aws

1.x 版開發人員指南

# 適用於 Java 的 AWS SDK 1.x



# 適用於 Java 的 AWS SDK 1.x: 1.x 版開發人員指南

# Table of Contents


..... viii
適用於 Java 的 AWS SDK 1.x ..... 1
發行的 SDK 第 2 版 ..... 1
其他文件與資源 ..... 1
Eclipse IDE 支援 ..... 2
開發 Android 應用程式 ..... 2
檢視開發套件的修訂歷史記錄 ..... 2
建置舊版 SDK 的 Java 參考文件 ..... 2
開始使用 ..... 3
基本設定 ..... 3
概觀 ..... 3
AWS 存取入口網站的登入功能 ..... 4
設定共用組態檔案 ..... 4
安裝 Java 開發環境 ..... 6
取得 的方式 適用於 Java 的 AWS SDK ..... 6
先決條件 ..... 6
使用建置工具 ..... 6
下載預先建置的 jar ..... 6
從來源建置 ..... 7
使用建置工具 ..... 8
使用軟體開發套件搭配 Apache Maven ..... 8
使用軟體開發套件搭配 Gradle ..... 11
暫時登入資料和區域 ..... 14
設定臨時登入資料 ..... 14
重新整理 IMDS 登入資料 ..... 15
設定 AWS 區域 ..... 16
使用 適用於 Java 的 AWS SDK ..... 18
使用 AWS 開發的最佳實務 適用於 Java 的 AWS SDK ..... 18
S3 ..... 18
建立服務用戶端 ..... 19
取得用戶端建置器 ..... 19
建立非同步用戶端 ..... 20
使用 DefaultClient ..... 21
用戶端生命週期 ..... 21

提供臨時登入資料 ..... 22
使用預設登入資料供應商鏈結 ..... 22
指定登入資料提供者或提供者鏈 ..... 25
明確指定臨時登入資料 ..... 26
詳細資訊 ..... 26
AWS 區域 選擇 ..... 26
檢查區域中的服務可用性 ..... 26
選擇區域 ..... 27
選擇特定端點 ..... 27
從環境自動判斷區域 ..... 28
例外狀況處理 ..... 29
為什麼使用未檢查的例外狀況？ ..... 29
AmazonServiceException (和子類別) ..... 30
AmazonClientException ..... 30
非同步程式設計 ..... 30
Java 未來 ..... 31
非同步回呼 ..... 32
最佳實務 ..... 34
記錄 適用於 Java 的 AWS SDK 通話 ..... 34
下載 Log4J JAR ..... 35
設定 Classpath ..... 35
服務特定錯誤與警告 ..... 35
請求/回應摘要記錄 ..... 36
詳細連線記錄 ..... 37
延遲指標記錄 ..... 37
客戶端組態 ..... 38
代理組態 ..... 38
HTTP 傳輸組態 ..... 38
TCP Socket 緩衝區大小提示 ..... 39
存取控制政策 ..... 40
Amazon S3 範例 ..... 40
Amazon SQS 範例 ..... 41
Amazon SNS 範例 ..... 41
設定 DNS 名稱查詢的 JVM TTL ..... 42
如何設定 JVM TTL ..... 42
啟用 的指標 適用於 Java 的 AWS SDK ..... 44

如何啟用 Java SDK 指標產生 ..... 44
可用的指標類型 ..... 45
詳細資訊 ..... 48
程式碼範例 ..... 49
適用於 Java 的 AWS SDK 2.x ..... 49
Amazon CloudWatch 範例 ..... 49
從 CloudWatch 取得指標 ..... 50
發佈自訂指標資料 ..... 51
使用 CloudWatch 警示 ..... 53
在 CloudWatch 中使用警示動作 ..... 56
傳送事件至 CloudWatch ..... 57
Amazon DynamoDB 範例 ..... 60
使用帳戶 AWS 型端點 ..... 60
在 中使用資料表 DynamoDB ..... 61
在 中使用項目 DynamoDB ..... 68
Amazon EC2 範例 ..... 75
教學課程：啟動 EC2 執行個體 ..... 75
使用 IAM 角色授予 上 AWS 資源的存取權 Amazon EC2 ..... 80
教學課程： Amazon EC2 Spot 執行個體 ..... 85
教學課程：進階 Amazon EC2 Spot 請求管理 ..... 95
管理 Amazon EC2 執行個體 ..... 111
在 中使用彈性 IP 地址 Amazon EC2 ..... 117
使用區域和可用區域 ..... 120
使用 Amazon EC2 金鑰對 ..... 122
在 中使用安全群組 Amazon EC2 ..... 125
AWS Identity and Access Management (IAM) 範例 ..... 128
管理 IAM 存取金鑰 ..... 129
管理 IAM 使用者 ..... 133
使用 IAM 帳戶別名 ..... 136
處理 IAM 政策 ..... 139
處理 IAM 伺服器憑證 ..... 143
Amazon Lambda 範例 ..... 147
服務操作 ..... 147
Amazon Pinpoint 範例 ..... 151
在 中建立和刪除應用程式 Amazon Pinpoint ..... 151
在 中建立端點 Amazon Pinpoint ..... 153

在 中建立客群 Amazon Pinpoint ..... 155
在 中建立行銷活動 Amazon Pinpoint ..... 157
在 中更新頻道 Amazon Pinpoint ..... 158
Amazon S3 範例 ..... 159
建立、列出和刪除 Amazon S3 儲存貯體 ..... 160
在 Amazon S3 物件上執行操作 ..... 165
管理儲存貯體和物件的 Amazon S3 存取許可 ..... 170
使用 Amazon S3 儲存貯體政策管理對儲存貯體的存取 ..... 174
使用 TransferManager 進行 Amazon S3 操作 ..... 177
將 Amazon S3 儲存貯體設定為網站 ..... 189
使用 Amazon S3 用戶端加密 ..... 192
Amazon SQS 範例 ..... 198
使用 Amazon SQS 訊息佇列 ..... 198
傳送、接收和刪除 Amazon SQS 訊息 ..... 201
啟用 Amazon SQS 訊息佇列的長輪詢 ..... 203
在 中設定可見性逾時 Amazon SQS ..... 206
在 中使用無效字母佇列 Amazon SQS ..... 208
Amazon SWF 範例 ..... 210
SWF 基本概念 ..... 211
建置簡單的 Amazon SWF 應用程式 ..... 212
Lambda 任務 ..... 230
正常關閉活動和工作流程工作者 ..... 234
註冊網域 ..... 237
列出網域 ..... 237
軟體開發套件隨附的程式碼範例 ..... 238
如何取得範例 ..... 238
使用命令列建置和執行範例 ..... 238
使用 Eclipse IDE 建置和執行範例 ..... 240
安全 ..... 241
資料保護 ..... 241
強制執行最低 TLS 版本 ..... 242
如何檢查 TLS 版本 ..... 242
強制執行最低 TLS 版本 ..... 243
身分和存取權管理 ..... 243
目標對象 ..... 243
使用身分驗證 ..... 244

使用政策管理存取權 ..... 245
AWS 服務 如何使用 IAM ..... 246
對 AWS 身分和存取進行故障診斷 ..... 247
合規驗證 ..... 248
恢復能力 ..... 249
基礎設施安全性 ..... 249
S3 加密用戶端遷移 ..... 250
先決條件 ..... 250
遷移概觀 ..... 250
更新現有用戶端以讀取新格式 ..... 250
將加密和解密用戶端遷移至 V2 ..... 251
其他範例 ..... 254
OpenPGP 金鑰 ..... 255
目前金鑰 ..... 255
上一個索引鍵 ..... 261
文件歷史記錄 ..... 268

適用於 Java 的 AWS SDK 1.x 已於 2025 年 12 月 31 日end-of-support。我們建議您遷移至 AWS SDK for Java 2.x，以繼續接收新功能、可用性改善和安全性更新。

本文為英文版的機器翻譯版本，如內容有任何歧義或不一致之處，概以英文版為準。

# 開發人員指南 - 適用於 Java 的 AWS SDK 1.x

為 AWS 服務適用於 Java 的 AWS SDK提供 Java API。使用 SDK，您可以輕鬆建置可使用 Amazon S3 Amazon EC2 DynamoDB等的 Java 應用程式。我們會定期新增新服務的支援到 適用於 Java 的 AWS SDK。如需軟體開發套件每個版本隨附的支援服務及其 API 版本清單，請檢視您正在使用的版本版本備註。

## 發行的 SDK 第 2 版

前往 https://github.com/aws/aws-sdk-java-v2/ 適用於 Java 的 AWS SDK ：//。它包含許多等待的功能，例如插入 HTTP 實作的方式。若要開始使用，請參閱 適用於 Java 的 AWS SDK 2.x 開發人員指南。

## 其他文件與資源

除了本指南之外，以下是 適用於 Java 的 AWS SDK 開發人員的寶貴線上資源：

- 適用於 Java 的 AWS SDK API 參考
- Java 開發人員部落格
- Java 開發人員論壇
- GitHub:
  - 文件來源
  - 文件問題
  - 開發套件來源
  - SDK 問題
  - SDK 範例
  - 發射器頻道
- AWS 程式碼範例目錄
- @awsforjava (Twitter)
- 版本備註

# Eclipse IDE 支援

如果您使用 Eclipse IDE 開發程式碼，您可以使用 AWS Toolkit for Eclipse將 適用於 Java 的 AWS SDK 新增至現有的 Eclipse 專案，或建立新的 適用於 Java 的 AWS SDK 專案。此工具組也支援建立和上傳 Lambda 函數、啟動和監控 Amazon EC2 執行個體、管理 IAM 使用者和安全群組、 AWS CloudFormation 範本編輯器等。

如需完整文件，請參閱 AWS Toolkit for Eclipse 使用者指南。

# 開發 Android 應用程式

如果您是 Android 開發人員， 會 Amazon Web Services 發佈專為 Android 開發而打造的 SDK：Amplify Android （ 適用於 Android 的AWS Mobile SDK)。

# 檢視開發套件的修訂歷史記錄

若要檢視 的發行歷史記錄 適用於 Java 的 AWS SDK，包括每個 SDK 版本的變更和支援服務，請參閱 SDK 的版本備註。

# 建置舊版 SDK 的 Java 參考文件

適用於 Java 的 AWS SDK API 參考代表 SDK 1.x 版的最新建置。如果您使用的是舊版 1.x，您可能想要存取與您正在使用的版本相符的 SDK 參考文件。

建置文件的最簡單方法是使用 Apache 的 Maven 建置工具。如果您還沒有 Maven，請先下載並安裝 Maven，然後使用下列指示來建置參考文件。

1. 在 GitHub 上 SDK 儲存庫的版本頁面上，找到並選取您正在使用的 SDK 版本。
2. 選擇 `zip`（大多數平台，包括 Windows) 或 `tar.gz`(Linux、macOS 或 Unix) 連結，將 SDK 下載到您的電腦。
3. 將封存解壓縮至本機目錄。
4. 在命令列上，導覽至您解壓縮封存的目錄，然後輸入以下內容。

```
mvn javadoc:javadoc
```

5. 建置完成後，您可以在 `aws-java-sdk/target/site/apidocs/`目錄中找到產生的 HTML 文件。

# 開始使用

本節提供如何安裝、設定和使用 適用於 Java 的 AWS SDK的相關資訊。

主題


- 使用 的基本設定 AWS 服務
- 取得 的方式 適用於 Java 的 AWS SDK
- 使用建置工具
- 設定 AWS 臨時登入資料和 AWS 區域 以進行開發


# 使用 的基本設定 AWS 服務

## 概觀

若要成功開發 AWS 服務 使用 存取的應用程式 適用於 Java 的 AWS SDK，需要下列條件：

- 您必須能夠登入 中提供的 AWS 存取入口網站 AWS IAM Identity Center。
- 為 SDK 設定的 IAM 角色許可必須允許存取 AWS 服務 您的應用程式所需的 。與 PowerUserAccess AWS 受管政策相關聯的許可足以滿足大多數開發需求。
- 具有下列元素的開發環境：
  - 以下列方式設定的共用組態檔案：
    - `config` 檔案包含指定 的預設設定檔 AWS 區域。
    - `credentials` 檔案包含臨時登入資料做為預設設定檔的一部分。
  - Java 的適當安裝。
  - Maven 或 Gradle 等建置自動化工具。
  - 使用程式碼的文字編輯器。
  - （選用，但建議使用） IDE （整合式開發環境），例如 IntelliJ IDEA、Eclipse 或 NetBeans。

    使用 IDE 時，您也可以整合 AWS 工具組以更輕鬆地使用 AWS 服務。AWS Toolkit for IntelliJ 和 AWS Toolkit for Eclipse是兩個工具組，可用於 Java 開發。

> **Important**
>
> 此設定區段中的指示假設您或組織使用 IAM Identity Center。如果您的組織使用獨立於 IAM Identity Center 運作的外部身分提供者，請了解如何取得適用於 Java 的 SDK 暫時登入資料。請依照這些指示，將臨時登入資料新增至 `~/.aws/credentials` 檔案。
> 如果您的身分提供者自動將臨時登入資料新增至 `~/.aws/credentials` 檔案，請確定設定檔名稱為 ，`[default]`這樣您就不需要提供設定檔名稱給 SDK 或 AWS CLI。

## AWS 存取入口網站的登入功能

AWS 存取入口網站是您手動登入 IAM Identity Center 的 Web 位置。URL 的格式為 `d-xxxxxxxxxx.awsapps.com/start`或 *`your_subdomain`*`.awsapps.com/start`。

如果您不熟悉 AWS 存取入口網站，請遵循 AWS SDKs 和工具參考指南中 IAM Identity Center 身分驗證主題的步驟 1 中的帳戶存取指引。請勿遵循步驟 2，因為 適用於 Java 的 AWS SDK 1.x 不支援針對步驟 2 描述的 SDK 自動權杖重新整理和自動擷取臨時登入資料。

## 設定共用組態檔案

共用組態檔案位於開發工作站上，並包含 AWS SDKs和 AWS Command Line Interface (CLI) 使用的基本設定。共用組態檔案可以包含許多設定，但這些指示會設定使用 SDK 所需的基本元素。

### 設定共用`config`檔案

下列範例顯示共用`config`檔案的內容。

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

基於開發目的，請使用您計劃執行程式碼 AWS 區域 的最接近 。如需要在 `config` 檔案中使用的區域代碼清單，請參閱 Amazon Web Services 一般參考 指南。輸出格式`json`的設定是數個可能的值之一。

遵循本節中的指引來建立 `config` 檔案。

## 設定 SDK 的臨時登入資料

您可以透過存取 AWS 入口網站存取 AWS 帳戶 和 IAM 角色之後，請使用暫時登入資料來設定開發環境，以便 SDK 存取。

使用臨時登入資料設定本機**credentials**檔案的步驟

1. 建立共用credentials檔案。
2. 在 credentials 檔案中，貼上下列預留位置文字，直到您貼上運作中的暫時登入資料為止。

```
[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. 儲存檔案。檔案現在~/.aws/credentials應該存在於您的本機開發系統上。如果未指定特定具名設定檔，此檔案包含適用於 Java 的 SDK 使用的 【預設】 設定檔。
4. 登入 AWS 存取入口網站。
5. 請遵循手動登入資料重新整理標題下的這些指示，從 AWS 存取入口網站複製 IAM 角色登入資料。

   a. 針對連結指示中的步驟 4，選擇 IAM 角色名稱，為您的開發需求授予存取權。此角色通常具有 PowerUserAccess 或 Developer 等名稱。

   b. 針對步驟 7，選取手動將設定檔新增至您的 AWS 登入資料檔案選項，然後複製內容。

6. 將複製的登入資料貼到您的本機credentials檔案中，並移除任何貼上的設定檔名稱。您的 檔案應該類似以下內容：

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

7. 儲存credentials檔案

適用於 Java 的 SDK 會在建立服務用戶端時存取這些臨時登入資料，並用於每個請求。在步驟 5a 中選擇的 IAM 角色設定會決定臨時憑證的有效時長。最長持續時間為 12 小時。

臨時憑證過期後，請重複步驟 4 至 7。

## 安裝 Java 開發環境

適用於 Java 的 AWS SDK V1 需要 Java 7 JDK 或更新版本，並支援所有 Java LTS （長期支援） JDK 版本。如果您使用 1.12.767 版或更早版本的開發套件，則可以使用 Java 7，但如果您使用 1.12.768 版或更新版本的開發套件，則需要 Java 8。Maven 中央儲存庫會列出適用於 Java 的 SDK 最新版本。

適用於 Java 的 AWS SDK 適用於 Oracle Java SE 開發套件和 Open Java 開發套件 (OpenJDK) 的發行版本，例如 Amazon Corretto、Red Hat OpenJDK 和 Adoptium。

# 取得 的方式 適用於 Java 的 AWS SDK

## 先決條件

若要使用 適用於 Java 的 AWS SDK，您必須具有：

- 您必須能夠登入 中提供的 AWS 存取入口網站 AWS IAM Identity Center。
- Java 的適當安裝。
- 在您的本機共用`credentials`檔案中設定的臨時登入資料。

如需如何設定以使用適用於 Java 的 SDK 的說明，請參閱 the section called “基本設定”主題。

## 使用建置工具來管理適用於 Java 的 SDK 相依性 （建議）

我們建議您將 Apache Maven 或 Gradle 與專案搭配使用，以存取適用於 Java 的 SDK 所需的相依性。本節說明如何使用這些工具。

## 下載並擷取 SDK （不建議）

我們建議您使用建置工具來存取專案的 SDK，不過，您可以下載 SDK 的預先建置 jar 的最新版本。

> **Note**
>
> 如需有關如何下載和建置舊版 SDK 的資訊，請參閱安裝舊版 SDK。

1. 從 https://sdk-for-java.amazonwebservices.com/latest/aws-java-sdk.zip：// 下載開發套件。

2. 下載 SDK 後，將內容擷取至本機目錄。

開發套件包含下列目錄：

- `documentation`- 包含 API 文件（也可在 Web 上取得：適用於 Java 的 AWS SDK API 參考）。
- `lib`- 包含 SDK `.jar` 檔案。
- `samples`- 包含工作範例程式碼，示範如何使用 SDK。
- `third-party/lib`- 包含開發套件使用的第三方程式庫，例如 Apache 通用日誌記錄、AspectJ 和 Spring 架構。

若要使用 SDK，請將 `lib`和 `third-party` 目錄的完整路徑新增至建置檔案中的相依性，並將其新增至您的 Java `CLASSPATH`以執行程式碼。

## 從來源建置舊版 SDK（不建議）

只有最新版的完整 SDK 會以可下載的 jar 格式提供。不過，您可以使用 Apache Maven（開放原始碼）建置舊版 SDK。Maven 會在一個步驟中下載所有必要的相依性、建置和安裝 SDK。如需安裝說明和詳細資訊，請造訪 http://maven.apache.org/：//。

1. 前往開發套件的 GitHub 頁面，網址為： 適用於 Java 的 AWS SDK (GitHub)。
2. 選擇與您想要的 SDK 版本編號對應的標籤。例如 `1.6.10`。
3. 按一下下載 ZIP 按鈕，下載您選取的 SDK 版本。
4. 將檔案解壓縮至開發系統上的目錄。在許多系統上，您可以使用圖形檔案管理員來執行此操作，或在終端機視窗中使用 `unzip` 公用程式。
5. 在終端機視窗中，導覽至您解壓縮 SDK 來源的目錄。
6. 使用下列命令建置並安裝 SDK（需要 Maven）：

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

   產生的 `.jar` 檔案內建至 `target` 目錄。
7. （選用）使用以下命令建置 API 參考文件：

   ```
   mvn javadoc:javadoc
   ```

   文件內建於 `target/site/apidocs/`目錄中。

# 使用建置工具

使用建置工具有助於管理 Java 專案的開發。有數種建置工具可供使用，但我們示範如何使用兩種熱門的建置工具來啟動和執行：Maven 和 Gradle。本主題說明如何使用這些建置工具來管理專案所需的適用於 Java 的 SDK 相依性。

主題


- 使用軟體開發套件搭配 Apache Maven
- 使用軟體開發套件搭配 Gradle


## 使用軟體開發套件搭配 Apache Maven

您可以使用 Apache Maven 來設定和建置 適用於 Java 的 AWS SDK 專案，或建置 SDK 本身。

> **Note**
>
> 您必須已安裝 Maven 才能使用本主題中的指導方針。如果尚未安裝，請造訪 http://maven.apache.org/ 進行下載和安裝。

### 建立新的 Maven 套件

若要建立基本 Maven 套件，請開啟終端機（命令列）視窗並執行：

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

將 org.example.basicapp 取代為應用程式的完整套件命名空間，並將 myapp 取代為專案的名稱（這會成為專案的目錄名稱）。

根據預設， 會使用 quickstart 原型為您建立專案範本，這是許多專案的理想起點。有更多可用的原型；請造訪 Maven 原型頁面以取得 封裝的原型清單。您可以將 `-DarchetypeArtifactId` 引數新增到 `archetype:generate` 命令，選擇使用特定原型。例如：

```
mvn archetype:generate \
  -DarchetypeGroupId=org.apache.maven.archetypes \
```

```
  -DarchetypeArtifactId=maven-archetype-webapp \
  -DgroupId=org.example.webapp \
  -DartifactId=mywebapp
```

> **Note**
>
> Maven 入門指南提供建立和設定專案的更多相關資訊。

## 將 SDK 設定為 Maven 相依性

若要在專案 適用於 Java 的 AWS SDK 中使用 ，您需要在專案的 pom.xml 檔案中將其宣告為相依性。從 1.9.0 版開始，您可以匯入個別元件或整個 SDK。

指定個別 SDK 模組

若要選取個別 SDK 模組，請使用 Maven 的物料 適用於 Java 的 AWS SDK 清單 (BOM)，這將確保您指定的模組使用相同版本的 SDK，且彼此相容。

若要使用 BOM，請將 <dependencyManagement>區段新增至應用程式的 pom.xml 檔案，新增 aws-java-sdk-bom做為相依性，並指定您要使用的 SDK 版本：

```
<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>
```

若要檢視 Maven Central 上提供的 適用於 Java 的 AWS SDK BOM 最新版本，請造訪：https：//https://mvnrepository.com/artifact/com.amazonaws/aws-java-sdk-bom。您也可以使用此頁面來查看哪些模組 （相依性） 是由 BOM 管理，您可以包含在專案pom.xml檔案的 <dependencies>區段中。

您現在可以從應用程式中使用的 SDK 中選取個別模組。由於您已經在 BOM 中宣告開發套件版本，所以不需要指定每個元件的版本編號。

```
<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>
```

您也可以參閱 AWS 程式碼範例目錄 ，以了解要用於指定 的相依性 AWS 服務。請參閱特定服務範例下的 POM 檔案。例如，如果您對 AWS S3 服務的相依性感興趣，請參閱 GitHub 上的完整範例。（查看 /java/example_code/s3 下的 pom)。

匯入所有 SDK 模組

如果您想要提取整個開發套件做為相依性，請不要使用 BOM 方法，只需`pom.xml`像這樣在 中宣告：

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

## 建立專案

設定專案後，您可以使用 Maven 的`package`命令建置專案：

```
mvn package
```

這將在 `target`目錄中建立您的 `0jar` 檔案。

## 使用 Maven 建置 SDK

您可以使用 Apache Maven 從來源建置 SDK。若要這樣做，請從 GitHub 下載 SDK 程式碼，在本機解壓縮，然後執行下列 Maven 命令：

```
mvn clean install
```

# 使用軟體開發套件搭配 Gradle

若要管理 Gradle 專案的 SDK 相依性，請將 的 Maven BOM 匯入 適用於 Java 的 AWS SDK 應用程式的 `build.gradle` 檔案。

> **Note**
>
> 在下列範例中，將建置檔案中的 *`1.12.529`* 取代為有效的 版本 適用於 Java 的 AWS SDK。在 Maven 中央儲存庫中尋找最新版本。

## Gradle 4.6 或更新版本的專案設定

自 Gradle 4.6 以來，您可以使用 Gradle 改善的 POM 支援功能，透過宣告對 BOM 的相依性來匯入物料清單 (BOM) 檔案。

1. 如果您使用 Gradle 5.0 或更新版本，請跳至步驟 2。否則，請在 `settings.gradle` 檔案中啟用 IMPROVED_POM_SUPPORT 功能。

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

2. 將 BOM 新增至應用程式`build.gradle`檔案的相依性區段。

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

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

3. 指定要在 dependencies (相依性) 區段中使用的開發套件模組。例如，以下包含 Amazon Simple Storage Service () 的相依性Amazon S3。

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

Gradle 會使用 BOM 的資訊，自動解析您開發套件相依性的正確版本。

以下是包含其相依性的完整`build.gradle`檔案範例 Amazon S3。

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

apply plugin: 'java'

sourceCompatibility = 1.8

repositories {
  mavenCentral()
}

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

> **Note**
>
> 在上一個範例中，將 的相依性取代 Amazon S3 為您將在專案中使用的 AWS 服務的相依性。由 適用於 Java 的 AWS SDK BOM 管理的模組 （ 相依性 ） 會列在 Maven 中央儲存庫上。

## 4.6 之前 Gradle 版本的專案設定

4.6 之前的 Gradle 版本缺少原生 BOM 支援。若要管理專案的 適用於 Java 的 AWS SDK 相依性，請使用 Spring for Gradle 的相依性管理外掛程式來匯入 SDK 的 Maven BOM。

1. 將相依性管理外掛程式新增至應用程式的 `build.gradle` 檔案。

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

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

2. 新增 BOM 到檔案的 dependencyManagement 區段。

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

3. 指定您將在相依性區段中使用的 SDK 模組。例如，以下內容包含 Amazon S3的相依性。

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

Gradle 會使用 BOM 的資訊，自動解析您開發套件相依性的正確版本。

以下是包含其相依性的完整`build.gradle`檔案範例 Amazon S3。

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

apply plugin: 'java'

sourceCompatibility = 1.8

repositories {
  mavenCentral()
}

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

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

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

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

> **Note**
>
> 在上述範例中，將 的相依性取代 Amazon S3 為您將在專案中使用的 AWS 服務相依性。由 適用於 Java 的 AWS SDK BOM 管理的模組（相依性）會列在 Maven 中央儲存庫上。

如需使用 BOM 指定 SDK 相依性的詳細資訊，請參閱搭配 Apache Maven 使用 SDK。

# 設定 AWS 臨時登入資料和 AWS 區域 以進行開發

若要使用 連線到任何支援的 服務 適用於 Java 的 AWS SDK，您必須提供 AWS 暫時登入資料。 AWS SDKs 和 CLIs 使用提供者鏈在許多不同的位置尋找 AWS 臨時登入資料，包括系統/使用者環境變數和本機 AWS 組態檔案。

本主題提供使用 為本機應用程式開發設定 AWS 臨時登入資料的基本資訊 適用於 Java 的 AWS SDK。如果您需要設定登入資料以在 EC2 執行個體內使用，或者您使用 Eclipse IDE 進行開發，請改為參考下列主題：

- 使用 EC2 執行個體時，請建立 IAM 角色，然後讓 EC2 執行個體存取該角色，如使用 IAM 角色授予 AWS 資源的存取權 Amazon EC2中所示。
- 使用 在 Eclipse 中設定 AWS 登入資料AWS Toolkit for Eclipse。如需詳細資訊，請參閱AWS Toolkit for Eclipse 《使用者指南》中的設定 AWS 登入資料。

## 設定臨時登入資料

您可以透過 適用於 Java 的 AWS SDK 多種方式設定 的臨時登入資料，但以下是建議的方法：

- 在本機系統的登入資料設定檔中設定臨時 AWS 登入資料，位於：

- `~/.aws/credentials`在 Linux、macOS 或 Unix
- Windows 上的 `C:\Users\USERNAME\.aws\credentials`

如需如何取得臨時登入資料的說明，請參閱本指南the section called “設定 SDK 的臨時登入資料”中的 。

- 設定 `AWS_ACCESS_KEY_IDAWS_SECRET_ACCESS_KEY`、 和 `AWS_SESSION_TOKEN`環境變數。

  若要在 Linux、macOS 或 Unix 上設定這些變數，請使用 ：

  ```
  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
  ```

  若要在 Windows 上設定這些變數，請使用 ：

  ```
  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
  ```

- 對於 EC2 執行個體，請指定 IAM 角色然後提供您的 EC2 執行個體存取權給該角色。如需此運作方式的詳細討論，請參閱《Linux 執行個體 Amazon EC2 使用者指南》中的適用於 的 IAM 角色 Amazon EC2。

使用這些方法之一設定 AWS 臨時登入資料後， 將使用預設 適用於 Java 的 AWS SDK 登入資料提供者鏈結自動載入臨時登入資料。如需在 Java 應用程式中使用 AWS 登入資料的詳細資訊，請參閱使用 AWS 登入資料。

## 重新整理 IMDS 登入資料

無論憑證過期時間為何， 支援每 1 分鐘在背景 適用於 Java 的 AWS SDK 中選擇加入重新整理 IMDS 憑證。這可讓您更頻繁地重新整理登入資料，並降低未達到 IMDS 影響感知 AWS 可用性的機會。

```
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();
```

## 設定 AWS 區域

您應該設定用於使用 AWS 區域 存取 AWS 服務的預設值 適用於 Java 的 AWS SDK。為了獲得最佳的網路效能，請選擇地理位置上靠近您 (或您的客戶) 的區域。如需每個服務的區域清單，請參閱《 Amazon Web Services 一般參考》中的區域和端點。

> **Note**
>
> 如果您未選取區域，則預設會使用 us-east-1。

您可以使用類似的技術來設定登入資料，以設定您的預設 AWS 區域：

- 在本機系統的組態檔案中設定 AWS 區域 AWS ，位於：
  - Linux、macOS 或 Unix 上的 ~/.aws/config
  - Windows 上的 C：\Users\USERNAME\.aws\config

  此檔案應該包含下列格式的行：

  +

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

  +

  將您想要的 AWS 區域 （ 例如，「us-east-1」) 替換為 your_aws_region。
- 設定 AWS_REGION 環境變數。

  在 Linux、macOS 或 Unix 上，使用 ：

```
export AWS_REGION=your_aws_region
```

在 Windows 上，使用 ：

```
set AWS_REGION=your_aws_region
```

其中 your_aws_region 是所需的 AWS 區域 名稱。

# 使用 適用於 Java 的 AWS SDK

本節提供使用 進行程式設計的重要一般資訊 適用於 Java 的 AWS SDK ，適用於您可能搭配 SDK 使用的所有 服務。

如需服務特定的程式設計資訊和範例 （ 例如 Amazon S3 Amazon SWF等 ） Amazon EC2，請參閱適用於 Java 的 AWS SDK 程式碼範例。

主題


- 使用 AWS 開發的最佳實務 適用於 Java 的 AWS SDK
- 建立服務用戶端
- 提供臨時登入資料給 適用於 Java 的 AWS SDK
- AWS 區域 選擇
- 例外狀況處理
- 非同步程式設計
- 記錄 適用於 Java 的 AWS SDK 通話
- 客戶端組態
- 存取控制政策
- 設定 DNS 名稱查詢的 JVM TTL
- 啟用 的指標 適用於 Java 的 AWS SDK


# 使用 AWS 開發的最佳實務 適用於 Java 的 AWS SDK

下列最佳實務可協助您在使用 開發 AWS 應用程式時避免問題或麻煩 適用於 Java 的 AWS SDK。我們已依服務組織最佳實務。

## S3

### 避免 ResetExceptions

當您 Amazon S3 使用串流 （ 透過 `AmazonS3`用戶端或 `TransferManager`) 將物件上傳至 時，您可能會遇到網路連線或逾時問題。根據預設， 適用於 Java 的 AWS SDK 嘗試重試失敗的傳輸，方法是在傳輸開始之前標記輸入串流，然後在重試之前重設它。

如果串流不支援標記和重設，當發生暫時性故障並啟用重試時，開發套件會擲回 ResetException。

最佳實務

我們建議您使用支援標記和重設操作的串流。

避免 ResetException 的最可靠方法是使用 檔案或 FileInputStream 來提供資料，而 適用於 Java 的 AWS SDK 可以處理這些資料，而不會受限於標記和重設限制。

如果串流不是 FileInputStream，但確實支援標記和重設，您可以使用 RequestClientOptions `setReadLimit`的方法設定標記限制。其預設值為 128 KB。將讀取限制值設定為大於串流大小的一個位元組，可以可靠地避免 ResetException。

例如，如果串流的預期大小上限為 100，000 個位元組，請將讀取限制設定為 100，001 (100，000 + 1) 個位元組。標記和重設一律適用於 100，000 個位元組或更少的位元組。請注意，這可能會導致某些串流緩衝記憶體中的位元組數。

# 建立服務用戶端

若要向 提出請求 Amazon Web Services，您必須先建立服務用戶端物件。建議的方式是使用 服務用戶端建置器。

每個 AWS 服務 都有一個服務介面，其中包含服務 API 中每個動作的方法。例如，DynamoDB 的服務界面名為 AmazonDynamoDBClient。每個服務界面都有對應的用戶端建置器，可用來建構服務界面的實作。的用戶端建置器類別 DynamoDB 名為 AmazonDynamoDBClientBuilder。

## 取得用戶端建置器

若要取得用戶端建置器的執行個體，請使用靜態原廠方法 `standard`，如下列範例所示。

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

擁有建置器後，您可以在建置器 API 中使用許多流暢的設定器來自訂用戶端的屬性。例如，您可以設定自訂區域和自訂登入資料提供者，如下所示。

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

> **Note**
>
> 流暢withXXX的方法會傳回 builder 物件，以便您可以鏈結方法呼叫以方便使用，並取得更易讀的程式碼。設定您要的屬性後，您可以呼叫 build 方法來建立用戶端。建立用戶端後，用戶端是不可變的，任何對 setRegion或 的呼叫setEndpoint都會失敗。

建置器可以建立具有相同組態的多個用戶端。當您撰寫應用程式時，請注意建置器是可變的，且不安全執行緒。

下列程式碼使用 建置器做為用戶端執行個體的工廠。

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

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

建置器也會公開 ClientConfiguration 和 RequestMetricCollector 的流暢設定器，以及 RequestHandler2 的自訂清單。

以下是覆寫所有可設定屬性的完整範例。

```
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();
```

## 建立非同步用戶端

每個服務 適用於 Java 的 AWS SDK 都有非同步 （或非同步） 用戶端 ( 除外 Amazon S3)，每個服務都有對應的非同步用戶端建置器。

### 使用預設 ExecutorService 建立非同步 DynamoDB 用戶端

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

除了同步 （或同步） 用戶端建置器支援的組態選項之外，非同步用戶端可讓您設定自訂 ExecutorFactory 來變更非同步用戶端使用的 ExecutorService 。 ExecutorFactory 是功能介面，因此與 Java 8 lambda 表達式和方法參考互通。

### 使用自訂執行器建立非同步用戶端

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

## 使用 DefaultClient

同步和非同步用戶端建置器都有另一個名為 的原廠方法defaultClient。此方法會使用預設提供者鏈來載入登入資料和 ，以預設組態建立服務用戶端 AWS 區域。如果無法從應用程式執行的環境判斷登入資料或區域，則對 defaultClient 的呼叫失敗。如需如何判斷AWS 登入資料和區域的詳細資訊，請參閱使用登入資料和AWS 區域 選擇。

### 建立預設服務用戶端

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

## 用戶端生命週期

開發套件中的服務用戶端是執行緒安全的，為了獲得最佳效能，您應該將它們視為長期物件。每個用戶端都有自己的連線集區資源。當不再需要用戶端以避免資源洩漏時，請明確將其關閉。

若要明確關閉用戶端，請呼叫 shutdown方法。呼叫 後shutdown，會釋出所有用戶端資源，且用戶端無法使用。

### 關閉用戶端

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

