regions.DEFAULT_REGION).build();
```

3. 활동으로 사용할 메서드를 추가합니다.

```
private static String sayHello(String input) throws Throwable {
    return "Hello, " + input + "!";
}
```

이 활동은 단순히 문자열을 받아서 인사말로 결합하고 결과를 반환합니다. 이 활동은 예외가 발생할 가능성이 매우 희박하지만, 잘못되었을 경우 오류를 일으킬 수 있는 활동을 설계하는 것이 바람직합니다.

4. 활동 작업 폴링 메서드로 사용할 main 메서드를 추가합니다. 활동 작업의 작업 목록을 폴링하기 위한 몇 가지 코드를 추가하는 것부터 시작하겠습니다.

```
        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();
```

활동은 SWF 클라이언트의 pollForActivityTask 메서드를 호출하고 전달된 PollForActivityTaskRequest에서 사용할 도메인과 작업 목록을 지정하여 Amazon SWF에서 작업을 받습니다.

작업이 수신되고 나면 작업의 getTaskToken 메서드를 호출하여 작업의 고유 식별자를 가져옵니다.

5. 다음에는 들어오는 작업을 처리하는 코드를 작성합니다. 작업을 폴링하고 관련 작업 토큰을 가져오는 코드 바로 뒤에 다음을 main 메서드에 추가합니다.

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

작업 토큰이 null이 아니면 작업과 함께 전송된 입력 데이터를 지정하여 활동 메서드(sayHello)의 실행을 시작할 수 있습니다.

작업이 성공하면(오류가 생성되지 않은 경우) 작업자는 작업 토큰과 활동의 결과 데이터가 포함된 RespondActivityTaskCompletedRequest 객체를 사용하여 SWF 클라이언트의 respondActivityTaskCompleted 메서드를 호출하여 SWF에 응답합니다.

반면, 작업이 실패하면 RespondActivityTaskFailedRequest 객체를 사용하여 respondActivityTaskFailed 메서드를 호출하고 작업 토큰과 오류에 대한 정보를 전달하여 응답합니다.

> **Note**
>
> 이 활동은 중지될 경우 정상적으로 종료되지 않습니다. 이 자습서의 범위에는 포함되지 않지만, 이 활동 작업자의 대체 구현이 함께 제공되는 항목인 활동 및 워크플로 작업자를 정상적으로 종료하기 단원에 설명되어 있습니다.

워크플로 작업자 구현

워크플로 논리는 워크플로 작업자라고 하는 코드 부분에 상주합니다. 워크플로 작업자는 워크플로 유형이 등록된 기본 작업 목록과 도메인에서 Amazon SWF에서 전송하는 결정 작업을 폴링합니다.

워크플로 작업자는 작업을 수신할 경우 결정을 내리고(일반적으로 새 활동을 예약할지 여부) 적절한 작업(예: 활동 예약)을 수행합니다.

1. 텍스트 편집기를 열고 `WorkflowWorker.java` 파일을 생성하여 공통 단계에 따라 패키지 선언과 가져오기를 추가합니다.
2. 파일에 몇 가지 추가적인 가져오기를 추가합니다.

```
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. `WorkflowWorker` 클래스를 선언하고 SWF 메서드에 액세스하는 데 사용되는 AmazonSimpleWorkflowClient 클래스의 인스턴스를 생성합니다.

```
    private static final AmazonSimpleWorkflow swf =

AmazonSimpleWorkflowClientBuilder.standard().withRegion(Regions.DEFAULT_REGION).build();
```

4. `main` 메서드를 추가합니다. 이 메서드는 SWF 클라이언트의 `pollForDecisionTask` 메서드를 사용하여 결정 작업을 폴링하며 계속 반복됩니다. PollForDecisionTaskRequest에서 세부 정보를 제공합니다.

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

작업이 수신되고 나면 `getTaskToken` 메서드를 호출합니다. 이 메서드는 작업을 식별하는 데 사용할 수 있는 문자열을 반환합니다. 반환된 토큰이 아닌 `null` 경우 `executeDecisionTask` 메서드에서 추가로 처리하여 작업 토큰과, 작업과 함께 전송된 HistoryEvent 객체 목록을 전달합니다.

5. 작업 토큰(`String`)과 `HistoryEvent` 목록을 사용하는 `executeDecisionTask` 메서드를 추가합니다.

```
    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;
```

또한 다음과 같은 항목을 추적하기 위해 데이터 멤버를 설정할 수도 있습니다.

- 작업 처리 결과를 보고하는 데 사용되는 Decision 객체 목록
- "WorkflowExecutionStarted" 이벤트에서 제공하는 워크플로 입력을 포함할 문자열
- 이미 예약되었거나 현재 실행 중인 경우 동일 활동의 예약을 방지하기 위한 예약 및 열린(실행 중인) 활동 수
- 활동이 완료되었음을 나타내는 부울 값

- 워크플로 결과로서 반환하기 위한, 활동 결과를 포함할 문자열

6. 다음에는 executeDecisionTask 메서드에 의해 보고되는 이벤트 유형에 근거하여 작업과 함께 전송된 HistoryEvent 객체를 처리할 코드를 getEventType에 추가합니다.

```
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("]");
```

이 워크플로의 목적상, 우리에게 중요한 부분은 다음과 같습니다.

- "WorkflowExecutionStarted" 이벤트 - 워크플로 실행이 시작되었음(일반적으로 워크플로에서 첫 번째 활동을 실행해야 함을 의미함)을 나타내고 워크플로에 제공되는 초기 입력을 제공합니다. 이 경우, 인사말의 이름 부분이므로 실행할 활동을 예약할 때 사용할 문자열에 저장됩니다.
- "ActivityTaskCompleted" 이벤트 - 예약된 활동이 완료되고 나서 전송됩니다. 이벤트 데이터에는 완료된 활동의 반환 값도 포함됩니다. 활동이 하나뿐이므로 해당 값을 전체 워크플로의 결과로 사용하겠습니다.

그 밖의 이벤트 유형은 워크플로에 필요할 경우 사용할 수 있습니다. 각 이벤트 유형에 대한 자세한 내용은 HistoryEvent 클래스 설명을 참조하십시오.

+ 참조: `switch` 문의 문자열은 Java 7에서 도입되었습니다. Java의 이전 버전을 사용 중인 경우 EventType 클래스를 사용하여 `history_event.getType()`에 의해 반환되는 `String`을 열거 값으로 변환했다가 필요에 따라 다시 `String`으로 변환할 수 있습니다.

```
EventType et = EventType.fromValue(event.getEventType());
```

1. `switch` 문 뒤에, 수신된 작업에 따라 해당하는 결정으로 응답하는 코드를 추가합니다.

```
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);
```

- 활동이 아직 예정되지 않은 경우 ScheduleActivityTask 결정을 내려 응답합니다. 이 결정은 Amazon SWF에서 다음 일정을 잡아야 하는 활동에 대한 ScheduleActivityTaskDecisionAttributes 구조로 된 정보와 Amazon SWF가 활동으로 전송해야 하는 모든 데이터를 포함합니다.
- 활동이 완료된 경우 전체 워크플로가 완료된 것으로 간주하고 CompleteWorkExecutionDecisionAttributes 구조를 작성하여 완료된 워크플로에 대한 세부 정보를 제공하여 CompletedWorkflowExecution 의사 결정에 응답합니다. 이 경우 활동의 결과를 반환합니다.

어느 경우든, 결정 정보는 메서드 맨 위에 선언된 Decision 목록에 추가됩니다.

2. 작업을 처리하면서 수집된 Decision 객체의 목록을 반환하여 결정 작업을 완료합니다. 작성 중이었던 executeDecisionTask 메서드의 끝에 이 코드를 추가합니다.

```
swf.respondDecisionTaskCompleted(
    new RespondDecisionTaskCompletedRequest()
        .withTaskToken(taskToken)
        .withDecisions(decisions));
```

SWF 클라이언트의 respondDecisionTaskCompleted 메서드는 작업을 식별하는 작업 토큰과 Decision 객체의 목록을 사용합니다.

워크플로 시작자 구현

마지막으로 워크플로 실행을 시작하는 몇 가지 코드를 작성해 보겠습니다.

1. 텍스트 편집기를 열고 WorkflowStarter.java 파일을 생성하여 공통 단계에 따라 패키지 선언과 가져오기를 추가합니다.
2. WorkflowStarter 클래스를 추가합니다.

```
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() + "'.");
    }
}
```

`WorkflowStarter` 클래스는 단일 메서드인 `main`으로 이루어지는데, 이 메서드는 워크플로의 입력 데이터로서 명령줄을 통해 전달되는 인수(선택 사항)를 사용합니다.

SWF 클라이언트 메서드인 `startWorkflowExecution`은 StartWorkflowExecutionRequest 객체를 입력으로 사용합니다. 여기서는 실행할 워크플로 유형과 도메인을 지정하는 것 외에, 다음을 제공합니다.

- 사람이 읽을 수 있는 워크플로 실행 이름
- 워크플로 입력 데이터(여기 예제에서는 명령줄을 통해 제공됨)
- 전체 워크플로를 실행하는 데 걸릴 시간(초)을 나타내는 제한 시간 값

`startWorkflowExecution`가 반환하는 Run 객체는 워크플로 실행에 대한 Amazon SWF의 기록에서 해당 특정 워크플로 실행을 식별하는 데 사용할 수 있는 값인 실행 ID를 제공합니다.

+ 참조: 실행 ID는 Amazon SWF에서 생성하며 워크플로 실행 시작 시 전달하는 워크플로 실행 이름과 동일하지 않습니다.

예제 빌드

Maven을 사용하여 예제 프로젝트를 빌드하려면 `helloswf` 디렉터리 및 유형으로 이동합니다.

```
mvn package
```

결과적으로 `helloswf-1.0.jar`가 `target` 디렉터리에 생성됩니다.

예제 실행

이 예제는 서로 독립적으로 실행되는 네 가지 개별 실행 가능 클래스로 구성되어 있습니다.

> **Note**
>
> Linux, macOS 또는 Unix 시스템을 사용 중인 경우 단일 터미널 창에서 이들 클래스 모드를 차례로 하나씩 실행할 수 있습니다. Windows를 실행 중이면 추가 명령줄 인스턴스 두 개를 열어서 각각 `helloswf` 디렉터리로 이동해야 합니다.