```
// Client is now unusable
```

# 提供臨時登入資料給 適用於 Java 的 AWS SDK

若要向 提出請求 Amazon Web Services，您必須提供 AWS 暫時性登入資料 適用於 Java 的 AWS SDK ，讓 在呼叫 服務時使用。您可採用以下方式：

- 使用預設登入資料供應者鏈結 (建議)。
- 使用特定的登入資料供應者或供應者鏈結 (或建立您自己的項目)。
- 在程式碼中自行提供臨時登入資料。

## 使用預設登入資料供應商鏈結

當您初始化新的服務用戶端而不提供任何引數時， 適用於 Java 的 AWS SDK 會嘗試使用 DefaultAWSCredentialsProviderChain 類別實作的預設登入資料提供者鏈結來尋找臨時登入資料。預設登入資料供應者鏈結會依以下順序尋找登入資料：

1. 環境變數 -AWS_ACCESS_KEY_ID、 AWS_SECRET_KEY或 AWS_SECRET_ACCESS_KEY，以及 AWS_SESSION_TOKEN。 適用於 Java 的 AWS SDK 使用 EnvironmentVariableCredentialsProvider 類別載入這些登入資料。
2. Java 系統屬性-aws.accessKeyId、 aws.secretKey ( 但不是 aws.secretAccessKey) 和 aws.sessionToken。 適用於 Java 的 AWS SDK 使用 SystemPropertiesCredentialsProvider 載入這些登入資料。
3. 來自環境或容器的 Web Identity Token 登入資料。
4. 預設登入資料設定檔檔案 - 通常位於 ~/.aws/credentials ( 每個平台的位置可能不同 ) ，並由許多 AWS SDKs 和 共用 AWS CLI。 適用於 Java 的 AWS SDK 使用 ProfileCredentialsProvider 載入這些登入資料。

   您可以使用 提供的aws configure命令來建立登入資料檔案 AWS CLI，或使用文字編輯器編輯檔案來建立登入資料檔案。如需登入資料檔案格式的資訊，請參閱AWS 登入資料檔案格式。
5. 如果已AWS_CONTAINER_CREDENTIALS_RELATIVE_URI設定環境變數，則從 Amazon ECS 載入的 Amazon ECS 容器憑證。 適用於 Java 的 AWS SDK 使用 ContainerCredentialsProvider 載入這些登入資料。您可以為此值指定 IP 地址。
6. 執行個體描述檔登入資料 - 用於 EC2 執行個體，並透過 Amazon EC2 中繼資料服務傳遞。 適用於 Java 的 AWS SDK 使用 InstanceProfileCredentialsProvider 載入這些登入資料。您可以為此值指定 IP 地址。

> **Note**
>
> 只有在AWS_CONTAINER_CREDENTIALS_RELATIVE_URI未設定 時，才會使用執行個體描述檔登入資料。如需詳細資訊，請參閱 EC2ContainerCredentialsProviderWrapper。

### 設定暫時登入資料

若要能夠使用 AWS 臨時登入資料，它們必須至少設定在上述其中一個位置。如需設定登入資料的相關資訊，請參閱下列主題：

- 若要在環境或預設憑證設定檔檔案中指定憑證，請參閱 the section called “設定臨時登入資料” 。
- 若要設定 Java 系統屬性，請參閱官方 Java 教學課程網站的系統屬性教學課程。
- 若要在 EC2 執行個體中設定和使用執行個體描述檔登入資料，請參閱使用 IAM 角色授予 AWS 資源的存取權 Amazon EC2。

### 設定替代登入資料設定檔

根據預設， 適用於 Java 的 AWS SDK 會使用預設的設定檔，但有方法可以自訂從登入資料檔案取得的設定檔。

您可以使用 AWS 設定檔環境變數來變更 SDK 載入的設定檔。

例如，在 Linux、macOS 或 Unix 上執行下列命令，將設定檔變更為 myProfile。

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

在 Windows 上，您會使用下列項目。

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

設定AWS_PROFILE環境變數會影響所有官方支援的 AWS SDKs和工具（包括 AWS CLI 和 AWS Tools for Windows PowerShell) 的登入資料載入。若要只變更 Java 應用程式的設定檔，您可以aws.profile改用 系統屬性。

> **Note**
>
> 環境變數會優先於系統屬性。

### 設定替代登入資料檔案位置

會自動從預設登入資料檔案位置 適用於 Java 的 AWS SDK 載入 AWS 臨時登入資料。不過，您也可以設定 AWS_CREDENTIAL_PROFILES_FILE 環境變數搭配登入資料檔案的完整路徑，來指定位置。

您可以使用此功能暫時變更 適用於 Java 的 AWS SDK 尋找登入資料檔案的位置 （例如，使用命令列設定此變數）。或者，您也可以在使用者或系統環境中設定環境變數，來為該使用者進行變更或是進行全系統變更。

若要覆寫預設登入資料檔案位置

- 將AWS_CREDENTIAL_PROFILES_FILE環境變數設定為登入 AWS 資料檔案的位置。
  - 在 Linux、macOS 或 Unix 上，使用：

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

  - 在 Windows 上，使用：

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

## `Credentials` 檔案格式

遵循本指南的基本設定中的指示，您的登入資料檔案應具有下列基本格式。

```
[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>
```

設定檔名稱在方括號中指定 (例如 [default])，後接該設定檔的可設定欄位，以索引鍵值組形式表示。您可以在 credentials 檔案中有多個設定檔，可使用 新增或編輯aws configure --profile *PROFILE_NAME* ，以選取要設定的設定檔。

您可以指定其他欄位，例如 metadata_service_timeout、 和 metadata_service_num_attempts。這些無法透過 CLI 設定 - 如果您想要使用檔案，您必須手動編輯檔案。如需組態檔案及其可用欄位的詳細資訊，請參閱 AWS Command Line Interface 《 使用者指南》中的設定 AWS Command Line Interface 。

### 載入登入資料

設定臨時登入資料後，軟體開發套件會使用預設登入資料提供者鏈結載入這些登入資料。

若要這樣做，您可以執行個體化 AWS 服務 用戶端，而不明確提供憑證給建置器，如下所示。

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

## 指定登入資料提供者或提供者鏈

您可以使用用戶端建置器，指定與預設登入資料供應者鏈結不同的登入資料供應者。

您可以將登入資料提供者或提供者鏈結的執行個體提供給用戶端建置器，該建置器會將 AWSCredentialsProvider 介面做為輸入。下列範例說明如何具體使用環境登入資料。

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

如需 適用於 Java 的 AWS SDK所提供登入資料提供者和提供者鏈的完整清單，請參閱 AWSCredentialsProvider 中的所有已知實作類別。

> **Note**
>
> 您可以使用此技術，透過使用實作AWSCredentialsProvider界面的登入資料提供者，或將 AWSCredentialsProviderChain 類別子類別，來提供您建立的登入資料提供者或提供者鏈。

## 明確指定臨時登入資料

如果預設登入資料鏈或特定或自訂提供者或提供者鏈不適用於您的程式碼，您可以設定明確提供的登入資料。如果您已使用 擷取臨時登入資料 AWS STS，請使用此方法指定登入資料以進行 AWS 存取。

1. 執行個體化 BasicSessionCredentials 類別，並提供 AWS 開發套件用於連線的存取金鑰、 AWS 私密金鑰和 AWS 工作階段字符。
2. 使用 `AWSCredentials` 物件建立 AWSStaticCredentialsProvider。
3. 使用 `AWSStaticCredentialsProvider` 設定用戶端建置器並建置用戶端。

下列是 範例。

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

## 詳細資訊

- 註冊 AWS 和建立 IAM 使用者
- 設定用於開發的 AWS 登入資料和區域
- 使用 IAM 角色授予 上 AWS 資源的存取權 Amazon EC2

# AWS 區域 選擇

區域可讓您存取 AWS 實際位於特定地理區域的服務。這對於備援以及讓您的資料和應用程式在靠近您和您的使用者存取位置附近執行，都很有用。

## 檢查區域中的服務可用性

若要查看特定 AWS 服務 是否可在區域中使用，請在您想要使用的區域上使用 `isServiceSupported`方法。

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

請參閱可指定區域的區域類別文件，並使用 服務的端點字首進行查詢。每個服務的端點字首都會在服務界面中定義。例如， DynamoDB 端點字首是在 AmazonDynamoDB 中定義。

## 選擇區域

從 1.4 版開始 適用於 Java 的 AWS SDK，您可以指定區域名稱，開發套件會自動為您選擇適當的端點。若要自行選擇端點，請參閱選擇特定端點。

若要明確設定區域，建議您使用區域列舉。這是所有公開可用區域的列舉。若要從列舉建立具有區域的用戶端，請使用下列程式碼。

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

如果您嘗試使用的區域不在Regions列舉中，您可以使用代表區域名稱的字串來設定區域。

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

> **Note**
>
> 使用建置器建置用戶端後，這是不可變的，而且區域無法變更。如果您 AWS 區域 針對相同的服務使用多個 ，您應該建立多個用戶端，每個區域一個。

## 選擇特定端點

每個 AWS 用戶端都可以設定為在建立用戶端時呼叫 withEndpointConfiguration方法，以使用區域內的特定端點。

例如，若要設定 Amazon S3 用戶端使用歐洲 （ 愛爾蘭 ） 區域，請使用下列程式碼。

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

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

如需所有 AWS 服務的目前區域清單及其對應端點，請參閱區域和端點。

## 從環境自動判斷區域

> **Important**
>
> 本節僅適用於使用用戶端建置器存取 AWS services。使用用戶端建構函數 AWS 建立的用戶端不會自動從環境判斷區域，而是使用預設 SDK 區域 (USEast1)。

在 Amazon EC2 或 Lambda 上執行時，您可能想要將用戶端設定為使用與程式碼執行所在的相同區域。這會讓您的程式碼與其執行環境分離，也更容易將應用程式部署到多個區域，以降低延遲或提供備援。

您必須使用用戶端建置器，讓 SDK 自動偵測您程式碼執行所在的區域。

若要使用預設的登入資料/區域供應者鏈結來從環境判斷區域，請使用用戶端建置器的 `defaultClient` 方法。

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

這與使用 `standard`後接 相同`build`。

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

如果您沒有使用 `withRegion`方法明確設定區域，開發套件會諮詢預設區域提供者鏈結，以嘗試並判斷要使用的區域。

### 預設區域供應者鏈結

以下是區域查詢程序：

1. 在建置器本身上使用 `withRegion`或 `setRegion` 設定的任何明確區域優先於任何其他區域。
2. 檢查 `AWS_REGION` 環境變數。如果有設定，會使用該區域來設定用戶端。

> **Note**
>
> 此環境變數是由 Lambda 容器設定。

3. SDK 會檢查 AWS 共用組態檔案 （通常位於 `~/.aws/config`)。如果 region 屬性存在，開發套件會予以使用。
   - `AWS_CONFIG_FILE` 環境變數可用於自訂共用組態檔的位置。
   - `AWS_PROFILE` 環境變數或`aws.profile`系統屬性可用來自訂 SDK 載入的設定檔。
4. SDK 會嘗試使用 Amazon EC2 執行個體中繼資料服務來判斷目前執行中 Amazon EC2 執行個體的區域。
5. 如果開發套件在這個時候仍找不到區域，用戶端建立會失敗，並出現例外狀況。

開發 AWS 應用程式時，常見的方法是使用共用組態檔案 （如使用預設登入資料提供者鏈中所述） 來設定本機開發的區域，並在 AWS 基礎設施上執行時依賴預設區域提供者鏈來判斷區域。這可大幅簡化用戶端建立並讓您的應用程式保持可攜式。

# 例外狀況處理

了解 適用於 Java 的 AWS SDK 拋出例外狀況的方式和時間，對於使用 SDK 建置高品質應用程式至關重要。以下章節說明開發套件會擲回的不同例外狀況案例，以及如何正確處理這些狀況。

## 為什麼使用未檢查的例外狀況？

由於這些原因， 適用於 Java 的 AWS SDK 使用執行時間 （或未核取） 例外狀況，而不是核取的例外狀況：

- 為了讓開發人員能夠更精確的控制他們想處理的錯誤，而非強制他們處理不在乎的例外情況 (因而使得程式碼過於冗長)
- 為了避免大型應用程式中已檢查例外狀況的固有擴展性問題

一般而言，已檢查例外狀況在小規模上可運作良好，但會隨著應用程式增長且更複雜而變得棘手。

如需使用已勾選和未勾選例外狀況的詳細資訊，請參閱：

- 未檢查的例外狀況 - 爭用
- 已檢查例外狀況的問題

- Java 檢查的例外狀況是錯誤 （以下是我想要處理的問題）

## AmazonServiceException (和子類別)

AmazonServiceException 是使用 時最常見的例外狀況 適用於 Java 的 AWS SDK。此例外狀況代表來自 的錯誤回應 AWS 服務。例如，如果您嘗試終止不存在的 Amazon EC2 執行個體，EC2 會傳回錯誤回應，而該錯誤回應的所有詳細資訊都會包含在AmazonServiceException擲出的 中。在某些情況下，會擲回 AmazonServiceException 的子類別以讓開發人員透過 catch 區塊更精確的控制錯誤情況處理。

當您遇到 時AmazonServiceException，您知道您的請求已成功傳送到 ， AWS 服務 但無法成功處理。這可能是因為請求參數中的錯誤，或因為服務端的問題。

AmazonServiceException 為您提供資訊，例如：

- 傳回 HTTP 狀態碼
- 傳回的 AWS 錯誤碼
- 來自該服務的詳細錯誤訊息
- AWS 失敗請求的請求 ID

AmazonServiceException 也包含有關失敗請求是發起人的錯誤 （具有非法值的請求） 還是 AWS 服務錯誤 （內部服務錯誤） 的資訊。

## AmazonClientException

AmazonClientException 指出在嘗試傳送請求至 或嘗試剖析回應時，Java AWS 用戶端程式碼內發生問題 AWS。AmazonClientException 通常比 更嚴重AmazonServiceException，並指出導致用戶端無法對 AWS 服務進行服務呼叫的主要問題。例如，當您嘗試呼叫其中一個用戶端上的 操作時，AmazonClientException如果沒有可用的網路連線， 會 適用於 Java 的 AWS SDK 擲回 。

# 非同步程式設計

您可以使用同步或非同步方法來呼叫 AWS 服務上的操作。同步方法會封鎖您的執行緒執行，直到用戶端收到服務的回應。非同步方法會立即傳回，將控制權回歸給呼叫端執行緒，無需等待回應。

由於非同步方法會在有可用回應之前傳回，您需要一個方法在回應準備好時取得回應。 適用於 Java 的 AWS SDK 提供兩種方式：未來物件和回呼方法。

# Java 未來

中的非同步方法會 適用於 Java 的 AWS SDK 傳回未來物件，其中包含未來非同步操作的結果。

呼叫 FutureisDone()方法，以查看服務是否已提供回應物件。當回應準備就緒時，您可以透過呼叫 Futureget()方法取得回應物件。您可以使用此機制定期輪詢非同步操作的結果，同時您的應用程式會繼續處理其他項目。

以下是呼叫 Lambda 函數的非同步操作範例，接收Future可以容納 InvokeResult 物件的 。只有在isDone()為 之後，才會擷取InvokeResult物件true。

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

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

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

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

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

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

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

## 非同步回呼

除了使用 Java Future 物件來監控非同步請求的狀態之外， SDK 還可讓您實作使用AsyncHandler 介面的類別。 AsyncHandler提供兩種呼叫方法，取決於請求的完成方式： onSuccess和 onError。

回呼界面方法的主要優點是讓您無需輪詢Future物件，即可了解請求何時完成。相反地，您的程式碼可以立即開始下一個活動，並依賴軟體開發套件在正確的時間呼叫您的處理常式。

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

## 最佳實務

### 回呼執行

您的 實作AsyncHandler會在非同步用戶端擁有的執行緒集區中執行。快速執行的簡短程式碼最適合您的AsyncHandler實作。處理常式方法內的長時間執行或封鎖程式碼可能會導致非同步用戶端使用的執行緒集區發生爭用，並可防止用戶端執行請求。如果您有需要從回呼開始的長期執行任務，請讓回呼在新的執行緒或由應用程式管理的執行緒集區中執行其任務。

### 執行緒集區組態

中的非同步用戶端 適用於 Java 的 AWS SDK 提供預設執行緒集區，應適用於大多數應用程式。您可以實作自訂 ExecutorService，並將其傳遞給 適用於 Java 的 AWS SDK 非同步用戶端，以進一步控制執行緒集區的管理方式。

例如，您可以提供使用自訂 ThreadFactory 來控制集區中執行緒命名方式的ExecutorService實作，或記錄有關執行緒用量的其他資訊。

### 非同步存取

開發套件中的 TransferManager 類別提供使用 的非同步支援 Amazon S3。 TransferManager管理非同步上傳和下載、提供有關傳輸的詳細進度報告，以及支援對不同事件的回呼。

# 記錄 適用於 Java 的 AWS SDK 通話

使用 Apache Commons Logging 進行 適用於 Java 的 AWS SDK 檢測，Apache Commons Logging 是一種抽象層，可在執行時間使用多個記錄系統中的任何一個。

支援的記錄系統包括 Java Logging Framework 和 Apache Log4j 等等。本主題說明如何使用 Log4j。您可以使用開發套件的記錄功能，而無需更改您的應用程式程式碼。

若要進一步了解 Log4j，請參閱 Apache 網站。

> **Note**
>
> 本主題著重於 Log4j 1.x。Log4j2 不直接支援 Apache Commons Logging，但提供轉接器，使用 Apache Commons Logging 介面自動將記錄呼叫導向 Log4j2。如需詳細資訊，請參閱 Log4j2 文件中的 Commons Logging Bridge。

## 下載 Log4J JAR

若要搭配 SDK 使用 Log4j，您需要從 Apache 網站下載 Log4j JAR。開發套件不包含 JAR。將 JAR 檔案複製到 classpath 上的位置。

Log4j 使用組態檔案 log4j.properties。範例組態檔案如下所示。將此組態檔案複製到 classpath 上的目錄。Log4j JAR 和 log4j.properties 檔案不必位於相同的目錄中。

log4j.properties 組態檔案會指定屬性，例如記錄層級、記錄輸出的傳送位置（例如檔案或主控台），以及輸出的格式。記錄層級是記錄器所產生輸出的精細度。Log4j 支援多個記錄階層的概念。每個階層的記錄層級都是獨立設定。以下兩個記錄階層可用於 適用於 Java 的 AWS SDK：

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

## 設定 Classpath

Log4j JAR 和 log4j.properties 檔案都必須位於 classpath 上。如果您使用的是 Apache Ant，請在 Ant 檔案中的 `path`元素中設定 classpath。下列範例顯示來自 Ant 檔案的路徑元素， Amazon S3 適用於軟體開發套件隨附的範例。

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

如果您是使用 Eclipse IDE，可以開啟選單並導覽到 Project (專案) | Properties (屬性) | Java Build Path (Java 建置路徑) 來設定 classpath。

## 服務特定錯誤與警告

我們建議您一律將 "com.amazonaws" 記錄器階層設定為 "WARN"，以從用戶端程式庫擷取任何重要訊息。例如，如果 Amazon S3 用戶端偵測到您的應用程式未正確關閉 `InputStream`並可能洩漏資源，S3 用戶端會透過警告訊息向日誌回報。這也可確保在用戶端處理請求或回應發生任何問題時，會記錄訊息。

下列 log4j.properties 檔案會將 `rootLogger` 設定為 WARN，這會導致包含 "com.amazonaws" 階層中所有記錄器的警告和錯誤訊息。或者，您可以明確地將 com.amazonaws 記錄器設定為 WARN。

```
log4j.rootLogger=WARN, A1
log4j.appender.A1=org.apache.log4j.ConsoleAppender
log4j.appender.A1.layout=org.apache.log4j.PatternLayout
log4j.appender.A1.layout.ConversionPattern=%d [%t] %-5p %c -  %m%n
# Or you can explicitly enable WARN and ERROR messages for the {AWS} Java clients
log4j.logger.com.amazonaws=WARN
```

## 請求/回應摘要記錄

如果您遇到 如何處理 AWS 請求的問題，對 的每個請求都會 AWS 服務 產生唯一的 AWS 服務 請求 ID。對於任何失敗的服務呼叫，可以透過 SDK 中的例外物件以程式設計方式存取 AWS 請求 IDs，也可以透過 "com.amazonaws.request" 記錄器中的 DEBUG 日誌層級進行報告。

下列 log4j.properties 檔案可啟用請求和回應的摘要，包括 AWS 請求 IDs。

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

以下為日誌輸出的範例。

```
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
```

## 詳細連線記錄

在某些情況下，查看 適用於 Java 的 AWS SDK 傳送和接收的確切請求和回應會很有用。您不應該在生產系統中啟用此記錄，因為寫入大型請求 （例如，上傳到的檔案 Amazon S3) 或回應可能會大幅降低應用程式的速度。如果您真的需要存取此資訊，您可以透過 Apache HttpClient 4 記錄器暫時啟用它。啟用 `org.apache.http.wire` 記錄器的 DEBUG 層級，可以啟用所有請求和回應資料的記錄。

下列 log4j.properties 檔案會開啟 Apache HttpClient 4 中的完整線路記錄，並且只應暫時開啟，因為它可能會對您的應用程式造成重大效能影響。

```
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
```

## 延遲指標記錄

如果您要進行故障診斷，並想要查看哪些程序耗時最多的指標，或伺服器或用戶端的延遲是否更大，延遲記錄器會很有幫助。將`com.amazonaws.latency`記錄器設定為 DEBUG 以啟用此記錄器。

> **Note**
>
> 只有在啟用 SDK 指標時，才能使用此記錄器。若要進一步了解 SDK 指標套件，請參閱啟用 的指標 適用於 Java 的 AWS SDK。

```
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
```

以下為日誌輸出的範例。

```
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]
```

# 客戶端組態

適用於 Java 的 AWS SDK 可讓您變更預設用戶端組態，當您想要：

- 透過代理連線到網際網路
- 變更 HTTP 傳輸設定，例如連線逾時和請求重試
- 指定 TCP 通訊端緩衝區大小提示

## 代理組態

建構用戶端物件時，您可以傳入選用的 ClientConfiguration 物件，以自訂用戶端的組態。

如果您透過代理伺服器連線至網際網路，則需要透過 `ClientConfiguration` 物件設定代理伺服器設定（代理主機、連接埠和使用者名稱/密碼）。

## HTTP 傳輸組態

您可以使用 ClientConfiguration 物件來設定數個 HTTP 傳輸選項。系統偶爾會新增新選項；若要查看您可以擷取或設定的完整選項清單，請參閱 適用於 Java 的 AWS SDK API 參考。

> **Note**
>
> 每個可設定的值都有一個由常數定義的預設值。如需 的恆定值清單`ClientConfiguration`，請參閱 適用於 Java 的 AWS SDK API 參考中的恆定欄位值。

### 連線數上限

您可以使用 ClientConfiguration.setMaxConnections 方法來設定允許的開啟 HTTP 連線數目上限。

> **Important**
>
> 設定連線數上限到並行交易，避免連線失敗和效能不佳。如需預設的最大連線值，請參閱 適用於 Java 的 AWS SDK API 參考中的恆定欄位值。

### 逾時和錯誤處理

您可以設定與逾時和使用 HTTP 連線處理錯誤相關的選項。

- 連線逾時

  連線逾時是 HTTP 連線在放棄之前等待建立連線的時間量 （以毫秒為單位）。預設值為 10，000 毫秒。

  若要自行設定此值，請使用 ClientConfiguration.setConnectionTimeout 方法。
- 連線存留時間 (TTL)

  根據預設，軟體開發套件會盡可能嘗試重複使用 HTTP 連線。在與已停止服務的伺服器建立連線的失敗情況下，具有有限的 TTL 有助於應用程式復原。例如，設定 15 分鐘的 TTL 可確保即使您已建立與發生問題的伺服器的連線，您也會在 15 分鐘內重新建立與新伺服器的連線。

  若要設定 HTTP 連線 TTL，請使用 ClientConfiguration.setConnectionTTL 方法。
- 錯誤重試次數上限

  可重試錯誤的預設重試計數上限為 3。您可以使用 ClientConfiguration.setMaxErrorRetry 方法設定不同的值。

### 本機地址

若要設定 HTTP 用戶端將繫結的本機地址，請使用 ClientConfiguration.setLocalAddress。

## TCP Socket 緩衝區大小提示

想要調校低階 TCP 參數的進階使用者可以透過 ClientConfiguration 物件額外設定 TCP 緩衝區大小提示。大多數使用者永遠不需要調整這些值，但會提供給進階使用者。

應用程式的最佳 TCP 緩衝區大小高度取決於網路和作業系統組態和功能。例如，大多數現代作業系統為 TCP 緩衝區大小提供自動調校邏輯。這可能會對 TCP 連線的效能產生重大影響，而 TCP 連線的開放時間足以讓自動調校最佳化緩衝區大小。

大型緩衝區大小（例如 2 MB) 可讓作業系統緩衝記憶體中的更多資料，而不需要遠端伺服器確認收到該資訊，因此在網路具有高延遲時特別有用。

這只是一個提示，作業系統可能不會遵守它。使用此選項時，使用者應一律檢查作業系統的已設定限制和預設值。大多數作業系統已設定 TCP 緩衝區大小上限，除非您明確提高 TCP 緩衝區大小上限，否則不會讓您超過該限制。

許多資源可協助您設定 TCP 緩衝區大小和作業系統特定的 TCP 設定，包括下列項目：

- 主機調校

# 存取控制政策

AWS 存取控制政策可讓您在 AWS 資源上指定精細存取控制。存取控制政策包含陳述式的集合，其格式如下：

帳戶 A 有權在條件 D 適用的資源 C 上執行動作 B。

其中：

- 是委託人 - 正在請求存取或修改您的其中一個 AWS 資源 AWS 帳戶 的 。
- B 是動作 - 存取或修改 AWS 資源的方式，例如傳送訊息至 Amazon SQS 佇列，或將物件存放在 Amazon S3 儲存貯體中。
- C 是資源 - 委託人想要存取的 AWS 實體，例如 Amazon SQS 佇列或存放於其中的物件 Amazon S3。
- D 是一組條件 - 選擇性限制，指定何時允許或拒絕委託人存取您的資源。有許多表達式條件可供使用，有些是針對每個服務。例如，您可以使用日期條件，僅允許在特定時間之後或之前存取您的資源。

## Amazon S3 範例

下列範例示範政策，允許任何人讀取儲存貯體中的所有物件，但將物件上傳到該儲存貯體的存取權限制為兩個特定的 AWS 帳戶（除了儲存貯體擁有者的帳戶之外）。

```
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 範例

政策的一個常見用途是授權 Amazon SQS 佇列接收來自 Amazon SNS 主題的訊息。

```
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 範例

有些 服務提供額外的條件，可用於 政策。Amazon SNS 提供條件，根據訂閱主題的請求的通訊協定（例如電子郵件、HTTP、HTTPS Amazon SQS) 和端點（例如電子郵件地址、URL、 Amazon SQS ARN)，允許或拒絕訂閱 SNS 主題。

```
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()));
```

# 設定 DNS 名稱查詢的 JVM TTL

Java 虛擬機器 (JVM) 會快取 DNS 名稱查詢。當 JVM 將主機名稱解析為 IP 位址時，它會在指定的時間段內快取 IP 位址，稱為存留時間 (TTL)。

由於 AWS 資源使用偶爾變更的 DNS 名稱項目，我們建議您將 JVM 設定為 5 秒的 TTL 值。這可確保當資源的 IP 位址變更時，您的應用程式將可透過重新查詢 DNS 來接收並使用資源的新 IP 位址。

在一些 Java 組態上，JVM 的預設 TTL 會如此設定，在重新啟動 JVM 之前，「絕不」重新整理 DNS 項目。因此，如果 AWS 資源的 IP 地址在您的應用程式仍在執行時變更，在您手動重新啟動 JVM 並重新整理快取的 IP 資訊之前，將無法使用該資源。在此情況下，設定 JVM 的 TTL 至為關鍵，以便其定期重新整理快取的 IP 資訊。

## 如何設定 JVM TTL

若要修改 JVM 的 TTL，請設定 networkaddress.cache.ttl 安全屬性值。請注意，`networkaddress.cache.ttl` 是安全屬性，而不是系統屬性，即無法使用`-D`命令列旗標來設定。

### 選項 1：在應用程式中以程式設計方式設定

在應用程式啟動的`java.security.Security.setProperty()`早期、建立任何 AWS SDK 用戶端之前，以及提出任何網路請求之前呼叫 ：

```
import java.security.Security;

public class MyApplication {
    public static void main(String[] args) {
        Security.setProperty("networkaddress.cache.ttl", "5");

        // ... create SDK clients and run application
    }
```

```
}
```

### 選項 2：在 java.security 檔案中設定

在 Java 8 的 `$JAVA_HOME/jre/lib/security/java.security` 檔案中設定 `networkaddress.cache.ttl` 屬性，或在 Java 11 或更新版本的 `$JAVA_HOME/conf/security/java.security`檔案中設定 屬性。

以下是 檔案的程式碼片段`java.security`，顯示 TTL 快取設定為 5 秒。

```
#
# 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
...
```

在`$JAVA_HOME`環境變數所代表的 JVM 上執行的所有應用程式都會使用此設定。

### 選項 3：使用 JDK 系統屬性備用 （命令列）

如果您無法修改安全組態或程式碼，您可以使用 JDK 系統屬性。如果未定義安全屬性，這些會充當備用。

- `sun.net.inetaddr.ttl` – 控制成功查詢 （正 TTL)
- `sun.net.inetaddr.negative.ttl` – 控制查詢失敗 （負 TTL)

```
java -Dsun.net.inetaddr.ttl=5 -Dsun.net.inetaddr.negative.ttl=1 -jar myapp.jar
```

> **Note**
>
> 這些是 Oracle Java 8 網路屬性參考中記載的 JDK 內部屬性，做為「未來版本可能不支援」的私有屬性。盡可能使用選項 1-2。

# 啟用 的指標 適用於 Java 的 AWS SDK

適用於 Java 的 AWS SDK 可以使用 Amazon CloudWatch 產生視覺化和監控的指標，以測量：

- 存取應用程式的效能 AWS
- 搭配 使用時，JVMs的效能 AWS
- 執行時間環境詳細資訊，例如堆積記憶體、執行緒數目和開啟的檔案描述項

## 如何啟用 Java SDK 指標產生

您需要新增下列 Maven 相依性，才能讓 SDK 將指標傳送至 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>
```

*將版本編號取代為 Maven Central 提供的最新版本 SDK。

適用於 Java 的 AWS SDK 指標預設為停用。若要為本機開發環境啟用此功能，請在啟動 JVM 時包含指向 AWS 安全登入資料檔案的系統屬性。例如：

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

您需要指定登入資料檔案的路徑，讓 SDK 可以將收集的資料點上傳到 CloudWatch 以供日後分析。

> **Note**
>
> 如果您使用 Amazon EC2 執行個體中繼資料服務 AWS 從 Amazon EC2 執行個體存取 ，則不需要指定登入資料檔案。在這種情況下，您只需指定：
>
> ```
> -Dcom.amazonaws.sdk.enableDefaultMetrics
> ```

擷取的所有指標 適用於 Java 的 AWS SDK 都位於命名空間 AWSSDK/Java 下，並上傳至 CloudWatch 預設區域 (us-east-1)。若要變更區域，請使用系統屬性中的 `cloudwatchRegion` 屬性來指定區域。例如，若要將 CloudWatch 區域設定為 us-east-1，請使用：

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

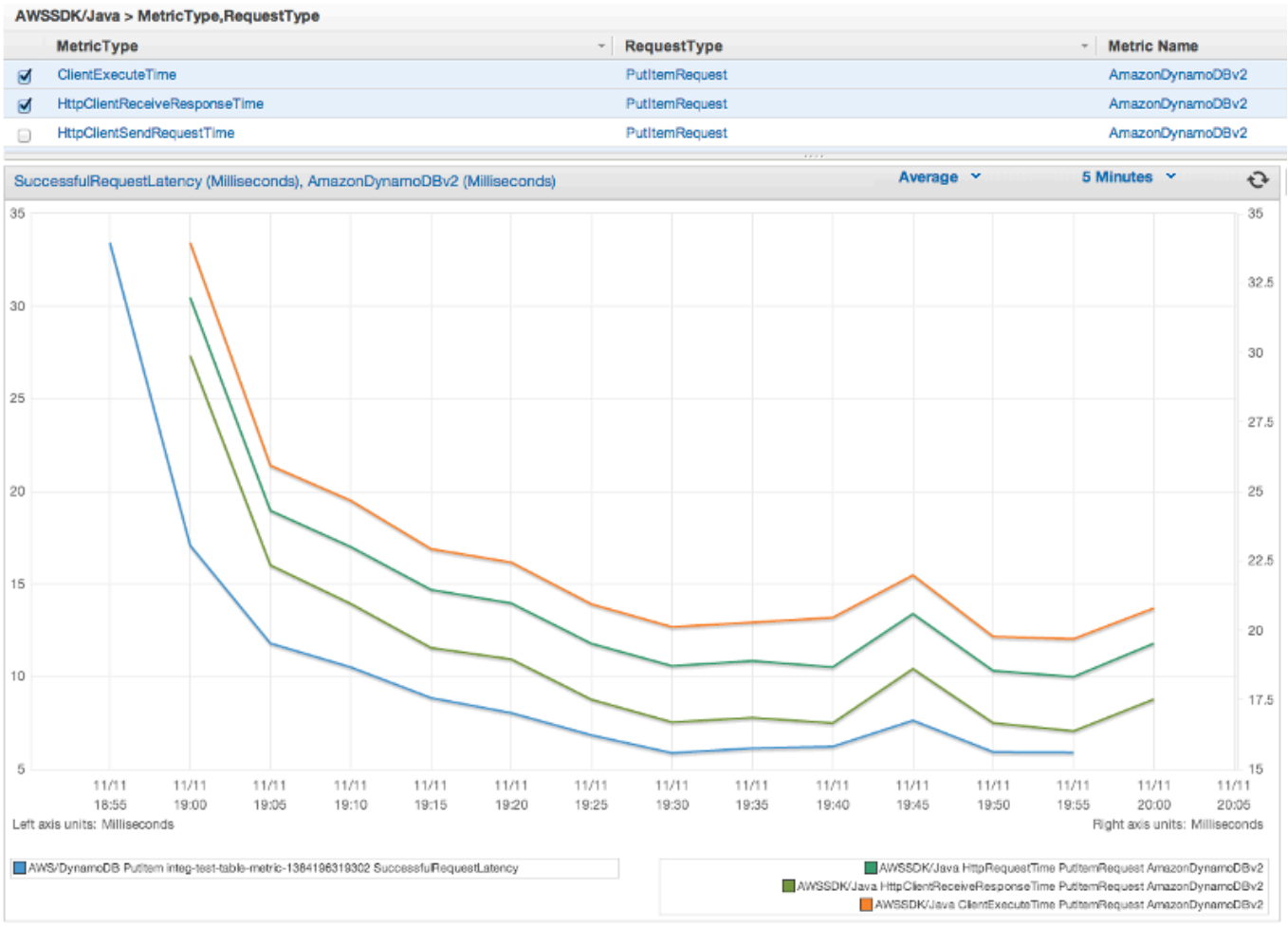
啟用此功能後，每次 AWS 有來自 的服務請求時 適用於 Java 的 AWS SDK，都會產生指標資料點、排入統計摘要佇列，並以非同步方式上傳至 CloudWatch，大約每分鐘一次。上傳指標後，您可以使用 將其視覺化，AWS 管理主控台並針對記憶體洩漏、檔案描述項洩漏等潛在問題設定警示。

## 可用的指標類型

預設指標集分為三個主要類別：

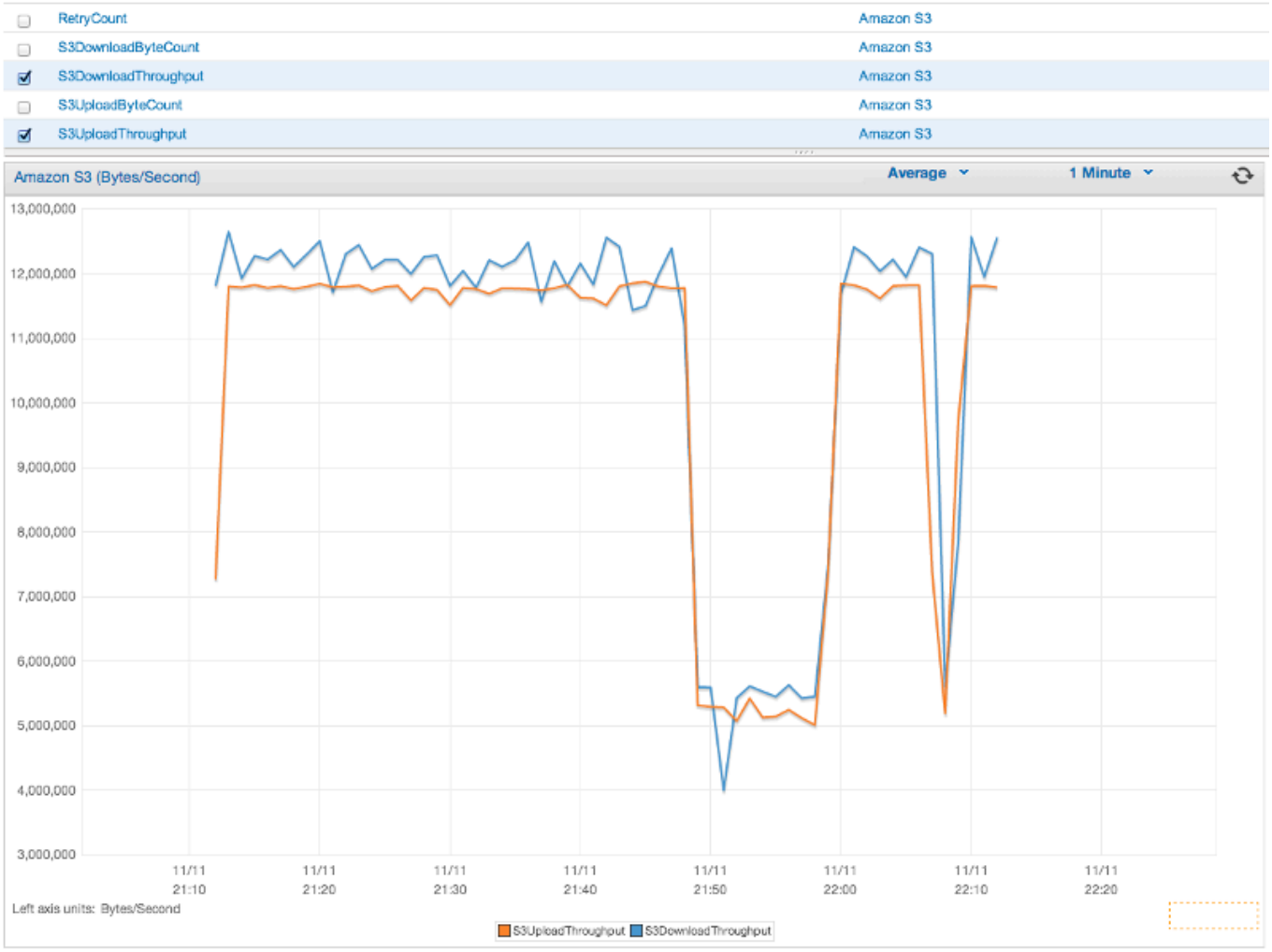
AWS 請求指標

- 涵蓋 HTTP 請求/回應的延遲、請求數量、例外狀況和重試等領域。



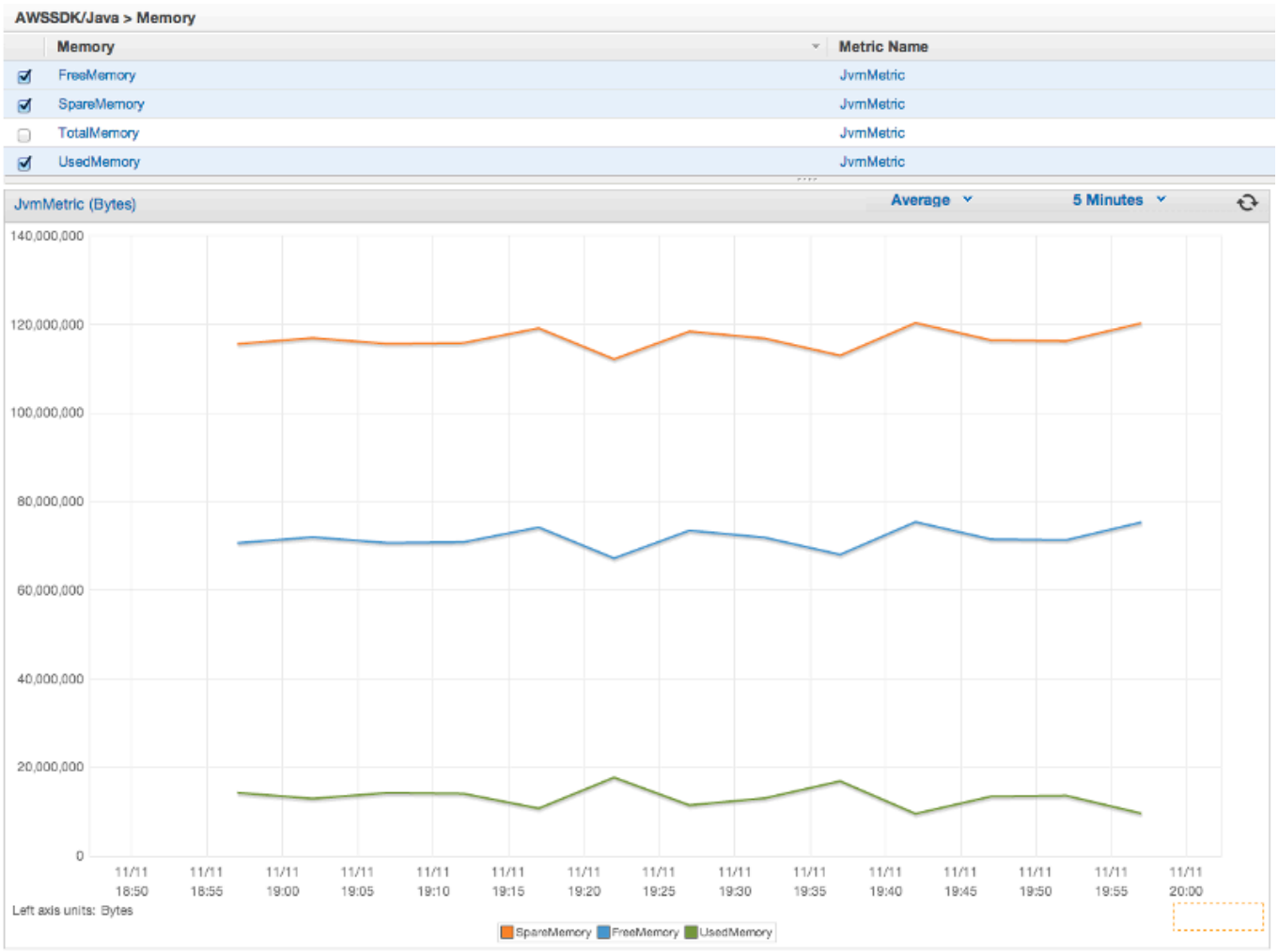

## AWS 服務 指標

- 包含 AWS 服務特定資料，例如 S3 上傳和下載的輸送量和位元組數。




## 機器指標

- 涵蓋執行時間環境，包括堆積記憶體、執行緒數目和開啟的檔案描述項。




如果您想要排除機器指標，請將 `excludeMachineMetrics`新增至系統屬性：

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

## 詳細資訊

- 如需預先定義核心指標類型的完整清單，請參閱 amazonaws/metrics 套件摘要。
- 了解如何使用 CloudWatch 範例中的 使用 適用於 Java 的 AWS SDK CloudWatch。 CloudWatch 適用於 Java 的 AWS SDK
- 進一步了解調校 適用於 Java 的 AWS SDK 以改善彈性部落格文章中的效能調校。

# 適用於 Java 的 AWS SDK 程式碼範例

本節提供使用 適用於 Java 的 AWS SDK v1 編寫程式 AWS 服務的教學課程和範例。

在 GitHub 上的 AWS 文件程式碼範例儲存庫中尋找這些範例和其他範例的原始程式碼。 GitHub

若要提議新的程式碼範例，讓 AWS 文件團隊考慮生產，請建立新的請求。該團隊想要產生比僅涵蓋個別 API 呼叫之簡易程式碼更為廣泛的程式碼範例，以涵蓋更為廣泛的案例和使用案例。如需說明，請參閱 GitHub 上程式碼範例儲存庫中的貢獻指導方針。 GitHub..

## 適用於 Java 的 AWS SDK 2.x

在 2018 年， AWS 發行了 AWS SDK for Java 2.x。本指南包含使用最新 Java 開發套件以及範例程式碼的指示。

> **Note**
>
> 如需 適用於 Java 的 AWS SDK 開發人員可用的更多範例和其他資源，請參閱其他文件和資源！

## 使用 的 CloudWatch 範例 適用於 Java 的 AWS SDK

本節提供使用適用於 Java 的 AWS SDK編寫 CloudWatch 程式的範例。

Amazon CloudWatch AWS 會即時監控您的 Amazon Web Services (AWS) 資源和您在 上執行的應用程式。您可以使用 CloudWatch 收集和追蹤指標，這些是您可以為您的資源和應用程式測量的變數。CloudWatch 警示會根據您定義的規則，傳送通知或自動變更您要監控的資源。

如需 的詳細資訊 CloudWatch，請參閱Amazon CloudWatch 《 使用者指南》。

> **Note**
>
> 這些範例僅包含示範每種技術所需的程式碼。GitHub 上提供完整程式碼範例。您可以從那裡下載單一原始檔案或將儲存庫複製到本機，以取得建置和執行的所有範例。

主題

- 從 CloudWatch 取得指標
- 發佈自訂指標資料
- 使用 CloudWatch 警示
- 在 CloudWatch 中使用警示動作
- 傳送事件至 CloudWatch

# 從 CloudWatch 取得指標

## 列出指標

若要列出 CloudWatch 指標，請建立 ListMetricsRequest 並呼叫 AmazonCloudWatchClient 的 `listMetrics`方法。您可以使用 `ListMetricsRequest` 來根據命名空間、指標名稱或維度，篩選傳回的指標。

> **Note**
>
> AWS 服務發佈的指標和維度清單，請參閱 Amazon CloudWatch 《 使用者指南》中的 {https---docs-aws-amazon-com-AmazonCloudWatch-latest-monitoring-CW-Support-For-AWS html}【Amazon CloudWatch 指標和維度參考】。

### 匯入

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

### Code

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

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

```
boolean done = false;

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

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

    request.setNextToken(response.getNextToken());

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

指標會透過呼叫其getMetrics方法，在 ListMetricsResult 中傳回。結果可能會分頁。若要擷取下一批結果，setNextToken請對原始請求物件呼叫 ，並使用ListMetricsResult物件getNextToken方法的傳回值，並將修改後的請求物件傳遞回另一個呼叫給 listMetrics。

### 詳細資訊

- Amazon CloudWatch API 參考中的 ListMetrics。

## 發佈自訂指標資料

許多 AWS 服務會在以「AWS」開頭的命名空間中發佈自己的指標。您也可以使用自己的命名空間（只要不是以「AWS」開頭）來發佈自訂指標資料。

### 發佈自訂指標資料

若要發佈您自己的指標資料，請使用 PutMetricDataRequest 呼叫 AmazonCloudWatchClient 的putMetricData方法。PutMetricDataRequest 必須在 MetricDatum 物件中包含用於資料的自訂命名空間，以及資料點本身的相關資訊。

> **Note**
>
> 您無法指定開頭為 " AWS " 的命名空間。以 " AWS " 開頭的命名空間保留供 Amazon Web Services 產品使用。

匯入

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

Code

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

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

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

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

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

## 詳細資訊

- Amazon CloudWatch 《使用者指南》中的使用 Amazon CloudWatch 指標。
- Amazon CloudWatch 《使用者指南AWS》中的命名空間。
- 《Amazon CloudWatch API 參考》中的 PutMetricData。

# 使用 CloudWatch 警示

## 建立警示

若要根據 CloudWatch 指標建立警示，請呼叫 AmazonCloudWatchClient 的 `putMetricAlarm`方法，並將 PutMetricAlarmRequest 填入警示條件。

### 匯入

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

### Code

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

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

PutMetricAlarmRequest request = new PutMetricAlarmRequest()
    .withAlarmName(alarmName)
    .withComparisonOperator(
        ComparisonOperator.GreaterThanThreshold)
    .withEvaluationPeriods(1)
    .withMetricName("CPUUtilization")
    .withNamespace("{AWS}/EC2")
    .withPeriod(60)
    .withStatistic(Statistic.Average)
    .withThreshold(70.0)
    .withActionsEnabled(false)
    .withAlarmDescription(
        "Alarm when server CPU utilization exceeds 70%")
    .withUnit(StandardUnit.Seconds)
    .withDimensions(dimension);
```

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

## 列出警示

若要列出您已建立的 CloudWatch 警示，請使用 DescribeAlarmsRequest 呼叫 AmazonCloudWatchClient 的 `describeAlarms`方法，以用於設定結果的選項。

匯入

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

Code

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

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

while(!done) {

    DescribeAlarmsResult response = cw.describeAlarms(request);

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

    request.setNextToken(response.getNextToken());

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

您可以在 傳回的 DescribeAlarmsResult `getMetricAlarms`上呼叫 ，以取得警示清單`describeAlarms`。

結果可能會分頁。若要擷取下一批結果，setNextToken請對原始請求物件呼叫 ，並使用DescribeAlarmsResult物件getNextToken方法的傳回值，並將修改後的請求物件傳遞回另一個呼叫給 describeAlarms。

> **Note**
>
> 您也可以使用 AmazonCloudWatchClient 的 describeAlarmsForMetric方法，擷取特定指標的警示。其用法類似於 describeAlarms。

## 刪除警示

若要刪除 CloudWatch 警示，請使用 DeleteAlarmsRequest 呼叫 AmazonCloudWatchClient 的 deleteAlarms方法，其中包含您要刪除的一或多個警示名稱。

匯入

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

Code

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

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

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

## 詳細資訊

- Amazon CloudWatch 《 使用者指南》中的建立 Amazon CloudWatch 警示
- Amazon CloudWatch API 參考中的 PutMetricAlarm
- Amazon CloudWatch API 參考中的 DescribeAlarms
- Amazon CloudWatch API 參考中的 DeleteAlarms

# 在 CloudWatch 中使用警示動作

使用 CloudWatch 警示動作，您可以建立警示來執行自動停止、終止、重新啟動或復原 Amazon EC2 執行個體等動作。

> **Note**
>
> 在建立警示時，可以使用 `setAlarmActions`PutMetricAlarmRequest 的 方法，將警示動作新增到警示。

## 啟用警示動作

若要啟用 CloudWatch 警示的警示動作，`enableAlarmActions`請使用 EnableAlarmActionsRequest 呼叫 AmazonCloudWatchClient 的 ，其中包含您想要啟用其動作的一或多個警示名稱。

匯入

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

Code

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

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

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

## 停用警示動作

若要停用 CloudWatch 警示的警示動作，`disableAlarmActions`請使用 DisableAlarmActionsRequest 呼叫 AmazonCloudWatchClient 的 ，其中包含您想要停用其動作的一或多個警示名稱。

匯入

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

Code

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

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

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

### 詳細資訊

- 《Amazon CloudWatch 使用者指南》中的建立警示以停止、終止、重新啟動或復原執行個體
- Amazon CloudWatch API 參考中的 PutMetricAlarm
- Amazon CloudWatch API 參考中的 EnableAlarmActions
- API 參考中的 DisableAlarmActions Amazon CloudWatch

## 傳送事件至 CloudWatch

CloudWatch 事件提供近乎即時的系統事件串流，描述 Amazon EC2 執行個體、 Lambda 函數、 Kinesis 串流、 Amazon ECS 任務、 Step Functions 狀態機器、 Amazon SNS 主題、 Amazon SQS 佇列或內建目標 AWS 的資源變更。您可以使用簡單的規則，來比對事件，並將這些事件轉傳到一或多個目標函數或串流。

### 新增事件

若要新增自訂 CloudWatch 事件，請使用 PutEventsRequest 物件呼叫 AmazonCloudWatchEventsClient 的 `putEvents`方法，該物件包含一或多個 PutEventsRequestEntry 物件，提供每個事件的詳細資訊。您可以指定項目的多個參數，例如事件的來源和類型、與事件相關聯的資源等等。

> **Note**
>
> 對 putEvents 的每個呼叫最多可以指定 10 個事件。

匯入

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

Code

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

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

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

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

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

## 新增規則

若要建立或更新規則，請使用 PutRuleRequest 呼叫 AmazonCloudWatchEventsClient 的 putRule方法，其中包含規則名稱和選用參數，例如事件模式、與規則建立關聯 IAM 的角色，以及描述規則執行頻率的排程表達式。

匯入

```
import com.amazonaws.services.cloudwatchevents.AmazonCloudWatchEvents;
```

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

Code

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

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

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

## 新增目標

目標是觸發規則時叫用的資源。範例目標包括 Amazon EC2 執行個體、 Lambda 函數、 Kinesis 串流、 Amazon ECS 任務、 Step Functions 狀態機器和內建目標。

若要將目標新增至規則，請使用 PutTargetsRequest 呼叫 AmazonCloudWatchEventsClient 的 `putTargets`方法，其中包含要更新的規則和要新增至規則的目標清單。

匯入

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

Code

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

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

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

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

### 詳細資訊

- 《Amazon CloudWatch Events 使用者指南》中的使用 PutEvents 新增事件
- Amazon CloudWatch Events 《使用者指南》中的規則排程表達式
- 《Amazon CloudWatch Events 使用者指南》中的CloudWatch 事件的事件類型
- Amazon CloudWatch Events 《使用者指南》中的事件和事件模式
- Amazon CloudWatch Events API 參考中的 PutEvents
- Amazon CloudWatch Events API 參考中的 PutTargets
- Amazon CloudWatch Events API 參考中的 PutRule

# DynamoDB 使用 的範例 適用於 Java 的 AWS SDK

本節提供使用適用於 Java 的 AWS SDK編寫 DynamoDB 程式的範例。

> **Note**
>
> 這些範例僅包含示範每種技術所需的程式碼。GitHub 上提供完整程式碼範例。您可以從那裡下載單一原始檔案或將儲存庫複製到本機，以取得建置和執行的所有範例。

主題

- 使用帳戶 AWS 型端點
- 在 中使用資料表 DynamoDB
- 在 中使用項目 DynamoDB

## 使用帳戶 AWS 型端點

DynamoDB 提供以AWS 帳戶為基礎的端點，可透過使用 AWS 您的帳戶 ID 來簡化請求路由來改善效能。

若要利用此功能，您需要使用 版本 1.12.771 或更新版本的 版本 1 適用於 Java 的 AWS SDK。您可以在 Maven 中央儲存庫中找到最新版本的 SDK。支援版本的 SDK 處於作用中狀態後，會自動使用新的端點。

如果您想要選擇退出以帳戶為基礎的路由，您有四個選項：

- 將 DynamoDB 服務用戶端`AccountIdEndpointMode`設定為 `DISABLED`。
- 設定環境變數。
- 設定 JVM 系統屬性。
- 更新共用 AWS 組態檔案設定。

下列程式碼片段示範如何透過設定 DynamoDB 服務用戶端來停用帳戶型路由：

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

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

AWS SDKs 和工具參考指南提供有關最後三個組態選項的詳細資訊。

## 在 中使用資料表 DynamoDB

資料表是 DynamoDB 資料庫中所有項目的容器。您必須先建立資料表 DynamoDB，才能從中新增或移除資料。

對於每個資料表，您必須定義：

- 對於您的帳戶和區域都具有獨一性的資料表名稱。
- 每個值的主索引鍵都必須獨一無二，資料表中任兩個項目不能有相同的主索引鍵值。

  主索引鍵可以是簡單的，包含單一分割區 (HASH) 索引鍵；也可以是複合的，包含分割區和排序 (RANGE) 索引鍵。

每個索引鍵值都有一個關聯的資料類型，由 ScalarAttributeType 類別列舉。索引鍵值可以是二進位 (B)、數值 (N)、或字串 (S)。如需詳細資訊，請參閱《 Amazon DynamoDB 開發人員指南》中的命名規則和資料類型。

- 定義資料表預留讀取/寫入容量單位數量的佈建輸送量值。

> **Note**
>
> Amazon DynamoDB 定價是以您在資料表上設定的佈建輸送量值為基礎，因此請只預留您認為資料表所需的容量。

您可以隨時修改資料表的佈建輸送量，以便在需要變更時調整容量。

## 建立資料表

使用DynamoDB 用戶端的 `createTable`方法來建立新的 DynamoDB 資料表。您需要建構資料表屬性和資料表結構描述，這兩項都會用來識別資料表的主索引鍵。您也必須提供初始佈建的輸送量值和資料表名稱。只有在建立資料表時，才定義索引鍵 DynamoDB 資料表屬性。

> **Note**
>
> 如果具有您所選名稱的資料表已存在，則會擲回 AmazonServiceException。

匯入

```
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;
```

使用簡單主索引鍵建立資料表

此程式碼會使用簡單主索引鍵 ("Name") 來建立資料表。

Code

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

final AmazonDynamoDB ddb = AmazonDynamoDBClientBuilder.defaultClient();

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

請參閱 GitHub 上的完整範例。

使用複合主索引鍵建立資料表

新增另一個 AttributeDefinition 和 KeySchemaElement 到 CreateTableRequest。

Code

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

請參閱 GitHub 上的完整範例。

## 列出資料表

您可以呼叫DynamoDB 用戶端的 `listTables`方法，列出特定區域中的資料表。

> **Note**
>
> 如果您的帳戶和區域不存在指定的資料表，會擲出 ResourceNotFoundException。

### 匯入

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

### Code

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

ListTablesRequest request;

boolean more_tables = true;
String last_name = null;

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

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

        if (table_names.size() > 0) {
            for (String cur_name : table_names) {
```

```
                System.out.format("* %s\n", cur_name);
            }
        } else {
            System.out.println("No tables found!");
            System.exit(0);
        }

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

根據預設，每次呼叫最多傳回 100 個資料表 - 在傳回的 ListTablesResult 物件`getLastEvaluatedTableName`上使用 ，以取得上次評估的資料表。您可以使用這個值，在前次列表最後傳回值之後開始列表。

請參閱 GitHub 上的完整範例。

## 說明資料表 (取得相關資訊)

呼叫DynamoDB 用戶端的 `describeTable`方法。

> **Note**
>
> 如果您的帳戶和區域不存在指定的資料表，會擲出 ResourceNotFoundException。

匯入

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

Code

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

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

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

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

        List<AttributeDefinition> attributes =
           table_info.getAttributeDefinitions();
        System.out.println("Attributes");
        for (AttributeDefinition a : attributes) {
            System.out.format("  %s (%s)\n",
                  a.getAttributeName(), a.getAttributeType());
        }
    }
} catch (AmazonServiceException e) {
    System.err.println(e.getErrorMessage());
    System.exit(1);
}
```

請參閱 GitHub 上的完整範例。

## 修改 (更新) 資料表

您可以隨時呼叫DynamoDB 用戶端的 updateTable方法，修改資料表的佈建輸送量值。

> **Note**
>
> 如果您的帳戶和區域不存在指定的資料表，會擲出 ResourceNotFoundException。

匯入

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

Code

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

final AmazonDynamoDB ddb = AmazonDynamoDBClientBuilder.defaultClient();

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

請參閱 GitHub 上的完整範例。

## 刪除資料表

呼叫DynamoDB 用戶端的 `deleteTable`方法，並將資料表的名稱傳遞給用戶端。

> **Note**
>
> 如果您的帳戶和區域不存在指定的資料表，會擲出 ResourceNotFoundException。

匯入

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

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

Code

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

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

請參閱 GitHub 上的完整範例。

### 詳細資訊

- 《 Amazon DynamoDB 開發人員指南》中的使用資料表的指導方針
- 《 Amazon DynamoDB 開發人員指南》中的在 中使用資料表 DynamoDB

## 在 中使用項目 DynamoDB

在 中 DynamoDB，項目是屬性的集合，每個屬性都有名稱和值。屬性值可以是純量、集合或文件類型。如需詳細資訊，請參閱《 Amazon DynamoDB 開發人員指南》中的命名規則和資料類型。

### 從資料表擷取 (取得) 項目

呼叫 AmazonDynamoDB 的 `getItem`方法，並傳遞具有資料表名稱的 GetItemRequest 物件，以及您想要的項目主索引鍵值。它會傳回 GetItemResult 物件。

您可以使用所傳回 `GetItemResult` 物件的 `getItem( )` 方法來擷取與項目關聯之索引鍵 (字串) 和值的對應 (AttributeValue) 組。

匯入

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

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

### Code

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

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

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

final AmazonDynamoDB ddb = AmazonDynamoDBClientBuilder.defaultClient();

try {
    Map<String,AttributeValue> returned_item =
       ddb.getItem(request).getItem();
    if (returned_item != null) {
        Set<String> keys = returned_item.keySet();
        for (String key : keys) {
            System.out.format("%s: %s\n",
                    key, returned_item.get(key).toString());
        }
    } else {
        System.out.format("No item found with the key %s!\n", name);
    }
} catch (AmazonServiceException e) {
    System.err.println(e.getErrorMessage());
    System.exit(1);
```

請參閱 GitHub 上的完整範例。

## 新增項目到資料表

建立代表項目屬性的索引鍵值組對應。這些項目必須包含資料表主索引鍵欄位的值。如果主索引鍵識別的項目已存在，其欄位會透過請求更新。

> **Note**
>
> 如果您的帳戶和區域不存在指定的資料表，會擲出 ResourceNotFoundException。

### 匯入

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.services.dynamodbv2.model.AttributeValue;
import com.amazonaws.services.dynamodbv2.model.ResourceNotFoundException;
import java.util.ArrayList;
```

### Code

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

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

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

final AmazonDynamoDB ddb = AmazonDynamoDBClientBuilder.defaultClient();

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

請參閱 GitHub 上的完整範例。

## 更新資料表中的現有項目

您可以使用 AmazonDynamoDB 的 `updateItem`方法，提供資料表名稱、主索引鍵值和要更新的欄位映射，來更新已存在於資料表中的項目屬性。

> **Note**
>
> 如果您的帳戶和區域不存在指定的資料表，或者如果您傳遞的主索引鍵所識別的項目不存在，會擲出 ResourceNotFoundException。

### 匯入

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.services.dynamodbv2.model.AttributeAction;
import com.amazonaws.services.dynamodbv2.model.AttributeValue;
import com.amazonaws.services.dynamodbv2.model.AttributeValueUpdate;
import com.amazonaws.services.dynamodbv2.model.ResourceNotFoundException;
import java.util.ArrayList;
```

### Code

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

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

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

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

final AmazonDynamoDB ddb = AmazonDynamoDBClientBuilder.defaultClient();

try {
```

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

請參閱 GitHub 上的完整範例。

## 使用 DynamoDBMapper 類別

適用於 Java 的 AWS SDK 提供 DynamoDBMapper 類別，可讓您將用戶端類別映射至 Amazon DynamoDB 資料表。若要使用 DynamoDBMapper 類別，您可以使用註釋 （如下列程式碼範例所示），定義 DynamoDB 資料表中項目與程式碼中對應物件執行個體之間的關係。DynamoDBMapper 類別可讓您存取資料表；執行各種建立、讀取、更新和刪除 (CRUD) 操作；以及執行查詢。

> **Note**
>
> DynamoDBMapper 類別不允許您建立、更新或刪除資料表。

匯入

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

Code

下列 Java 程式碼範例示範如何使用 DynamoDBMapper 類別將內容新增至 Music 資料表。將內容新增至資料表後，請注意，會使用分割區和排序索引鍵載入項目。然後，獎勵項目會更新。如需建立音樂資料表的資訊，請參閱《 Amazon DynamoDB 開發人員指南》中的建立資料表。

```
        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;
        }
    }
```

請參閱 GitHub 上的完整範例。

## 詳細資訊

- 《 Amazon DynamoDB 開發人員指南》中的使用項目的指導方針
- 《 Amazon DynamoDB 開發人員指南》中的使用 中的項目 DynamoDB

# Amazon EC2 使用 的範例 適用於 Java 的 AWS SDK

本節提供Amazon EC2使用 進行程式設計的範例 適用於 Java 的 AWS SDK。

主題


- 教學課程：啟動 EC2 執行個體
- 使用 IAM 角色授予 上 AWS 資源的存取權 Amazon EC2
- 教學課程： Amazon EC2 Spot 執行個體
- 教學課程：進階 Amazon EC2 Spot 請求管理
- 管理 Amazon EC2 執行個體
- 在 中使用彈性 IP 地址 Amazon EC2
- 使用區域和可用區域
- 使用 Amazon EC2 金鑰對
- 在 中使用安全群組 Amazon EC2


## 教學課程：啟動 EC2 執行個體

本教學課程示範如何使用 適用於 Java 的 AWS SDK 來啟動 EC2 執行個體。

主題


- 先決條件
- 建立 Amazon EC2 安全群組
- 建立金鑰對
- 執行 Amazon EC2 執行個體


### 先決條件

開始之前，請確定您已建立 ， AWS 帳戶 且已設定您的 AWS 登入資料。如需詳細資訊，請參閱 入門。

# 建立 Amazon EC2 安全群組

EC2-Classic 正在淘汰

> **Warning**
>
> 我們將於 2022 年 8 月 15 日淘汰 EC2-Classic。建議您從 EC2-Classic 遷移至 VPC。如需詳細資訊，請參閱部落格文章 EC2-Classic-Classic Networking 正在淘汰 – 以下說明如何準備。

建立安全群組，做為虛擬防火牆，控制一或多個 EC2 執行個體的網路流量。根據預設， 會將您的執行個體與不允許傳入流量的安全群組建立 Amazon EC2 關聯。您可以建立允許您的 EC2 執行個體接受特定連接的安全群組。例如，如果您需要連線到 Linux 執行個體，您必須設定安全群組以允許 SSH 流量。您可以使用 Amazon EC2 主控台或 建立安全群組 適用於 Java 的 AWS SDK。

您可以建立安全群組，提供於 EC2-Classic 或 EC2-VPC 使用。如需 EC2-Classic 和 EC2-VPC 的詳細資訊，請參閱《Linux 執行個體 Amazon EC2 使用者指南》中的支援的平台。

如需使用 Amazon EC2 主控台建立安全群組的詳細資訊，請參閱《Linux 執行個體 Amazon EC2 使用者指南》中的Amazon EC2 安全群組。

1. 建立和初始化 CreateSecurityGroupRequest 執行個體。使用 withGroupName 方法來設定安全群組名稱，並使用 withDescription 方法來設定安全群組描述，如下所示：

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

   安全群組名稱在您初始化 Amazon EC2 用戶端的 AWS 區域中必須是唯一的。您必須使用 US-ASCII 字元做為安全群組名稱和描述。

2. 將請求物件做為參數傳遞至 createSecurityGroup 方法。方法會傳回 CreateSecurityGroupResult 物件，如下所示：

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

   如果您嘗試建立與現有安全群組同名的安全群組， 會`createSecurityGroup`擲回例外狀況。

根據預設，新的安全群組不允許任何傳入流量到您的 Amazon EC2 執行個體。若要允許傳入流量，您必須明確授權安全群組傳入。您可以為個別 IP 地址、一系列 IP 地址、特定通訊協定和 TCP/UDP 連接埠授權輸入。

1. 建立和初始化 IpPermission 執行個體。使用 withIpv4Ranges 方法設定 IP 地址的範圍以授權輸入，並使用 withIpProtocol 方法設定 IP 通訊協定。使用 withFromPort 和 withToPort 方法指定連接埠範圍以授權輸入，如下所示：

```
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);
```

   必須符合您在 IpPermission 物件中指定的所有條件，才能允許輸入。

   使用 CIDR 表示法指定 IP 地址。如果您將通訊協定指定為 TCP/UDP，則必須提供來源連接埠和目的地連接埠。只有在您指定 TCP 或 UDP 時，才能授權連接埠。

2. 建立和初始化 AuthorizeSecurityGroupIngressRequest 執行個體。使用 withGroupName方法指定安全群組名稱，並將您先前初始化的IpPermission物件傳遞至 withIpPermissions 方法，如下所示：

```
AuthorizeSecurityGroupIngressRequest authorizeSecurityGroupIngressRequest =
    new AuthorizeSecurityGroupIngressRequest();

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

3. 將請求物件傳遞至 authorizeSecurityGroupIngress 方法，如下所示：

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

   如果您authorizeSecurityGroupIngress使用已授權輸入 IP 地址呼叫 ，則 方法會擲回例外狀況。在呼叫 之前，建立並初始化新的IpPermission物件，以授權不同 IPs、連接埠和通訊協定的輸入AuthorizeSecurityGroupIngress。

每當您呼叫 authorizeSecurityGroupIngress 或 authorizeSecurityGroupEgress 方法時，就會將規則新增至您的安全群組。

## 建立金鑰對

您必須在啟動 EC2 執行個體時指定金鑰對，然後在連線至執行個體時指定金鑰對的私有金鑰。您可以建立金鑰對，或使用啟動其他執行個體時使用的現有金鑰對。如需詳細資訊，請參閱《Linux 執行個體 Amazon EC2 使用者指南》中的Amazon EC2 金鑰對。

1. 建立和初始化 CreateKeyPairRequest 執行個體。使用 withKeyName 方法來設定金鑰對名稱，如下所示：

```
CreateKeyPairRequest createKeyPairRequest = new CreateKeyPairRequest();

createKeyPairRequest.withKeyName(keyName);
```

> **Important**
>
> 金鑰對名稱必須是唯一的。如果您嘗試建立與現有金鑰對具有相同金鑰名稱的金鑰對，則會收到例外狀況。

2. 將請求物件傳遞至 createKeyPair 方法。方法會傳回 CreateKeyPairResult 執行個體，如下所示：

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

3. 呼叫結果物件的 getKeyPair 方法以取得 KeyPair 物件。呼叫KeyPair物件的 getKeyMaterial 方法，以取得未加密的 PEM 編碼私有金鑰，如下所示：

```
KeyPair keyPair = new KeyPair();

keyPair = createKeyPairResult.getKeyPair();

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

## 執行 Amazon EC2 執行個體

使用下列程序，從相同的 Amazon Machine Image (AMI) 啟動一或多個設定相同的 EC2 執行個體。建立 EC2 執行個體後，您可以查看他們的狀態。執行 EC2 執行個體之後，您可以連線到執行個體。

1. 建立和初始化 RunInstancesRequest 執行個體。請確定您在建立用戶端物件時指定的區域中存在您指定的 AMI、金鑰對和安全群組。