Java classpath 설정

Maven에서 개발자를 대신하여 종속성을 처리했더라도 이 예제를 실행하려면 AWS SDK 라이브러리와 Java 클래스 경로에 대한 관련 종속성을 제공해야 합니다. `CLASSPATH` 환경 변수를 AWS SDK 라이브러리와 `third-party/lib` 디렉터리의 위치(필요 종속성이 포함됨)로 설정할 수 있습니다.

```
export CLASSPATH='target/helloswf-1.0.jar:/path/to/sdk/lib/*:/path/to/sdk/third-party/
lib/*'
java example.swf.hello.HelloTypes
```

또는 **java** 명령의 -cp 옵션을 사용하여 각 애플리케이션을 실행하는 동안 classpath를 설정할 수 있습니다.

```
java -cp target/helloswf-1.0.jar:/path/to/sdk/lib/*:/path/to/sdk/third-party/lib/* \
  example.swf.hello.HelloTypes
```

사용할 스타일은 개발자 재량에 따라 선택할 수 있습니다. 코드 빌드 시에는 문제가 발생하지 않았지만 예제를 실행할 때 일련의 "NoClassDefFound" 오류가 발생하는 경우 이는 classpath가 올바르게 설정되지 않았기 때문일 가능성이 높습니다.

도메인, 워크플로 및 활동 유형 등록

작업자 및 워크플로 시작자를 실행하기 전에 도메인 및 워크플로와 활동 유형을 등록해야 합니다. 이 작업을 수행하는 코드가 도메인, 워크플로 및 활동 유형 등록에 구현되었습니다.

빌드 후 CLASSPATH를 설정한 경우, 다음 명령을 실행하여 등록 코드를 실행할 수 있습니다.

```
    echo 'Supply the name of one of the example classes as an argument.'
```

활동 및 워크플로 작업자 시작

이제 유형이 등록되었으며, 활동 및 워크플로 작업자를 시작할 수 있습니다. 이들 작업자는 종료될 때까지 계속 실행되며 작업을 폴링하므로, 별도의 터미널 창에서 작업자를 실행합니다. 또는 Linux, macOS나 Unix 에서 실행 중인 경우에는 & 연산자를 사용하여 각 작업자가 실행 시 개별 프로세스를 생성하도록 할 수 있습니다.

```
    echo 'If there are arguments to the class, put them in quotes after the class
name.'
    exit 1
```

개별 창에서 이러한 명령을 실행 중인 경우에는 각 줄에서 마지막 & 연산자를 생략합니다.

워크플로 실행 시작

이제 활동 및 워크플로 작업자가 폴링 중이며, 워크플로 실행을 시작할 수 있습니다. 이 프로세스는 워크플로가 완료 상태를 반환할 때까지 실행할 수 있습니다. (& 연산자를 사용하여 작업자를 새로 생성된 프로세스로 실행한 경우가 아니면) 워크플로를 새 터미널 창에서 실행해야 합니다.

```
fi
```

> **Note**
>
> 고유의 입력 데이터를 제공하려면 입력 데이터를 명령줄에 추가합니다. 이 데이터는 워크플로에 먼저 전달되고 나서 활동에 전달됩니다. 예:
>
> ```
> echo "## Running $className..."
> ```

워크플로 실행을 시작하고 나면 작업자와 워크플로 실행 자체에서 전달되는 출력 보기를 시작해야 합니다. 워크플로가 최종적으로 완료되면 출력 내용이 화면으로 출력됩니다.

이 예제의 전체 소스

Github의 aws-java-developer-guide 리포지토리에서 이 예제의 전체 소스를 찾아볼 수 있습니다.

자세한 정보

- 여기에 제시된 작업자는 워크플로 폴이 여전히 진행 중인 동안 종료되는 경우 작업이 손실될 수 있습니다. 작업자를 정상적으로 종료하는 방법을 알아보려면 Shutting Down Activity and Workflow Workers Gracefully 단원을 참조하십시오.
- Amazon SWF에 대해 자세히 알아보려면 Amazon SWF 홈 페이지 또는 Amazon SWF 개발자 안내서를 참조하세요.
- Java용 AWS Flow Framework를 사용하면 주석을 사용하여 멋진 Java 스타일의 고급 워크플로를 작성할 수 있습니다. 자세히 알아보려면 Java용 AWS Flow Framework 개발자 안내서를 참조하세요.

Lambda 작업

Amazon SWF 활동의 대안으로 또는 이러한 활동과 함께, Lambda 함수를 사용하여 워크플로의 작업 단위를 표현하고 활동처럼 예약할 수 있습니다.

이 항목에서는 AWS SDK for Java를 사용하여 Amazon SWF Lambda 작업을 구현하는 방법을 중점적으로 설명합니다. Lambda 작업에 대한 자세한 내용은 Amazon SWF 개발자 안내서의 AWS Lambda 작업을 참조하세요.

Lambda 함수를 실행하도록 교차 서비스 IAM 역할 설정

Amazon SWF에서 Lambda 함수를 실행하려면 먼저 사용자를 대신하여 Lambda 함수를 실행할 권한을 Amazon SWF에 부여하도록 IAM 역할을 설정해야 합니다. 이와 같이 설정하는 방법에 대한 자세한 내용은 AWS Lambda 작업 단원을 참조하세요.

Lambda 작업을 사용할 워크플로를 등록할 때 이 IAM 역할의 Amazon 리소스 이름(ARN)이 필요합니다.

Lambda 함수 생성

Java를 비롯하여 다양한 언어로 Lambda 함수를 작성할 수 있습니다. Lambda 함수 작성, 배포 및 사용 방법에 대한 자세한 내용은 AWS Lambda 개발자 안내서를 참조하세요.

> **Note**
>
> Lambda 함수를 작성하는 데 사용하는 언어와 무관합니다. 즉, 워크플로 코드의 작성 언어와 관계없이 모든 Amazon SWF 워크플로에서 예약하고 실행할 수 있습니다. Amazon SWF는 함수 실행 및 함수와 주고받는 데이터 전달에 대한 세부 사항을 처리합니다.

다음은 간단한 Amazon SWF 애플리케이션 구축 작업 대신 사용할 수 있는 간단한 Lambda 함수입니다.

- 이 버전은 JavaScript로 작성되었으며, AWS Management Console을 사용하여 직접 입력할 수 있습니다.

```
exports.handler = function(event, context) {
    context.succeed("Hello, " + event.who + "!");
};
```

- 다음은 Java로 작성된 동일 함수로, Lambda에서 배포 및 실행할 수도 있습니다.

```
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**
>
> Lambda에 Java 함수를 배포하는 방법에 대해 자세히 알아보려면 AWS Lambda 개발자 안내서의 배포 패키지 생성(Java)을 참조하세요. 또한 Java로 Lambda 함수를 작성하기 위한 프로그래밍 모델 단원을 살펴볼 수도 있습니다.

Lambda 함수는 event 또는 input 객체를 첫 번째 매개 변수로 사용하고, context 객체를 두 번째 매개 변수로 사용하여 Lambda 함수 실행 요청에 대한 정보를 제공합니다. 이 특수한 함수에서는 입력이 JSON 형식이어야 하며, who 필드가 인사말을 생성하는 데 사용된 이름으로 설정되어야 합니다.

Lambda에 사용할 워크플로 등록

워크플로에서 Lambda 함수를 예약하려면 Lambda 함수 호출 권한을 가진 권한을 Amazon SWF에 제공하는 IAM 역할의 이름을 지정해야 합니다. RegisterWorkflowTypeRequest의 `withDefaultLambdaRole` 또는 `setDefaultLambdaRole` 메서드를 사용하여 워크플로를 등록하는 동안 이를 설정할 수 있습니다.

```
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) {
```

Lambda 작업 예약

Lambda 작업 예약은 활동 예약과 비슷합니다. '스케줄 Lambda 함수' `ScheduleLambdaFunction`DecisionType 및 ScheduleLambdaFunctionDecisionAttributes와 함께 결정을 제공합니다.

```
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(
```

`ScheduleLambdaFuntionDecisionAttributes`에, 이름(호출할 Lambda 함수의 ARN)과, id(Amazon SWF가 기록 로그에서 Lambda 함수를 식별하는 데 사용할 이름)를 지정해야 합니다.

또한 Lambda 함수에 선택적 입력을 제공하고 시작에서 종료까지 제한 시간 값을 설정할 수 있습니다. 이 값은 `LambdaFunctionTimedOut` 이벤트를 생성하기 전에 Lambda 함수를 실행할 수 있는 시간(초)입니다.

> **Note**
>
> 이 코드는 함수 이름이 지정된 경우 AWSLambdaClient를 사용하여 Lambda 함수의 ARN을 가져옵니다. 이 기법을 사용하여 코드에서 전체 ARN(AWS 계정 ID 포함)가 하드코딩되는 것을 방지할 수 있습니다.

결정자에서 Lambda 함수 이벤트 처리

Lambda 작업은 워크플로 워커의 의사 결정 작업을 폴링할 때 Lambda 작업 수명 주기에 따라 조치를 취할 수 있는 여러 이벤트를 생성합니다. 이 이벤트는 EventType 값(예: `LambdaFunctionScheduled`, `LambdaFunctionStarted` 및 `LambdaFunctionCompleted`)으로 지정됩니다. Lambda 함수가 실패하거나 설정된 제한 시간 값보다 실행하는 데 더 오래 걸리면 각각 `LambdaFunctionFailed` 또는 `LambdaFunctionTimedOut` 이벤트 유형이 수신됩니다.

```
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;
```

Lambda 함수로부터 출력 수신

HistoryEvent에서 LambdaFunctionCompleted`EventType, you can retrieve your 0 function's return value by first calling `getLambdaFunctionCompletedEventAttributes를 수신하여 LambdaFunctionCompletedEventAttributes 객체를 가져온 다음 해당 getResult 메서드를 호출하여 Lambda 함수의 출력을 검색하는 경우:

```
 LambdaFunctionCompleted:
running_functions--;
```

이 예제의 전체 소스

Github의 aws-java-developer-guide 리포지토리에서 이 예제의 전체 소스 :github:`<awsdocs/aws-java-developer-guide/tree/master/doc_source/snippets/helloswf_lambda/>를 찾아볼 수 있습니다.

활동 및 워크플로 작업자 정상 종료

간단한 Amazon SWF 애플리케이션 구축 주제에서는 등록 애플리케이션, 활동과 워크플로 작업자, 워크플로 시작자로 이루어진 단순 워크플로 애플리케이션의 전체 구현을 제공했습니다.

작업자 클래스는 지속적으로 실행하며 Amazon SWF에서 전송된 작업을 폴링하여 활동을 실행하거나 결정을 반환하도록 설계되었습니다. 폴링 요청이 생성되면 Amazon SWF는 폴러를 기록하고 이 폴러에 작업을 할당하려고 시도합니다.

워크플로 작업자가 긴 폴링 기간 동안 종료되는 경우 Amazon SWF가 여전히 작업을 종료된 작업자로 전송하려고 시도하여 (작업 제한 시간이 경과될 때까지) 작업 손실이 발생할 수도 있습니다.

이 상황을 처리하는 한 가지 방법은 작업자가 종료되기 전에 모든 긴 폴링 요청이 반환될 때까지 기다리는 것입니다.

이 주제에서는 Java의 종료 후크를 사용하여 활동 작업자의 정상 종료를 시도하도록 helloswf의 활동 작업자를 다시 작성하겠습니다.

전체 코드는 다음과 같습니다.

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

이 버전에서는 원래 버전의 `main` 함수에 있었던 폴링 코드가 고유 메서드인 `pollAndExecute`로 이동되었습니다.

이제 `main` 함수는 종료 후크와 함께 CountDownLatch를 사용하여 스레드를 종료하기 전에 스레드가 종료 요청 후 최대 60초 동안 대기하도록 합니다.

여러 도메인 등록하기

Amazon SWF의 모든 워크플로 및 활동에는 실행할 도메인이 필요합니다.

1. 새 RegisterDomainRequest 객체를 생성하고, 하나 이상의 도메인 이름 및 워크플로 실행 보존 기간을 지정합니다. 이러한 파라미터는 둘 다 필수 파라미터입니다.
2. RegisterDomainRequest 객체를 사용하여 AmazonSimpleWorkflowClient.registerDomain 메서드를 호출합니다.
3. 요청하는 도메인이 이미 존재하는 경우 DomainAlreadyExistsException을 확인합니다(이 경우, 일반적으로 아무런 조치가 필요하지 않음).

다음 코드에서는 이 절차를 보여줍니다.

```
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!");
    }
}
```

도메인 나열

등록 유형별로 계정과 연결된 Amazon SWF 도메인과 AWS 리전을 나열할 수 있습니다.

1. ListDomainsRequest 객체를 만들고 관심 있는 도메인의 등록 상태를 지정하세요. 이는 필수입니다.
2. ListDomainRequest 객체를 사용하여 AmazonSimpleWorkflowClient.listDomains를 호출합니다. 결과는 DomainInfos 객체에 제공됩니다.
3. 반환된 객체에 대해 getDomainInfos를 호출하여 DomainInfo 객체의 목록을 가져옵니다.
4. 각 DomainInfo 객체에서 getName을 호출하여 이름을 가져옵니다.

다음 코드에서는 이 절차를 보여줍니다.

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

SDK에 포함된 코드 샘플

AWS SDK for Java는 SDK의 여러 기능을 보여주는 빌드 및 실행 가능한 프로그램 형태의 코드 샘플과 함께 패키징되어 제공됩니다. 이를 검토하거나 수정하여 AWS SDK for Java를 사용하여 자신의 AWS 솔루션을 구현할 수 있습니다.

샘플을 가져오는 방법

AWS SDK for Java 코드 샘플은 SDK의 samples 디렉터리에 제공됩니다. AWS SDK for Java 설정의 정보를 사용하여 SDK를 다운로드하여 설치한 경우 샘플이 이미 시스템에 있습니다.

또한 src/samples 디렉토리의 AWS SDK for Java GitHub 리포지토리에서 최신 샘플을 볼 수 있습니다.

명령줄을 사용하여 샘플 빌드 및 실행

이러한 샘플에는 Ant 빌드 스크립트가 포함되므로 명령줄에서 이러한 샘플을 쉽게 빌드 및 실행할 수 있습니다. 각 샘플에는 각 샘플 관련 정보를 포함하는 HTML 형식의 README 파일도 들어 있습니다.

> **Note**
>
> GitHub에서 샘플 코드를 찾아보려면 샘플의 README.html 파일을 보면서 소스 코드 디스플레이에서 원시 버튼을 클릭합니다. 원시 모드에서는 HTML이 브라우저에 의도한 대로 렌더링됩니다.

필수 조건

AWS SDK for Java 샘플을 실행하기 전에 개발을 위한 AWS 자격 증명 및 리전 설정에 지정된 대로 환경에서 또는 AWS CLI를 사용하여 AWS 자격 증명을 설정해야 합니다. 샘플에는 가능하면 언제든 기본 자격 증명 공급자 체인이 사용됩니다. 따라서 이러한 방식으로 자격 증명을 설정하면 소스 코드 디렉터리에 있는 파일에 AWS 자격 증명을 삽입하는(이 경우 잘못 체크인되어 공개적으로 공유될 수도 있음) 위험한 관행을 방지할 수 있습니다.

샘플 실행

1. 샘플 코드를 포함하는 디렉터리로 변경합니다. 예를 들어 AWS SDK 다운로드의 루트 디렉터리에 있으며 `AwsConsoleApp` 샘플을 실행하려는 경우 다음을 입력합니다.

   ```
   cd samples/AwsConsoleApp
   ```

2. Ant를 사용하여 샘플을 빌드하고 실행합니다. 기본 빌드 대상에서 두 작업이 모두 수행되므로 다음만 입력하면 됩니다.

   ```
   ant
   ```

샘플은 정보를 표준 출력으로 인쇄합니다. 예를 들면 다음과 같습니다.

```
===========================================

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.
```

Eclipse IDE를 사용하여 샘플 빌드 및 실행

AWS Toolkit for Eclipse를 사용하는 경우 AWS SDK for Java를 기반으로 Eclipse에서 새 프로젝트를 시작하거나 기존 Java 프로젝트에 SDK를 추가할 수 있습니다.

필수 조건

AWS Toolkit for Eclipse를 설치한 후에는 보안 자격 증명을 사용하여 이 도구 키트를 구성하는 것이 좋습니다. Eclipse의 창 메뉴에서 기본 설정을 선택한 다음 AWS 툴킷 섹션을 선택하여 언제든지 이 작업을 수행할 수 있습니다.

샘플 실행

1. Eclipse를 엽니다.
2. 새 AWS Java 프로젝트를 생성합니다. Eclipse의 [File] 메뉴에서 [New]를 선택한 다음, [Project]를 클릭합니다. [New Project] 마법사가 열립니다.
3. AWS 범주를 확장하고 나서 AWS Java 프로젝트를 선택합니다.
4. 다음(Next)을 선택합니다. 프로젝트 설정 페이지가 표시됩니다.
5. [Project Name] 상자에 이름을 입력합니다. 앞에서 설명한 바와 같이 SDK에서 사용할 수 있는 AWS SDK for Java 샘플이 샘플 그룹에 표시됩니다.
6. 각 확인란을 선택하여 프로젝트에 포함할 샘플을 선택합니다.
7. AWS 보안 인증을 입력합니다. 자격 증명을 사용하여 AWS Toolkit for Eclipse를 이미 구성한 경우 해당 정보가 자동으로 채워집니다.
8. [마침]을 클릭합니다. 프로젝트가 생성되어 [Project Explorer]에 추가됩니다.
9. 실행할 샘플 `.java` 파일을 선택합니다. 예를 들어 Amazon S3 샘플의 경우 `S3Sample.java`를 선택합니다.
10. [Run] 메뉴에서 [Run]을 선택합니다.

11. [Project Explorer]에서 프로젝트를 마우스 오른쪽 버튼으로 클릭하고 [Build Path]를 가리킨 다음 [Add Libraries]를 선택합니다.
12. AWS Java SDK를 선택하고 다음을 선택한 다음, 화면의 나머지 지침을 따릅니다.

보안을 위한 AWS SDK for Java

Amazon Web Services(AWS)에서 가장 우선순위가 높은 것이 클라우드 보안입니다. AWS 고객으로서 여러분은 가장 높은 보안 요구 사항을 충족하기 위해 설계된 데이터 센터 및 네트워크 아키텍처의 혜택을 받게 됩니다. 보안은 사용자와 사용자 간의 공동 책임입니다. AWS 공동 책임 모델은 이 사항을 클라우드 내 보안 및 클라우드의 보안으로 설명합니다.

클라우드 보안 — AWS 클라우드에서 제공되는 모든 서비스를 실행하는 인프라를 보호하고 안전하게 사용할 수 있는 서비스를 제공하는 역할을 합니다. AWS 당사의 보안 책임은 최우선 과제이며 AWS, AWS 규정 준수 프로그램의 일환으로 타사 감사자가 보안 효과를 정기적으로 테스트하고 검증합니다.

클라우드에서의 보안 — 사용자의 책임은 사용 중인 AWS 서비스와 데이터의 민감도, 조직의 요구 사항, 관련 법률 및 규정을 비롯한 기타 요인에 따라 결정됩니다.

이 AWS 제품 또는 서비스는 지원하는 특정 Amazon Web Services (AWS) 서비스를 통해 공동 책임 모델을 따릅니다. AWS 서비스 보안 정보는 AWS 서비스 보안 설명서 페이지 및 AWS 규정 준수 프로그램의 규정 준수 노력 범위에 속하는AWS 서비스를 참조하십시오.

주제

AWS SDK for Java 1.x의 데이터 보호

공동 책임 모델은 이 AWS 제품 또는 서비스의 데이터 보호에 적용됩니다. 이 모델에 설명된 대로 AWS는 모든 AWS 클라우드를 실행하는 글로벌 인프라를 보호하는 역할을 합니다. 이 인프라에서 호스팅되는 콘텐츠에 대한 통제를 유지하는 것은 사용자의 책임입니다. 이 콘텐츠에는 사용하는 AWS 서비스에 대한 보안 구성 및 관리 작업이 포함됩니다. 데이터 프라이버시에 대한 자세한 내용은 데이터 프라이버시 FAQ를 참조하세요. 유럽의 데이터 보호에 대한 자세한 내용은 AWS 보안 블로그의 AWS 공동 책임 모델 및 GDPR 블로그 게시물을 참조하십시오.

데이터 보호를 위해 AWS Identity and Access Management (IAM) 를 AWS 계정 사용하여 자격 증명을 보호하고 개별 사용자 계정을 설정하는 것이 좋습니다. 이러한 방식에서는 각 사용자에게 자신의 직무를 충실히 이행하는 데 필요한 권한만 부여됩니다. 또한 다음과 같은 방법으로 데이터를 보호하는 것이 좋습니다.

- 각 계정에 멀티 팩터 인증 설정(MFA)을 사용하세요.
- SSL/TLS를 사용하여 리소스와 통신하세요. AWS
- 를 사용하여 API 및 사용자 활동 로깅을 설정합니다. AWS CloudTrail
- AWS 서비스 내 모든 기본 보안 제어가 포함된 AWS 암호화 솔루션을 사용하십시오.
- Amazon S3에 저장된 개인 데이터를 검색하고 보호하는 데 도움이 되는 Amazon Macie와 같은 고급 관리형 보안 서비스를 사용합니다.
- 명령줄 인터페이스 또는 API를 AWS 통해 액세스할 때 FIPS 140-2로 검증된 암호화 모듈이 필요한 경우 FIPS 엔드포인트를 사용하십시오. 사용 가능한 FIPS 엔드포인트에 대한 자세한 내용은 Federal Information Processing Standard(FIPS) 140-2 섹션을 참조하세요.

명칭 필드와 같은 자유 형식 필드에 고객 계정 번호와 같은 중요 식별 정보를 절대 입력하지 마세요. 여기에는 콘솔, API 또는 SDK를 사용하여 이 AWS 제품이나 서비스 또는 기타 AWS 서비스를 사용하는 경우가 포함됩니다. AWS CLI AWS 이 AWS 제품이나 서비스 또는 기타 서비스에 입력하는 모든 데이터는 진단 로그에 포함되도록 선택될 수 있습니다. 외부 서버에 URL을 제공할 때 해당 서버에 대한 요청을 검증하기 위해 자격 증명 정보를 URL에 포함하지 마십시오.

AWS SDK for Java TLS 지원

다음 정보는 Java SSL 구현 (의 기본 SSL 구현 AWS SDK for Java) 에만 적용됩니다. 다른 SSL 구현을 사용하는 경우 해당 SSL 구현을 참조하여 TLS 버전을 적용하는 방법을 알아보십시오.

TLS 버전을 확인하는 방법

플랫폼에서 지원되는 TLS 버전을 확인하려면 Java 가상 머신(JVM) 공급자의 설명서를 참조하세요. 일부 JVM의 경우 다음 코드는 지원되는 SSL 버전을 인쇄합니다.