```
RunInstancesRequest runInstancesRequest =
   new RunInstancesRequest();

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

withImageId

- AMI 的 ID。若要了解如何尋找 Amazon 提供的公AMIs 或建立您自己的 AMI，請參閱 Amazon Machine Image (AMI)。

withInstanceType

- 執行個體類型與所指定的 AMI 相容。如需詳細資訊，請參閱《Linux 執行個體使用者指南》中的執行個體類型。 Amazon EC2

withMinCount

- 要啟動執行個體的最少數量。如果這比 Amazon EC2 可在目標可用區域中啟動的執行個體更多，則 不會 Amazon EC2 啟動任何執行個體。

withMaxCount

- 要啟動執行個體的最大數量。如果這比 Amazon EC2 可在目標可用區域中啟動的執行個體更多， 會在 上方 Amazon EC2 啟動最多的執行個體數量MinCount。您可以啟動的範圍數量介於 1 到執行個體類型允許的執行個體最大數量。如需詳細資訊，請參閱 Amazon EC2 一般常見問答集 Amazon EC2 中的我可以在 中執行多少執行個體。

withKeyName

- EC2 金鑰對的名稱。如果您未指定金鑰對而啟動執行個體，則就無法與它連線。如需詳細資訊，請參閱建立金鑰對。

withSecurityGroups

- 一個或多個安全群組。如需詳細資訊，請參閱建立 Amazon EC2 安全群組。

2. 透過將請求物件傳遞至 runInstances 方法來啟動執行個體。方法會傳回 RunInstancesResult 物件，如下所示：

```
RunInstancesResult result = amazonEC2Client.runInstances(
```

```
                                runInstancesRequest);
```

執行個體執行後，您可以使用 金鑰對來連接至執行個體。如需詳細資訊，請參閱《Linux 執行個體使用者指南》中的連線至您的 Linux 執行個體。 Amazon EC2

# 使用 IAM 角色授予 上 AWS 資源的存取權 Amazon EC2

所有對 Amazon Web Services (AWS) 的請求都必須使用 發出的登入資料進行密碼編譯簽署 AWS。您可以使用 IAM 角色，方便地從 Amazon EC2 執行個體授予 AWS 資源的安全存取權。

本主題提供如何搭配執行 的 Java SDK 應用程式使用 IAM 角色的相關資訊 Amazon EC2。如需 IAM 執行個體的詳細資訊，請參閱《Linux 執行個體 Amazon EC2 使用者指南》中的適用於 的 IAM 角色 Amazon EC2。

## 預設提供者鏈和 EC2 執行個體描述檔

如果您的應用程式使用預設建構函數建立 AWS 用戶端，則用戶端將依下列順序使用預設憑證提供者鏈結搜尋憑證：

1. 在 Java 系統屬性中：`aws.accessKeyId` 和 `aws.secretKey`。
2. 在系統環境變數中：`AWS_ACCESS_KEY_ID` 和 `AWS_SECRET_ACCESS_KEY`。
3. 在預設登入資料檔案中 (此檔案的位置因平台而異)。
4. 如果已設定`AWS_CONTAINER_CREDENTIALS_RELATIVE_URI`環境變數，且安全管理員具有存取變數的許可，則透過 Amazon EC2 容器服務傳遞的登入資料。
5. 在執行個體描述檔登入資料中，這存在於與 EC2 執行個體的 IAM 角色關聯的執行個體中繼資料內。
6. 來自環境或容器的 Web Identity Token 登入資料。

預設供應商鏈中的執行個體描述檔登入資料步驟只有在 Amazon EC2 執行個體上執行應用程式時才能使用，但在使用 Amazon EC2 執行個體時提供最大的使用便利性和最佳安全性。您也可以直接傳遞 InstanceProfileCredentialsProvider 執行個體給用戶端建構函數來取得執行個體描述檔登入資料，而無須繼續進行整個預設供應者鏈結。

例如：

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

```
                .build();
```

使用此方法時，開發套件會擷取暫時 AWS 登入資料，其許可與執行個體描述檔中與 Amazon EC2 執行個體相關聯之 IAM 角色的許可相同。雖然這些登入資料是暫時的，最終會過期，`InstanceProfileCredentialsProvider`會定期為您重新整理，以便取得的登入資料繼續允許存取 AWS。

> **Important**
>
> 自動登入資料重新整理只會在您使用預設用戶端建構函數時發生，這會建立自己的 `InstanceProfileCredentialsProvider` 做為預設提供者鏈結的一部分，或當您將`InstanceProfileCredentialsProvider`執行個體直接傳遞給用戶端建構函數時。如果您使用其他方法來取得或傳遞執行個體描述檔登入資料，則需負責檢查和重新整理過期的登入資料。

如果用戶端建構函式無法使用登入資料提供者鏈結找到登入資料，則會擲回 AmazonClientException。

## 逐步解說：針對 EC2 執行個體使用 IAM 角色

下列逐步解說說明如何 Amazon S3 使用 IAM 角色從 擷取物件來管理存取。

### 建立 IAM 角色

建立授予唯讀存取權的 IAM 角色 Amazon S3。

1. 開啟 IAM 主控台。
2. 在導覽窗格中，選取角色，然後選取建立新角色。
3. 輸入角色的名稱，然後選擇 Next Step (下一步)。請記住此名稱，因為當您啟動 Amazon EC2 執行個體時，您將需要此名稱。
4. 在選取角色類型頁面 AWS 服務 的角色下，選取  Amazon EC2 。
5. 在設定許可頁面的選取政策範本下，選取 Amazon S3 唯讀存取，然後選取下一步。
6. 在檢閱頁面上，選取建立角色。

### 啟動 EC2 執行個體時並指定 IAM 角色

您可以使用 Amazon EC2 主控台或 啟動具有 IAM 角色的 Amazon EC2 執行個體 適用於 Java 的 AWS SDK。

- 若要使用主控台啟動 Amazon EC2 執行個體，請遵循 Amazon EC2 Linux 執行個體使用者指南中 Linux 執行個體入門的指示。 Amazon EC2

  當您到達 Review Instance Launch (檢閱執行個體啟動) 頁面時，選取 Edit instance details (編輯執行個體詳細資訊)。在 IAM 角色中，選擇您先前建立的 IAM 角色。依照指示完成程序。

  > **Note**
  >
  > 您需要建立或使用現有的安全群組與金鑰對，以連接到執行個體。

- 若要使用 啟動具有 IAM 角色的 Amazon EC2 執行個體 適用於 Java 的 AWS SDK，請參閱執行 Amazon EC2 執行個體。

建立您的應用程式

讓我們建置要在 EC2 執行個體上執行的範例應用程式。首先，建立可用來保存教學課程檔案的目錄（例如 `GetS3ObjectApp`）。

接著，將 適用於 Java 的 AWS SDK 程式庫複製到新建立的目錄。如果您將 下載 適用於 Java 的 AWS SDK 到您的`~/Downloads`目錄，您可以使用下列命令來複製它們：

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

開啟新檔案、呼叫 `GetS3Object.java`，然後新增下列程式碼：

```
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();
  }
}
```

開啟新檔案、呼叫 build.xml，然後新增下列行：

```
<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>
```

建置並執行修改過的程式。請注意，程式中不會儲存任何登入資料。因此，除非您已指定 AWS 登入資料，否則程式碼會擲回 AmazonServiceException。例如：

```
$ 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
```

將編譯的程式傳輸至您的 EC2 執行個體

使用安全複本 () 以及 適用於 Java 的 AWS SDK 程式庫，將程式傳輸至您的 Amazon EC2 執行個體。命令的序列如下所示。

```
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**
>
> 根據您使用的 Linux 發行版本，使用者名稱可能是 "ec2-user"、"root" 或 "ubuntu"。若要取得執行個體的公有 DNS 名稱，請開啟 EC2 主控台，並在描述索引標籤中尋找公有 DNS 值（例如，ec2-198-51-100-1.compute-1.amazonaws.com)。

在上述命令中：

- `GetS3Object.class` 是您編譯的程式
- `build.xml` 是用來建置和執行程式的 ant 檔案
- `lib` 和 `third-party`目錄是來自 的對應程式庫資料夾 適用於 Java 的 AWS SDK。
- `-r` 切換表示 `scp`應該對 適用於 Java 的 AWS SDK 分佈中 `library`和 `third-party`目錄的所有內容進行遞迴複製。
- `-p` 切換會指出 `scp`應在來源檔案複製到目的地時保留其許可。

> **Note**
>
> `-p` 交換器僅適用於 Linux、macOS 或 Unix。如果您要從 Windows 複製檔案，您可能需要使用以下命令修正執行個體上的檔案許可：

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

在 EC2 執行個體上執行範例程式

若要執行程式，請連線至您的 Amazon EC2 執行個體。如需詳細資訊，請參閱《Linux 執行個體使用者指南》中的連線至您的 Linux 執行個體。 Amazon EC2

如果您的執行個體 **`ant`** 無法使用 ，請使用下列命令安裝它：

```
sudo yum install ant
```

然後，使用 執行程式`ant`，如下所示：

```
ant run
```

程式會將 Amazon S3 物件的內容寫入命令視窗。

# 教學課程： Amazon EC2 Spot 執行個體

## 概觀

競價型執行個體可讓您以最高 90% 的未使用 Amazon Elastic Compute Cloud (Amazon EC2) 容量競價，並執行取得的執行個體，只要您的競價超過目前的 Spot 價格。 會根據供需定期 Amazon EC2 變

更競價型價格，且競價符合或超過該價格的客戶可以存取可用的 Spot 執行個體。如同隨需執行個體和預留執行個體，Spot 執行個體為您提供另一個選項，讓您取得更多運算容量。

Spot 執行個體可以大幅降低批次處理、科學研究、影像處理、影片編碼、資料和網路爬取、財務分析和測試 Amazon EC2 的成本。此外，Spot 執行個體可讓您在不需要該容量時，存取大量的額外容量。

若要使用 Spot 執行個體，您可以提出 Spot 執行個體請求，並指定您願意支付每一個小時使用一個執行個體的最高預算；此為您的出價。如果您出價符合或超出目前競價型價格，則會履行您的請求，您的執行個體將會執行，直到選擇終止或競價型價格增加到高於出價 (以先到者為準)。

請務必注意：

- 您通常每小時支付的費用比出價低。 會在請求進來時定期 Amazon EC2 調整 Spot 價格，並提供可用的供應變更。無論出價多高，在該期間每個人支付相同的 Spot 價格。因此，您支付的費用可能會低於您的出價，但永遠不會超過您的出價。
- 如果您正在執行 Spot 執行個體，且您的出價不再符合或超過目前的 Spot 價格，您的執行個體將會終止。這表示您會希望確保您的工作負載和應用程式有足夠的彈性，以利用此機會容量。

Spot 執行個體在執行時與其他 Amazon EC2 執行個體執行完全相同，而與其他 Amazon EC2 執行個體一樣，當您不再需要 Spot 執行個體時，可以終止這些執行個體。如果您終止了您的執行個體，不足一小時則按一小時支付費用，就像使用 (如您為隨需執行個體或預留執行個體所做的那樣)。不過，如果 Spot 價格高於您的出價，且您的執行個體被 終止 Amazon EC2，則不會向您收取任何部分使用時數的費用。

本教學課程說明如何使用 適用於 Java 的 AWS SDK 執行下列動作。

- 提交 Spot 請求
- 判斷 Spot 請求何時履行
- 取消 Spot 請求
- 終止關聯的執行個體

## 先決條件

若要使用此教學課程，您必須 適用於 Java 的 AWS SDK 安裝 ，並符合其基本安裝先決條件。如需詳細資訊，請參閱設定 適用於 Java 的 AWS SDK 。

### 步驟 1：設定您的登入資料

若要開始使用此程式碼範例，您需要設定 AWS 登入資料。如需如何執行此作業的說明，請參閱設定開發的 AWS 登入資料和區域。

> **Note**
>
> 我們建議您使用 IAM 使用者的登入資料來提供這些值。如需詳細資訊，請參閱註冊 AWS 和建立 IAM 使用者。

現在您已設定 設定，您可以開始使用範例中的程式碼。

### 步驟 2：設定安全群組

安全群組可做為防火牆，控制允許進出一組執行個體的流量。根據預設，執行個體會在沒有任何安全群組的情況下啟動，這表示在任何 TCP 連接埠上的所有傳入 IP 流量都會遭到拒絕。因此，在提交 Spot 請求之前，我們會設定允許必要網路流量的安全群組。基於本教學的目的，我們將建立新的安全群組，稱為「GettingStarted」，允許來自您執行應用程式的 IP 地址的 Secure Shell (SSH) 流量。若要設定新的安全群組，您需要包含或執行下列程式碼範例，以程式設計方式設定安全群組。

建立AmazonEC2用戶端物件之後，我們會建立名稱為「GettingStarted」的`CreateSecurityGroupRequest`物件，以及安全群組的描述。然後，我們呼叫 `ec2.createSecurityGroup` API 來建立群組。

為了啟用對群組的存取，我們會建立 IP 地址範圍設為本機電腦子網路 CIDR 表示的`ipPermission`物件；IP 地址上的 "/10" 尾碼表示指定 IP 地址的子網路。我們也使用 TCP 通訊協定和連接埠 22 (SSH) 設定`ipPermission`物件。最後一個步驟是`ec2.authorizeSecurityGroupIngress`使用安全群組的名稱和 `ipPermission` 物件呼叫 。

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

請注意，您只需要執行此應用程式一次，即可建立新的安全群組。

您也可以使用 建立安全群組 AWS Toolkit for Eclipse。如需詳細資訊，請參閱從 管理安全群組 AWS Cost Explorer。

## 步驟 3：提交 Spot 請求

若要提交 Spot 請求，您必須先判斷要使用的執行個體類型、Amazon Machine Image (AMI) 和最高出價。您還必須包含我們先前設定的安全群組，以便您可以視需要登入執行個體。

有多種執行個體類型可供選擇；請前往 Amazon EC2 執行個體類型以取得完整清單。在本教學課程中，我們將使用 t1.micro，這是最便宜的可用執行個體類型。接下來，我們將決定要使用的 AMI 類型。我們將使用 ami-a9d09ed1，這是撰寫本教學課程時可用的up-to-date Amazon Linux AMI。最新的 AMI 可能會隨著時間而變更，但您可以始終遵循以下步驟來確定最新版本的 AMI：

1. 開啟 Amazon EC2 主控台。
2. 選擇啟動執行個體按鈕。
3. 第一個視窗會顯示可用的 AMIs。AMI ID 會列在每個 AMI 標題旁。或者，您可以使用 `DescribeImages` API，但利用該命令超出本教學課程的範圍。

有多種方法可以對 Spot 執行個體進行競價；若要全面了解各種方法，您應該檢視競價 Spot 執行個體影片。不過，若要開始使用，我們將描述三種常見策略：競價以確保成本低於隨需定價；根據產生的運算值競價；競價以盡快取得運算容量。

- 降低隨需成本 您有批次處理任務，需要數小時或數天才能執行。不過，在啟動和完成時，您具有彈性。您想要查看是否可以以比隨需執行個體更低的成本完成它。您可以使用 AWS 管理主控台 或 Amazon EC2 API 來檢查執行個體類型的 Spot 價格歷史記錄。如需詳細資訊，請至 查閱 Spot 歷史價格。在指定可用區域中分析所需執行個體類型的價格歷史記錄之後，您有兩種替代的競價方法：
  - 您可以出價此 Spot 價格範圍的上限 (仍低於隨需執行個體的價格)、期待您的一次性 Spot 要求可以履行並執行，有一段足夠的連續時間完成工作。
  - 或者，您可以指定您願意為 Spot 執行個體支付的金額，做為隨需執行個體價格 的 %，並計劃透過持久性請求結合隨時間啟動的許多執行個體。如果超過指定的價格，則 Spot 執行個體將會終止。(我們將說明如何使這個任務自動進行。)
- 支付不超過結果的值 您有要執行的資料處理任務。您很了解此任務結果的價值，換言之您知道成本為多少。分析執行個體類型的 Spot 價格歷史記錄後，您可以選擇出價，計算時間的成本不超過任務結果的值。您建立可以長久出價的方式，且允許它間歇性地執行，當 Spot 價格出現波動，或 Spot 價格低於您的出價時。
- 快速取得運算容量 您對無法透過隨需執行個體取得的額外容量有非預期的短期需求。分析執行個體類型的 Spot 價格歷史記錄後，您競價高於最高歷史價格，以提供快速履行請求的高度可能性，並繼續運算，直到完成為止。

在您選擇您的出價金額後，您可以要求 Spot 執行個體了。在本教學課程中，我們將出價隨需價格 (0.03 USD)，以最大限度地提高完成出價的機會。您可以前往 Amazon EC2 定價頁面，判斷可用執行個體的類型和執行個體的隨需價格。當 Spot 執行個體執行時，您需要支付執行個體執行期間生效的 Spot 價格。Spot 執行個體價格由 設定 Amazon EC2 ，並根據 Spot 執行個體容量供需的長期趨勢逐步調整。您也可以指定您願意為 Spot 執行個體支付的金額，做為隨需執行個體 price.To 請求 Spot 執行個體的 %，您只需要使用您稍早選擇的參數來建置您的請求即可。我們從建立`RequestSpotInstanceRequest`物件開始。請求物件需要您要啟動的執行個體數量和出價。此外，您需要`LaunchSpecification`為請求設定 ，其中包含要使用的執行個體類型、AMI ID 和安全群組。填入請求後，您可以在 `AmazonEC2Client` 物件上呼叫 `requestSpotInstances`方法。下列範例示範如何請求 Spot 執行個體。

```
// 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);
```

執行此程式碼將啟動新的 Spot 執行個體請求。您可以使用其他選項來設定 Spot 請求。若要進一步了解，請造訪 適用於 Java 的 AWS SDK API 參考中的教學課程：進階 Amazon EC2 Spot 請求管理或 RequestSpotInstances 類別。

> **Note**
>
> 您將需要支付實際啟動的任何 Spot 執行個體的費用，因此請務必取消任何請求並終止啟動的任何執行個體，以減少任何相關費用。

### 步驟 4：判斷 Spot 請求的狀態

接下來，我們希望建立程式碼以等待 Spot 請求達到「作用中」狀態，再繼續進行最後一個步驟。為了判斷 Spot 請求的狀態，我們會輪詢 describeSpotInstanceRequests 方法，以取得要監控的 Spot 請求 ID 狀態。

在步驟 2 中建立的請求 ID 內嵌在我們`requestSpotInstances`請求的回應中。下列範例程式碼示範如何從`requestSpotInstances`回應中收集請求 IDs，並使用它們來填入 `ArrayList`。

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

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

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

若要監控您的請求 ID，請呼叫 `describeSpotInstanceRequests`方法來判斷請求的狀態。然後循環直到請求未處於「開啟」狀態。請注意，我們監控的狀態不是「開啟」，而是「作用中」，因為如果您的請求引數有問題，請求可能會直接進入「關閉」。下列程式碼範例提供如何完成此任務的詳細資訊。

```
// 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);
```

執行此程式碼後，您的 Spot 執行個體請求將已完成或失敗，並出現錯誤，將輸出至畫面。在任何一種情況下，我們都可以繼續進行下一個步驟，以清除任何作用中的請求並終止任何執行中的執行個體。

## 步驟 5：清除 Spot 請求和執行個體

最後，我們需要清除請求和執行個體。請務必同時取消任何未完成的請求並終止任何執行個體。只要取消請求，您的執行個體就不會終止，這表示您將繼續支付這些請求的費用。如果您終止執行個體，Spot 請求可能會遭到取消，但在某些情況下，例如如果您使用持續競價，而終止執行個體不足以阻止您的請求重新履行。因此，最好的方式是取消所有作用中的競價和終止所有執行中的執行個體。

下列程式碼示範如何取消您的請求。

```
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());
}
```

若要終止任何未完成的執行個體，您將需要與啟動執行個體的請求相關聯的執行個體 ID。下列程式碼範例採用原始程式碼來監控執行個體，並新增 `ArrayList`，其中存放與`describeInstance`回應相關聯的執行個體 ID。

```
// 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);
```

使用存放在 中的執行個體 IDsArrayList，使用以下程式碼片段終止任何執行中的執行個體。

```
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());
}
```

### 將一切結合在一起

為了整合所有這些，我們提供更以物件為導向的方法，結合先前顯示的步驟：初始化 EC2 用戶端、提交 Spot 請求、判斷 Spot 請求何時不再處於開啟狀態，以及清除任何保留的 Spot 請求和相關聯的執行個體。我們建立名為 的類別Requests，以執行這些動作。

我們也建立一個GettingStartedApp類別，它具有主要方法，用於執行高階函數呼叫。具體而言，我們會初始化前述的Requests物件。我們提交 Spot 執行個體請求。然後，我們等待 Spot 請求達到「作用中」狀態。最後，我們會清除請求和執行個體。

您可以在 GitHub 檢視或下載此範例的完整原始程式碼。

恭喜您！您剛完成使用 開發 Spot 執行個體軟體的入門教學課程 適用於 Java 的 AWS SDK。

### 後續步驟

繼續教學課程：進階 Amazon EC2 Spot 請求管理。

## 教學課程：進階 Amazon EC2 Spot 請求管理

Amazon EC2 Spot 執行個體可讓您競價未使用的 Amazon EC2 容量，並在競價超過目前 Spot 價格時執行這些執行個體。 會根據供需定期 Amazon EC2 變更 Spot 價格。如需 Spot 執行個體的詳細資訊，請參閱《Linux 執行個體使用者指南》中的 Spot 執行個體。 Amazon EC2

## 先決條件

若要使用此教學課程，您必須 適用於 Java 的 AWS SDK 安裝 ，並符合其基本安裝先決條件。如需詳細資訊，請參閱設定 適用於 Java 的 AWS SDK 。

## 設定您的登入資料

若要開始使用此程式碼範例，您需要設定 AWS 登入資料。如需如何執行此作業的說明，請參閱設定開發的 AWS 登入資料和區域。

> **Note**
>
> 我們建議您使用 IAM 使用者的登入資料來提供這些值。如需詳細資訊，請參閱註冊 AWS 和建立 IAM 使用者。

現在您已設定好設定，您可以開始使用範例中的程式碼。

## 設定安全群組

安全群組可做為防火牆，控制允許進出一組執行個體的流量。根據預設，執行個體會在沒有任何安全群組的情況下啟動，這表示在任何 TCP 連接埠上的所有傳入 IP 流量都會遭到拒絕。因此，在提交 Spot 請求之前，我們會設定允許必要網路流量的安全群組。基於本教學的目的，我們將建立新的安全群組，名為「GettingStarted」，允許來自您執行應用程式的 IP 地址的安全殼層 (SSH) 流量。若要設定新的安全群組，您需要包含或執行下列程式碼範例，以程式設計方式設定安全群組。

建立AmazonEC2用戶端物件之後，我們會建立名稱為「GettingStarted」的`CreateSecurityGroupRequest`物件，以及安全群組的描述。然後，我們呼叫 `ec2.createSecurityGroup` API 來建立 群組。

為了啟用群組的存取，我們會建立 IP 地址範圍設為本機電腦子網路 CIDR 表示法的`ipPermission`物件；IP 地址上的 "/10" 尾碼表示指定 IP 地址的子網路。我們也使用 TCP 通訊協定和連接埠 22 (SSH) 設定`ipPermission`物件。最後一個步驟是`ec2 .authorizeSecurityGroupIngress`使用安全群組的名稱和 `ipPermission` 物件呼叫 。

（下列程式碼與我們在第一個教學課程中所使用的程式碼相同。）

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

您可以在程式碼範例中檢視整個advanced.CreateSecurityGroupApp.java程式碼範例。請注意，您只需要執行此應用程式一次，即可建立新的安全群組。

> **Note**
>
> 您也可以使用 建立安全群組 AWS Toolkit for Eclipse。如需詳細資訊，請參閱 AWS Toolkit for Eclipse 《 使用者指南》中的從 管理安全群組 AWS Cost Explorer。

## 詳細的 Spot 執行個體請求建立選項

如我們在教學課程： Amazon EC2 Spot 執行個體中所說明，您需要使用執行個體類型、Amazon Machine Image (AMI) 和最高出價來建置請求。

讓我們從建立RequestSpotInstanceRequest物件開始。請求物件需要您想要的執行個體數量和出價。此外，我們需要LaunchSpecification為 請求設定 ，其中包括您想要使用的執行個體類型、AMI ID 和安全群組。填入請求後，我們會在 AmazonEC2Client 物件上呼叫requestSpotInstances方法。以下範例說明如何請求 Spot 執行個體。

( 下列程式碼與我們在第一個教學課程中所使用的程式碼相同。)

```
// 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);
```

## 持久性與一次性請求

建置 Spot 請求時，您可以指定數個選用參數。首先是您的請求是一次性還是持久性。根據預設，這是一次性請求。一次性請求只能履行一次，在請求的執行個體終止後，請求將會關閉。每當相同請求沒有執行的 Spot 執行個體時，就會考慮持續請求。若要指定請求類型，您只需在 Spot 請求上設定類型。這可以使用下列程式碼來完成。

```
// 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);
```

## 限制請求的持續時間

您也可以選擇性地指定請求將保持有效的時間長度。您可以指定此期間的開始和結束時間。根據預設，Spot 請求將從建立開始考慮履行，直到您履行或取消為止。不過，如有需要，您可以限制有效期間。以下程式碼顯示如何指定此期間的範例。

```
// 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);
```

## 分組 Spot Amazon EC2 執行個體請求

您可以選擇以數種不同的方式分組 Spot 執行個體請求。我們將探討使用啟動群組、可用區域群組和置放群組的優點。

如果您想要確保 Spot 執行個體一起啟動和終止，則可以選擇利用啟動群組。啟動群組是將一組出價分組在一起的標籤。啟動群組中的所有執行個體會同時啟動和終止。請注意，如果已滿足啟動群組中的執行個體，則無法保證也會滿足使用相同啟動群組啟動的新執行個體。下列程式碼範例顯示如何設定啟動群組的範例。

```
// 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);
```

如果您想要確保在相同可用區域中啟動請求中的所有執行個體，而且您不在乎哪個執行個體，您可以利用可用區域群組。可用區域群組是將一組執行個體分組在相同可用區域中的標籤。所有共用可用區域群組且同時履行的執行個體都會在相同的可用區域中開始。以下範例說明如何設定可用區域群組。

```
// 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);
```

您可以指定 Spot 執行個體所需的可用區域。下列程式碼範例示範如何設定可用區域。

```
// 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);
```

最後，如果您使用的是高效能運算 (HPC) Spot 執行個體，例如叢集運算執行個體或叢集 GPU 執行個體，您可以指定置放群組。置放群組可在執行個體之間提供較低的延遲和高頻寬連線。以下範例說明如何設定置放群組。

```
// 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);
```

本節顯示的所有參數都是選用的。也請務必了解，除了您的出價是一次性還是持久性之外，大多數這些參數都可能降低出價履行的可能性。因此，只有在您需要時才利用這些選項非常重要。上述所有程式碼範例都會合併為一個長程式碼範例，可在 `com.amazonaws.codesamples.advanced.InlineGettingStartedCodeSampleApp.java`類別中找到。

## 如何在中斷或終止後保留根分割區

管理 Spot 執行個體中斷最簡單的方式之一，就是確保您的資料定期檢查點至 Amazon Elastic Block Store (Amazon Amazon EBS) 磁碟區。透過定期檢查點，如果發生中斷，您只會遺失自上次檢查點之後建立的資料（假設之間沒有執行其他非等冪動作）。若要讓此程序更簡單，您可以設定 Spot 請求，以確保根分割區不會在中斷或終止時遭到刪除。我們已在下列範例中插入新的程式碼，示範如何啟用此案例。

在新增的程式碼中，我們會建立`BlockDeviceMapping`物件，並將其 associated Amazon Elastic Block Store (Amazon EBS) 設定為在 Spot 執行個體終止時，已設定`not`刪除的 Amazon EBS 物件。然後`BlockDeviceMapping`，我們會將此新增至我們在啟動規格中包含的映射 ArrayList。

```
// 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);
```

假設您想要在啟動時將此磁碟區重新連接至執行個體，您也可以使用區塊型設備映射設定。或者，如果您連接了非根分割區，您可以在 Spot 執行個體恢復後指定要連接到 Spot 執行個體的 Amazon Amazon EBS 磁碟區。您只需在 中指定快照 ID，`EbsBlockDevice`並在 `BlockDeviceMapping` 物件中指定替代裝置名稱即可。透過利用區塊型設備映射，您可以更輕鬆地引導執行個體。

使用根分割區來檢查點您的關鍵資料，是管理執行個體中斷可能性的絕佳方式。如需管理中斷可能性的更多方法，請造訪管理中斷影片。

## 如何標記 Spot 請求和執行個體

將標籤新增至 Amazon EC2 資源可以簡化雲端基礎設施的管理。標籤是一種中繼資料形式，可用來建立易用的名稱、增強可搜尋性，並改善多個使用者之間的協調性。您也可以使用標籤來自動化指令碼和部分程序。若要進一步了解標記 Amazon EC2 資源，請前往《Linux 執行個體 Amazon EC2 使用者指南》中的使用標籤。

### 標記 請求

若要將標籤新增至 Spot 請求，您需要在請求之後標記它們。的傳回值`requestSpotInstances()`為您提供 RequestSpotInstancesResult 物件，您可以用來取得標記的 Spot IDs：

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

```
}
```

擁有 IDs後，您可以將請求IDs 新增至 CreateTagsRequest 並呼叫 Amazon EC2 用戶端的 `createTags()`方法，以標記請求：

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

標記執行個體

與 spot 請求本身類似，您只能在執行個體建立後標記執行個體，這會在 Spot 請求滿足後發生（不再處於開啟狀態）。

您可以使用 DescribeSpotInstanceRequestsRequest 物件呼叫 Amazon EC2 用戶端的 `describeSpotInstanceRequests()`方法，來檢查請求的狀態。傳回的 DescribeSpotInstanceRequestsResult 物件包含 SpotInstanceRequest 物件清單，您可以用來查詢 Spot 請求的狀態，並在執行個體不再處於開啟狀態時取得其執行個體 IDs。

一旦 Spot 請求不再開啟，您可以透過呼叫其`getInstanceId()`方法，從`SpotInstanceRequest`物件擷取其執行個體 ID。

```
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);
```

現在您可以標記傳回的執行個體：

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

## 取消 Spot 請求和終止執行個體

### 取消 Spot 請求

若要取消 Spot 執行個體請求，請在 Amazon EC2 用戶端`cancelSpotInstanceRequests`上使用 CancelSpotInstanceRequestsRequest 物件呼叫 。

```
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());
}
```

終止 Spot 執行個體

您可以透過將任何正在執行的 Spot 執行個體 IDs傳遞至 Amazon EC2 用戶端的 `terminateInstances()`方法來終止這些執行個體。

```
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());
}
```

### 全部整合

為了將這些整合在一起，我們提供更以物件為導向的方法，將我們在本教學課程中顯示的步驟合併為一個易於使用的類別。我們會執行個體化名為 的類別`Requests`，以執行這些動作。我們也建立一個`GettingStartedApp`類別，它具有主要方法，用於執行高階函數呼叫。

您可以在 GitHub 檢視或下載此範例的完整原始程式碼。

恭喜您！您已完成使用 開發 Spot 執行個體軟體的進階請求功能教學課程 適用於 Java 的 AWS SDK。

## 管理 Amazon EC2 執行個體

### 建立執行個體

透過呼叫 AmazonEC2Client 的 `runInstances`方法建立新的 Amazon EC2 執行個體，提供 RunInstancesRequest，其中包含要使用的 Amazon Machine Image (AMI) 和執行個體類型。

匯入

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

Code

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

RunInstancesResult run_response = ec2.runInstances(run_request);

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

請參閱完整範例。

## 啟動執行個體

若要啟動 Amazon EC2 執行個體，請呼叫 AmazonEC2Client 的 `startInstances`方法，為其提供 StartInstancesRequest，其中包含要啟動的執行個體 ID。

匯入

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

Code

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

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

ec2.startInstances(request);
```

請參閱完整範例。

## 停止執行個體

若要停止 Amazon EC2 執行個體，請呼叫 AmazonEC2Client 的 `stopInstances`方法，為其提供 StopInstancesRequest，其中包含要停止的執行個體 ID。

匯入

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

Code

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

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

ec2.stopInstances(request);
```

請參閱完整範例。

## 重新啟動執行個體

若要重新啟動 Amazon EC2 執行個體，請呼叫 AmazonEC2Client 的 `rebootInstances`方法，為其提供 RebootInstancesRequest，其中包含要重新啟動的執行個體 ID。

匯入

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

Code

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

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

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

請參閱完整範例。

## 描述執行個體

若要列出您的執行個體，請建立 DescribeInstancesRequest 並呼叫 AmazonEC2Client 的 `describeInstances`方法。它會傳回 DescribeInstancesResult 物件，您可以用來列出您帳戶和區域的 Amazon EC2 執行個體。

執行個體依照保留分組。每個保留對應到呼叫 `startInstances`，用以啟動執行個體。若要列出您的執行個體，您必須先在每個傳回的預留物件`getReservations' method, and then call `getInstances`上呼叫 `DescribeInstancesResult`類別。

匯入

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

Code

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

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

    for(Reservation reservation : response.getReservations()) {
        for(Instance instance : reservation.getInstances()) {
            System.out.printf(
                "Found instance with id %s, " +
                "AMI %s, " +
                "type %s, " +
                "state %s " +
                "and monitoring state %s",
                instance.getInstanceId(),
                instance.getImageId(),
                instance.getInstanceType(),
                instance.getState().getName(),
                instance.getMonitoring().getState());
        }
```

```
    }

    request.setNextToken(response.getNextToken());

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

結果是分頁的；您可以將結果物件的 getNextToken方法傳回的值傳遞至原始請求物件的 setNextToken方法，然後在下一次呼叫 時使用相同的請求物件，以取得進一步的結果describeInstances。

請參閱完整範例。

## 監控執行個體

您可以監控 Amazon EC2 執行個體的各個層面，例如 CPU 和網路使用率、可用的記憶體，以及剩餘的磁碟空間。若要進一步了解執行個體監控，請參閱《Linux 執行個體 Amazon EC2 使用者指南》中的監控 Amazon EC2。

若要開始監控執行個體，您必須使用要監控的執行個體 ID 建立 MonitorInstancesRequest，並將其傳遞至 AmazonEC2Client 的 monitorInstances方法。

匯入

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

Code

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

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

ec2.monitorInstances(request);
```

請參閱完整範例。

## 停止執行個體監控

若要停止監控執行個體，請使用執行個體的 ID 建立 UnmonitorInstancesRequest 以停止監控，並將其傳遞至 AmazonEC2Client 的 unmonitorInstances方法。