```
System.out.println(Arrays.toString(SSLContext.getDefault().getSupportedSSLParameters().getProto
```

작동 중인 SSL 핸드셰이크와 사용된 TLS 버전을 보려면 시스템 속성 javax.net.debug를 사용하면 됩니다.

```
java app.jar -Djavax.net.debug=ssl
```

> **Note**
>
> TLS 1.3은 Java 버전 1.9.5~1.10.31용 SDK와 호환되지 않습니다. 자세한 내용은 다음 블로그 게시물을 참조하세요.
> https://aws.amazon.com/blogs/developer/tls-1-3 - incompatibility-with-aws-sdk - for-java-versions -1-9-5-to-1-10-31/

최소 TLS 버전 적용

SDK는 항상 플랫폼 및 서비스에서 지원하는 최신 TLS 버전을 선호합니다. 특정 최소 TLS 버전을 적용하려면 JVM 설명서를 참조하세요. OpenJDK 기반 JVM의 경우 시스템 속성 `jdk.tls.client.protocols`을 사용할 수 있습니다.

```
java app.jar -Djdk.tls.client.protocols=PROTOCOLS
```

지원되는 PROTOCOLS 값은 JVM 설명서를 참조하세요.

ID 및 액세스 관리

AWS Identity and Access Management (IAM) 은 관리자가 리소스에 대한 액세스를 안전하게 제어할 수 AWS 서비스 있도록 AWS 도와줍니다. IAM 관리자는 리소스를 사용할 수 있는 인증 (로그인) 및 권한 부여 (권한 보유) 를 받을 수 있는 사용자를 제어합니다. AWS IAM은 추가 AWS 서비스 비용 없이 사용할 수 있습니다.

주제

- 고객
- ID를 통한 인증
- 정책을 사용한 액세스 관리
- AWS 서비스 IAM을 사용하는 방법
- AWS ID 및 액세스 문제 해결

고객

사용하는 방식 AWS Identity and Access Management (IAM) 은 수행하는 작업에 따라 다릅니다. AWS

서비스 사용자 - 작업을 수행하는 AWS 서비스 데 사용하는 경우 관리자가 필요한 자격 증명과 권한을 제공합니다. 더 많은 AWS 기능을 사용하여 작업을 수행함에 따라 추가 권한이 필요할 수 있습니다. 액세스 권한 관리 방식을 이해하면 적절한 권한을 관리자에게 요청할 수 있습니다. 에서 AWS기능에 액세스할 수 없는 경우 사용 중인 기능의 사용 설명서를 참조하십시오AWS ID 및 액세스 문제 해결. AWS 서비스

서비스 관리자 — 회사에서 AWS 리소스를 담당하는 경우 전체 액세스 권한이 있을 수 AWS있습니다. 서비스 사용자가 액세스해야 하는 AWS 기능과 리소스를 결정하는 것은 여러분의 몫입니다. 그런 다음, IAM 관리자에게 요청을 제출하여 서비스 사용자의 권한을 변경해야 합니다. 이 페이지의 정보를 검토하여 IAM의 기본 개념을 이해하십시오. 회사에서 IAM을 어떻게 사용할 수 있는지 자세히 알아보려면 사용 중인 사용 설명서를 참조하십시오. AWS AWS 서비스

IAM 관리자 - IAM 관리자라면 AWS에 대한 액세스 권한 관리 정책 작성 방법을 자세히 알고 싶을 것입니다. IAM에서 사용할 수 있는 AWS ID 기반 정책의 예를 보려면 사용 중인 사용 설명서를 참조하십시오. AWS 서비스

ID를 통한 인증

인증은 자격 증명 자격 증명을 AWS 사용하여 로그인하는 방법입니다. IAM 사용자로 인증 (로그인 AWS) 하거나 IAM 역할을 맡아 인증 (로그인) 해야 합니다. AWS 계정 루트 사용자

ID 소스를 통해 제공된 자격 증명을 사용하여 페더레이션 ID로 로그인할 수 있습니다. AWS AWS IAM Identity Center (IAM ID 센터) 사용자, 회사의 싱글 사인온 인증, Google 또는 Facebook 자격 증명이 페더레이션 ID의 예입니다. 연동 자격 증명으로 로그인할 때 관리자가 이전에 IAM 역할을 사용하여 ID 페더레이션을 설정했습니다. 페더레이션을 사용하여 액세스하는 경우 AWS 간접적으로 역할을 맡게 됩니다.

사용자 유형에 따라 AWS Management Console 또는 AWS 액세스 포털에 로그인할 수 있습니다. 로그인에 대한 자세한 내용은 AWS 로그인 사용 설명서의 내 로그인 방법을 참조하십시오. AWS AWS 계정

AWS 프로그래밍 방식으로 액세스하는 경우 자격 증명을 사용하여 요청에 암호화 방식으로 서명할 수 있는 소프트웨어 개발 키트 (SDK) 와 명령줄 인터페이스 (CLI) 를 AWS 제공합니다. AWS 도구를 사용하지 않는 경우 요청에 직접 서명해야 합니다. 권장 방법을 사용하여 직접 요청에 서명하는 방법에 대한 자세한 내용은 IAM 사용 설명서의 AWS API 요청 서명을 참조하십시오.

사용하는 인증 방법에 상관없이 추가 보안 정보를 제공해야 할 수도 있습니다. 예를 들어, AWS 계정의 보안을 강화하기 위해 다단계 인증 (MFA) 을 사용할 것을 권장합니다. 자세한 내용은 AWS IAM Identity Center 사용 설명서의 다중 인증 및 IAM 사용 설명서의 AWS에서 다중 인증(MFA) 사용을 참조하세요.

AWS 계정 루트 사용자

계정을 AWS 계정만들 때는 먼저 계정의 모든 AWS 서비스 리소스에 대한 완전한 액세스 권한을 가진 하나의 로그인 ID로 시작합니다. 이 ID를 AWS 계정 루트 사용자라고 하며, 계정을 만들 때 사용한 이메일 주소와 비밀번호로 로그인하여 액세스할 수 있습니다. 일상적인 태스크에 루트 사용자를 사용하지 않을 것을 강력히 권장합니다. 루트 사용자 보안 인증 정보를 보호하고 루트 사용자만 수행할 수 있는 태스크를 수행하는 데 사용하세요. 루트 사용자로 로그인해야 하는 태스크의 전체 목록은 IAM 사용자 안내서의 루트 사용자 보안 인증이 필요한 태스크를 참조하세요.

연동 자격 증명

가장 좋은 방법은 관리자 액세스가 필요한 사용자를 비롯한 수동 AWS 서비스 사용자가 ID 공급자와의 페더레이션을 사용하여 임시 자격 증명을 사용하여 액세스하도록 하는 것입니다.

페더레이션 ID는 기업 사용자 디렉토리, 웹 ID 공급자, Identity Center 디렉터리의 사용자 또는 ID 소스를 통해 제공된 자격 증명을 사용하여 액세스하는 AWS 서비스 모든 사용자를 말합니다. AWS Directory Service페더레이션 ID에 AWS 계정액세스하면 이들이 역할을 맡고 역할은 임시 자격 증명을 제공합니다.

중앙 집중식 액세스 관리를 위해 AWS IAM Identity Center(을)를 사용하는 것이 좋습니다. IAM Identity Center에서 사용자 및 그룹을 생성하거나 자체 ID 소스의 사용자 및 그룹 집합에 연결하고 동기화하여 모든 사용자 및 애플리케이션에서 사용할 수 있습니다. AWS 계정 IAM Identity Center에 대한 자세한 내용은AWS IAM Identity Center 사용 설명서에서 IAM Identity Center란 무엇입니까?를 참조하세요.

IAM 사용자 및 그룹

IAM 사용자는 단일 사용자 또는 애플리케이션에 대한 특정 권한을 AWS 계정 가진 사용자 내 자격 증명입니다. 가능하면 암호 및 액세스 키와 같은 장기 자격 증명이 있는 IAM 사용자를 생성하는 대신 임시 자격 증명을 사용하는 것이 좋습니다. 하지만 IAM 사용자의 장기 자격 증명이 필요한 특정 사용 사례가 있는 경우 액세스 키를 교체하는 것이 좋습니다. 자세한 내용은 IAM 사용 설명서의 장기 보안 인증이 필요한 사용 사례의 경우 정기적으로 액세스 키 교체를 참조하세요.

IAM 그룹은 IAM 사용자 컬렉션을 지정하는 자격 증명입니다. 사용자는 그룹으로 로그인할 수 없습니다. 그룹을 사용하여 여러 사용자의 권한을 한 번에 지정할 수 있습니다. 그룹을 사용하면 대규모 사용

자 집합의 권한을 더 쉽게 관리할 수 있습니다. 예를 들어, IAMAdmins라는 그룹이 있고 이 그룹에 IAM 리소스를 관리할 권한을 부여할 수 있습니다.

사용자는 역할과 다릅니다. 사용자는 한 사람 또는 애플리케이션과 고유하게 연결되지만, 역할은 해당 역할이 필요한 사람이라면 누구나 수임할 수 있습니다. 사용자는 영구적인 장기 보안 인증을 가지고 있지만, 역할은 임시 보안 인증만 제공합니다. 자세한 정보는 IAM 사용 설명서의 IAM 사용자를 만들어야 하는 경우(역할이 아님)를 참조하세요.

IAM 역할

IAM 역할은 특정 권한을 가진 사용자 AWS 계정 내의 자격 증명입니다. IAM 사용자와 유사하지만, 특정 개인과 연결되지 않습니다. 역할을 AWS Management Console 전환하여 에서 일시적으로 IAM 역할을 맡을 수 있습니다. AWS CLI 또는 AWS API 작업을 호출하거나 사용자 지정 URL을 사용하여 역할을 수임할 수 있습니다. 역할 사용 방법에 대한 자세한 정보는 IAM 사용 설명서의 IAM 역할 사용을 참조하세요.

임시 보안 인증이 있는 IAM 역할은 다음과 같은 상황에서 유용합니다.

- 페더레이션 사용자 액세스 - 연동 자격 증명에 권한을 부여하려면 역할을 생성하고 해당 역할의 권한을 정의합니다. 연동 자격 증명이 인증되면 역할이 연결되고 역할에 정의된 권한이 부여됩니다. 페더레이션 역할에 대한 자세한 내용은 IAM 사용 설명서의 타사 자격 증명 공급자의 역할 만들기를 참조하세요. IAM Identity Center를 사용하는 경우 권한 세트를 구성합니다. 인증 후 아이덴티티가 액세스할 수 있는 항목을 제어하기 위해 IAM Identity Center는 권한 세트를 IAM의 역할과 연관 짓습니다. 권한 세트에 대한 자세한 내용은 AWS IAM Identity Center 사용 설명서의 권한 세트를 참조하세요.
- 임시 IAM 사용자 권한 - IAM 사용자 또는 역할은 IAM 역할을 수임하여 특정 태스크에 대한 다양한 권한을 임시로 받을 수 있습니다.
- 크로스 계정 액세스 - IAM 역할을 사용하여 다른 계정의 사용자(신뢰할 수 있는 보안 주체)가 내 계정의 리소스에 액세스하도록 허용할 수 있습니다. 역할은 계정 간 액세스를 부여하는 기본적인 방법입니다. 그러나 일부 AWS 서비스경우에는 역할을 프록시로 사용하는 대신 정책을 리소스에 직접 연결할 수 있습니다. 크로스 계정 액세스를 위한 역할과 리소스 기반 정책의 차이점을 알아보려면 IAM 사용 설명서의 IAM 역할과 리소스 기반 정책의 차이를 참조하세요.
- 서비스 간 액세스 — 일부는 다른 AWS 서비스서비스의 기능을 AWS 서비스 사용합니다. 예컨대, 어떤 서비스에서 호출을 수행하면 일반적으로 해당 서비스는 Amazon EC2에서 애플리케이션을 실행하거나 Amazon S3에 객체를 저장합니다. 서비스는 호출하는 보안 주체의 권한을 사용하거나, 서비스 역할을 사용하거나, 또는 서비스 연결 역할을 사용하여 이 작업을 수행할 수 있습니다.
  - 순방향 액세스 세션 (FAS) — IAM 사용자 또는 역할을 사용하여 작업을 수행하는 경우 보안 AWS 주체로 간주됩니다. 일부 서비스를 사용하는 경우 다른 서비스에서 다른 작업을 시작하는 작업을

수행할 수 있습니다. FAS는 전화를 거는 주체의 권한을 다운스트림 AWS 서비스서비스에 AWS 서비스 요청하기 위한 요청과 결합하여 사용합니다. FAS 요청은 다른 서비스 AWS 서비스 또는 리소스와의 상호 작용이 필요한 요청을 서비스가 수신한 경우에만 이루어집니다. 이 경우 두 작업을 모두 수행할 수 있는 권한이 있어야 합니다. FAS 요청 시 정책 세부 정보는 전달 액세스 세션을 참조하세요.

- 서비스 역할 - 서비스 역할은 서비스가 사용자를 대신하여 태스크를 수행하기 위해 맡는 IAM 역할입니다. IAM 관리자는 IAM 내에서 서비스 역할을 생성, 수정 및 삭제할 수 있습니다. 자세한 정보는 IAM 사용 설명서의 AWS 서비스에 대한 권한을 위임할 역할 생성을 참조하세요.
- 서비스 연결 역할 — 서비스 연결 역할은 에 연결된 서비스 역할의 한 유형입니다. AWS 서비스서비스는 사용자를 대신하여 작업을 수행하기 위해 역할을 수임할 수 있습니다. 서비스 연결 역할은 사용자에게 AWS 계정 표시되며 해당 서비스가 소유합니다. IAM 관리자는 서비스 링크 역할의 권한을 볼 수 있지만 편집은 할 수 없습니다.

- Amazon EC2에서 실행되는 애플리케이션 — IAM 역할을 사용하여 EC2 인스턴스에서 실행되고 API 요청을 AWS CLI 하는 애플리케이션의 임시 자격 증명을 관리할 수 있습니다. AWS 이는 EC2 인스턴스 내에 액세스 키를 저장할 때 권장되는 방법입니다. EC2 인스턴스에 AWS 역할을 할당하고 모든 애플리케이션에서 사용할 수 있게 하려면 인스턴스에 연결된 인스턴스 프로필을 생성합니다. 인스턴스 프로파일에는 역할이 포함되어 있으며 EC2 인스턴스에서 실행되는 프로그램이 임시 보안 인증을 얻을 수 있습니다. 자세한 정보는 IAM 사용 설명서의 IAM 역할을 사용하여 Amazon EC2 인스턴스에서 실행되는 애플리케이션에 권한 부여를 참조하세요.

IAM 역할을 사용할지 또는 IAM 사용자를 사용할지를 알아보려면 IAM 사용 설명서의 IAM 역할(사용자 대신)을 생성하는 경우를 참조하세요.

정책을 사용한 액세스 관리

정책을 생성하고 이를 AWS ID 또는 리소스에 AWS 연결하여 액세스를 제어할 수 있습니다. 정책은 ID 또는 리소스와 연결될 때 AWS 해당 권한을 정의하는 객체입니다. AWS 주도자 (사용자, 루트 사용자 또는 역할 세션) 가 요청할 때 이러한 정책을 평가합니다. 정책에서 권한은 요청이 허용되거나 거부되는 지를 결정합니다. 대부분의 정책은 JSON 문서로 AWS 저장됩니다. JSON 정책 문서의 구조와 콘텐츠에 대한 자세한 정보는 IAM 사용 설명서의 JSON 정책 개요를 참조하세요.

관리자는 AWS JSON 정책을 사용하여 누가 무엇에 액세스할 수 있는지 지정할 수 있습니다. 즉, 어떤 보안 주체가 어떤 리소스와 어떤 조건에서 작업을 수행할 수 있는지를 지정할 수 있습니다.

기본적으로, 사용자와 역할에는 어떠한 권한도 없습니다. 사용자에게 사용자가 필요한 리소스에서 작업을 수행할 권한을 부여하려면 IAM 관리자가 IAM 정책을 생성하면 됩니다. 그런 다음 관리자가 IAM 정책을 역할에 추가하고, 사용자가 역할을 수임할 수 있습니다.

IAM 정책은 작업을 수행하기 위해 사용하는 방법과 상관없이 작업에 대한 권한을 정의합니다. 예를 들어, `iam:GetRole`태스크를 허용하는 정책이 있다고 가정합니다. 해당 정책을 사용하는 사용자는 AWS Management Console, AWS CLI, 또는 AWS API에서 역할 정보를 가져올 수 있습니다.

ID 기반 정책

ID 기반 정책은 IAM 사용자, 사용자 그룹 또는 역할과 같은 자격 증명에 연결할 수 있는 JSON 권한 정책 문서입니다. 이러한 정책은 사용자와 역할이 어떤 리소스와 어떤 조건에서 어떤 작업을 수행할 수 있는 지를 제어합니다. 자격 증명 기반 정책을 생성하는 방법을 알아보려면 IAM 사용 설명서의 IAM 정책 생성을 참조하세요.

자격 증명 기반 정책은 인라인 정책 또는 관리형 정책으로 한층 더 분류할 수 있습니다. 인라인 정책은 단일 사용자, 그룹 또는 역할에 직접 포함됩니다. 관리형 정책은 내 여러 사용자, 그룹 및 역할에 연결할 수 있는 독립형 정책입니다. AWS 계정관리형 정책에는 AWS 관리형 정책과 고객 관리형 정책이 포함됩니다. 관리형 정책 또는 인라인 정책을 선택하는 방법을 알아보려면 IAM 사용 설명서의 관리형 정책과 인라인 정책의 선택을 참조하세요.

리소스 기반 정책

리소스 기반 정책은 리소스에 연결하는 JSON 정책 설명서입니다. 리소스 기반 정책의 예는 IAM 역할 신뢰 정책과 Amazon S3 버킷 정책입니다. 리소스 기반 정책을 지원하는 서비스에서 서비스 관리자는 이러한 정책을 사용하여 특정 리소스에 대한 액세스를 통제할 수 있습니다. 정책이 연결된 리소스의 경우 정책은 지정된 보안 주체가 해당 리소스와 어떤 조건에서 어떤 작업을 수행할 수 있는 지를 정의합니다. 리소스 기반 정책에서 보안 주체를 지정해야 합니다. 보안 주체에는 계정, 사용자, 역할, 연동 사용자 등이 포함될 수 있습니다. AWS 서비스

리소스 기반 정책은 해당 서비스에 있는 인라인 정책입니다. IAM의 AWS 관리형 정책은 리소스 기반 정책에 사용할 수 없습니다.

액세스 제어 목록(ACLs)

액세스 제어 목록(ACL)은 어떤 보안 주체(계정 멤버, 사용자 또는 역할)가 리소스에 액세스할 수 있는 권한을 가지고 있는지를 제어합니다. ACLs는 JSON 정책 문서 형식을 사용하지 않지만 리소스 기반 정책과 유사합니다.

ACL을 지원하는 서비스의 예로는 아마존 S3와 아마존 VPC가 있습니다. AWS WAF ACL에 대해 자세히 알아보려면 Amazon Simple Storage Service 개발자 안내서의 액세스 제어 목록(ACL) 개요를 참조하세요.

기타 정책 타입

AWS 일반적이지 않은 추가 정책 유형을 지원합니다. 이러한 정책 타입은 더 일반적인 정책 타입에 따라 사용자에게 부여되는 최대 권한을 설정할 수 있습니다.

- 권한 경계 – 권한 경계는 보안 인증 기반 정책에 따라 IAM 엔터티(IAM 사용자 또는 역할)에 부여할 수 있는 최대 권한을 설정하는 고급 기능입니다. 개체에 대한 권한 경계를 설정할 수 있습니다. 그 결과로 얻는 권한은 엔터티의 자격 증명 기반 정책과 그 권한 경계의 교집합입니다. `Principal` 필드에서 사용자나 역할을 보안 주체로 지정하는 리소스 기반 정책은 권한 경계를 통해 제한되지 않습니다. 이러한 정책 중 하나에 포함된 명시적 거부는 허용을 재정의합니다. 권한 경계에 대한 자세한 정보는 IAM 사용 설명서의 IAM 엔터티에 대한 권한 경계를 참조하세요.
- 서비스 제어 정책 (SCP) - SCP는 조직 또는 조직 단위 (OU) 에 대한 최대 권한을 지정하는 JSON 정책입니다. AWS Organizations AWS Organizations 사업체가 소유한 여러 AWS 계정 개를 그룹화하고 중앙에서 관리하는 서비스입니다. 조직에서 모든 기능을 활성화할 경우 서비스 제어 정책 (SCP)을 임의의 또는 모든 계정에 적용할 수 있습니다. SCP는 구성원 계정의 엔티티 (각 엔티티 포함) 에 대한 권한을 제한합니다. AWS 계정 루트 사용자조직 및 SCP에 대한 자세한 정보는 AWS Organizations 사용 설명서의 SCP 작동 방식을 참조하세요.
- 세션 정책 – 세션 정책은 역할 또는 페더레이션 사용자에 대해 임시 세션을 프로그래밍 방식으로 생성할 때 파라미터로 전달하는 고급 정책입니다. 결과적으로 얻는 세션의 권한은 사용자 또는 역할 자격 증명 기반 정책의 교차 및 세션 정책입니다. 또한 권한을 리소스 기반 정책에서 가져올 수도 있습니다. 이러한 정책 중 하나에 포함된 명시적 거부는 허용을 재정의합니다. 자세한 정보는 IAM 사용 설명서의 세션 정책을 참조하세요.

여러 정책 타입

여러 정책 타입이 요청에 적용되는 경우 결과 권한은 이해하기가 더 복잡합니다. 여러 정책 유형이 관련되어 있을 때 요청을 허용할지 여부를 AWS 결정하는 방법을 알아보려면 IAM 사용 설명서의 정책 평가 로직을 참조하십시오.

AWS 서비스 IAM을 사용하는 방법

대부분의 IAM 기능을 어떻게 AWS 서비스 사용하는지 자세히 알아보려면 IAM 사용 설명서의 IAM과 연동되는AWS 서비스를 참조하십시오.

특정 기능을 AWS 서비스 IAM과 함께 사용하는 방법을 알아보려면 관련 서비스 사용 설명서의 보안 섹션을 참조하십시오.

AWS ID 및 액세스 문제 해결

다음 정보를 사용하면 및 IAM을 사용할 때 발생할 수 있는 일반적인 문제를 AWS 진단하고 해결하는 데 도움이 됩니다.

주제

저는 다음과 같은 작업을 수행할 권한이 없습니다. AWS

작업을 수행할 권한이 없다는 오류가 수신되면, 작업을 수행할 수 있도록 정책을 업데이트해야 합니다.

다음 예제 오류는 mateojacksonIAM 사용자가 콘솔을 사용하여 가상 *my-example-widget*리소스에 대한 세부 정보를 보려고 하지만 가상 awes:*GetWidget*권한이 없을 때 발생합니다.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 awes:GetWidget on resource: my-example-widget
```