匯入

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

Code

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

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

ec2.unmonitorInstances(request);
```

請參閱完整範例。

## 詳細資訊

- Amazon EC2 API 參考中的 RunInstances
- Amazon EC2 API 參考中的 DescribeInstances
- Amazon EC2 API 參考中的 StartInstances
- Amazon EC2 API 參考中的 StopInstances
- API Amazon EC2 參考中的 RebootInstances
- Amazon EC2 API 參考中的 MonitorInstances
- Amazon EC2 API 參考中的 UnmonitorInstances

# 在 中使用彈性 IP 地址 Amazon EC2

EC2-Classic 正在淘汰

> **Warning**
>
> 我們將於 2022 年 8 月 15 日淘汰 EC2-Classic。建議您從 EC2-Classic 遷移至 VPC。如需詳細資訊，請參閱部落格文章 EC2-Classic-Classic Networking 正在淘汰 – 以下說明如何準備。

## 配置彈性 IP 地址

若要使用彈性 IP 位址，您可以先將一個地址配置給帳戶，再將其與您的執行個體或網路介面建立關聯。

若要配置彈性 IP 地址，請使用包含 網路類型的 AllocateAddressRequest 物件 （ 傳統 EC2 或 VPC) 呼叫 AmazonEC2Client 的 `allocateAddress`方法。

傳回的 AllocateAddressResult 包含配置 ID，您可以透過將 AssociateAddressRequest 中的配置 ID 和執行個體 ID 傳遞至 AmazonEC2Client 的 `associateAddress`方法，來用來將地址與執行個體建立關聯。

匯入

```
import com.amazonaws.services.ec2.AmazonEC2;
import com.amazonaws.services.ec2.AmazonEC2ClientBuilder;
import com.amazonaws.services.ec2.model.AllocateAddressRequest;
import com.amazonaws.services.ec2.model.AllocateAddressResult;
import com.amazonaws.services.ec2.model.AssociateAddressRequest;
import com.amazonaws.services.ec2.model.AssociateAddressResult;
import com.amazonaws.services.ec2.model.DomainType;
```

Code

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

AllocateAddressRequest allocate_request = new AllocateAddressRequest()
    .withDomain(DomainType.Vpc);

AllocateAddressResult allocate_response =
    ec2.allocateAddress(allocate_request);
```

```
String allocation_id = allocate_response.getAllocationId();

AssociateAddressRequest associate_request =
    new AssociateAddressRequest()
        .withInstanceId(instance_id)
        .withAllocationId(allocation_id);

AssociateAddressResult associate_response =
    ec2.associateAddress(associate_request);
```

請參閱完整範例。

## 描述彈性 IP 地址

若要列出指派給您帳戶的彈性 IP 地址，請呼叫 AmazonEC2Client 的 `describeAddresses`方法。它會傳回 DescribeAddressesResult，您可以使用它來取得代表您帳戶中彈性 IP 地址的位址物件清單。

匯入

```
import com.amazonaws.services.ec2.AmazonEC2;
import com.amazonaws.services.ec2.AmazonEC2ClientBuilder;
import com.amazonaws.services.ec2.model.Address;
import com.amazonaws.services.ec2.model.DescribeAddressesResult;
```

Code

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

DescribeAddressesResult response = ec2.describeAddresses();

for(Address address : response.getAddresses()) {
    System.out.printf(
            "Found address with public IP %s, " +
            "domain %s, " +
            "allocation id %s " +
            "and NIC id %s",
            address.getPublicIp(),
            address.getDomain(),
            address.getAllocationId(),
            address.getNetworkInterfaceId());
}
```

請參閱完整範例。

## 釋放彈性 IP 地址

若要釋出彈性 IP 地址，請呼叫 AmazonEC2Client 的 `releaseAddress`方法，將包含您要釋出之彈性 IP 地址配置 ID 的 ReleaseAddressRequest 傳遞給它。

匯入

```
import com.amazonaws.services.ec2.AmazonEC2;
import com.amazonaws.services.ec2.AmazonEC2ClientBuilder;
import com.amazonaws.services.ec2.model.ReleaseAddressRequest;
import com.amazonaws.services.ec2.model.ReleaseAddressResult;
```

Code

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

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

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

釋出彈性 IP 地址後，該地址會釋出到 AWS IP 地址集區，您之後可能無法使用。請務必更新您的 DNS 記錄以及與該地址通訊的任何伺服器或裝置。如果您嘗試釋出已釋出的彈性 IP 地址，如果地址已配置給另一個 ，則會收到 AuthFailure 錯誤 AWS 帳戶。

如果您使用 EC2-Classic 或預設 VPC，則釋出彈性 IP 地址會自動將其與任何已關聯的執行個體取消關聯。若要取消與彈性 IP 地址的關聯而不將其釋出，請使用 AmazonEC2Client 的 `disassociateAddress`方法。

如果您是使用非預設 VPC，在嘗試釋出該彈性 IP 地址前，您必須先使用 `disassociateAddress` 將其取消關聯。否則， 會 Amazon EC2 傳回錯誤 (InvalidIPAddress.InUse)。

請參閱完整範例。

## 詳細資訊

- Linux 執行個體 Amazon EC2 使用者指南中的彈性 IP 地址
- Amazon EC2 API 參考中的 AllocateAddress
- Amazon EC2 API 參考中的 DescribeAddresses
- Amazon EC2 API 參考中的 ReleaseAddress

# 使用區域和可用區域

## 描述區域

若要列出您的帳戶可用的區域，請呼叫 AmazonEC2Client 的 `describeRegions`方法。它會傳回 DescribeRegionsResult。呼叫傳回物件的 `getRegions` 方法以取得代表每個區域的 Region 物件清單。

匯入

```
import com.amazonaws.services.ec2.AmazonEC2;
import com.amazonaws.services.ec2.AmazonEC2ClientBuilder;
import com.amazonaws.services.ec2.model.DescribeRegionsResult;
import com.amazonaws.services.ec2.model.Region;
import com.amazonaws.services.ec2.model.AvailabilityZone;
import com.amazonaws.services.ec2.model.DescribeAvailabilityZonesResult;
```

Code

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

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

請參閱完整範例。

## 描述可用區域

若要列出您帳戶可用的每個可用區域，請呼叫 AmazonEC2Client 的 `describeAvailabilityZones`方法。它會傳回 DescribeAvailabilityZonesResult。呼叫其 `getAvailabilityZones` 方法以取得代表每個可用區域的 AvailabilityZone 物件清單。

匯入

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

```
import com.amazonaws.services.ec2.model.Region;
import com.amazonaws.services.ec2.model.AvailabilityZone;
import com.amazonaws.services.ec2.model.DescribeAvailabilityZonesResult;
```

Code

```
DescribeAvailabilityZonesResult zones_response =
    ec2.describeAvailabilityZones();

for(AvailabilityZone zone : zones_response.getAvailabilityZones()) {
    System.out.printf(
        "Found availability zone %s " +
        "with status %s " +
        "in region %s",
        zone.getZoneName(),
        zone.getState(),
        zone.getRegionName());
}
```

請參閱完整範例。

## 描述帳戶

若要描述您的帳戶，請呼叫 AmazonEC2Client 的 `describeAccountAttributes`方法。此方法會傳回 DescribeAccountAttributesResult 物件。叫用此物件的 `getAccountAttributes` 方法來取得 AccountAttribute 物件的清單。您可以逐一查看清單以擷取 AccountAttribute 物件。

您可以透過叫用 AccountAttribute 對象的 `getAttributeValues` 方法來取得您帳戶的屬性值。此方法會傳回 AccountAttributeValue 物件的清單。您可以逐一查看第二個清單以顯示屬性值 (請參閱下列程式碼範例)。

匯入

```
import com.amazonaws.services.ec2.AmazonEC2;
import com.amazonaws.services.ec2.AmazonEC2ClientBuilder;
import com.amazonaws.services.ec2.model.AccountAttributeValue;
import com.amazonaws.services.ec2.model.DescribeAccountAttributesResult;
import com.amazonaws.services.ec2.model.AccountAttribute;
import java.util.List;
import java.util.ListIterator;
```

Code

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

try{
    DescribeAccountAttributesResult accountResults = ec2.describeAccountAttributes();
    List<AccountAttribute> accountList = accountResults.getAccountAttributes();

    for (ListIterator iter = accountList.listIterator(); iter.hasNext(); ) {

        AccountAttribute attribute = (AccountAttribute) iter.next();
        System.out.print("\n The name of the attribute is
 "+attribute.getAttributeName());
        List<AccountAttributeValue> values = attribute.getAttributeValues();

         //iterate through the attribute values
        for (ListIterator iterVals = values.listIterator(); iterVals.hasNext(); ) {
            AccountAttributeValue myValue = (AccountAttributeValue) iterVals.next();
            System.out.print("\n The value of the attribute is
 "+myValue.getAttributeValue());
        }
    }
    System.out.print("Done");
}
catch (Exception e)
{
    e.getStackTrace();
}
```

請參閱 GitHub 上的完整範例。

### 其他資訊

- Linux 執行個體 Amazon EC2 使用者指南中的區域和可用區域
- Amazon EC2 API 參考中的 DescribeRegions
- Amazon EC2 API 參考中的 DescribeAvailabilityZones

## 使用 Amazon EC2 金鑰對

### 建立金鑰對

若要建立金鑰對，請使用包含金鑰名稱的 CreateKeyPairRequest 呼叫 AmazonEC2Client 的 `createKeyPair`方法。

匯入

```
import com.amazonaws.services.ec2.AmazonEC2;
import com.amazonaws.services.ec2.AmazonEC2ClientBuilder;
import com.amazonaws.services.ec2.model.CreateKeyPairRequest;
import com.amazonaws.services.ec2.model.CreateKeyPairResult;
```

Code

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

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

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

請參閱完整範例。

## 描述金鑰對

若要列出金鑰對或取得相關資訊，請呼叫 AmazonEC2Client 的 `describeKeyPairs`方法。它會傳回 DescribeKeyPairsResult，您可以透過呼叫其`getKeyPairs`方法來存取金鑰對的清單，這會傳回 KeyPairInfo 物件的清單。

匯入

```
import com.amazonaws.services.ec2.AmazonEC2;
import com.amazonaws.services.ec2.AmazonEC2ClientBuilder;
import com.amazonaws.services.ec2.model.DescribeKeyPairsResult;
import com.amazonaws.services.ec2.model.KeyPairInfo;
```

Code

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

DescribeKeyPairsResult response = ec2.describeKeyPairs();

for(KeyPairInfo key_pair : response.getKeyPairs()) {
    System.out.printf(
        "Found key pair with name %s " +
```

```
        "and fingerprint %s",
        key_pair.getKeyName(),
        key_pair.getKeyFingerprint());
}
```

請參閱完整範例。

## 刪除金鑰對

若要刪除金鑰對，請呼叫 AmazonEC2Client 的 `deleteKeyPair`方法，並向其傳遞 DeleteKeyPairRequest，其中包含要刪除的金鑰對名稱。

匯入

```
import com.amazonaws.services.ec2.AmazonEC2;
import com.amazonaws.services.ec2.AmazonEC2ClientBuilder;
import com.amazonaws.services.ec2.model.DeleteKeyPairRequest;
import com.amazonaws.services.ec2.model.DeleteKeyPairResult;
```

Code

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

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

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

請參閱完整範例。

## 詳細資訊

- Linux 執行個體 Amazon EC2 使用者指南中的Amazon EC2 金鑰對
- Amazon EC2 API 參考中的 CreateKeyPair
- Amazon EC2 API 參考中的 DescribeKeyPairs
- Amazon EC2 API 參考中的 DeleteKeyPair

# 在 中使用安全群組 Amazon EC2

## 建立安全群組

若要建立安全群組，請使用包含金鑰名稱的 CreateSecurityGroupRequest 呼叫 AmazonEC2Client 的 `createSecurityGroup`方法。

匯入

```
import com.amazonaws.services.ec2.AmazonEC2;
import com.amazonaws.services.ec2.AmazonEC2ClientBuilder;
import com.amazonaws.services.ec2.model.CreateSecurityGroupRequest;
import com.amazonaws.services.ec2.model.CreateSecurityGroupResult;
```

Code

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

CreateSecurityGroupRequest create_request = new
    CreateSecurityGroupRequest()
        .withGroupName(group_name)
        .withDescription(group_desc)
        .withVpcId(vpc_id);

CreateSecurityGroupResult create_response =
    ec2.createSecurityGroup(create_request);
```

請參閱完整範例。

## 設定安全群組

安全群組可以控制執行個體的傳入（傳入）和傳出（傳出）流量 Amazon EC2 。

若要將輸入規則新增至您的安全群組，請使用 AmazonEC2Client 的 `authorizeSecurityGroupIngress`方法，提供安全群組的名稱，以及您想要在 AuthorizeSecurityGroupIngressRequest 物件中為其指派的存取規則 (IpPermission)。以下範例說明如何將 IP 許可新增至安全群組。

匯入

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

```
import com.amazonaws.services.ec2.model.CreateSecurityGroupRequest;
import com.amazonaws.services.ec2.model.CreateSecurityGroupResult;
```

Code

```
IpRange ip_range = new IpRange()
    .withCidrIp("0.0.0.0/0");

IpPermission ip_perm = new IpPermission()
    .withIpProtocol("tcp")
    .withToPort(80)
    .withFromPort(80)
    .withIpv4Ranges(ip_range);

IpPermission ip_perm2 = new IpPermission()
    .withIpProtocol("tcp")
    .withToPort(22)
    .withFromPort(22)
    .withIpv4Ranges(ip_range);

AuthorizeSecurityGroupIngressRequest auth_request = new
    AuthorizeSecurityGroupIngressRequest()
        .withGroupName(group_name)
        .withIpPermissions(ip_perm, ip_perm2);

AuthorizeSecurityGroupIngressResult auth_response =
    ec2.authorizeSecurityGroupIngress(auth_request);
```

若要將輸出規則新增至安全群組，請將 AuthorizeSecurityGroupEgressRequest 中的類似資料提供給 AmazonEC2Client 的 `authorizeSecurityGroupEgress`方法。

請參閱完整範例。

## 描述安全群組

若要描述您的安全群組或取得相關資訊，請呼叫 AmazonEC2Client 的 `describeSecurityGroups`方法。它會傳回 DescribeSecurityGroupsResult，您可以透過呼叫其`getSecurityGroups`方法來存取安全群組清單，這會傳回 SecurityGroup 物件清單。

匯入

```
import com.amazonaws.services.ec2.AmazonEC2;
```

```
import com.amazonaws.services.ec2.AmazonEC2ClientBuilder;
import com.amazonaws.services.ec2.model.DescribeSecurityGroupsRequest;
import com.amazonaws.services.ec2.model.DescribeSecurityGroupsResult;
```

Code

```
final String USAGE =
    "To run this example, supply a group id\n" +
    "Ex: DescribeSecurityGroups <group-id>\n";

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

String group_id = args[0];
```

請參閱完整範例。

## 刪除安全群組

若要刪除安全群組，請呼叫 AmazonEC2Client 的 `deleteSecurityGroup`方法，並向其傳遞 DeleteSecurityGroupRequest，其中包含要刪除的安全群組 ID。

匯入

```
import com.amazonaws.services.ec2.AmazonEC2;
import com.amazonaws.services.ec2.AmazonEC2ClientBuilder;
import com.amazonaws.services.ec2.model.DeleteSecurityGroupRequest;
import com.amazonaws.services.ec2.model.DeleteSecurityGroupResult;
```

Code

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

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

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

請參閱完整範例。

### 詳細資訊

- Linux 執行個體 Amazon EC2 使用者指南中的Amazon EC2 安全群組
- 《Linux 執行個體使用者指南》中的授權 Linux 執行個體的傳入流量 Amazon EC2
- Amazon EC2 API 參考中的 CreateSecurityGroup
- Amazon EC2 API 參考中的 DescribeSecurityGroups
- Amazon EC2 API 參考中的 DeleteSecurityGroup
- Amazon EC2 API 參考中的 AuthorizeSecurityGroupIngress

# 使用 的 IAM 範例 適用於 Java 的 AWS SDK

本節提供使用 程式設計 IAM 的範例適用於 Java 的 AWS SDK。

AWS Identity and Access Management (IAM) 可讓您安全地控制使用者對 AWS 服務和資源的存取。使用 IAM，您可以建立和管理 AWS 使用者和群組，並使用許可來允許和拒絕他們存取 AWS 資源。如需 IAM 的完整指南，請造訪 IAM 使用者指南。

> **Note**
>
> 這些範例僅包含示範每種技術所需的程式碼。GitHub 上提供完整程式碼範例。您可以從那裡下載單一原始檔案或將儲存庫複製到本機，以取得建置和執行的所有範例。

主題

- 管理 IAM 存取金鑰
- 管理 IAM 使用者
- 使用 IAM 帳戶別名
- 處理 IAM 政策
- 處理 IAM 伺服器憑證

# 管理 IAM 存取金鑰

## 建立存取金鑰

若要建立 IAM 存取金鑰，請使用 CreateAccessKeyRequest 物件呼叫 AmazonIdentityManagementClient`createAccessKey` 方法。

`CreateAccessKeyRequest` 有兩個建構函數：一個採用使用者名稱，另一個沒有參數。如果您使用不使用參數的版本，則必須使用 `withUserName` setter 方法設定使用者名稱，然後再將其傳遞給 `createAccessKey`方法。

匯入

```
import com.amazonaws.services.identitymanagement.AmazonIdentityManagement;
import com.amazonaws.services.identitymanagement.AmazonIdentityManagementClientBuilder;
import com.amazonaws.services.identitymanagement.model.CreateAccessKeyRequest;
import com.amazonaws.services.identitymanagement.model.CreateAccessKeyResult;
```

Code

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

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

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

請參閱 GitHub 上的完整範例。

## 列出存取金鑰

若要列出指定使用者的存取金鑰，請建立 ListAccessKeysRequest 物件，其中包含要列出金鑰的使用者名稱，並將其傳遞至 AmazonIdentityManagementClient 的 `listAccessKeys`方法。

> **Note**
>
> 如果您未提供使用者名稱給 `listAccessKeys`，則會嘗試列出與簽署請求 AWS 帳戶 之 相關聯的存取金鑰。

匯入

```
import com.amazonaws.services.identitymanagement.AmazonIdentityManagement;
import com.amazonaws.services.identitymanagement.AmazonIdentityManagementClientBuilder;
import com.amazonaws.services.identitymanagement.model.AccessKeyMetadata;
import com.amazonaws.services.identitymanagement.model.ListAccessKeysRequest;
import com.amazonaws.services.identitymanagement.model.ListAccessKeysResult;
```

Code

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

boolean done = false;
ListAccessKeysRequest request = new ListAccessKeysRequest()
        .withUserName(username);

while (!done) {

    ListAccessKeysResult response = iam.listAccessKeys(request);

    for (AccessKeyMetadata metadata :
            response.getAccessKeyMetadata()) {
        System.out.format("Retrieved access key %s",
                metadata.getAccessKeyId());
    }

    request.setMarker(response.getMarker());

    if (!response.getIsTruncated()) {
        done = true;
    }
}
```

`listAccessKeys` 的結果會分頁 (每個呼叫預設最多 100 個記錄)。您可以在傳回的 ListAccessKeysResult 物件`getIsTruncated`上呼叫 ，以查看查詢是否傳回較少的結果，則可供使用。如果是這樣，請在 `setMarker`上呼叫 ，`ListAccessKeysRequest`並將其傳遞給下一次叫用 `listAccessKeys`。

請參閱 GitHub 上的完整範例。

## 擷取存取金鑰的上次使用時間

若要取得上次使用存取金鑰的時間，請呼叫 AmazonIdentityManagementClient 的 `getAccessKeyLastUsed`方法與存取金鑰的 ID （可以使用 GetAccessKeyLastUsedRequest 物件傳入，或直接傳送到直接取得存取金鑰 ID 的過載。

然後，您可以使用傳回的 GetAccessKeyLastUsedResult 物件來擷取金鑰的上次使用時間。

匯入

```
import com.amazonaws.services.identitymanagement.AmazonIdentityManagement;
import com.amazonaws.services.identitymanagement.AmazonIdentityManagementClientBuilder;
import com.amazonaws.services.identitymanagement.model.GetAccessKeyLastUsedRequest;
import com.amazonaws.services.identitymanagement.model.GetAccessKeyLastUsedResult;
```

Code

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

GetAccessKeyLastUsedRequest request = new GetAccessKeyLastUsedRequest()
    .withAccessKeyId(access_id);

GetAccessKeyLastUsedResult response = iam.getAccessKeyLastUsed(request);

System.out.println("Access key was last used at: " +
        response.getAccessKeyLastUsed().getLastUsedDate());
```

請參閱 GitHub 上的完整範例。

## 啟用或停用存取金鑰

您可以透過建立 UpdateAccessKeyRequest 物件、提供存取金鑰 ID、選擇性地提供使用者名稱和所需狀態來啟用或停用存取金鑰，然後將請求物件傳遞至 AmazonIdentityManagementClient 的 `updateAccessKey`方法。

匯入

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

```
import com.amazonaws.services.identitymanagement.model.UpdateAccessKeyResult;
```

Code

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

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

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

請參閱 GitHub 上的完整範例。

## 刪除存取金鑰

若要永久刪除存取金鑰，請呼叫 AmazonIdentityManagementClient 的 `deleteKey`方法，為其提供 DeleteAccessKeyRequest，其中包含存取金鑰的 ID 和使用者名稱。

> **Note**
>
> 金鑰一旦刪除，就不能再擷取或使用。若要暫時停用金鑰，稍後再行啟動，請改為使用 updateAccessKey 方法。

匯入

```
import com.amazonaws.services.identitymanagement.AmazonIdentityManagement;
import com.amazonaws.services.identitymanagement.AmazonIdentityManagementClientBuilder;
import com.amazonaws.services.identitymanagement.model.DeleteAccessKeyRequest;
import com.amazonaws.services.identitymanagement.model.DeleteAccessKeyResult;
```

Code

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

DeleteAccessKeyRequest request = new DeleteAccessKeyRequest()
    .withAccessKeyId(access_key)
```

```
    .withUserName(username);

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

請參閱 GitHub 上的完整範例。

### 詳細資訊

- IAM API 參考中的 CreateAccessKey
- IAM API 參考中的 ListAccessKeys
- IAM API 參考中的 GetAccessKeyLastUsed
- IAM API 參考中的 UpdateAccessKey
- IAM API 參考中的 DeleteAccessKey

## 管理 IAM 使用者

### 建立使用者

直接或使用包含使用者名稱的 CreateUserRequest 物件，將使用者名稱提供給 AmazonIdentityManagementClient 的 `createUser`方法，以建立新的 IAM 使用者。

匯入

```
import com.amazonaws.services.identitymanagement.AmazonIdentityManagement;
import com.amazonaws.services.identitymanagement.AmazonIdentityManagementClientBuilder;
import com.amazonaws.services.identitymanagement.model.CreateUserRequest;
import com.amazonaws.services.identitymanagement.model.CreateUserResult;
```

Code

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

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

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

請參閱 GitHub 上的完整範例。

## 列出使用者

若要列出帳戶的 IAM 使用者，請建立新的 ListUsersRequest，並將其傳遞至 AmazonIdentityManagementClient 的 `listUsers`方法。您可以在傳回的 ListUsersResult 物件`getUsers`上呼叫 來擷取使用者清單。

`listUsers` 傳回的使用者清單會分頁。您可以呼叫回應物件的 `getIsTruncated` 方法，檢查是否有更多可擷取的結果。如果傳回 `true`，則呼叫請求物件的 `setMarker()`方法，傳遞回應物件`getMarker()`方法的傳回值。

匯入

```
import com.amazonaws.services.identitymanagement.AmazonIdentityManagement;
import com.amazonaws.services.identitymanagement.AmazonIdentityManagementClientBuilder;
import com.amazonaws.services.identitymanagement.model.ListUsersRequest;
import com.amazonaws.services.identitymanagement.model.ListUsersResult;
import com.amazonaws.services.identitymanagement.model.User;
```

Code

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

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

while(!done) {
    ListUsersResult response = iam.listUsers(request);

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

    request.setMarker(response.getMarker());

    if(!response.getIsTruncated()) {
        done = true;
    }
}
```

請參閱 GitHub 上的完整範例。

## 更新使用者

若要更新使用者，請呼叫 AmazonIdentityManagementClient 物件的 `updateUser`方法，該方法採用 UpdateUserRequest 物件，您可以用來變更使用者名稱或路徑。

匯入

```
import com.amazonaws.services.identitymanagement.AmazonIdentityManagement;
import com.amazonaws.services.identitymanagement.AmazonIdentityManagementClientBuilder;
import com.amazonaws.services.identitymanagement.model.UpdateUserRequest;
import com.amazonaws.services.identitymanagement.model.UpdateUserResult;
```

Code

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

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

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

請參閱 GitHub 上的完整範例。

## 刪除使用者

若要刪除使用者，請使用 UpdateUserRequest 物件集呼叫 AmazonIdentityManagementClient 的`deleteUser`請求，並設定要刪除的使用者名稱。

匯入

```
import com.amazonaws.services.identitymanagement.AmazonIdentityManagement;
import com.amazonaws.services.identitymanagement.AmazonIdentityManagementClientBuilder;
import com.amazonaws.services.identitymanagement.model.DeleteConflictException;
import com.amazonaws.services.identitymanagement.model.DeleteUserRequest;
```

Code

```
final AmazonIdentityManagement iam =
```

```
    AmazonIdentityManagementClientBuilder.defaultClient();

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

try {
    iam.deleteUser(request);
} catch (DeleteConflictException e) {
    System.out.println("Unable to delete user. Verify user is not" +
            " associated with any resources");
    throw e;
}
```

請參閱 GitHub 上的完整範例。

### 詳細資訊

- 《使用者指南》中的 IAM 使用者 IAM
- 《使用者指南》中的管理 IAM 使用者 IAM
- IAM API 參考中的 CreateUser
- IAM API 參考中的 ListUsers
- IAM API 參考中的 UpdateUser
- IAM API 參考中的 DeleteUser

## 使用 IAM 帳戶別名

如果您希望登入頁面的 URL 包含您的公司名稱或其他易記識別符，而不是您的 AWS 帳戶 ID，您可以為您的 建立別名 AWS 帳戶。

> **Note**
>
> AWS 每個帳戶僅支援一個帳戶別名。

### 建立帳戶別名

若要建立帳戶別名，請使用包含別名名稱的 CreateAccountAliasRequest 物件呼叫 AmazonIdentityManagementClient 的 `createAccountAlias`方法。

匯入

```
import com.amazonaws.services.identitymanagement.AmazonIdentityManagement;
import com.amazonaws.services.identitymanagement.AmazonIdentityManagementClientBuilder;
import com.amazonaws.services.identitymanagement.model.CreateAccountAliasRequest;
import com.amazonaws.services.identitymanagement.model.CreateAccountAliasResult;
```

Code

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

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

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

請參閱 GitHub 上的完整範例。

## 列出帳戶別名

若要列出您帳戶的別名，如果有的話，請呼叫 AmazonIdentityManagementClient 的 `listAccountAliases`方法。

> **Note**
>
> 傳回的 ListAccountAliasesResult 支援與其他 適用於 Java 的 AWS SDK 清單`getMarker`方法相同的 `getIsTruncated`和 方法，但 只能 AWS 帳戶 有一個帳戶別名。

匯入

```
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);
}
```

請參閱 GitHub 上的完整範例。

## 刪除帳戶別名

若要刪除帳戶的別名，請呼叫 AmazonIdentityManagementClient 的 `deleteAccountAlias`方法。刪除帳戶別名時，您必須使用 DeleteAccountAliasRequest 物件提供其名稱。

匯入

```
import com.amazonaws.services.identitymanagement.AmazonIdentityManagement;
import com.amazonaws.services.identitymanagement.AmazonIdentityManagementClientBuilder;
import com.amazonaws.services.identitymanagement.model.DeleteAccountAliasRequest;
import com.amazonaws.services.identitymanagement.model.DeleteAccountAliasResult;
```

Code

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

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

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

請參閱 GitHub 上的完整範例。

## 詳細資訊

- IAM 《 使用者指南》中的AWS 您的帳戶 ID 及其別名
- IAM API 參考中的 CreateAccountAlias
- IAM API 參考中的 ListAccountAliases
- IAM API 參考中的 DeleteAccountAlias

# 處理 IAM 政策

## 建立政策

若要建立新的政策，請在 CreatePolicyRequest 中提供政策的名稱和 JSON 格式的政策文件給 AmazonIdentityManagementClient 的 `createPolicy`方法。

匯入

```
import com.amazonaws.services.identitymanagement.AmazonIdentityManagement;
import com.amazonaws.services.identitymanagement.AmazonIdentityManagementClientBuilder;
import com.amazonaws.services.identitymanagement.model.CreatePolicyRequest;
import com.amazonaws.services.identitymanagement.model.CreatePolicyResult;
```

Code

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

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

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

IAM 政策文件是具有妥善記錄語法的 JSON 字串。以下範例提供存取權以便對 DynamoDB提出特定請求。

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

請參閱 GitHub 上的完整範例。

## 取得政策

若要擷取現有政策，請呼叫 AmazonIdentityManagementClient 的 `getPolicy`方法，在 GetPolicyRequest 物件中提供政策的 ARN。

匯入

```
import com.amazonaws.services.identitymanagement.AmazonIdentityManagement;
import com.amazonaws.services.identitymanagement.AmazonIdentityManagementClientBuilder;
import com.amazonaws.services.identitymanagement.model.GetPolicyRequest;
import com.amazonaws.services.identitymanagement.model.GetPolicyResult;
```

Code

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

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

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

請參閱 GitHub 上的完整範例。

## 附加角色政策

您可以透過呼叫 AmazonIdentityManagementClient 的 `attachRolePolicy`方法，在 AttachRolePolicyRequest 中提供角色名稱和政策 ARN，將政策連接到 IAMhttp：// docs.aws.amazon.com/IAM/latest/UserGuide/id_roles.html【角色】。

匯入

```
import com.amazonaws.services.identitymanagement.AmazonIdentityManagement;
import com.amazonaws.services.identitymanagement.AmazonIdentityManagementClientBuilder;
import com.amazonaws.services.identitymanagement.model.AttachRolePolicyRequest;
import com.amazonaws.services.identitymanagement.model.AttachedPolicy;
```

Code

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

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

iam.attachRolePolicy(attach_request);
```

請參閱 GitHub 上的完整範例。

## 列出附加的角色政策

呼叫 AmazonIdentityManagementClient 的 `listAttachedRolePolicies`方法，列出角色上的附加政策。它採用 ListAttachedRolePoliciesRequest 物件，其中包含要列出其政策的角色名稱。

在傳回的 ListAttachedRolePoliciesResult 物件`getAttachedPolicies`上呼叫 ，以取得連接的政策清單。結果可能會截斷；如果`ListAttachedRolePoliciesResult`物件的 `getIsTruncated`方法傳回 `true`，請呼叫`ListAttachedRolePoliciesRequest`物件的 `setMarker`方法，並使用它`listAttachedRolePolicies`再次呼叫 以取得下一批結果。

匯入

```
import com.amazonaws.services.identitymanagement.AmazonIdentityManagement;
import com.amazonaws.services.identitymanagement.AmazonIdentityManagementClientBuilder;
import com.amazonaws.services.identitymanagement.model.ListAttachedRolePoliciesRequest;
import com.amazonaws.services.identitymanagement.model.ListAttachedRolePoliciesResult;
import java.util.ArrayList;
import java.util.List;
import java.util.stream.Collectors;
```

Code

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

ListAttachedRolePoliciesRequest request =
    new ListAttachedRolePoliciesRequest()
        .withRoleName(role_name);

List<AttachedPolicy> matching_policies = new ArrayList<>();

boolean done = false;

while(!done) {
    ListAttachedRolePoliciesResult response =
        iam.listAttachedRolePolicies(request);

    matching_policies.addAll(
            response.getAttachedPolicies()
                    .stream()
                    .filter(p -> p.getPolicyName().equals(role_name))
                    .collect(Collectors.toList()));

    if(!response.getIsTruncated()) {
        done = true;
    }
    request.setMarker(response.getMarker());
}
```

請參閱 GitHub 上的完整範例。

## 分離角色政策

若要從角色分離政策，請呼叫 AmazonIdentityManagementClient 的 `detachRolePolicy`方法，在 DetachRolePolicyRequest 中提供角色名稱和政策 ARN。

**匯入**

```
import com.amazonaws.services.identitymanagement.AmazonIdentityManagement;
import com.amazonaws.services.identitymanagement.AmazonIdentityManagementClientBuilder;
import com.amazonaws.services.identitymanagement.model.DetachRolePolicyRequest;
import com.amazonaws.services.identitymanagement.model.DetachRolePolicyResult;
```

Code

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

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

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

請參閱 GitHub 上的完整範例。

### 詳細資訊

- 《使用者指南》中的 IAM 政策概觀。 IAM
- IAM 《使用者指南》中的 AWS IAM 政策參考。
- IAM API 參考中的 CreatePolicy
- IAM API 參考中的 GetPolicy
- IAM API 參考中的 AttachRolePolicy
- IAM API 參考中的 ListAttachedRolePolicies
- IAM API 參考中的 DetachRolePolicy

## 處理 IAM 伺服器憑證

若要在 上啟用網站或應用程式的 HTTPS 連線 AWS，您需要 SSL/TLS 伺服器憑證。您可以使用 AWS Certificate Manager 提供的伺服器憑證，或是從外部供應商取得的憑證。

建議您使用 ACM 來佈建、管理和部署伺服器憑證。使用 ACM，您可以請求憑證、將其部署到您的 AWS 資源，並讓 ACM 為您處理憑證續約。ACM 提供的憑證是免費的。如需 ACM 的詳細資訊，請參閱 ACM 使用者指南。

### 取得伺服器憑證

您可以透過呼叫 AmazonIdentityManagementClient 的 `getServerCertificate`方法來擷取伺服器憑證，並使用憑證的名稱傳遞 GetServerCertificateRequest。

匯入

```
import com.amazonaws.services.identitymanagement.AmazonIdentityManagement;
```

```
import com.amazonaws.services.identitymanagement.AmazonIdentityManagementClientBuilder;
import com.amazonaws.services.identitymanagement.model.GetServerCertificateRequest;
import com.amazonaws.services.identitymanagement.model.GetServerCertificateResult;
```

Code

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

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

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

請參閱 GitHub 上的完整範例。

## 列出伺服器憑證

若要列出您的伺服器憑證，請使用 ListServerCertificatesRequest 呼叫 AmazonIdentityManagementClient 的 `listServerCertificates`方法。它會傳回 ListServerCertificatesResult。

呼叫傳回 `ListServerCertificateResult` 物件的 `getServerCertificateMetadataList` 方法以取得 ServerCertificateMetadata 物件的清單，您可以用來取得每個憑證的相關資訊。

結果可能會截斷；如果`ListServerCertificateResult`物件的 `getIsTruncated`方法傳回 `true`，請呼叫`ListServerCertificatesRequest`物件的 `setMarker`方法，並用它`listServerCertificates`再次呼叫 以取得下一批結果。

匯入

```
import com.amazonaws.services.identitymanagement.AmazonIdentityManagement;
import com.amazonaws.services.identitymanagement.AmazonIdentityManagementClientBuilder;
import com.amazonaws.services.identitymanagement.model.ListServerCertificatesRequest;
import com.amazonaws.services.identitymanagement.model.ListServerCertificatesResult;
import com.amazonaws.services.identitymanagement.model.ServerCertificateMetadata;
```

Code

```
final AmazonIdentityManagement iam =
```

```
    AmazonIdentityManagementClientBuilder.defaultClient();

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

while(!done) {

    ListServerCertificatesResult response =
        iam.listServerCertificates(request);

    for(ServerCertificateMetadata metadata :
            response.getServerCertificateMetadataList()) {
        System.out.printf("Retrieved server certificate %s",
                metadata.getServerCertificateName());
    }

    request.setMarker(response.getMarker());

    if(!response.getIsTruncated()) {
        done = true;
    }
}
```

請參閱 GitHub 上的完整範例。

## 更新伺服器憑證

您可以透過呼叫 AmazonIdentityManagementClient 的 `updateServerCertificate`方法來更新伺服器憑證的名稱或路徑。它需要設定 UpdateServerCertificateRequest 物件，並搭配伺服器憑證的目前名稱以及要使用的新名稱或新路徑。

### 匯入