이 경우 awes:*GetWidget*작업을 사용하여 *my-example-widget*리소스에 액세스할 수 있도록 mateojackson사용자 정책을 업데이트해야 합니다.

도움이 필요하면 AWS 관리자에게 문의하세요. 관리자는 로그인 자격 증명을 제공한 사람입니다.

저는 IAM을 수행할 권한이 없습니다. PassRole

iam:PassRole 작업을 수행할 수 있는 권한이 없다는 오류가 수신되면 AWS에 역할을 전달할 수 있도록 정책을 업데이트해야 합니다.

새 서비스 역할 또는 서비스 연결 역할을 만드는 대신 기존 역할을 해당 서비스에 전달할 AWS 서비스 수 있는 기능도 있습니다. 이렇게 하려면 사용자가 서비스에 역할을 전달할 수 있는 권한을 가지고 있어야 합니다.

다음 예 오류는 marymajor라는 IAM 사용자가 콘솔을 사용하여 AWS에서 작업을 수행하려고 하는 경우에 발생합니다. 하지만 작업을 수행하려면 서비스 역할이 부여한 권한이 서비스에 있어야 합니다. Mary는 서비스에 역할을 전달할 수 있는 권한을 가지고 있지 않습니다.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

이 경우 Mary가 `iam:PassRole`작업을 수행할 수 있도록 Mary의 정책을 업데이트해야 합니다.

도움이 필요하면 관리자에게 문의하세요. AWS 관리자는 로그인 자격 증명을 제공한 사람입니다.

외부 사용자가 내 AWS 리소스에 액세스할 수 있도록 AWS 계정 허용하고 싶습니다.

다른 계정의 사용자 또는 조직 외부의 사람이 리소스에 액세스할 때 사용할 수 있는 역할을 생성할 수 있습니다. 역할을 수임할 신뢰할 수 있는 사람을 지정할 수 있습니다. 리소스 기반 정책 또는 액세스 제어 목록(ACL)을 지원하는 서비스의 경우 이러한 정책을 사용하여 다른 사람에게 리소스에 대한 액세스 권한을 부여할 수 있습니다.

자세히 알아보려면 다음을 참조하세요.

- 이러한 기능의 AWS 지원 여부를 알아보려면 을 참조하십시오AWS 서비스 IAM을 사용하는 방법.
- 소유한 리소스에 대한 액세스 권한을 AWS 계정 부여하는 방법을 알아보려면 IAM 사용 설명서에서 자신이 소유한 다른 AWS 계정 IAM 사용자에게 액세스 권한 제공을 참조하십시오.
- 제3자에게 리소스에 대한 액세스 권한을 제공하는 방법을 알아보려면 IAM 사용 설명서의 타사 AWS 계정AWS 계정 소유에 대한 액세스 제공을 참조하십시오.
- ID 페더레이션을 통해 액세스 권한을 제공하는 방법을 알아보려면 IAM 사용 설명서의 외부에서 인증된 사용자에게 액세스 권한 제공(ID 페더레이션)을 참조하세요.
- 크로스 계정 액세스를 위한 역할과 리소스 기반 정책 사용의 차이점을 알아보려면 IAM 사용 설명서의 IAM 역할과 리소스 기반 정책의 차이를 참조하세요.

이 AWS 제품 또는 서비스에 대한 규정 준수 검증

특정 규정 준수 프로그램의 범위 내에 AWS 서비스 있는지 알아보려면AWS 서비스 규정 준수 프로그램의AWS 서비스 범위별, 규정 참조하여 관심 있는 규정 준수 프로그램을 선택하십시오. 일반 정보는 AWS 규정 준수 프로그램AWS 보증 프로그램 규정AWS 참조하십시오.

를 사용하여 AWS Artifact타사 감사 보고서를 다운로드할 수 있습니다. 자세한 내용은 의 보고서 https://docs.aws.amazon.com/artifact/latest/ug/downloading-documents.html 참조하십시오 AWS Artifact.

사용 시 규정 준수 AWS 서비스 책임은 데이터의 민감도, 회사의 규정 준수 목표, 관련 법률 및 규정에 따라 결정됩니다. AWS 규정 준수에 도움이 되는 다음 리소스를 제공합니다.

- 보안 및 규정 준수 퀵 스타트 가이드 - 이 배포 가이드에서는 아키텍처 고려 사항을 설명하고 보안 및 규정 준수에 AWS 중점을 둔 기본 환경을 배포하기 위한 단계를 제공합니다.
- Amazon Web Services의 HIPAA 보안 및 규정 준수를 위한 설계 — 이 백서에서는 기업이 HIPAA 적격 애플리케이션을 만드는 AWS 데 사용할 수 있는 방법을 설명합니다.

  > **Note**
  >
  > 모든 AWS 서비스 사람이 HIPAA 자격을 갖춘 것은 아닙니다. 자세한 내용은 HIPAA 적격 서비스 참조를 참조하십시오.

- AWS 규정 준수 리소스AWS — 이 워크북 및 가이드 모음은 해당 산업 및 지역에 적용될 수 있습니다.
- AWS 고객 규정 준수 가이드 — 규정 준수의 관점에서 공동 책임 모델을 이해하십시오. 이 가이드에서는 보안을 유지하기 위한 모범 사례를 AWS 서비스 요약하고 여러 프레임워크 (미국 표준 기술 연구소 (NIST), 결제 카드 산업 보안 표준 위원회 (PCI), 국제 표준화기구 (ISO) 등) 에서 보안 제어에 대한 지침을 매핑합니다.
- AWS Config 개발자 안내서의 규칙을 통한 리소스 평가 — 이 AWS Config 서비스는 리소스 구성이 내부 관행, 업계 지침 및 규정을 얼마나 잘 준수하는지 평가합니다.
- AWS Security Hub— 이를 AWS 서비스 통해 내부 AWS보안 상태를 포괄적으로 파악할 수 있습니다. Security Hub는 보안 제어를 사용하여 AWS 리소스를 평가하고 보안 업계 표준 및 모범 사례에 대한 규정 준수를 확인합니다. 지원되는 서비스 및 제어 목록은 Security Hub 제어 참조를 참조하십시오.
- Amazon GuardDuty — 환경에 의심스럽고 악의적인 활동이 있는지 AWS 계정모니터링하여 워크로드, 컨테이너 및 데이터에 대한 잠재적 위협을 AWS 서비스 탐지합니다. GuardDuty 특정 규정 준수 프레임워크에서 요구하는 침입 탐지 요구 사항을 충족하여 PCI DSS와 같은 다양한 규정 준수 요구 사항을 해결하는 데 도움이 될 수 있습니다.
- AWS Audit Manager— 이를 AWS 서비스 통해 AWS 사용량을 지속적으로 감사하여 위험을 관리하고 규정 및 업계 표준을 준수하는 방법을 단순화할 수 있습니다.

이 AWS 제품 또는 서비스는 지원하는 특정 Amazon Web Services (AWS) 서비스를 통해 공동 책임 모델을 따릅니다. AWS 서비스 보안 정보는 AWS 서비스 보안 설명서 페이지 및 AWS 규정 준수 프로그램의 규정 준수 노력 범위에 속하는AWS 서비스를 참조하십시오.

이 AWS 제품 또는 서비스에 대한 복원력

AWS 글로벌 인프라는 가용 영역을 중심으로 AWS 리전 구축됩니다.

AWS 리전 물리적으로 분리되고 격리된 여러 가용 영역을 제공합니다. 이 가용 영역은 지연 시간이 짧고 처리량이 높으며 중복성이 높은 네트워킹으로 연결됩니다.

가용 영역을 사용하면 중단 없이 영역 간에 자동으로 장애 극복 조치가 이루어지는 애플리케이션 및 데이터베이스를 설계하고 운영할 수 있습니다. 가용 영역은 기존의 단일 또는 다중 데이터 센터 인프라보다 가용성, 내결함성, 확장성이 뛰어납니다.

AWS 지역 및 가용 영역에 대한 자세한 내용은 글로벌 인프라를 참조하십시오AWS .

이 AWS 제품 또는 서비스는 지원하는 특정 Amazon Web Services (AWS) 서비스를 통해 공동 책임 모델을 따릅니다. AWS 서비스 보안 정보는 AWS 서비스 보안 설명서 페이지 및 AWS 규정 준수 프로그램의 규정 준수 노력 범위에 속하는AWS 서비스를 참조하십시오.

이 AWS 제품 또는 서비스의 인프라 보안

이 AWS 제품 또는 서비스는 관리 서비스를 사용하므로 AWS 글로벌 네트워크 보안의 보호를 받습니다. AWS 보안 서비스 및 인프라 AWS 보호 방법에 대한 자세한 내용은 AWS 클라우드 보안을 참조하십시오. 인프라 보안 모범 사례를 사용하여 AWS 환경을 설계하려면 Security Pillar AWS Well-Architected Framework의 인프라 보호를 참조하십시오.

AWS 게시된 API 호출을 사용하여 네트워크를 통해 이 AWS 제품 또는 서비스에 액세스할 수 있습니다. 고객은 다음을 지원해야 합니다.

- 전송 계층 보안(TLS) TLS 1.2는 필수이며 TLS 1.3을 권장합니다.
- DHE(Ephemeral Diffie-Hellman) 또는 ECDHE(Elliptic Curve Ephemeral Diffie-Hellman)와 같은 완전 전송 보안(PFS)이 포함된 암호 제품군 Java 7 이상의 최신 시스템은 대부분 이러한 모드를 지원합니다.

또한 요청은 액세스 키 ID 및 IAM 주체와 관련된 비밀 액세스 키를 사용하여 서명해야 합니다. 또는 AWS Security Token Service(AWS STS)를 사용하여 임시 보안 인증을 생성하여 요청에 서명할 수 있습니다.

이 AWS 제품 또는 서비스는 지원하는 특정 Amazon Web Services (AWS) 서비스를 통해 공동 책임 모델을 따릅니다. AWS 서비스 보안 정보는 AWS 서비스 보안 설명서 페이지 및 AWS 규정 준수 프로그램의 규정 준수 노력 범위에 속하는AWS 서비스를 참조하십시오.

Amazon S3 암호화: 클라이언트 마이그레이션

이 주제에서는 () 암호화 클라이언트의 버전 1 (V1) 에서 버전 2 Amazon Simple Storage Service (V2 Amazon S3) 로 애플리케이션을 마이그레이션하고 마이그레이션 프로세스 전반에 걸쳐 애플리케이션 가용성을 보장하는 방법을 보여줍니다.

필수 조건

Amazon S3 클라이언트 측 암호화에는 다음이 필요합니다.

- Java 8 이상이 애플리케이션 환경에 설치되어 있어야 합니다. 는 오라클 자바 SE 개발 키트 및 Red Hat OpenJDK 및 JDK와 같은 Amazon Corretto오픈 자바 개발 키트 (OpenJDK) 배포판과 함께 AWS SDK for Java 작동합니다. AdoptOpen
- Bouncy Castle Crypto 패키지. Bouncy Castle.jar 파일을 애플리케이션 환경의 클래스 경로에 배치하거나 Maven `pom.xml` 파일에 artifactId `bcprov-ext-jdk15on`(`org.bouncycastle` groupID 사용)에 대한 종속성을 추가할 수 있습니다.

마이그레이션 개요

이 마이그레이션은 다음 두 단계로 진행됩니다.

1. 새 형식을 읽도록 기존 클라이언트를 업데이트하세요. 버전 1.11.837 이상을 사용하도록 애플리케이션을 업데이트하고 애플리케이션을 재배포하십시오. AWS SDK for Java 이렇게 하면 애플리케이션의 Amazon S3 클라이언트측 암호화 서비스 클라이언트가 V2 서비스 클라이언트에서 만든 객체를 해독할 수 있습니다. 애플리케이션에서 여러 AWS SDK를 사용하는 경우 각 SDK를 개별적으로 업데이트해야 합니다.
2. 암호화 및 복호화 클라이언트를 V2로 마이그레이션합니다. 모든 V1 암호화 클라이언트가 V2 암호화 형식을 읽을 수 있게 되면 애플리케이션 코드의 Amazon S3 클라이언트 측 암호화 및 암호 해독 클라이언트를 업데이트하여 해당하는 V2를 사용하도록 하십시오.

새 형식을 읽도록 기존 클라이언트를 업데이트하세요

V2 암호화 클라이언트는 이전 버전에서 지원하지 않는 암호화 알고리즘을 사용합니다. AWS SDK for Java

마이그레이션의 첫 번째 단계는 V1 암호화 클라이언트가 AWS SDK for Java의 버전 1.11.837 이상을 사용하도록 업데이트하는 것입니다. (Java API 참조 버전 1.x에서 찾을 수 있는 최신 릴리스 버전으로

업데이트하는 것이 좋습니다.) 이렇게 하려면 프로젝트 구성의 종속성을 업데이트합니다. 프로젝트 구성이 업데이트된 후 프로젝트를 다시 빌드하고 다시 배포하세요.

이 단계를 완료하면 애플리케이션의 V1 암호화 클라이언트가 V2 암호화 클라이언트가 작성한 객체를 읽을 수 있습니다.

프로젝트 구성의 종속성을 업데이트하세요.

AWS SDK for Java의 버전 1.11.837 이상을 사용하도록 프로젝트 구성 파일(예: pom.xml 또는 build.gradle)을 수정하세요. 그런 다음 프로젝트를 다시 빌드하고 다시 배포하세요.

새 애플리케이션 코드를 배포하기 전에 이 단계를 완료하면 마이그레이션 프로세스 중에 플릿 전체에서 암호화 및 암호 해독 작업을 일관되게 유지할 수 있습니다.

Maven 사용 사례

pom.xml 파일의 코드 조각:

```
<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>
```

Gradle 사용 사례

build.gradle 파일의 코드 조각:

```
dependencies {
  implementation platform('com.amazonaws:aws-java-sdk-bom:1.11.837')
  implementation 'com.amazonaws:aws-java-sdk-s3'
}
```

암호화 및 복호화 클라이언트를 V2로 마이그레이션

프로젝트가 최신 SDK 버전으로 업데이트되면 V2 클라이언트를 사용하도록 애플리케이션 코드를 수정할 수 있습니다. 이렇게 하려면 먼저 새 서비스 클라이언트 빌더를 사용하도록 코드를 업데이트하세요. 그런 다음 이름이 바뀐 빌더의 메서드를 사용하여 암호화 자료를 제공하고 필요에 따라 서비스 클라이언트를 추가로 구성하세요.

이 코드 스니펫은 에서 클라이언트 측 암호화를 사용하는 방법을 보여주고 V1과 AWS SDK for Java V2 암호화 클라이언트를 비교합니다.

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();
```

위 예시에서는 `cryptoMode`를 `AuthenticatedEncryption`로 설정합니다. 이는 V2 암호화 클라이언트가 V1 암호화 클라이언트가 작성한 객체를 읽을 수 있도록 하는 설정입니다. 클라이언트에 V1 클라이언트가 작성한 객체를 읽을 수 있는 기능이 필요하지 않은 경우에는 기본 설정인 `StrictAuthenticatedEncryption`를 대신 사용하는 것이 좋습니다.

V2 암호화 클라이언트 생성

AmazonS3 v2.EncryptionBuilder () 를 호출하여 V2 암호화 클라이언트를 구성할 수 있습니다. EncryptionClient

기존 V1 암호화 클라이언트를 모두 V2 암호화 클라이언트로 교체할 수 있습니다. V2 암호화 클라이언트는 V1 암호화 클라이언트가 작성한 객체를 항상 읽을 수 있습니다. 단, `를 사용하도록 V2 암호화 클라이언트를 구성하여 읽도록 허용하기만 하면 됩니다. AuthenticatedEncryption `cryptoMode

새 V2 암호화 클라이언트를 만드는 것은 V1 암호화 클라이언트를 만드는 방법과 매우 비슷합니다. 그러나 몇 가지 차이점이 있습니다.

- `CryptoConfiguration` 객체 대신 `CryptoConfigurationV2` 객체를 사용하여 클라이언트를 구성합니다. 이 파라미터는 필수 사항입니다.
- V2 암호화 클라이언트의 기본 `cryptoMode` 설정은 `StrictAuthenticatedEncryption`입니다. V1 암호화 클라이언트의 경우 `EncryptionOnly`입니다.
- 암호화 클라이언트 빌더의 메서드 withEncryptionMaterials() 의 이름이 withEncryptionMaterialsProvider () 로 변경되었습니다. 이는 단순히 인수 유형을 더 정확하게 반영하기 위한 외관상의 변경일 뿐입니다. 서비스 클라이언트를 구성할 때 새 메서드를 사용해야 합니다.

> **Note**
>
> AES-GCM으로 해독할 때는 해독된 데이터를 사용하기 전에 전체 객체를 끝까지 읽습니다. 이는 객체가 암호화된 이후 수정되지 않았는지 확인하기 위한 것입니다.

암호화 자료 제공업체 사용

V1 암호화 클라이언트에서는 이미 사용하고 있는 동일한 암호화 자료 제공자 및 암호화 자료 객체를 계속 사용할 수 있습니다. 이러한 클래스는 암호화 클라이언트가 데이터를 보호하는 데 사용하는 키를 제공하는 역할을 합니다. V2 및 V1 암호화 클라이언트 모두와 호환하여 사용할 수 있습니다.

V2 암호화 클라이언트 구성

V2 암호화 클라이언트는 `CryptoConfigurationV2` 객체로 구성됩니다. 이 객체는 기본 생성자를 호출한 다음 필요에 따라 기본값에서 해당 속성을 수정하여 생성할 수 있습니다.

`CryptoConfigurationV2`의 기본값은 다음과 같습니다.

- `cryptoMode` = `CryptoMode.StrictAuthenticatedEncryption`
- `storageMode` = `CryptoStorageMode.ObjectMetadata`
- `secureRandom` = `SecureRandom`의 인스턴스
- `rangeGetMode` = `CryptoRangeGetMode.DISABLED`

- `unsafeUndecryptableObjectPassthrough = false`

참고로 V2 암호화 `cryptoMode` 클라이언트에서는 EncryptionOnly지원되지 않습니다. V2 암호화 클라이언트는 항상 인증된 암호화를 사용하여 콘텐츠를 암호화하고 V2 `KeyWrap` 객체를 사용하여 콘텐츠 암호화 키(CEK)를 보호합니다.

다음 예제는 V1에서 암호화 구성을 지정하는 방법과 CryptoConfigurationV2 객체를 인스턴스화하여 V2 암호화 클라이언트 빌더로 전달하는 방법을 보여줍니다.

V1

```
CryptoConfiguration cryptoConfiguration = new CryptoConfiguration()
        .withCryptoMode(CryptoMode.StrictAuthenticatedEncryption);
```

V2

```
CryptoConfigurationV2 cryptoConfiguration = new CryptoConfigurationV2()
        .withCryptoMode(CryptoMode.StrictAuthenticatedEncryption);
```

추가 예제

다음 예제는 V1에서 V2로의 마이그레이션과 관련된 특정 사용 사례를 해결하는 방법을 보여줍니다.

V1 암호화 클라이언트가 생성한 객체를 읽도록 서비스 클라이언트를 구성합니다.

V1 암호화 클라이언트를 사용하여 이전에 작성된 객체를 읽으려면 `cryptoMode`를 `AuthenticatedEncryption`로 설정합니다. 다음 코드 코드 조각은 이 설정으로 구성 객체를 생성하는 방법을 보여줍니다.