```
import com.amazonaws.services.identitymanagement.AmazonIdentityManagement;
import com.amazonaws.services.identitymanagement.AmazonIdentityManagementClientBuilder;
import com.amazonaws.services.identitymanagement.model.UpdateServerCertificateRequest;
import com.amazonaws.services.identitymanagement.model.UpdateServerCertificateResult;
```

### Code

```
final AmazonIdentityManagement iam =
```

```
    AmazonIdentityManagementClientBuilder.defaultClient();

UpdateServerCertificateRequest request =
    new UpdateServerCertificateRequest()
        .withServerCertificateName(cur_name)
        .withNewServerCertificateName(new_name);

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

請參閱 GitHub 上的完整範例。

## 刪除伺服器憑證

若要刪除伺服器憑證，請使用包含憑證名稱的 DeleteServerCertificateRequest 呼叫 AmazonIdentityManagementClient 的 `deleteServerCertificate`方法。

匯入

```
import com.amazonaws.services.identitymanagement.AmazonIdentityManagement;
import com.amazonaws.services.identitymanagement.AmazonIdentityManagementClientBuilder;
import com.amazonaws.services.identitymanagement.model.DeleteServerCertificateRequest;
import com.amazonaws.services.identitymanagement.model.DeleteServerCertificateResult;
```

Code

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

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

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

請參閱 GitHub 上的完整範例。

## 詳細資訊

- IAM 《使用者指南》中的使用伺服器憑證

- IAM API 參考中的 GetServerCertificate
- IAM API 參考中的 ListServerCertificates
- IAM API 參考中的 UpdateServerCertificate
- IAM API 參考中的 DeleteServerCertificate
- ACM 使用者指南

# Lambda 使用 的範例 適用於 Java 的 AWS SDK

本節提供 Lambda 使用 進行程式設計的範例 適用於 Java 的 AWS SDK。

> **Note**
>
> 這些範例僅包含示範每種技術所需的程式碼。GitHub 上提供完整程式碼範例。您可以從那裡下載單一原始檔案或將儲存庫複製到本機，以取得建置和執行的所有範例。

主題

- 叫用、列出和刪除 Lambda 函數

## 叫用、列出和刪除 Lambda 函數

本節提供使用 搭配 Lambda 服務用戶端進行程式設計的範例 適用於 Java 的 AWS SDK。若要了解如何建立 Lambda 函數，請參閱如何建立 AWS Lambda 函數。

主題

- 呼叫函數
- 列出函數
- 刪除函數

### 呼叫函數

您可以透過建立 AWSLambda 物件並叫用其`invoke`方法來叫用 Lambda 函數。建立 InvokeRequest 物件以指定其他資訊，例如函數名稱和要傳遞給 Lambda 函數的承載。函數名稱會顯示為 arn：aws：lambda：us-east-1：555556330391：function：HelloFunction。您可以查看 中的 函數來擷取值 AWS 管理主控台。

若要將承載資料傳遞至函數，請呼叫 InvokeRequest 物件的 withPayload方法，並以 JSON 格式指定字串，如下列程式碼範例所示。

匯入

```
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.lambda.AWSLambda;
import com.amazonaws.services.lambda.AWSLambdaClientBuilder;
import com.amazonaws.services.lambda.model.InvokeRequest;
import com.amazonaws.services.lambda.model.InvokeResult;
import com.amazonaws.services.lambda.model.ServiceException;

import java.nio.charset.StandardCharsets;
```

Code

下列程式碼範例示範如何叫用 Lambda 函數。

```
        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());
```

請參閱 GitHub 上的完整範例。

## 列出函數

建置 AWSLambda 物件並叫用其`listFunctions`方法。此方法會傳回 ListFunctionsResult 物件。您可以叫用此物件的 `getFunctions` 方法來傳回 FunctionConfiguration 物件的清單。您可以逐一查看清單以擷取函數的相關資訊。例如，下列 Java 程式碼範例示範如何取得每個函數名稱。

匯入

```
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.lambda.AWSLambda;
import com.amazonaws.services.lambda.AWSLambdaClientBuilder;
import com.amazonaws.services.lambda.model.FunctionConfiguration;
import com.amazonaws.services.lambda.model.ListFunctionsResult;
import com.amazonaws.services.lambda.model.ServiceException;
import java.util.Iterator;
import java.util.List;
```

Code

下列 Java 程式碼範例示範如何擷取 Lambda 函數名稱清單。

```
        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);
    }
```

請參閱 GitHub 上的完整範例。

## 刪除函數

建置 AWSLambda 物件並叫用其deleteFunction方法。建立一個 DeleteFunctionRequest 物件並將其傳遞給 deleteFunction 方法。此物件包含資訊，例如要刪除的函數名稱。函數名稱會顯示為 arn：aws：lambda：us-east-1：555556330391：function：HelloFunction。您可以透過查看 中的 函數來擷取值 AWS 管理主控台。

### 匯入

```
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.lambda.AWSLambda;
import com.amazonaws.services.lambda.AWSLambdaClientBuilder;
import com.amazonaws.services.lambda.model.ServiceException;
import com.amazonaws.services.lambda.model.DeleteFunctionRequest;
```

### Code

下列 Java 程式碼示範如何刪除 Lambda 函數。

```
    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);
        }
```

請參閱 GitHub 上的完整範例。

# Amazon Pinpoint 使用 的範例 適用於 Java 的 AWS SDK

本節提供使用適用於 Java 的 AWS SDK編寫 Amazon Pinpoint 程式的範例。

> **Note**
>
> 這些範例僅包含示範每種技術所需的程式碼。GitHub 上提供完整程式碼範例。您可以從那裡下載單一原始檔案或將儲存庫複製到本機，以取得建置和執行的所有範例。

主題

- 在 中建立和刪除應用程式 Amazon Pinpoint
- 在 中建立端點 Amazon Pinpoint
- 在 中建立客群 Amazon Pinpoint
- 在 中建立行銷活動 Amazon Pinpoint
- 在 中更新頻道 Amazon Pinpoint

## 在 中建立和刪除應用程式 Amazon Pinpoint

應用程式是一種 Amazon Pinpoint 專案，您可以在其中定義不同應用程式的對象，並以量身打造的訊息吸引此對象。本頁面上的範例示範如何建立新的應用程式或刪除現有的應用程式。

### 建立應用程式

在 中提供應用程式名稱給 CreateAppRequest 物件，然後將該物件傳遞給 AmazonPinpointClient 的 `createApp`方法， Amazon Pinpoint 以在 中建立新的應用程式。

匯入

```
import com.amazonaws.services.pinpoint.AmazonPinpoint;
import com.amazonaws.services.pinpoint.AmazonPinpointClientBuilder;
import com.amazonaws.services.pinpoint.model.CreateAppRequest;
import com.amazonaws.services.pinpoint.model.CreateAppResult;
import com.amazonaws.services.pinpoint.model.CreateApplicationRequest;
```

Code

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

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

請參閱 GitHub 上的完整範例。

## 刪除應用程式

若要刪除應用程式，請呼叫 AmazonPinpointClient 的deleteApp請求，並呼叫 DeleteAppRequest 物件，該物件已設定為要刪除的應用程式名稱。

匯入

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

Code

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

pinpoint.deleteApp(deleteRequest);
```

請參閱 GitHub 上的完整範例。

## 詳細資訊

- Amazon Pinpoint API 參考中的應用程式

- Amazon Pinpoint API 參考中的應用程式

## 在 中建立端點 Amazon Pinpoint

端點可唯一識別可向其傳送推送通知的使用者裝置 Amazon Pinpoint。如果您的應用程式已啟用 Amazon Pinpoint 支援，則當新使用者開啟您的應用程式 Amazon Pinpoint 時，您的應用程式會自動向註冊端點。下列範例示範如何以程式設計方式新增端點。

### 建立端點

透過 Amazon Pinpoint 在 EndpointRequest 物件中提供端點資料，在 中建立新的端點。

匯入

```
import com.amazonaws.services.pinpoint.AmazonPinpoint;
import com.amazonaws.services.pinpoint.AmazonPinpointClientBuilder;
import com.amazonaws.services.pinpoint.model.UpdateEndpointRequest;
import com.amazonaws.services.pinpoint.model.UpdateEndpointResult;
import com.amazonaws.services.pinpoint.model.EndpointDemographic;
import com.amazonaws.services.pinpoint.model.EndpointLocation;
import com.amazonaws.services.pinpoint.model.EndpointRequest;
import com.amazonaws.services.pinpoint.model.EndpointResponse;
import com.amazonaws.services.pinpoint.model.EndpointUser;
import com.amazonaws.services.pinpoint.model.GetEndpointRequest;
import com.amazonaws.services.pinpoint.model.GetEndpointResult;
```

Code

```
HashMap<String, List<String>> customAttributes = new HashMap<>();
List<String> favoriteTeams = new ArrayList<>();
favoriteTeams.add("Lakers");
favoriteTeams.add("Warriors");
customAttributes.put("team", favoriteTeams);


EndpointDemographic demographic = new EndpointDemographic()
        .withAppVersion("1.0")
        .withMake("apple")
        .withModel("iPhone")
        .withModelVersion("7")
        .withPlatform("ios")
        .withPlatformVersion("10.1.1")
```

```
        .withTimezone("America/Los_Angeles");

EndpointLocation location = new EndpointLocation()
        .withCity("Los Angeles")
        .withCountry("US")
        .withLatitude(34.0)
        .withLongitude(-118.2)
        .withPostalCode("90068")
        .withRegion("CA");

Map<String,Double> metrics = new HashMap<>();
metrics.put("health", 100.00);
metrics.put("luck", 75.00);

EndpointUser user = new EndpointUser()
        .withUserId(UUID.randomUUID().toString());

DateFormat df = new SimpleDateFormat("yyyy-MM-dd'T'HH:mm'Z'"); // Quoted "Z" to
 indicate UTC, no timezone offset
String nowAsISO = df.format(new Date());

EndpointRequest endpointRequest = new EndpointRequest()
        .withAddress(UUID.randomUUID().toString())
        .withAttributes(customAttributes)
        .withChannelType("APNS")
        .withDemographic(demographic)
        .withEffectiveDate(nowAsISO)
        .withLocation(location)
        .withMetrics(metrics)
        .withOptOut("NONE")
        .withRequestId(UUID.randomUUID().toString())
        .withUser(user);
```

然後使用該 EndpointRequest 物件建立 UpdateEndpointRequest 物件。 EndpointRequest 最後，將 UpdateEndpointRequest 物件傳遞至 AmazonPinpointClient 的 `updateEndpoint`方法。

Code

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

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

請參閱 GitHub 上的完整範例。

### 詳細資訊

- 《 Amazon Pinpoint 開發人員指南》中的新增端點
- Amazon Pinpoint API 參考中的端點

## 在 中建立客群 Amazon Pinpoint

使用者客群代表以共用特性為基礎的使用者子集，例如使用者最近開啟您的應用程式或使用的裝置。下列範例示範如何定義使用者區段。

### 建立客群

透過在 SegmentDimensions 物件中定義區段的維度，在 Amazon Pinpoint 中建立新的區段。

匯入

```
import com.amazonaws.services.pinpoint.AmazonPinpoint;
import com.amazonaws.services.pinpoint.AmazonPinpointClientBuilder;
import com.amazonaws.services.pinpoint.model.CreateSegmentRequest;
import com.amazonaws.services.pinpoint.model.CreateSegmentResult;
import com.amazonaws.services.pinpoint.model.AttributeDimension;
import com.amazonaws.services.pinpoint.model.AttributeType;
import com.amazonaws.services.pinpoint.model.RecencyDimension;
import com.amazonaws.services.pinpoint.model.SegmentBehaviors;
import com.amazonaws.services.pinpoint.model.SegmentDemographics;
import com.amazonaws.services.pinpoint.model.SegmentDimensions;
import com.amazonaws.services.pinpoint.model.SegmentLocation;
import com.amazonaws.services.pinpoint.model.SegmentResponse;
import com.amazonaws.services.pinpoint.model.WriteSegmentRequest;
```

Code

```
Pinpoint pinpoint =
 AmazonPinpointClientBuilder.standard().withRegion(Regions.US_EAST_1).build();
```

```
Map<String, AttributeDimension> segmentAttributes = new HashMap<>();
segmentAttributes.put("Team", new
 AttributeDimension().withAttributeType(AttributeType.INCLUSIVE).withValues("Lakers"));

SegmentBehaviors segmentBehaviors = new SegmentBehaviors();
SegmentDemographics segmentDemographics = new SegmentDemographics();
SegmentLocation segmentLocation = new SegmentLocation();

RecencyDimension recencyDimension = new RecencyDimension();
recencyDimension.withDuration("DAY_30").withRecencyType("ACTIVE");
segmentBehaviors.setRecency(recencyDimension);

SegmentDimensions dimensions = new SegmentDimensions()
        .withAttributes(segmentAttributes)
        .withBehavior(segmentBehaviors)
        .withDemographic(segmentDemographics)
        .withLocation(segmentLocation);
```

接著，在 WriteSegmentRequest 中設定 SegmentDimensions 物件，接著會用來建立 CreateSegmentRequest 物件。然後將 CreateSegmentRequest 物件傳遞至 AmazonPinpointClient 的 `createSegment`方法。

Code

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

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

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

請參閱 GitHub 上的完整範例。

## 詳細資訊

- Amazon Pinpoint 《 使用者指南》中的Amazon Pinpoint 客群
- 《 Amazon Pinpoint 開發人員指南》中的建立客群
- Amazon Pinpoint API 參考中的區段
- Amazon Pinpoint API 參考中的客群

# 在 中建立行銷活動 Amazon Pinpoint

您可以使用行銷活動來協助提高應用程式與使用者之間的參與度。您可以建立行銷活動，透過量身打造的訊息或特殊促銷來聯絡使用者的特定客群。此範例示範如何建立新的標準行銷活動，將自訂推送通知傳送至指定的客群。

## 建立行銷活動

在建立新的行銷活動之前，您必須定義排程和訊息，並在 WriteCampaignRequest 物件中設定這些值。

匯入

```
import com.amazonaws.services.pinpoint.AmazonPinpoint;
import com.amazonaws.services.pinpoint.AmazonPinpointClientBuilder;
import com.amazonaws.services.pinpoint.model.CreateCampaignRequest;
import com.amazonaws.services.pinpoint.model.CreateCampaignResult;
import com.amazonaws.services.pinpoint.model.Action;
import com.amazonaws.services.pinpoint.model.CampaignResponse;
import com.amazonaws.services.pinpoint.model.Message;
import com.amazonaws.services.pinpoint.model.MessageConfiguration;
import com.amazonaws.services.pinpoint.model.Schedule;
import com.amazonaws.services.pinpoint.model.WriteCampaignRequest;
```

Code

```
Schedule schedule = new Schedule()
        .withStartTime("IMMEDIATE");

Message defaultMessage = new Message()
        .withAction(Action.OPEN_APP)
        .withBody("My message body.")
        .withTitle("My message title.");

MessageConfiguration messageConfiguration = new MessageConfiguration()
        .withDefaultMessage(defaultMessage);

WriteCampaignRequest request = new WriteCampaignRequest()
        .withDescription("My description.")
        .withSchedule(schedule)
        .withSegmentId(segmentId)
        .withName("MyCampaign")
```

```
        .withMessageConfiguration(messageConfiguration);
```

然後在 中透過向 CreateCampaignRequest 物件 Amazon Pinpoint 提供具有行銷活動組態的 CreateCampaignRequest 來建立新的行銷活動。 WriteCampaignRequest 最後，將 CreateCampaignRequest 物件傳遞至 AmazonPinpointClient 的 `createCampaign`方法。

Code

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

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

請參閱 GitHub 上的完整範例。

### 詳細資訊

- Amazon Pinpoint 《 使用者指南》中的Amazon Pinpoint 行銷活動
- 《 Amazon Pinpoint 開發人員指南》中的建立行銷活動
- Amazon Pinpoint API 參考中的行銷活動
- Amazon Pinpoint API 參考中的行銷活動
- Amazon Pinpoint API 參考中的行銷活動
- Amazon Pinpoint API 參考中的行銷活動版本
- Amazon Pinpoint API 參考中的行銷活動版本

## 在 中更新頻道 Amazon Pinpoint

頻道定義您可以傳送訊息的平台類型。此範例說明如何使用 APNs 頻道來傳送訊息。

### 更新頻道

Amazon Pinpoint 透過提供您要更新之頻道類型的應用程式 ID 和請求物件，在 中啟用頻道。此範例會更新需要 APNSChannelRequest 物件的 APNs 頻道。在 UpdateApnsChannelRequest 中設定這些物件，並將該物件傳遞至 AmazonPinpointClient 的 `updateApnsChannel`方法。

匯入

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

```
import com.amazonaws.services.pinpoint.AmazonPinpointClientBuilder;
import com.amazonaws.services.pinpoint.model.APNSChannelRequest;
import com.amazonaws.services.pinpoint.model.APNSChannelResponse;
import com.amazonaws.services.pinpoint.model.GetApnsChannelRequest;
import com.amazonaws.services.pinpoint.model.GetApnsChannelResult;
import com.amazonaws.services.pinpoint.model.UpdateApnsChannelRequest;
import com.amazonaws.services.pinpoint.model.UpdateApnsChannelResult;
```

Code

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

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

請參閱 GitHub 上的完整範例。

### 詳細資訊

- Amazon Pinpoint 《使用者指南》中的Amazon Pinpoint 頻道
- Amazon Pinpoint API 參考中的 ADM 頻道
- Amazon Pinpoint API 參考中的 APNs 頻道
- API 參考中的 APNs 沙盒頻道 Amazon Pinpoint
- API 參考中的 APNs VoIP 頻道 Amazon Pinpoint
- API 參考中的 APNs VoIP 沙盒頻道 Amazon Pinpoint
- Amazon Pinpoint API 參考中的百度頻道
- Amazon Pinpoint API 參考中的電子郵件管道
- Amazon Pinpoint API 參考中的 GCM 頻道
- Amazon Pinpoint API 參考中的簡訊管道

## Amazon S3 使用 的範例 適用於 Java 的 AWS SDK

本節提供使用適用於 Java 的 AWS SDK編寫 Amazon S3 程式的範例。

> **Note**
>
> 這些範例僅包含示範每種技術所需的程式碼。GitHub 上提供完整程式碼範例。您可以從那裡下載單一原始檔案或將儲存庫複製到本機，以取得建置和執行的所有範例。

主題


- 建立、列出和刪除 Amazon S3 儲存貯體
- 在 Amazon S3 物件上執行操作
- 管理儲存貯體和物件的 Amazon S3 存取許可
- 使用 Amazon S3 儲存貯體政策管理對儲存貯體的存取
- 使用 TransferManager 進行 Amazon S3 操作
- 將 Amazon S3 儲存貯體設定為網站
- 使用 Amazon S3 用戶端加密


## 建立、列出和刪除 Amazon S3 儲存貯體

中的每個物件（檔案）Amazon S3 都必須位於代表物件集合（容器）的儲存貯體中。每個儲存貯體都是由索引鍵（名稱）所知道，該索引鍵必須是唯一的。如需儲存貯體及其組態的詳細資訊，請參閱 Amazon Simple Storage Service 《使用者指南》中的使用 Amazon S3 儲存貯體。

> **Note**
>
> 最佳實務
> 建議您在 Amazon S3 儲存貯體上啟用 AbortIncompleteMultipartUpload 生命週期規則。
> 此規則 Amazon S3 會指示 中止未在啟動後指定天數內完成的分段上傳。超過設定的時間限制時， 會 Amazon S3 中止上傳，然後刪除不完整的上傳資料。
> 如需詳細資訊，請參閱 Amazon S3 《使用者指南》中的使用版本控制的儲存貯體生命週期組態。

> **Note**
>
> 這些程式碼範例假設您了解使用 適用於 Java 的 AWS SDK 中的資料，並使用設定 AWS 登入資料和開發區域中的資訊來設定預設 AWS 登入資料。

## 建立儲存貯體

使用 AmazonS3 用戶端的 `createBucket`方法。傳回新的儲存貯體。如果儲存貯體已存在，則 `createBucket`方法將引發例外狀況。

> **Note**
>
> 若要在嘗試建立具有相同名稱的儲存貯體之前檢查儲存貯體是否已存在，請呼叫 `doesBucketExist`方法。`true` 如果儲存貯體存在，則會傳回 ，`false`否則傳回 。

匯入

```
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.AmazonS3Exception;
import com.amazonaws.services.s3.model.Bucket;

import java.util.List;
```

Code

```
if (s3.doesBucketExistV2(bucket_name)) {
    System.out.format("Bucket %s already exists.\n", bucket_name);
    b = getBucket(bucket_name);
} else {
    try {
        b = s3.createBucket(bucket_name);
    } catch (AmazonS3Exception e) {
        System.err.println(e.getErrorMessage());
    }
}
return b;
```

請參閱 GitHub 上的完整範例。

## 列出儲存貯體

使用 AmazonS3 用戶端的 `listBucket`方法。如果成功，則會傳回儲存貯體清單。

匯入

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

import java.util.List;
```

Code

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

請參閱 GitHub 上的完整範例。

## 刪除儲存貯體

您必須先確認儲存 Amazon S3 貯體為空，否則將發生錯誤，才能刪除儲存貯體。如果您有版本控制的儲存貯體，您也必須刪除與儲存貯體相關聯的任何版本控制的物件。

> **Note**
>
> 完整範例依序包含這些步驟，提供刪除 儲存 Amazon S3 貯體及其內容的完整解決方案。

主題

- 從未版本控制的儲存貯體移除物件，然後再刪除
- 從已版本控制的儲存貯體移除物件，然後再將其刪除
- 刪除空的儲存貯體

從未版本控制的儲存貯體移除物件，然後再刪除

使用 AmazonS3 用戶端的 `listObjects`方法擷取物件清單`deleteObject`，並刪除每個物件清單。

匯入

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

import java.util.Iterator;
```

Code

```
System.out.println(" - removing objects from bucket");
ObjectListing object_listing = s3.listObjects(bucket_name);
while (true) {
    for (Iterator<?> iterator =
         object_listing.getObjectSummaries().iterator();
         iterator.hasNext(); ) {
        S3ObjectSummary summary = (S3ObjectSummary) iterator.next();
        s3.deleteObject(bucket_name, summary.getKey());
    }

    // more object_listing to retrieve?
    if (object_listing.isTruncated()) {
        object_listing = s3.listNextBatchOfObjects(object_listing);
    } else {
        break;
    }
}
```

請參閱 GitHub 上的完整範例。

從已版本控制的儲存貯體移除物件，然後再將其刪除

如果您使用的是版本控制的儲存貯體，您也需要移除儲存貯體中物件的任何存放版本，才能刪除儲存貯體。

使用類似於移除儲存貯體中物件時所使用的模式，透過使用 AmazonS3 用戶端的 `listVersions`方法來列出任何版本控制的物件，然後刪除每個物件`deleteVersion`，以移除版本控制的物件。

匯入

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

```
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.*;

import java.util.Iterator;
```

Code

```
System.out.println(" - removing versions from bucket");
VersionListing version_listing = s3.listVersions(
        new ListVersionsRequest().withBucketName(bucket_name));
while (true) {
    for (Iterator<?> iterator =
         version_listing.getVersionSummaries().iterator();
         iterator.hasNext(); ) {
        S3VersionSummary vs = (S3VersionSummary) iterator.next();
        s3.deleteVersion(
                bucket_name, vs.getKey(), vs.getVersionId());
    }

    if (version_listing.isTruncated()) {
        version_listing = s3.listNextBatchOfVersions(
                version_listing);
    } else {
        break;
    }
}
```

請參閱 GitHub 上的完整範例。

刪除空的儲存貯體

從儲存貯體移除物件後（包括任何版本控制的物件），您可以使用 AmazonS3 用戶端的 `deleteBucket`方法刪除儲存貯體本身。

匯入

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

```
import java.util.Iterator;
```

Code

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

請參閱 GitHub 上的完整範例。

## 在 Amazon S3 物件上執行操作

Amazon S3 物件代表檔案或資料集合。每個物件都必須位於儲存貯體中。

> **Note**
>
> 這些程式碼範例假設您了解使用 中的資料， 適用於 Java 的 AWS SDK並使用設定 AWS 登入資料和開發區域中的資訊來設定預設 AWS 登入資料。

主題

- 上傳物件
- 列出物件
- 下載物件
- 複製、移動或重新命名物件
- 刪除物件
- 一次刪除多個物件

### 上傳物件

使用 AmazonS3 用戶端的 `putObject`方法，提供儲存貯體名稱、金鑰名稱和要上傳的檔案。儲存貯體必須存在，否則會產生錯誤。

匯入

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

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

Code

```
System.out.format("Uploading %s to S3 bucket %s...\n", file_path, bucket_name);
final AmazonS3 s3 =
 AmazonS3ClientBuilder.standard().withRegion(Regions.DEFAULT_REGION).build();
try {
    s3.putObject(bucket_name, key_name, new File(file_path));
} catch (AmazonServiceException e) {
    System.err.println(e.getErrorMessage());
    System.exit(1);
}
```

請參閱 GitHub 上的完整範例。

## 列出物件

若要取得儲存貯體中的物件清單，請使用 AmazonS3 用戶端的 `listObjects`方法，並提供儲存貯體的名稱。

`listObjects` 方法會傳回 ObjectListing 物件，提供儲存貯體中物件的相關資訊。若要列出物件名稱（金鑰），請使用 `getObjectSummaries`方法取得 S3ObjectSummary 物件清單，每個物件都代表儲存貯體中的單一物件。然後呼叫其 `getKey`方法來擷取物件的名稱。

匯入

```
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.ListObjectsV2Result;
import com.amazonaws.services.s3.model.S3ObjectSummary;
```

Code

```
System.out.format("Objects in S3 bucket %s:\n", bucket_name);
final AmazonS3 s3 =
 AmazonS3ClientBuilder.standard().withRegion(Regions.DEFAULT_REGION).build();
ListObjectsV2Result result = s3.listObjectsV2(bucket_name);
List<S3ObjectSummary> objects = result.getObjectSummaries();
```

```
for (S3ObjectSummary os : objects) {
    System.out.println("* " + os.getKey());
}
```

請參閱 GitHub 上的完整範例。

## 下載物件

使用 AmazonS3 用戶端的 getObject方法，將要下載的儲存貯體和物件的名稱傳遞給用戶端。如果成功， 方法會傳回 S3Object。指定的儲存貯體和物件金鑰必須存在，否則會產生錯誤。

您可以在 getObjectContent上呼叫 ，以取得物件的內容S3Object。這會傳回做為標準 Java InputStream 物件的 S3ObjectInputStream。

下列範例會從 S3 下載物件，並將其內容儲存至檔案 （使用與物件金鑰相同的名稱）。

匯入

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.S3Object;
import com.amazonaws.services.s3.model.S3ObjectInputStream;

import java.io.File;
```

Code

```
System.out.format("Downloading %s from S3 bucket %s...\n", key_name, bucket_name);
final AmazonS3 s3 =
 AmazonS3ClientBuilder.standard().withRegion(Regions.DEFAULT_REGION).build();
try {
    S3Object o = s3.getObject(bucket_name, key_name);
    S3ObjectInputStream s3is = o.getObjectContent();
    FileOutputStream fos = new FileOutputStream(new File(key_name));
    byte[] read_buf = new byte[1024];
    int read_len = 0;
    while ((read_len = s3is.read(read_buf)) > 0) {
        fos.write(read_buf, 0, read_len);
    }
    s3is.close();
```

```
    fos.close();
} catch (AmazonServiceException e) {
    System.err.println(e.getErrorMessage());
    System.exit(1);
} catch (FileNotFoundException e) {
    System.err.println(e.getMessage());
    System.exit(1);
} catch (IOException e) {
    System.err.println(e.getMessage());
    System.exit(1);
}
```

請參閱 GitHub 上的完整範例。

## 複製、移動或重新命名物件

您可以使用 AmazonS3 用戶端的 copyObject方法，將物件從一個儲存貯體複製到另一個儲存貯體。它接受要複製的儲存貯體名稱、要複製的物件，以及目的地儲存貯體名稱。

匯入

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

Code

```
try {
    s3.copyObject(from_bucket, object_key, to_bucket, object_key);
} catch (AmazonServiceException e) {
    System.err.println(e.getErrorMessage());
    System.exit(1);
}
System.out.println("Done!");
```

請參閱 GitHub 上的完整範例。

> **Note**
>
> 您可以copyObject搭配 deleteObject 使用 來移動或重新命名物件，方法是先將物件複製到新名稱（您可以使用與來源和目的地相同的儲存貯體），然後從舊位置刪除物件。

## 刪除物件

使用 AmazonS3 用戶端的 `deleteObject`方法，將要刪除的儲存貯體和物件的名稱傳遞給用戶端。指定的儲存貯體和物件金鑰必須存在，否則會產生錯誤。

匯入

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

Code

```
final AmazonS3 s3 =
 AmazonS3ClientBuilder.standard().withRegion(Regions.DEFAULT_REGION).build();
try {
    s3.deleteObject(bucket_name, object_key);
} catch (AmazonServiceException e) {
    System.err.println(e.getErrorMessage());
    System.exit(1);
}
```

請參閱 GitHub 上的完整範例。

## 一次刪除多個物件

使用 AmazonS3 用戶端的 `deleteObjects`方法，您可以將多個物件從相同儲存貯體中刪除，方法是將它們的名稱傳遞至 link：sdk-for-java/v1/reference/com/amazonaws/services/s3/model/DeleteObjectsRequest.html 方法。

匯入

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

Code

```
final AmazonS3 s3 =
 AmazonS3ClientBuilder.standard().withRegion(Regions.DEFAULT_REGION).build();
try {
```

```
    DeleteObjectsRequest dor = new DeleteObjectsRequest(bucket_name)
            .withKeys(object_keys);
    s3.deleteObjects(dor);
} catch (AmazonServiceException e) {
    System.err.println(e.getErrorMessage());
    System.exit(1);
}
```

請參閱 GitHub 上的完整範例。

## 管理儲存貯體和物件的 Amazon S3 存取許可

您可以針對 Amazon S3 儲存貯體和物件使用存取控制清單 (ACLs)，以精細控制 Amazon S3 資源。

> **Note**
>
> 這些程式碼範例假設您了解使用 適用於 Java 的 AWS SDK 中的資料，並使用設定 AWS 登入資料和開發區域中的資訊來設定預設 AWS 登入資料。

### 取得儲存貯體的存取控制清單

若要取得儲存貯體的目前 ACL，請呼叫 AmazonS3 的 `getBucketAcl`方法，將儲存貯體名稱傳遞給它以進行查詢。此方法會傳回 AccessControlList 物件。若要取得清單中的每個存取授權，請呼叫其 `getGrantsAsList` 方法，這會傳回標準 Java 授予物件清單。

匯入

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

Code

```
final AmazonS3 s3 =
 AmazonS3ClientBuilder.standard().withRegion(Regions.DEFAULT_REGION).build();
try {
```

```
    AccessControlList acl = s3.getBucketAcl(bucket_name);
    List<Grant> grants = acl.getGrantsAsList();
    for (Grant grant : grants) {
        System.out.format("  %s: %s\n", grant.getGrantee().getIdentifier(),
                grant.getPermission().toString());
    }
} catch (AmazonServiceException e) {
    System.err.println(e.getErrorMessage());
    System.exit(1);
}
```

請參閱 GitHub 上的完整範例。

## 設定儲存貯體的存取控制清單

若要新增或修改儲存貯體的 ACL 許可，請呼叫 AmazonS3 的 `setBucketAcl`方法。它需要 AccessControlList 物件，其中包含要設定的承授者和存取層級清單。

匯入

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

Code

```
final AmazonS3 s3 =
 AmazonS3ClientBuilder.standard().withRegion(Regions.DEFAULT_REGION).build();
try {
    // get the current ACL
    AccessControlList acl = s3.getBucketAcl(bucket_name);
    // set access for the grantee
    EmailAddressGrantee grantee = new EmailAddressGrantee(email);
    Permission permission = Permission.valueOf(access);
    acl.grantPermission(grantee, permission);
    s3.setBucketAcl(bucket_name, acl);
} catch (AmazonServiceException e) {
    System.err.println(e.getErrorMessage());
    System.exit(1);
```

```
}
```

> **Note**
>
> 您可以直接使用承授者類別提供承授者的唯一識別符，或使用 EmailAddressGrantee 類別來透過電子郵件設定承授者，如同我們在這裡所做的。

請參閱 GitHub 上的完整範例。

## 取得物件的存取控制清單

若要取得物件的目前 ACL，請呼叫 AmazonS3 的 `getObjectAcl`方法，將要查詢的儲存貯體名稱和物件名稱傳遞給它。如同 `getBucketAcl`，此方法會傳回 AccessControlList 物件，您可以用來檢查每個授與。

匯入

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

Code

```
try {
    AccessControlList acl = s3.getObjectAcl(bucket_name, object_key);
    List<Grant> grants = acl.getGrantsAsList();
    for (Grant grant : grants) {
        System.out.format("  %s: %s\n", grant.getGrantee().getIdentifier(),
                grant.getPermission().toString());
    }
} catch (AmazonServiceException e) {
    System.err.println(e.getErrorMessage());
    System.exit(1);
}
```

請參閱 GitHub 上的完整範例。

## 設定物件的存取控制清單

若要新增或修改物件的 ACL 許可，請呼叫 AmazonS3 的 `setObjectAcl`方法。它需要 AccessControlList 物件，其中包含要設定的承授者和存取層級清單。

匯入

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

Code

```
    try {
        // get the current ACL
        AccessControlList acl = s3.getObjectAcl(bucket_name, object_key);
        // set access for the grantee
        EmailAddressGrantee grantee = new EmailAddressGrantee(email);
        Permission permission = Permission.valueOf(access);
        acl.grantPermission(grantee, permission);
        s3.setObjectAcl(bucket_name, object_key, acl);
    } catch (AmazonServiceException e) {
        System.err.println(e.getErrorMessage());
        System.exit(1);
    }
}
```

> **Note**
>
> 您可以直接使用承授者類別提供承授者的唯一識別符，或使用 EmailAddressGrantee 類別來透過電子郵件設定承授者，如同我們在這裡所做的。

請參閱 GitHub 上的完整範例。

## 詳細資訊

- Amazon S3 API 參考中的 GET 儲存貯體 acl

- Amazon S3 API 參考中的 PUT 儲存貯體 acl
- Amazon S3 API 參考中的 GET 物件 acl
- Amazon S3 API 參考中的 PUT 物件 acl

## 使用 Amazon S3 儲存貯體政策管理對儲存貯體的存取

您可以設定、取得或刪除儲存貯體政策，以管理對儲存 Amazon S3 貯體的存取。

### 設定儲存貯體政策

您可以透過下列方式設定特定 S3 儲存貯體的儲存貯體政策：

- 呼叫 AmazonS3 用戶端的 ，`setBucketPolicy`並提供 SetBucketPolicyRequest
- 使用採用儲存貯體名稱和政策文字的`setBucketPolicy`過載直接設定政策 (JSON 格式 )

匯入

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

Code

```
    s3.setBucketPolicy(bucket_name, policy_text);
} catch (AmazonServiceException e) {
    System.err.println(e.getErrorMessage());
    System.exit(1);
}
```

使用政策類別來產生或驗證政策

提供儲存貯體政策給 時`setBucketPolicy`，您可以執行下列動作：

- 直接將政策指定為 JSON 格式文字的字串
- 使用 政策類別建置政策

透過使用 `Policy`類別，您不需要擔心文字字串的格式是否正確。若要從 `Policy`類別取得 JSON 政策文字，請使用其 `toJson` 方法。

匯入

```
import com.amazonaws.auth.policy.Resource;
import com.amazonaws.auth.policy.Statement;
import com.amazonaws.auth.policy.actions.S3Actions;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
```

Code

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

Policy 類別也提供fromJson一種方法，可嘗試使用傳入的 JSON 字串來建置政策。方法會進行驗證，以確保文字可轉換為有效的政策結構，如果政策文字無效IllegalArgumentException，便會失敗。

```
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());
}
```

您可以使用此技術預先驗證從檔案或其他方式讀取的政策。

請參閱 GitHub 上的完整範例。

## 取得儲存貯體政策

若要擷取 儲存 Amazon S3 貯體的政策，請呼叫 AmazonS3 用戶端的 getBucketPolicy方法，將儲存貯體的名稱傳遞給它以從中取得政策。

匯入

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

Code

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

如果命名的儲存貯體不存在，如果您無法存取它，或如果它沒有儲存貯體政策，`AmazonServiceException`則會擲出 。

請參閱 GitHub 上的完整範例。

## 刪除儲存貯體政策

若要刪除儲存貯體政策，請呼叫 AmazonS3 用戶端的 `deleteBucketPolicy`，並提供儲存貯體名稱。

匯入

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

Code

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

即使儲存貯體還沒有政策，此方法也會成功。如果您指定的儲存貯體名稱不存在，或者您無法存取儲存貯體，`AmazonServiceException`則會擲回 。

請參閱 GitHub 上的完整範例。

### 詳細資訊

- Amazon Simple Storage Service 《 使用者指南》中的存取政策語言概觀
- Amazon Simple Storage Service 《 使用者指南》中的儲存貯體政策範例

## 使用 TransferManager 進行 Amazon S3 操作

您可以使用 適用於 Java 的 AWS SDK TransferManager 類別，可靠地將檔案從本機環境傳輸到另一個 S3 位置 Amazon S3 ，以及將物件複製到另一個位置。 `TransferManager`可以取得傳輸進度，並暫停或繼續上傳和下載。

> **Note**
>
> 最佳實務
> 建議您在 Amazon S3 儲存貯體上啟用 AbortIncompleteMultipartUpload 生命週期規則。此規則 Amazon S3 會指示 中止在啟動後指定天數內未完成的分段上傳。超過設定的時間限制時， 會 Amazon S3 中止上傳，然後刪除不完整的上傳資料。
> 如需詳細資訊，請參閱 Amazon S3 《 使用者指南》中的使用版本控制的儲存貯體生命週期組態。

> **Note**
>
> 這些程式碼範例假設您了解使用 適用於 Java 的 AWS SDK 中的資料，並使用設定 AWS 登入資料和開發區域中的資訊來設定預設 AWS 登入資料。

### 上傳檔案和目錄

TransferManager 可以將檔案、檔案清單和目錄上傳至您先前建立的任何儲存 Amazon S3 貯體。

主題

- 上傳單一檔案

- 上傳檔案清單
- 上傳目錄

### 上傳單一檔案

呼叫 TransferManager 的 upload方法，提供儲存 Amazon S3 貯體名稱、金鑰 （ 物件 ） 名稱，以及代表要上傳之檔案的標準 Java 檔案物件。

匯入

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.services.s3.transfer.MultipleFileUpload;
import com.amazonaws.services.s3.transfer.TransferManager;
import com.amazonaws.services.s3.transfer.TransferManagerBuilder;
import com.amazonaws.services.s3.transfer.Upload;

import java.io.File;
import java.util.ArrayList;
import java.util.Arrays;
```

Code

```
File f = new File(file_path);
TransferManager xfer_mgr = TransferManagerBuilder.standard().build();
try {
    Upload xfer = xfer_mgr.upload(bucket_name, key_name, f);
    // loop with Transfer.isDone()
    XferMgrProgress.showTransferProgress(xfer);
    //  or block with Transfer.waitForCompletion()
    XferMgrProgress.waitForCompletion(xfer);
} catch (AmazonServiceException e) {
    System.err.println(e.getErrorMessage());
    System.exit(1);
}
xfer_mgr.shutdownNow();
```

upload 方法會立即傳回，提供 Upload 物件以用來檢查傳輸狀態或等待它完成。

請參閱等待轉接完成，以取得在呼叫 TransferManager 的 shutdownNow方法之前waitForCompletion，使用 成功完成轉接的相關資訊。等待傳輸完成時，您可以輪詢或接聽其狀態和進度的更新。如需詳細資訊，請參閱取得傳輸狀態和進度。

請參閱 GitHub 上的完整範例。

### 上傳檔案清單

若要在一個操作中上傳多個檔案，請呼叫 TransferManageruploadFileList 方法，並提供下列項目：

- 儲存 Amazon S3 貯體名稱
- 金鑰字首，以建立物件的名稱開頭（在儲存貯體中放置物件的路徑）
- 代表從中建立檔案路徑之相對目錄的檔案物件
- 清單物件，其中包含一組要上傳的檔案物件

#### 匯入

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.services.s3.transfer.MultipleFileUpload;
import com.amazonaws.services.s3.transfer.TransferManager;
import com.amazonaws.services.s3.transfer.TransferManagerBuilder;
import com.amazonaws.services.s3.transfer.Upload;

import java.io.File;
import java.util.ArrayList;
import java.util.Arrays;
```

#### Code

```
ArrayList<File> files = new ArrayList<File>();
for (String path : file_paths) {
    files.add(new File(path));
}

TransferManager xfer_mgr = TransferManagerBuilder.standard().build();
try {
    MultipleFileUpload xfer = xfer_mgr.uploadFileList(bucket_name,
            key_prefix, new File("."), files);
    // loop with Transfer.isDone()
    XferMgrProgress.showTransferProgress(xfer);
    // or block with Transfer.waitForCompletion()
    XferMgrProgress.waitForCompletion(xfer);
} catch (AmazonServiceException e) {
    System.err.println(e.getErrorMessage());
```

```
    System.exit(1);
}
xfer_mgr.shutdownNow();
```

請參閱等待轉接完成，以取得在呼叫 TransferManager 的 shutdownNow方法之前waitForCompletion，使用 成功完成轉接的相關資訊。等待傳輸完成時，您可以輪詢或接聽其狀態和進度的更新。如需詳細資訊，請參閱取得傳輸狀態和進度。

傳回的 MultipleFileUpload 物件uploadFileList可用來查詢傳輸狀態或進度。如需詳細資訊，請參閱使用 ProgressListener 輪詢傳輸的目前進度和取得傳輸進度。 ProgressListener

您也可以使用 MultipleFileUpload的 getSubTransfers方法，取得每個要傳輸檔案的個別Upload物件。如需詳細資訊，請參閱取得子轉移進度。

請參閱 GitHub 上的完整範例。

上傳目錄

您可以使用 TransferManager 的 uploadDirectory方法上傳整個檔案目錄，並可選擇以遞迴方式複製子目錄中的檔案。您提供儲存 Amazon S3 貯體名稱、S3 金鑰字首、代表要複製之本機目錄的檔案物件，以及指出是否要以遞迴方式 (true 或 false) 複製子目錄boolean的值。

匯入

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.services.s3.transfer.MultipleFileUpload;
import com.amazonaws.services.s3.transfer.TransferManager;
import com.amazonaws.services.s3.transfer.TransferManagerBuilder;
import com.amazonaws.services.s3.transfer.Upload;

import java.io.File;
import java.util.ArrayList;
import java.util.Arrays;
```

Code

```
TransferManager xfer_mgr = TransferManagerBuilder.standard().build();
try {
    MultipleFileUpload xfer = xfer_mgr.uploadDirectory(bucket_name,
            key_prefix, new File(dir_path), recursive);
    // loop with Transfer.isDone()
    XferMgrProgress.showTransferProgress(xfer);
```

```
    // or block with Transfer.waitForCompletion()
    XferMgrProgress.waitForCompletion(xfer);
} catch (AmazonServiceException e) {
    System.err.println(e.getErrorMessage());
    System.exit(1);
}
xfer_mgr.shutdownNow();
```

請參閱等待轉接完成，以取得在呼叫 TransferManager 的 shutdownNow方法之前waitForCompletion，使用 成功完成轉接的相關資訊。等待傳輸完成時，您可以輪詢或接聽其狀態和進度的更新。如需詳細資訊，請參閱取得傳輸狀態和進度。

傳回的 MultipleFileUpload 物件uploadFileList可用來查詢傳輸狀態或進度。如需詳細資訊，請參閱使用 ProgressListener 輪詢傳輸的目前進度和取得傳輸進度。 ProgressListener

您也可以使用 MultipleFileUpload的 getSubTransfers方法，取得每個要傳輸檔案的個別Upload物件。如需詳細資訊，請參閱取得子轉移進度。

請參閱 GitHub 上的完整範例。

## 下載檔案或目錄

使用 TransferManager 類別從中下載單一檔案 (Amazon S3 物件） 或目錄 ( Amazon S3 儲存貯體名稱後接物件字首） Amazon S3。

主題

- 下載單一檔案
- 下載目錄

### 下載單一檔案

使用 TransferManager 的 download方法，提供儲存 Amazon S3 貯體名稱，其中包含您要下載的物件、金鑰 （ 物件） 名稱，以及代表要在本機系統上建立之檔案的檔案https://docs.oracle.com/javase/8/docs/api/index.html?java/io/File.html物件。

匯入

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.services.s3.transfer.Download;
import com.amazonaws.services.s3.transfer.MultipleFileDownload;
import com.amazonaws.services.s3.transfer.TransferManager;
```

```
import com.amazonaws.services.s3.transfer.TransferManagerBuilder;

import java.io.File;
```

Code

```
File f = new File(file_path);
TransferManager xfer_mgr = TransferManagerBuilder.standard().build();
try {
    Download xfer = xfer_mgr.download(bucket_name, key_name, f);
    // loop with Transfer.isDone()
    XferMgrProgress.showTransferProgress(xfer);
    // or block with Transfer.waitForCompletion()
    XferMgrProgress.waitForCompletion(xfer);
} catch (AmazonServiceException e) {
    System.err.println(e.getErrorMessage());
    System.exit(1);
}
xfer_mgr.shutdownNow();
```

請參閱等待轉接完成，以取得有關使用 `waitForCompletion` 成功完成轉接的資訊，然後再呼叫 TransferManager 的 `shutdownNow`方法。等待傳輸完成時，您可以輪詢或接聽其狀態和進度的更新。如需詳細資訊，請參閱取得傳輸狀態和進度。

請參閱 GitHub 上的完整範例。

下載目錄

若要從中下載一組共用常見金鑰字首（類似於檔案系統上的目錄）的檔案 Amazon S3，請使用 TransferManager`downloadDirectory` 方法。方法採用儲存 Amazon S3 貯體名稱，其中包含您要下載的物件、所有物件共用的物件字首，以及代表要在本機系統上下載檔案的目錄的檔案物件。如果名為的目錄尚不存在，則會建立該目錄。

匯入

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.services.s3.transfer.Download;
import com.amazonaws.services.s3.transfer.MultipleFileDownload;
import com.amazonaws.services.s3.transfer.TransferManager;
import com.amazonaws.services.s3.transfer.TransferManagerBuilder;

import java.io.File;
```

Code

```
TransferManager xfer_mgr = TransferManagerBuilder.standard().build();

try {
    MultipleFileDownload xfer = xfer_mgr.downloadDirectory(
            bucket_name, key_prefix, new File(dir_path));
    // loop with Transfer.isDone()
    XferMgrProgress.showTransferProgress(xfer);
    // or block with Transfer.waitForCompletion()
    XferMgrProgress.waitForCompletion(xfer);
} catch (AmazonServiceException e) {
    System.err.println(e.getErrorMessage());
    System.exit(1);
}
xfer_mgr.shutdownNow();
```

請參閱等待轉接完成，以取得在呼叫 TransferManager 的 shutdownNow方法之前waitForCompletion，使用 成功完成轉接的相關資訊。等待傳輸完成時，您可以輪詢或接聽其狀態和進度的更新。如需詳細資訊，請參閱取得傳輸狀態和進度。

請參閱 GitHub 上的完整範例。

## 複製物件

若要將物件從一個 S3 儲存貯體複製到另一個儲存貯體，請使用 TransferManagercopy 方法。

匯入

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.services.s3.transfer.Copy;
import com.amazonaws.services.s3.transfer.TransferManager;
import com.amazonaws.services.s3.transfer.TransferManagerBuilder;
```

Code

```
System.out.println("Copying s3 object: " + from_key);
System.out.println("      from bucket: " + from_bucket);
System.out.println("     to s3 object: " + to_key);
System.out.println("        in bucket: " + to_bucket);
```

```
TransferManager xfer_mgr = TransferManagerBuilder.standard().build();
try {
    Copy xfer = xfer_mgr.copy(from_bucket, from_key, to_bucket, to_key);
    // loop with Transfer.isDone()
    XferMgrProgress.showTransferProgress(xfer);
    // or block with Transfer.waitForCompletion()
    XferMgrProgress.waitForCompletion(xfer);
} catch (AmazonServiceException e) {
    System.err.println(e.getErrorMessage());
    System.exit(1);
}
xfer_mgr.shutdownNow();
```

請參閱 GitHub 上的完整範例。

## 等待轉接完成

如果您的應用程式（或執行緒）可以在傳輸完成之前封鎖，您可以使用 Transfer 介面的 `waitForCompletion`方法封鎖，直到傳輸完成或發生例外狀況為止。

```
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);
}
```

如果您在呼叫 之前輪詢事件`waitForCompletion`、在個別執行緒上實作輪詢機制，或使用 ProgressListener 非同步接收進度更新，就會取得傳輸進度。

請參閱 GitHub 上的完整範例。

## 取得傳輸狀態和進度

TransferManager`upload*`、 `download*`和 `copy`方法傳回的每個類別都會傳回下列其中一個類別的執行個體，視其為單一檔案或多檔案操作而定。

| 類別 | 傳回者 |
| --- | --- |
| Copy (複製) | `copy` |
| 下載 | `download` |
| MultipleFileDownload | `downloadDirectory` |
| 上傳 | `upload` |
| MultipleFileUpload | `uploadFileList` ，`uploadDirectory` |

所有這些類別都會實作 Transfer 界面。 `Transfer`提供實用方法來取得傳輸進度、暫停或繼續傳輸，以及取得傳輸的目前或最終狀態。

主題

- 輪詢傳輸的目前進度
- 使用 ProgressListener 取得傳輸進度
- 取得子轉移進度

輪詢傳輸的目前進度

此迴圈會列印傳輸的進度、在執行時檢查其目前的進度，並在完成時列印其最終狀態。

匯入

```
import com.amazonaws.AmazonClientException;
import com.amazonaws.AmazonServiceException;
import com.amazonaws.event.ProgressEvent;
import com.amazonaws.event.ProgressListener;
import com.amazonaws.services.s3.transfer.*;
import com.amazonaws.services.s3.transfer.Transfer.TransferState;

import java.io.File;
import java.util.ArrayList;
import java.util.Collection;
```

Code

```
// print the transfer's human-readable description
System.out.println(xfer.getDescription());
// print an empty progress bar...
printProgressBar(0.0);
// update the progress bar while the xfer is ongoing.
do {
    try {
        Thread.sleep(100);
    } catch (InterruptedException e) {
        return;
    }
    // Note: so_far and total aren't used, they're just for
    // documentation purposes.
    TransferProgress progress = xfer.getProgress();
    long so_far = progress.getBytesTransferred();
    long total = progress.getTotalBytesToTransfer();
    double pct = progress.getPercentTransferred();
    eraseProgressBar();
    printProgressBar(pct);
} while (xfer.isDone() == false);
// print the final state of the transfer.
TransferState xfer_state = xfer.getState();
System.out.println(": " + xfer_state);
```

請參閱 GitHub 上的完整範例。

使用 ProgressListener 取得傳輸進度

您可以使用 Transfer 介面的 addProgressListener方法，將 ProgressListener 連接到任何傳輸。

ProgressListener 只需要一個方法 progressChanged，它需要 ProgressEvent 物件。您可以使用 物件來呼叫getBytes操作的總位元組數，以及呼叫 到目前為止傳輸的位元組數getBytesTransferred。

匯入

```
import com.amazonaws.AmazonClientException;
import com.amazonaws.AmazonServiceException;
import com.amazonaws.event.ProgressEvent;
import com.amazonaws.event.ProgressListener;
import com.amazonaws.services.s3.transfer.*;
import com.amazonaws.services.s3.transfer.Transfer.TransferState;
```

```
import java.io.File;
import java.util.ArrayList;
import java.util.Collection;
```

Code

```
File f = new File(file_path);
TransferManager xfer_mgr = TransferManagerBuilder.standard().build();
try {
    Upload u = xfer_mgr.upload(bucket_name, key_name, f);
    // print an empty progress bar...
    printProgressBar(0.0);
    u.addProgressListener(new ProgressListener() {
        public void progressChanged(ProgressEvent e) {
            double pct = e.getBytesTransferred() * 100.0 / e.getBytes();
            eraseProgressBar();
            printProgressBar(pct);
        }
    });
    // block with Transfer.waitForCompletion()
    XferMgrProgress.waitForCompletion(u);
    // print the final state of the transfer.
    TransferState xfer_state = u.getState();
    System.out.println(": " + xfer_state);
} catch (AmazonServiceException e) {
    System.err.println(e.getErrorMessage());
    System.exit(1);
}
xfer_mgr.shutdownNow();
```

請參閱 GitHub 上的完整範例。

取得子轉移進度

MultipleFileUpload 類別可以透過呼叫其getSubTransfers方法傳回其子傳輸的相關資訊。它會傳回無法修改的上傳物件集合，提供每個子傳輸的個別傳輸狀態和進度。

匯入

```
import com.amazonaws.AmazonClientException;
import com.amazonaws.AmazonServiceException;
import com.amazonaws.event.ProgressEvent;
import com.amazonaws.event.ProgressListener;
```

```
import com.amazonaws.services.s3.transfer.*;
import com.amazonaws.services.s3.transfer.Transfer.TransferState;

import java.io.File;
import java.util.ArrayList;
import java.util.Collection;
```

### Code

```
Collection<? extends Upload> sub_xfers = new ArrayList<Upload>();
sub_xfers = multi_upload.getSubTransfers();

do {
    System.out.println("\nSubtransfer progress:\n");
    for (Upload u : sub_xfers) {
        System.out.println("  " + u.getDescription());
        if (u.isDone()) {
            TransferState xfer_state = u.getState();
            System.out.println("  " + xfer_state);
        } else {
            TransferProgress progress = u.getProgress();
            double pct = progress.getPercentTransferred();
            printProgressBar(pct);
            System.out.println();
        }
    }

    // wait a bit before the next update.
    try {
        Thread.sleep(200);
    } catch (InterruptedException e) {
        return;
    }
} while (multi_upload.isDone() == false);
// print the final state of the transfer.
TransferState xfer_state = multi_upload.getState();
System.out.println("\nMultipleFileUpload " + xfer_state);
```

請參閱 GitHub 上的完整範例。

## 詳細資訊

- Amazon Simple Storage Service 《使用者指南》中的物件金鑰

# 將 Amazon S3 儲存貯體設定為網站

您可以設定 儲存 Amazon S3 貯體做為網站運作。若要這樣做，您需要設定其網站組態。

> **Note**
>
> 這些程式碼範例假設您了解使用 適用於 Java 的 AWS SDK 中的資料，並使用設定 AWS 登入資料和開發區域中的資訊來設定預設 AWS 登入資料。

## 設定儲存貯體的網站組態

若要設定儲存 Amazon S3 貯體的網站組態，請使用儲存貯體名稱呼叫 AmazonS3 的 `setWebsiteConfiguration`方法來設定組態，以及包含儲存貯體網站組態的 BucketWebsiteConfiguration 物件。

需要設定索引文件；所有其他參數都是選用的。

匯入

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

Code

```
    String bucket_name, String index_doc, String error_doc) {
BucketWebsiteConfiguration website_config = null;

if (index_doc == null) {
    website_config = new BucketWebsiteConfiguration();
} else if (error_doc == null) {
    website_config = new BucketWebsiteConfiguration(index_doc);
} else {
    website_config = new BucketWebsiteConfiguration(index_doc, error_doc);
}

final AmazonS3 s3 =
 AmazonS3ClientBuilder.standard().withRegion(Regions.DEFAULT_REGION).build();
```

```
try {
    s3.setBucketWebsiteConfiguration(bucket_name, website_config);
} catch (AmazonServiceException e) {
    System.out.format(
            "Failed to set website configuration for bucket '%s'!\n",
            bucket_name);
    System.err.println(e.getErrorMessage());
    System.exit(1);
}
```

> **Note**
>
> 設定網站組態不會修改儲存貯體的存取許可。若要在 Web 上顯示您的檔案，您還需要設定儲存貯體政策，允許公開讀取儲存貯體中的檔案。如需詳細資訊，請參閱使用 Amazon S3 儲存貯體政策管理對儲存貯體的存取。

請參閱 GitHub 上的完整範例。

## 取得儲存貯體的網站組態

若要取得儲存 Amazon S3 貯體的網站組態，請呼叫 AmazonS3 的 `getWebsiteConfiguration`方法，並指定儲存貯體的名稱來擷取其組態。

組態會以 BucketWebsiteConfiguration 物件傳回。如果儲存貯體沒有網站組態，`null`則會傳回 。

匯入

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

Code

```
final AmazonS3 s3 =
 AmazonS3ClientBuilder.standard().withRegion(Regions.DEFAULT_REGION).build();
try {
    BucketWebsiteConfiguration config =
            s3.getBucketWebsiteConfiguration(bucket_name);
```

```
    if (config == null) {
        System.out.println("No website configuration found!");
    } else {
        System.out.format("Index document: %s\n",
                config.getIndexDocumentSuffix());
        System.out.format("Error document: %s\n",
                config.getErrorDocument());
    }
} catch (AmazonServiceException e) {
    System.err.println(e.getErrorMessage());
    System.out.println("Failed to get website configuration!");
    System.exit(1);
}
```

請參閱 GitHub 上的完整範例。

## 刪除儲存貯體的網站組態

若要刪除儲存 Amazon S3 貯體的網站組態，請使用儲存貯體的名稱呼叫 AmazonS3 的 `deleteWebsiteConfiguration`方法，從中刪除組態。

匯入

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

Code

```
final AmazonS3 s3 =
 AmazonS3ClientBuilder.standard().withRegion(Regions.DEFAULT_REGION).build();
try {
    s3.deleteBucketWebsiteConfiguration(bucket_name);
} catch (AmazonServiceException e) {
    System.err.println(e.getErrorMessage());
    System.out.println("Failed to delete website configuration!");
    System.exit(1);
}
```

請參閱 GitHub 上的完整範例。

### 詳細資訊

- Amazon S3 API 參考中的 PUT 儲存貯體網站
- Amazon S3 API 參考中的 GET 儲存貯體網站
- Amazon S3 API 參考中的 DELETE 儲存貯體網站

## 使用 Amazon S3 用戶端加密

使用加密用戶端 Amazon S3 加密資料是您可以為存放在其中的敏感資訊提供額外一層保護的方式之一 Amazon S3。本節中的範例示範如何為您的應用程式建立和設定 Amazon S3 加密用戶端。

如果您是初次使用密碼編譯，請參閱 AWS KMS 開發人員指南中的密碼編譯基本概念，以取得密碼編譯術語和演算法的基本概觀。如需有關跨 AWS SDKs加密支援的資訊，請參閱《 Amazon Web Services 一般參考》中的AWSAmazon S3 用戶端加密的 SDK 支援。

> **Note**
>
> 這些程式碼範例假設您了解使用 適用於 Java 的 AWS SDK 中的資料，並使用設定 AWS 登入資料和開發區域中的資訊來設定預設 AWS 登入資料。

如果您使用的是 1.11.836 版或更早的版本 適用於 Java 的 AWS SDK，請參閱Amazon S3 加密用戶端遷移，以取得將應用程式遷移至更新版本的相關資訊。如果您無法遷移，請參閱 GitHub 上的此完整範例。

否則，如果您使用的是 1.11.837 版或更新版本 適用於 Java 的 AWS SDK，請探索下列範例主題，以使用 Amazon S3 用戶端加密。

主題

- Amazon S3 使用用戶端主金鑰的用戶端加密
- Amazon S3 使用 AWS KMS 受管金鑰的用戶端加密

### Amazon S3 使用用戶端主金鑰的用戶端加密

下列範例使用 AmazonS3EncryptionClientV2Builder 類別來建立已啟用用戶端加密的 Amazon S3 用戶端。啟用後，您 Amazon S3 使用此用戶端上傳到 的任何物件都會加密。您從 Amazon S3 使用此用戶端取得的任何物件都會自動解密。

> **Note**
>
> 下列範例示範搭配客戶受管 Amazon S3 用戶端主金鑰使用用戶端加密。若要了解如何搭配 AWS KMS 受管金鑰使用加密，請參閱搭配 Amazon S3AWS KMS 受管金鑰的用戶端加密。

啟用用戶端加密時，您可以選擇兩種 Amazon S3 加密模式：嚴格驗證或驗證。下列各節說明如何啟用每種類型。若要了解每個模式使用哪些演算法，請參閱 CryptoMode 定義。

必要的匯入

匯入下列類別以取得這些範例。

匯入

```
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;
```

嚴格驗證加密

如果`CryptoMode`未指定 ，嚴格驗證加密是預設模式。

若要明確啟用此模式，請在 `withCryptoConfiguration`方法中指定 `StrictAuthenticatedEncryption`值。

> **Note**
>
> 若要使用用戶端驗證加密，您必須在應用程式的 classpath 中包含最新的 Bouncy Castle jar 檔案。

Code

```
AmazonS3EncryptionV2 s3Encryption = AmazonS3EncryptionClientV2Builder.standard()
```

```
        .withRegion(Regions.US_WEST_2)
        .withCryptoConfiguration(new
 CryptoConfigurationV2().withCryptoMode((CryptoMode.StrictAuthenticatedEncryption)))
        .withEncryptionMaterialsProvider(new StaticEncryptionMaterialsProvider(new
 EncryptionMaterials(secretKey)))
        .build();

s3Encryption.putObject(bucket_name, ENCRYPTED_KEY2, "This is the 2nd content to
 encrypt");
```

已驗證的加密模式

當您使用 AuthenticatedEncryption 模式時，會在加密期間套用改進的金鑰包裝演算法。在此模式下解密時，演算法可以驗證解密物件的完整性，並在檢查失敗時擲回例外狀況。如需驗證加密運作方式的詳細資訊，請參閱Amazon S3 用戶端驗證加密部落格文章。

> **Note**
>
> 若要使用用戶端驗證加密，您必須在應用程式的 classpath 中包含最新的 Bouncy Castle jar 檔案。

若要啟用此模式，請在 withCryptoConfiguration方法中指定 AuthenticatedEncryption值。

Code

```
AmazonS3EncryptionV2 s3EncryptionClientV2 =
 AmazonS3EncryptionClientV2Builder.standard()
        .withRegion(Regions.DEFAULT_REGION)
        .withClientConfiguration(new ClientConfiguration())
        .withCryptoConfiguration(new
 CryptoConfigurationV2().withCryptoMode(CryptoMode.AuthenticatedEncryption))
        .withEncryptionMaterialsProvider(new StaticEncryptionMaterialsProvider(new
 EncryptionMaterials(secretKey)))
        .build();

s3EncryptionClientV2.putObject(bucket_name, ENCRYPTED_KEY1, "This is the 1st content to
 encrypt");
```

## Amazon S3 使用 AWS KMS 受管金鑰的用戶端加密

下列範例使用 AmazonS3EncryptionClientV2Builder 類別來建立已啟用用戶端加密的 Amazon S3 用戶端。設定完成後，任何使用此用戶端上傳到 Amazon S3 的物件都會加密。您從 Amazon S3 使用此用戶端取得的任何物件都會自動解密。

> **Note**
>
> 下列範例示範如何搭配 AWS KMS 受管金鑰使用 Amazon S3 用戶端加密。若要了解如何使用加密搭配您自己的金鑰，請參閱Amazon S3 用戶端加密搭配用戶端主金鑰。

啟用用戶端加密時，您可以選擇兩種 Amazon S3 加密模式：嚴格驗證或驗證。下列各節說明如何啟用每種類型。若要了解每個模式使用哪些演算法，請參閱 CryptoMode 定義。

必要的匯入

匯入下列類別以取得這些範例。

匯入

```
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;
```

嚴格驗證加密

如果`CryptoMode`未指定 ，嚴格驗證加密是預設模式。

若要明確啟用此模式，請在 `withCryptoConfiguration`方法中指定 `StrictAuthenticatedEncryption`值。

> **Note**
>
> 若要使用用戶端驗證加密，您必須在應用程式的 classpath 中包含最新的 Bouncy Castle jar 檔案。

Code

```
AmazonS3EncryptionV2 s3Encryption = AmazonS3EncryptionClientV2Builder.standard()
         .withRegion(Regions.US_WEST_2)
         .withCryptoConfiguration(new
 CryptoConfigurationV2().withCryptoMode((CryptoMode.StrictAuthenticatedEncryption)))
         .withEncryptionMaterialsProvider(new KMSEncryptionMaterialsProvider(keyId))
         .build();

s3Encryption.putObject(bucket_name, ENCRYPTED_KEY3, "This is the 3rd content to encrypt
 with a key created in the {console}");
System.out.println(s3Encryption.getObjectAsString(bucket_name, ENCRYPTED_KEY3));
```

在 Amazon S3 加密用戶端上呼叫 `putObject`方法以上傳物件。

Code

```
s3Encryption.putObject(bucket_name, ENCRYPTED_KEY3, "This is the 3rd content to encrypt
 with a key created in the {console}");
```

您可以使用相同的用戶端擷取物件。此範例會呼叫 `getObjectAsString`方法，以擷取存放的字串。

Code

```
System.out.println(s3Encryption.getObjectAsString(bucket_name, ENCRYPTED_KEY3));
```

已驗證的加密模式

當您使用 `AuthenticatedEncryption` 模式時，會在加密期間套用改進的金鑰包裝演算法。在此模式下解密時，演算法可以驗證解密物件的完整性，並在檢查失敗時擲回例外狀況。如需驗證加密運作方式的詳細資訊，請參閱Amazon S3 用戶端驗證加密部落格文章。

> **Note**
>
> 若要使用用戶端驗證加密，您必須在應用程式的 classpath 中包含最新的 Bouncy Castle jar 檔案。

若要啟用此模式，請在 `withCryptoConfiguration`方法中指定 `AuthenticatedEncryption`值。

Code

```
AmazonS3EncryptionV2 s3Encryption = AmazonS3EncryptionClientV2Builder.standard()
         .withRegion(Regions.US_WEST_2)
         .withCryptoConfiguration(new
 CryptoConfigurationV2().withCryptoMode((CryptoMode.AuthenticatedEncryption)))
         .withEncryptionMaterialsProvider(new KMSEncryptionMaterialsProvider(keyId))
         .build();
```

設定 AWS KMS 用戶端

除非明確指定，否則 Amazon S3 加密用戶端預設會建立 AWS KMS 用戶端。

若要為此自動建立的 AWS KMS 用戶端設定區域，請設定 `awsKmsRegion`。

Code

```
Region kmsRegion = Region.getRegion(Regions.AP_NORTHEAST_1);

AmazonS3EncryptionV2 s3Encryption = AmazonS3EncryptionClientV2Builder.standard()
        .withRegion(Regions.US_WEST_2)
        .withCryptoConfiguration(new
 CryptoConfigurationV2().withAwsKmsRegion(kmsRegion))
        .withEncryptionMaterialsProvider(new KMSEncryptionMaterialsProvider(keyId))
        .build();
```

或者，您可以使用自己的 AWS KMS 用戶端來初始化加密用戶端。

Code

```
AWSKMS kmsClient = AWSKMSClientBuilder.standard()
        .withRegion(Regions.US_WEST_2);
        .build();
```

```
AmazonS3EncryptionV2 s3Encryption = AmazonS3EncryptionClientV2Builder.standard()
        .withRegion(Regions.US_WEST_2)
        .withKmsClient(kmsClient)
        .withCryptoConfiguration(new
 CryptoConfigurationV2().withCryptoMode((CryptoMode.AuthenticatedEncryption)))
        .withEncryptionMaterialsProvider(new KMSEncryptionMaterialsProvider(keyId))
        .build();
```

# Amazon SQS 使用 的範例 適用於 Java 的 AWS SDK

本節提供使用適用於 Java 的 AWS SDK編寫 Amazon SQS 程式的範例。

> **Note**
>
> 這些範例僅包含示範每種技術所需的程式碼。GitHub 上提供完整程式碼範例。您可以從那裡下載單一原始檔案或將儲存庫複製到本機，以取得建置和執行的所有範例。

主題

- 使用 Amazon SQS 訊息佇列
- 傳送、接收和刪除 Amazon SQS 訊息
- 啟用 Amazon SQS 訊息佇列的長輪詢
- 在 中設定可見性逾時 Amazon SQS
- 在 中使用無效字母佇列 Amazon SQS

## 使用 Amazon SQS 訊息佇列

訊息佇列是用來可靠地傳送訊息的邏輯容器 Amazon SQS。有兩種佇列類型：標準和先進先出 (FIFO)。若要進一步了解佇列和這些類型之間的差異，請參閱 Amazon SQS 開發人員指南。

本主題說明如何使用 建立、列出、刪除和取得 Amazon SQS 佇列的 URL 適用於 Java 的 AWS SDK。

### 建立佇列

使用 AmazonSQS 用戶端的 `createQueue`方法，提供描述佇列參數的 CreateQueueRequest 物件。

匯入

```
import com.amazonaws.services.sqs.AmazonSQS;
import com.amazonaws.services.sqs.AmazonSQSClientBuilder;
import com.amazonaws.services.sqs.model.AmazonSQSException;
import com.amazonaws.services.sqs.model.CreateQueueRequest;
```

Code

```
AmazonSQS sqs = AmazonSQSClientBuilder.defaultClient();
CreateQueueRequest create_request = new CreateQueueRequest(QUEUE_NAME)
        .addAttributesEntry("DelaySeconds", "60")
        .addAttributesEntry("MessageRetentionPeriod", "86400");

try {
    sqs.createQueue(create_request);
} catch (AmazonSQSException e) {
    if (!e.getErrorCode().equals("QueueAlreadyExists")) {
        throw e;
    }
}
```

您可以使用簡化形式的 `createQueue`，只需要佇列名稱來建立標準佇列。

```
sqs.createQueue("MyQueue" + new Date().getTime());
```

請參閱 GitHub 上的完整範例。

## 列出佇列

若要列出帳戶的 Amazon SQS 佇列，請呼叫 AmazonSQS 用戶端的 `listQueues`方法。

匯入

```
import com.amazonaws.services.sqs.AmazonSQS;
import com.amazonaws.services.sqs.AmazonSQSClientBuilder;
import com.amazonaws.services.sqs.model.ListQueuesResult;
```

Code

```
AmazonSQS sqs = AmazonSQSClientBuilder.defaultClient();
ListQueuesResult lq_result = sqs.listQueues();
System.out.println("Your SQS Queue URLs:");
```

```
for (String url : lq_result.getQueueUrls()) {
    System.out.println(url);
}
```