```
CryptoConfigurationV2 cryptoConfiguration = new CryptoConfigurationV2()
        .withCryptoMode(CryptoMode.AuthenticatedEncryption);
```

객체의 바이트 범위를 가져오도록 서비스 클라이언트를 구성합니다.

암호화된 S3 객체에서 get 바이트 범위를 지정할 수 있으려면 새 구성 설정 `rangeGetMode`을 활성화하세요. V2 암호화 클라이언트에서는 이 설정이 기본적으로 비활성화되어 있습니다. 활성화된 경우에도 범위 지정된 `get`는 클라이언트의 `cryptoMode` 설정에서 지원하는 알고리즘을 사용하여 암호화된 개체에서만 작동합니다. 자세한 내용은 API CryptoRangeGetMode참조를 참조하십시오. AWS SDK for Java

를 사용하여 V2 암호화 클라이언트를 사용하여 암호화된 Amazon S3 객체의 멀티파트 다운로드를 Amazon S3 TransferManager 수행하려는 경우 먼저 V2 암호화 클라이언트에서 `rangeGetMode` 설정을 활성화해야 합니다.

다음 코드 코드 조각은 범위 `get`을 수행하도록 V2 클라이언트를 구성하는 방법을 보여줍니다.

```
// 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);
```

AWS SDK for Java용 OpenPGP 키

AWS SDK for Java에 대해 공개적으로 사용 가능한 모든 Maven 아티팩트는 OpenPGP 표준을 사용하여 서명됩니다. 아티팩트의 서명을 확인하는 데 필요한 공개 키는 다음 섹션에서 확인할 수 있습니다.

현재 키

다음 표는 현재 릴리즈의 SDK for Java 1x 및 SDK for Java 2.x에 대한 OpenPGP 키 정보를 보여줍니다.

키 ID	0xAC107B386692DADD
유형	RSA
크기	4096/4096
Created	2016-06-30
Expires	2024-10-08
사용자 ID	AWS SDK 및 도구 <aws-dr-tools@amazon.com>
키 지문	FEB9 209F 2F2F 3F46 6484 1E55 AC10 7B38 6692 DADD

SDK for Java의 다음 OpenPGP 공개 키를 클립보드에 복사하려면 오른쪽 상단 모서리에 있는 "복사" 아이콘을 선택합니다.

```
-----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-----
```

문서 기록

이 항목에서는 AWS SDK for Java 개발자 안내서의 역사 전반에 걸쳐 변경된 주요 내용을 설명합니다.

이 설명서는 2024년 5월 21일에 작성되었습니다.

2024년 5월 21일

Java 명령줄 `networkaddress.cache.ttl` 시스템 속성을 사용하여 보안 속성을 설정하는 지침을 제거합니다. JVM TTL을 설정하는 방법 단원을 참조하세요.

2024년 1월 12일

v1.x 지원 종료를 알리는 배너를 추가하십시오. AWS SDK for Java

2023년 12월 6일

- 현재 OpenPGP 키를 입력합니다.

2023년 3월 14일

- IAM 모범 사례에 따라 가이드가 업데이트되었습니다. 자세한 내용은 IAM의 보안 모범 사례를 참조하십시오.

2022년 7월 28일

- EC2-Classic은 2022년 8월 15일에 사용 중지될 예정이라는 경고가 추가되었습니다.

2018년 3월 22일

- DynamoDB 예를 들어 Tomcat 세션 관리 도구는 더 이상 지원되지 않으므로 제거되었습니다.

2017년 11월 2일

- 새 항목: AWS KMS 관리 키를 사용한 클라이언트 측 Amazon S3 암호화 및 클라이언트 측 암호화 사용, Amazon S3 클라이언트 마스터 키를 사용한 클라이언트 측 암호화 사용 등 암호화 Amazon S3 클라이언트에 대한 암호화 예제가 추가되었습니다.Amazon S3

2017년 8월 14일

- 버킷 및 객체에 대한 Amazon S3 액세스 권한 관리 및 버킷을 웹 사이트로 구성이라는 새 주제를 포함하여Amazon S3 예제 사용 AWS SDK for Java 섹션을 여러 번 업데이트했습니다. Amazon S3

2017년 8월 4일

- 새 항목인 AWS SDK for Java용 지표 활성화에서는 AWS SDK for Java에 대한 애플리케이션 및 SDK 성능 지표 생성 방법을 설명합니다.

2017년 8월 3일

- AWS SDK for Java섹션 사용 CloudWatch CloudWatch 예제에 지표 가져오기, 사용자 지정 지표 데이터 게시 CloudWatch, 경보 관련 작업, 경보 작업, 이벤트 전송 등에 새 예제가 추가되었습니다. CloudWatch CloudWatch CloudWatch

2017년 3월 27일

- AWS SDK for Java섹션 사용 Amazon EC2Amazon EC2 예시 (Amazon EC2 인스턴스 관리, 엘라스틱 IP 주소 사용, 지역 및 가용 영역 사용 Amazon EC2, Amazon EC2 키 페어 사용, 보안 그룹 사용) 에 Amazon EC2 더 많은 예제를 추가했습니다.

2017년 3월 21일

- IAM 액세스 키 관리, IAM 사용자 관리, IAM 계정 별칭 사용, IAM정책 사용, IAM서버 인증서 사용 AWS SDK for Java섹션에 새로운 IAM 예제 세트를 추가했습니다.

2017년 3월 13일

- Amazon SQS 섹션에 Amazon SQS 메시지 대기열의 긴 폴링 활성화, 가시성 제한 시간 설정, 데드레터 대기열 사용이라는 세 가지 새로운 주제를 추가했습니다. Amazon SQSAmazon SQS

2017년 1월 26일

- TransferManager Amazon S3 작업에 사용이라는 새 Amazon S3 항목과 함께 사용 섹션의 AWS SDK for Java항목과 함께 새로운 AWS 개발 모범 사례를 추가했습니다. AWS SDK for Java

2017년 1월 16일

- 버킷 정책을 사용한 버킷 액세스 관리라는 새 Amazon S3 Amazon SQS 항목과 Amazon SQS 메시지 대기열 작업, 메시지 수신 및 삭제 등 두 개의 새 항목이 추가되었습니다. Amazon S3 Amazon SQS

2016년 12월 16일

- 에서 테이블 작업 DynamoDB 및 항목 작업에 대한 새 예제 항목이 추가되었습니다 DynamoDB. DynamoDB

2016년 9월 26일

- 고급 섹션의 항목은 SDK 사용의 핵심이기 때문에 사용법으로 옮겨졌습니다. AWS SDK for Java

2016년 8월 25일

- 클라이언트 빌더를 사용하여 클라이언트 생성을 간소화하는 방법을 보여주는 새 AWS SDK for Java항목인 서비스 클라이언트 만들기가 Using the 에 추가되었습니다. AWS 서비스

  AWS SDK for Java 코드 예제 섹션이 전체 예제 코드가 들어 있는 리포지토리를 기반으로 GitHub 하는 새로운 S3 예제로 업데이트되었습니다.

2016년 5월 02일

- 새 항목인 비동기 프로그래밍이 AWS SDK for Java사용 단원에 추가되었으며 `Future` 객체를 반환하거나 `AsyncHandler`를 사용하는 비동기 클라이언트 메서드 사용 방법을 설명합니다.

2016년 4월 26일

- SSL 인증서 요구 사항 항목은 더 이상 관련이 없으므로 제거되었습니다. SHA-1 서명 인증서에 대한 지원은 2015년부터 더 이상 제공되지 않으며 테스트 스크립트를 포함했던 사이트가 제거되었습니다.

2016년 3월 14일

- 기존 활동을 사용하는 대신 함수를 작업으로 Lambda 호출하는 워크플로를 구현하는 Amazon SWF 방법을 설명하는 Lambda Tasks라는 새 주제를 Amazon SWF 섹션에 추가했습니다. Amazon SWF

2016년 3월 4일

- AWS SDK for Java를 사용하는Amazon SWF 예제 섹션을 다음과 같이 새로운 내용으로 업데이트했습니다.
  - Amazon SWF 기초 - 프로젝트에 SWF를 포함하는 방법에 대한 기본 정보를 제공합니다.
  - 간단한 Amazon SWF 응용 프로그램 만들기 - Java를 처음 접하는 개발자를 위한 step-by-step 지침을 제공하는 새 자습서입니다. Amazon SWF
  - 활동 및 워크플로 작업자 정상 종료는 Java의 동시 클래스를 사용하여 Amazon SWF 작업자 클래스를 정상적으로 종료하는 방법에 대해 설명합니다.

2016년 2월 23일

- AWS SDK for Java 개발자 안내서의 소스가 로 이동되었습니다 aws-java-developer-guide.

2015년 12월 28일

- the section called “DNS 이름 조회를 위한 JVM TTL 설정”고급에서 Using the 로 옮겨졌으며 명확성을 위해 다시 작성되었습니다. AWS SDK for Java

  Apache Maven으로 SDK 사용하기가 프로젝트에 SDK의 BOM을 포함하는 방법에 대한 정보로 업데이트되었습니다.

2015년 8월 4일

- SSL 인증서 요구 사항은 시작하기 단원에 새로 추가된 항목으로, SSL 연결용으로 SHA256 서명 인증서로 전환한다는 AWS과 이러한 인증서(2015년 9월 30일 이후 AWS 액세스 시 필수)를 사용하도록 이전 1.6 이하 Java 환경을 수정하는 방법에 대해 설명합니다.

> **Note**
>
> Java 1.7+에서는 SHA256 서명 인증서를 사용할 수 있습니다.

2014년 5월 14일

- 소개 및 시작 자료는 새로운 가이드 구조를 지원하기 위해 대대적으로 수정되었으며, 이제 AWS 자격 증명 및 개발 지역을 설정하는 방법에 대한 지침이 포함되어 있습니다.

  코드 샘플에 대한 내용을 추가 설명서 및 리소스 단원으로 옮겼습니다.

  SDK의 개정 기록 보기 방법에 대한 설명을 소개 단원으로 옮겼습니다.

2014년 5월 9일

- AWS SDK for Java 설명서의 전체 구조가 단순화되었으며 시작하기 및 추가 설명서 및 리소스 항목이 업데이트되었습니다.

  새 항목이 추가되었습니다.

  - AWS 자격 증명 작업에서는 AWS SDK for Java을 사용하여 사용할 자격 증명을 지정할 수 있는 다양한 방법을 설명합니다.
  - IAM 역할을 사용하여 AWS 리소스에 대한 액세스 권한 부여 Amazon EC2 - EC2 인스턴스에서 실행되는 애플리케이션의 자격 증명을 안전하게 지정하는 방법에 대한 정보를 제공합니다.

2013년 9월 9일

- 문서 이력 항목에서는 AWS SDK for Java 개발자 가이드에 대한 변경 사항을 추적합니다. 이는 릴리스 정보 기록을 비교하기 위한 것입니다.