使用無任何參數的`listQueues`過載會傳回所有佇列。您可以透過傳遞`ListQueuesRequest`物件來篩選傳回的結果。

匯入

```
import com.amazonaws.services.sqs.AmazonSQS;
import com.amazonaws.services.sqs.AmazonSQSClientBuilder;
import com.amazonaws.services.sqs.model.ListQueuesRequest;
```

Code

```
AmazonSQS sqs = AmazonSQSClientBuilder.defaultClient();
String name_prefix = "Queue";
lq_result = sqs.listQueues(new ListQueuesRequest(name_prefix));
System.out.println("Queue URLs with prefix: " + name_prefix);
for (String url : lq_result.getQueueUrls()) {
    System.out.println(url);
}
```

請參閱 GitHub 上的完整範例。

## 取得佇列 URL

呼叫 AmazonSQS 用戶端的 `getQueueUrl`方法。

匯入

```
import com.amazonaws.services.sqs.AmazonSQS;
import com.amazonaws.services.sqs.AmazonSQSClientBuilder;
```

Code

```
AmazonSQS sqs = AmazonSQSClientBuilder.defaultClient();
String queue_url = sqs.getQueueUrl(QUEUE_NAME).getQueueUrl();
```

請參閱 GitHub 上的完整範例。

### 刪除佇列

將佇列的 URL 提供給 AmazonSQS 用戶端的 deleteQueue方法。

匯入

```
import com.amazonaws.services.sqs.AmazonSQS;
import com.amazonaws.services.sqs.AmazonSQSClientBuilder;
```

Code

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

請參閱 GitHub 上的完整範例。

### 詳細資訊

- 《Amazon SQS 開發人員指南》中的佇列如何運作 Amazon SQS
- Amazon SQS API 參考中的 CreateQueue
- API Amazon SQS 參考中的 GetQueueUrl
- Amazon SQS API 參考中的 ListQueues
- Amazon SQS API 參考中的 DeleteQueues

## 傳送、接收和刪除 Amazon SQS 訊息

本主題說明如何傳送、接收和刪除 Amazon SQS 訊息。訊息一律使用 SQS 佇列來傳送。

### 傳送訊息

呼叫 AmazonSQS 用戶端的 sendMessage方法，將單一訊息新增至 Amazon SQS 佇列。提供 SendMessageRequest 物件，其中包含佇列的 URL、訊息本文，以及選用的延遲值 (以秒為單位)。

匯入

```
import com.amazonaws.services.sqs.AmazonSQS;
import com.amazonaws.services.sqs.AmazonSQSClientBuilder;
import com.amazonaws.services.sqs.model.SendMessageRequest;
```

Code

```
SendMessageRequest send_msg_request = new SendMessageRequest()
        .withQueueUrl(queueUrl)
        .withMessageBody("hello world")
        .withDelaySeconds(5);
sqs.sendMessage(send_msg_request);
```

請參閱 GitHub 上的完整範例。

一次傳送多個訊息

您可以在單一請求中傳送多個訊息。若要傳送多則訊息，請使用 AmazonSQS 用戶端的 `sendMessageBatch`方法，該方法會採用 SendMessageBatchRequest，其中包含要傳送的佇列 URL 和訊息清單（每個都包含一個 SendMessageBatchRequestEntry)。您也可以為每個訊息設定選用的延遲值。

匯入

```
import com.amazonaws.services.sqs.model.SendMessageBatchRequest;
import com.amazonaws.services.sqs.model.SendMessageBatchRequestEntry;
```

Code

```
SendMessageBatchRequest send_batch_request = new SendMessageBatchRequest()
        .withQueueUrl(queueUrl)
        .withEntries(
                new SendMessageBatchRequestEntry(
                        "msg_1", "Hello from message 1"),
                new SendMessageBatchRequestEntry(
                        "msg_2", "Hello from message 2")
                        .withDelaySeconds(10));
sqs.sendMessageBatch(send_batch_request);
```

請參閱 GitHub 上的完整範例。

## 接收訊息

呼叫 AmazonSQS 用戶端的 `receiveMessage`方法，將佇列的 URL 傳遞給佇列，以擷取目前佇列中的任何訊息。訊息會以 Message 物件清單的形式傳回。

匯入

```
import com.amazonaws.services.sqs.AmazonSQSClientBuilder;
import com.amazonaws.services.sqs.model.AmazonSQSException;
import com.amazonaws.services.sqs.model.SendMessageBatchRequest;
```

Code

```
List<Message> messages = sqs.receiveMessage(queueUrl).getMessages();
```

### 在收到訊息後刪除訊息

收到訊息並處理其內容後，將訊息的接收控點和佇列 URL 傳送至 AmazonSQS 用戶端的 `deleteMessage`方法，從佇列刪除訊息。

Code

```
for (Message m : messages) {
    sqs.deleteMessage(queueUrl, m.getReceiptHandle());
}
```

請參閱 GitHub 上的完整範例。

### 詳細資訊

- 《Amazon SQS 開發人員指南》中的佇列如何運作 Amazon SQS
- API Amazon SQS 參考中的 SendMessage
- API Amazon SQS 參考中的 SendMessageBatch
- Amazon SQS API 參考中的 ReceiveMessage
- Amazon SQS API 參考中的 DeleteMessage

## 啟用 Amazon SQS 訊息佇列的長輪詢

Amazon SQS 根據預設， 會使用短輪詢，根據加權隨機分佈僅查詢伺服器子集，以判斷回應中是否包含任何訊息。

當沒有訊息可傳回以回覆傳送至 Amazon SQS 佇列的 ReceiveMessage 請求並消除誤報時，長輪詢有助於 Amazon SQS 降低使用 的成本。

> **Note**
>
> 您可以將長輪詢頻率設定為 1-20 秒。

## 建立佇列時啟用長輪詢

若要在建立 Amazon SQS 佇列時啟用長輪詢，請在 CreateQueueRequest 物件上設定 `ReceiveMessageWaitTimeSeconds` 屬性，然後再呼叫 AmazonSQS 類別的 `createQueue`方法。

匯入

```
import com.amazonaws.services.sqs.AmazonSQS;
import com.amazonaws.services.sqs.AmazonSQSClientBuilder;
import com.amazonaws.services.sqs.model.AmazonSQSException;
import com.amazonaws.services.sqs.model.CreateQueueRequest;
```

Code

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

// Enable long polling when creating a queue
CreateQueueRequest create_request = new CreateQueueRequest()
        .withQueueName(queue_name)
        .addAttributesEntry("ReceiveMessageWaitTimeSeconds", "20");

try {
    sqs.createQueue(create_request);
} catch (AmazonSQSException e) {
    if (!e.getErrorCode().equals("QueueAlreadyExists")) {
        throw e;
    }
}
```

請參閱 GitHub 上的完整範例。

## 在現有佇列上啟用長輪詢

除了在建立佇列時啟用長輪詢之外，您也可以在呼叫 AmazonSQS 類別的 `setQueueAttributes`方法之前，在 SetQueueAttributesRequest `ReceiveMessageWaitTimeSeconds`上設定 ，在現有佇列上啟用它。

匯入

```
import com.amazonaws.services.sqs.model.SetQueueAttributesRequest;
```

Code

```
SetQueueAttributesRequest set_attrs_request = new SetQueueAttributesRequest()
        .withQueueUrl(queue_url)
        .addAttributesEntry("ReceiveMessageWaitTimeSeconds", "20");
sqs.setQueueAttributes(set_attrs_request);
```

請參閱 GitHub 上的完整範例。

## 在收到訊息時啟用長輪詢

您可以在接收訊息時啟用長輪詢，方法是在您提供給 AmazonSQS 類別的 ReceiveMessageRequest `receiveMessage`方法上設定等待時間，以秒為單位。

> **Note**
>
> 您應該確保 AWS 用戶端的請求逾時大於最長輪詢時間 (20 秒），以便您的`receiveMessage`請求在等待下一個輪詢事件時不會逾時！

匯入

```
import com.amazonaws.services.sqs.model.ReceiveMessageRequest;
```

Code

```
ReceiveMessageRequest receive_request = new ReceiveMessageRequest()
        .withQueueUrl(queue_url)
        .withWaitTimeSeconds(20);
sqs.receiveMessage(receive_request);
```

請參閱 GitHub 上的完整範例。

## 詳細資訊

- 《Amazon SQS 開發人員指南》中的Amazon SQS 長輪詢

- Amazon SQS API 參考中的 CreateQueue
- Amazon SQS API 參考中的 ReceiveMessage
- Amazon SQS API 參考中的 SetQueueAttributes

## 在 中設定可見性逾時 Amazon SQS

在 中接收訊息時 Amazon SQS，訊息會保留在佇列中，直到刪除為止，以確保接收。在指定的可見性逾時後，後續請求中將可取得已接收但未刪除的訊息，以協助防止訊息在處理和刪除之前收到超過一次。

> **Note**
>
> 使用標準佇列時，可見性逾時並不能保證接收訊息兩次。如果您使用的是標準佇列，請確定您的程式碼可以處理相同訊息已交付多次的情況。

### 設定單一訊息的訊息可見性逾時

當您收到訊息時，您可以透過在傳遞給 AmazonSQS 類別`changeMessageVisibility`的 ChangeMessageVisibilityRequest 中傳遞其接收控點，來修改其可見性逾時。

匯入

```
import com.amazonaws.services.sqs.AmazonSQS;
import com.amazonaws.services.sqs.AmazonSQSClientBuilder;
```

Code

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

// Get the receipt handle for the first message in the queue.
String receipt = sqs.receiveMessage(queue_url)
                    .getMessages()
                    .get(0)
                    .getReceiptHandle();

sqs.changeMessageVisibility(queue_url, receipt, timeout);
```

請參閱 GitHub 上的完整範例。

## 一次設定多個訊息的訊息可見性逾時

若要一次設定多個訊息的訊息可見性逾時，請建立 ChangeMessageVisibilityBatchRequestEntry 物件清單，每個物件都包含唯一的 ID 字串和接收控點。然後，將清單傳遞至 Amazon SQS 用戶端類別的 changeMessageVisibilityBatch方法。

匯入

```
import com.amazonaws.services.sqs.AmazonSQS;
import com.amazonaws.services.sqs.AmazonSQSClientBuilder;
import com.amazonaws.services.sqs.model.ChangeMessageVisibilityBatchRequestEntry;
import java.util.ArrayList;
import java.util.List;
```

Code

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

List<ChangeMessageVisibilityBatchRequestEntry> entries =
    new ArrayList<ChangeMessageVisibilityBatchRequestEntry>();

entries.add(new ChangeMessageVisibilityBatchRequestEntry(
            "unique_id_msg1",
            sqs.receiveMessage(queue_url)
               .getMessages()
               .get(0)
               .getReceiptHandle())
        .withVisibilityTimeout(timeout));

entries.add(new ChangeMessageVisibilityBatchRequestEntry(
            "unique_id_msg2",
            sqs.receiveMessage(queue_url)
               .getMessages()
               .get(0)
               .getReceiptHandle())
        .withVisibilityTimeout(timeout + 200));

sqs.changeMessageVisibilityBatch(queue_url, entries);
```

請參閱 GitHub 上的完整範例。

### 詳細資訊

- 《Amazon SQS 開發人員指南》中的可見性逾時
- Amazon SQS API 參考中的 SetQueueAttributes
- Amazon SQS API 參考中的 GetQueueAttributes
- Amazon SQS API 參考中的 ReceiveMessage
- Amazon SQS API 參考中的 ChangeMessageVisibility
- Amazon SQS API 參考中的 ChangeMessageVisibilityBatch

## 在 中使用無效字母佇列 Amazon SQS

Amazon SQS 支援無效字母佇列。無效字母佇列是其他（來源）佇列可以針對無法成功處理的訊息設定目標的佇列。您可以在無效字母佇列中擱置並隔離這類訊息，以確定無法成功處理訊息的原因。

### 建立無效字母佇列

無效字母佇列的建立方式與一般佇列相同，但有下列限制：

- 無效字母佇列必須與來源佇列的佇列類型相同 (FIFO 或標準）。
- 無效字母佇列必須使用與來源佇列相同的 AWS 帳戶 和 區域建立。

在這裡，我們建立兩個相同的 Amazon SQS 佇列，其中一個將做為無效字母佇列：

匯入

```
import com.amazonaws.services.sqs.AmazonSQS;
import com.amazonaws.services.sqs.AmazonSQSClientBuilder;
import com.amazonaws.services.sqs.model.AmazonSQSException;
```

Code

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

// Create source queue
try {
    sqs.createQueue(src_queue_name);
} catch (AmazonSQSException e) {
    if (!e.getErrorCode().equals("QueueAlreadyExists")) {
```

```
        throw e;
    }
}

// Create dead-letter queue
try {
    sqs.createQueue(dl_queue_name);
} catch (AmazonSQSException e) {
    if (!e.getErrorCode().equals("QueueAlreadyExists")) {
        throw e;
    }
}
```

請參閱 GitHub 上的完整範例。

## 指定來源佇列的無效字母佇列

若要指定無效字母佇列，您必須先建立再驅動政策，然後在佇列的屬性中設定政策。再驅動政策是在 JSON 中指定，並指定無效字母佇列的 ARN，以及在傳送至無效字母佇列之前可接收和處理訊息的次數上限。

若要設定來源佇列的再驅動政策，請使用 SetQueueAttributesRequest 物件呼叫 AmazonSQS 類別的 `setQueueAttributes`方法，而您已使用 JSON 再驅動政策設定 `RedrivePolicy` 屬性。

匯入

```
import com.amazonaws.services.sqs.model.GetQueueAttributesRequest;
import com.amazonaws.services.sqs.model.GetQueueAttributesResult;
import com.amazonaws.services.sqs.model.SetQueueAttributesRequest;
```

Code

```
String dl_queue_url = sqs.getQueueUrl(dl_queue_name)
                         .getQueueUrl();

GetQueueAttributesResult queue_attrs = sqs.getQueueAttributes(
        new GetQueueAttributesRequest(dl_queue_url)
            .withAttributeNames("QueueArn"));

String dl_queue_arn = queue_attrs.getAttributes().get("QueueArn");

// Set dead letter queue with redrive policy on source queue.
```

```
String src_queue_url = sqs.getQueueUrl(src_queue_name)
                          .getQueueUrl();

SetQueueAttributesRequest request = new SetQueueAttributesRequest()
        .withQueueUrl(src_queue_url)
        .addAttributesEntry("RedrivePolicy",
                "{\"maxReceiveCount\":\"5\", \"deadLetterTargetArn\":\""
                + dl_queue_arn + "\"}");

sqs.setQueueAttributes(request);
```

請參閱 GitHub 上的完整範例。

### 詳細資訊

- 《 Amazon SQS 開發人員指南》中的使用 Amazon SQS 無效字母佇列
- Amazon SQS API 參考中的 SetQueueAttributes

## Amazon SWF 使用 的範例 適用於 Java 的 AWS SDK

Amazon SWF 是一種工作流程管理服務，可協助開發人員建置和擴展分散式工作流程，這些工作流程可以具有由活動、子工作流程或甚至是 Lambda 任務組成的平行或循序步驟。

Amazon SWF 使用 適用於 Java 的 AWS SDK、使用 SWF 用戶端物件或使用 AWS Flow Framework 適用於 Java 的 有兩種方式。 AWS Flow Framework 適用於 Java 的 一開始較難設定，因為它會大量使用註釋，並依賴其他程式庫，例如 AspectJ 和 Spring Framework。不過，對於大型或複雜的專案，您可以使用 AWS Flow Framework 適用於 Java 的 來節省編碼時間。如需詳細資訊，請參閱適用於 AWS Flow Framework Java 的 開發人員指南。

本節提供直接 Amazon SWF 使用 適用於 Java 的 AWS SDK 用戶端的程式設計範例。

**主題**

- SWF 基本概念
- 建置簡單的 Amazon SWF 應用程式
- Lambda 任務
- 正常關閉活動和工作流程工作者
- 註冊網域
- 列出網域

# SWF 基本概念

這些是 Amazon SWF 使用 的一般模式 適用於 Java 的 AWS SDK。它主要用於參考。如需更完整的簡介教學課程，請參閱建置簡易 Amazon SWF 應用程式。

## 相依性

基本 Amazon SWF 應用程式將需要下列相依性，這些相依性包含在 中 適用於 Java 的 AWS SDK：

- 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**
>
> 這些套件的版本編號會根據您擁有的開發套件版本而有所不同，但開發套件隨附的版本已經過相容性測試，且是您應使用的版本。

AWS Flow Framework for Java 應用程式需要額外的設定和其他相依性。如需使用架構的詳細資訊，請參閱AWS Flow Framework 適用於 Java 的 開發人員指南。

## 匯入

一般而言，您可以使用下列匯入進行程式碼開發：

```
import com.amazonaws.services.simpleworkflow.AmazonSimpleWorkflowClientBuilder;
import com.amazonaws.services.simpleworkflow.model.*;
```

不過，最好只匯入您需要的類別。最後，您可能會在`com.amazonaws.services.simpleworkflow.model`工作區中指定特定類別：

```
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;
```

如果您使用的是 AWS Flow Framework 適用於 Java 的 ，則會從`com.amazonaws.services.simpleworkflow.flow`工作區匯入類別。例如：

```
import com.amazonaws.services.simpleworkflow.AmazonSimpleWorkflow;
import com.amazonaws.services.simpleworkflow.flow.ActivityWorker;
```

> **Note**
>
> AWS Flow Framework 適用於 Java 的 具有 基礎以外的其他需求 適用於 Java 的 AWS SDK。如需詳細資訊，請參閱適用於 AWS Flow Framework Java 的 開發人員指南。

### 使用 SWF 用戶端類別

您的 基本界面 Amazon SWF 是透過 AmazonSimpleWorkflowClient 或 AmazonSimpleWorkflowAsyncClient 類別。兩者之間的主要區別在於 `\*AsyncClient`類別會傳回未來物件，以進行並行（非同步）程式設計。

```
AmazonSimpleWorkflowClient swf = AmazonSimpleWorkflowClientBuilder.defaultClient();
```

## 建置簡單的 Amazon SWF 應用程式

本主題將介紹如何使用 編寫Amazon SWF應用程式 適用於 Java 的 AWS SDK，同時在整個過程中介紹幾個重要概念。

### 關於範例

此範例專案會建立工作流程，其中包含單一活動，接受透過 AWS 雲端傳遞的工作流程資料（在 HelloWorld 的傳統中，它會是有人打招呼的名稱），然後列印問候語以回應。

雖然這在表面上看起來非常簡單， Amazon SWF 但應用程式由多個組件一起運作：

- 網域，用作工作流程執行資料的邏輯容器。

- 一或多個工作流程，代表定義工作流程活動和子工作流程之邏輯執行順序的程式碼元件。
- 工作流程工作者，也稱為決策者，會輪詢決策任務並排程活動或子工作流程以回應。
- 一或多個活動，每個活動代表工作流程中的工作單位。
- 輪詢活動任務並執行活動方法以回應的活動工作者。
- 一或多個任務清單，這些是 維護的佇列， Amazon SWF 用於向工作流程和活動工作者發出請求。任務清單上適用於工作流程工作者的任務稱為決策任務。適用於活動工作者的那些任務稱為活動任務。
- 開始工作流程執行的工作流程啟動者。

在幕後， Amazon SWF 會協調這些元件的操作、協調其來自 AWS 雲端的流程、在它們之間傳遞資料、處理逾時和活動訊號通知，以及記錄工作流程執行歷史記錄。

## 先決條件

### 開發環境

本教學中使用的開發環境包含：

- 適用於 Java 的 AWS SDK。
- Apache Maven (3.3.1)。
- JDK 1.7 或更新版本。本教學課程是使用 JDK 1.8.0 進行開發和測試。
- 良好的 Java 文字編輯器 （ 由您選擇 ）。

> **Note**
>
> 如果您使用與 Maven 不同的建置系統，您仍然可以使用適合您環境的步驟建立專案，並使用此處提供的概念來遵循。有關設定和使用 適用於 Java 的 AWS SDK 搭配各種建置系統的詳細資訊，請參閱 入門。
> 同樣地，但只要付出更多努力，就可以使用任何支援 AWS SDKs 實作此處顯示的步驟 Amazon SWF。

包含所有必要的外部相依性 適用於 Java 的 AWS SDK，因此無需下載任何額外的項目。

### AWS 存取

若要成功完成本教學課程，您必須能夠存取 AWS 存取入口網站，如本指南的基本設定一節所述。

這些指示說明如何存取您複製並貼到本機共用credentials檔案的臨時登入資料。您貼上的臨時登入資料必須與 中的 IAM 角色相關聯 AWS IAM Identity Center ，該角色具有存取 Amazon SWF 的許可。貼上臨時登入資料後，您的credentials檔案看起來會類似以下內容。

```
[default]
aws_access_key_id=AKIAIOSFODNN7EXAMPLE
aws_secret_access_key=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
aws_session_token=IQoJb3JpZ2luX2IQoJb3JpZ2luX2IQoJb3JpZ2luX2IQoJb3JpZ2luX2IQoJb3JpZVERYLONGSTRI
```

這些臨時登入資料與default設定檔相關聯。

## 建立 SWF 專案

1. 使用 Maven 啟動新專案：

   ```
   mvn archetype:generate -DartifactId=helloswf \
   -DgroupId=aws.example.helloswf -DinteractiveMode=false
   ```

   這會建立具有標準 maven 專案結構的新專案：

   ```
   helloswf
   ### pom.xml
   ### src
       ### main
       #   ### java
       #       ### aws
       #           ### example
       #               ### helloswf
       #                   ### App.java
       ### test
           ### ...
   ```

   您可以忽略或刪除test目錄及其包含的所有內容，我們不會將其用於本教學課程。您也可以刪除App.java，因為我們會將它取代為新的類別。
2. 編輯專案pom.xml的檔案，並在 <dependencies>區塊中新增其相依性，以新增 aws-java-sdk-simpleworkflow 模組。

   ```
   <dependencies>
     <dependency>
       <groupId>com.amazonaws</groupId>
   ```

```
    <artifactId>aws-java-sdk-simpleworkflow</artifactId>
    <version>1.11.1000</version>
  </dependency>
</dependencies>
```

3. 請確定 Maven 使用 JDK 1.7+ 支援建置您的專案。在 中將下列項目新增至您的專案 (<dependencies>區塊之前或之後）pom.xml：

```
<build>
  <plugins>
    <plugin>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>3.6.1</version>
      <configuration>
          <source>1.8</source>
          <target>1.8</target>
      </configuration>
    </plugin>
  </plugins>
</build>
```

## 為專案編寫程式碼

範例專案將包含四個不同的應用程式，我們將逐一造訪：

- HelloTypes.java - 包含與其他元件共用的專案網域、活動和工作流程類型資料。它還處理向 SWF 註冊這些類型。
- ActivityWorker.java - 包含活動工作者，它會輪詢活動任務並執行活動以回應。
- WorkflowWorker.java - 包含工作流程工作者 （決策者），它會輪詢決策任務並排程新活動。
- WorkflowStarter.java - 包含工作流程啟動器，這會啟動新的工作流程執行，這將導致 SWF 開始產生決策和工作流程任務，以供工作者使用。

### 所有來源檔案的常見步驟

您為容納 Java 類別而建立的所有檔案都會有幾個共同點。基於時間考量，每次將新檔案新增至專案時，這些步驟都會隱含：

1. 在專案src/main/java/aws/example/helloswf/目錄中的 中建立 檔案。
2. 將package宣告新增至每個檔案的開頭，以宣告其命名空間。範例專案使用：

```
package aws.example.helloswf;
```

3. 新增 AmazonSimpleWorkflowClient 類別和 `com.amazonaws.services.simpleworkflow.model` 命名空間中多個類別的`import`宣告。為了簡化物件，我們將使用：

```
import com.amazonaws.regions.Regions;
import com.amazonaws.services.simpleworkflow.AmazonSimpleWorkflow;
import com.amazonaws.services.simpleworkflow.AmazonSimpleWorkflowClientBuilder;
import com.amazonaws.services.simpleworkflow.model.*;
```

### 註冊網域、工作流程和活動類型

首先，我們會建立新的可執行檔類別 `HelloTypes.java`。此檔案將包含工作流程不同部分需要知道的共用資料，例如活動的名稱和版本，以及工作流程類型、網域名稱和任務清單名稱。

1. 開啟文字編輯器並建立檔案 `HelloTypes.java`，根據常見步驟新增套件宣告和匯入。
2. 宣告 `HelloTypes`類別，並提供用於已註冊活動和工作流程類型的值：

```
    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";
```

   這些值將在整個程式碼中使用。

3. 在字串宣告之後，建立 AmazonSimpleWorkflowClient 類別的執行個體。這是 所提供 Amazon SWF 方法的基本界面 適用於 Java 的 AWS SDK。

```
private static final AmazonSimpleWorkflow swf =

 AmazonSimpleWorkflowClientBuilder.standard().withRegion(Regions.DEFAULT_REGION).build();
```

   上一個程式碼片段假設暫時登入資料與`default`設定檔相關聯。如果您使用不同的設定檔，請修改上述程式碼，如下所示，並將 *`profile_name`* 取代為實際的設定檔名稱。

```
private static final AmazonSimpleWorkflow swf =
```

```
        AmazonSimpleWorkflowClientBuilder
                .standard()
                .withCredentials(new ProfileCredentialsProvider("profile_name"))
                .withRegion(Regions.DEFAULT_REGION)
                .build();
```

4. 新增函數以註冊 SWF 網域。網域是許多相關 SWF 活動和工作流程類型的邏輯容器。SWF 元件只能在相同網域中存在時互相通訊。

```
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!");
}
```

註冊網域時，您會提供名稱（一組 1 到 256 個字元，不包括 :、/、|、控制字元或常值字串 \`arn') 和保留期間，這是工作流程執行完成後 Amazon SWF ，工作流程執行歷史記錄資料的保留天數。工作流程執行保留期上限為 90 天。如需詳細資訊，請參閱 RegisterDomainRequest。

如果具有該名稱的網域已存在，則會引發 DomainAlreadyExistsException。由於我們未考量網域是否已建立，因此可以忽略例外狀況。

> **Note**
>
> 此程式碼示範使用 適用於 Java 的 AWS SDK 方法時的常見模式，方法的資料是由 `simpleworkflow.model` 命名空間中的類別提供，您可以使用可鏈結`0with*`的方法執行個體化和填入。

5. 新增函數以註冊新的活動類型。活動代表工作流程中的工作單位。

```
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!");
    }
```

活動類型由名稱和版本識別，用於唯一識別其註冊網域中任何其他人的活動。活動也包含許多選用參數，例如用於從 SWF 接收任務和資料的預設任務清單，以及可用於限制活動執行不同部分需要多長時間的不同逾時。如需詳細資訊，請參閱 RegisterActivityTypeRequest。

> **Note**
>
> 所有逾時值都以秒為單位指定。如需Amazon SWF 逾時如何影響工作流程執行的完整說明，請參閱逾時類型。

如果您嘗試註冊的活動類型已存在，則會引發 TypeAlreadyExistsException。新增 函數以註冊新的工作流程類型。工作流程也稱為決策者，代表工作流程執行的邏輯。

+

```
    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!");
    }
```

+

與活動類型類似，工作流程類型會依名稱和版本識別，並具有可設定的逾時。如需詳細資訊，請參閱 RegisterWorkflowTypeRequest。

+

如果您嘗試註冊的工作流程類型已存在，則會引發 TypeAlreadyExistsException。最後，提供 main方法讓類別可執行，進而註冊網域、活動類型和工作流程類型：

+

```
    registerDomain();
    registerWorkflowType();
    registerActivityType();
```

您可以立即建置並執行應用程式以執行註冊指令碼，或繼續編碼活動和工作流程工作者。網域、工作流程和活動註冊完成後，就不需要再次執行此操作，這些類型會持續存在，直到您自行棄用它們為止。

實作活動工作者

活動是工作流程中的基本工作單位。工作流程提供邏輯、排程要執行的活動 （或其他要採取的動作）以回應決策任務。典型的工作流程通常包含許多可以同步、非同步或兩者組合執行的活動。

活動工作者是輪詢 為 Amazon SWF 回應工作流程決策而產生之活動任務的程式碼位元。收到活動任務時，它會執行對應的活動，並將成功/失敗回應傳回工作流程。

我們將實作簡單的活動工作者，以推動單一活動。

1. 開啟您的文字編輯器並建立檔案 ActivityWorker.java，根據常見步驟新增套件宣告和匯入。

```
import com.amazonaws.regions.Regions;
import com.amazonaws.services.simpleworkflow.AmazonSimpleWorkflow;
import com.amazonaws.services.simpleworkflow.AmazonSimpleWorkflowClientBuilder;
import com.amazonaws.services.simpleworkflow.model.*;
```

2. 將 ActivityWorker類別新增至 檔案，並為其提供資料成員，以保留我們將用來與之互動的 SWF 用戶端 Amazon SWF：

```
    private static final AmazonSimpleWorkflow swf =

 AmazonSimpleWorkflowClientBuilder.standard().withRegion(Regions.DEFAULT_REGION).build();
```

3. 新增我們將用作活動的方法：

```
private static String sayHello(String input) throws Throwable {
    return "Hello, " + input + "!";
```

```
}
```

活動只需要一個字串，將其合併為問候語並傳回結果。雖然此活動不太可能引發例外狀況，但最好設計活動，以便在發生錯誤時引發錯誤。

4. 新增main我們將用作活動任務輪詢方法的方法。我們會新增一些程式碼來輪詢活動任務的任務清單，以開始執行此作業：

```
        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();
```

活動 Amazon SWF 透過呼叫 SWF 用戶端的 pollForActivityTask方法接收來自 的任務，指定要在傳入 PollForActivityTaskRequest 中使用的網域和任務清單。

收到任務後，我們會呼叫任務的 getTaskToken方法來擷取其唯一識別符。

5. 接著，撰寫一些程式碼來處理進來的任務。在輪詢任務並擷取其任務字符的程式碼之後，立即將以下內容新增至您的main方法。

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

如果任務字符不是 null，我們可以開始執行活動方法 (sayHello)，提供它與任務一起傳送的輸入資料。

如果任務成功（未產生錯誤），則工作者會使用包含任務字符和活動結果資料的 RespondActivityTaskCompletedRequest 物件呼叫 SWF 用戶端的 respondActivityTaskCompleted方法來回應 SWF。

另一方面，如果任務失敗，我們會透過使用 RespondActivityTaskFailedRequest 物件呼叫 respondActivityTaskFailed方法，將任務權杖和錯誤的相關資訊傳遞給它來回應。

> **Note**
>
> 如果終止，此活動將不會正常關閉。雖然它超出本教學課程的範圍，但此活動工作者的替代實作會在隨附的主題中提供，即關閉活動和工作流程工作者 Gracefully。

實作工作流程工作者

您的工作流程邏輯位於稱為工作流程工作者的程式碼中。工作流程工作者會輪詢網域 Amazon SWF 中由 傳送的決策任務，以及在預設任務清單上註冊工作流程類型的決策任務。

當工作流程工作者收到任務時，會做出某種決策（通常是是否排程新活動），並採取適當的動作（例如排程活動）。

1. 開啟文字編輯器並建立檔案 WorkflowWorker.java，根據常見步驟新增套件宣告和匯入。

2. 將一些額外的匯入新增至 檔案：

```
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. 宣告 WorkflowWorker類別，並建立用於存取 SWF 方法的 AmazonSimpleWorkflowClient 類別執行個體。

```
    private static final AmazonSimpleWorkflow swf =

 AmazonSimpleWorkflowClientBuilder.standard().withRegion(Regions.DEFAULT_REGION).build();
```

4. 新增 main方法。方法會持續循環，使用 SWF 用戶端的 pollForDecisionTask方法輪詢決策任務。PollForDecisionTaskRequest 提供詳細資訊。

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

一旦收到任務，我們就會呼叫其getTaskToken方法，傳回可用來識別任務的字串。如果傳回的字符不是 null，我們會在 executeDecisionTask方法中進一步處理它，並傳遞任務字符和隨任務傳送的 HistoryEvent 物件清單。

5. 新增 executeDecisionTask方法，取得任務字符 (String) 和 HistoryEvent清單。

```
    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;
```

我們也設定一些資料成員來追蹤下列項目：

- 用於報告處理任務結果的決策物件清單。
- 保留「WorkflowExecutionStarted」事件提供之工作流程輸入的字串
- 排程和開啟（執行中）活動的計數，以避免在已排程或目前正在執行時排程相同的活動。
- 布林值，表示活動已完成。
- 保留活動結果的字串，用於將其作為我們的工作流程結果傳回。

6. 接下來，新增一些程式碼到 executeDecisionTask ，根據 getEventType方法報告的事件類型，處理與任務一起傳送的HistoryEvent物件。

```
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("]");
```

基於工作流程的目的，我們最感興趣的是：

- 「WorkflowExecutionStarted」事件，表示工作流程執行已開始（通常表示您應該在工作流程中執行第一個活動），並提供提供給工作流程的初始輸入。在這種情況下，它是我們問候語的名稱部分，因此會儲存在字串中，以便在排程要執行的活動時使用。
- 「ActivityTaskCompleted」事件，會在排程活動完成後傳送。事件資料也包含已完成活動的傳回值。由於我們只有一個活動，我們將使用該值作為整個工作流程的結果。

如果您的工作流程需要，則可以使用其他事件類型。如需每個事件類型的相關資訊，請參閱 HistoryEvent 類別描述。

+ 注意：Java 7 中引入switch了陳述式中的字串。如果您使用的是舊版 Java，則可以使用 EventType 類別，將 String傳回的 history_event.getType() 轉換為列舉值，然後String視需要返回 ：

```
EventType et = EventType.fromValue(event.getEventType());
```

1. 在switch陳述式之後，新增更多程式碼，根據收到的任務做出適當的決策來回應。

```
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);
```

- 如果活動尚未排程，我們會回應決策，該ScheduleActivityTask決策會在 ScheduleActivityTaskDecisionAttributes 結構中提供 Amazon SWF 後續排程活動的相關資訊，也包括 Amazon SWF 應傳送至活動的任何資料。
- 如果活動已完成，我們將整個工作流程視為已完成，並以CompletedWorkflowExecution決策回應，填寫 CompleteWorkflowExecutionDecisionAttributes 結構，以提供已完成工作流程的詳細資訊。在這種情況下，我們會傳回活動的結果。

無論哪種情況，決策資訊都會新增至方法頂端宣告的Decision清單。

2. 傳回處理任務時收集的Decision物件清單，以完成決策任務。在我們撰寫executeDecisionTask的方法結尾新增此程式碼：

```
swf.respondDecisionTaskCompleted(
    new RespondDecisionTaskCompletedRequest()
```

```
        .withTaskToken(taskToken)
        .withDecisions(decisions));
```

SWF 用戶端的 respondDecisionTaskCompleted方法會取得識別任務的任務字符，以及Decision物件清單。

### 實作工作流程啟動者

最後，我們將編寫一些程式碼來開始工作流程執行。

1. 開啟文字編輯器並建立檔案 WorkflowStarter.java，根據常見步驟新增套件宣告和匯入。
2. 新增 WorkflowStarter類別：

```
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` 類別由單一方法 組成`main`，採用在命令列上傳遞的選用引數做為工作流程的輸入資料。

SWF 用戶端方法 `startWorkflowExecution`採用 StartWorkflowExecutionRequest 物件做為輸入。在這裡，除了指定要執行的網域和工作流程類型之外，我們還提供它：

- 人類可讀取的工作流程執行名稱
- 工作流程輸入資料 （ 在範例中的命令列提供 ）
- 逾時值，代表整個工作流程執行所需的時間，以秒為單位。

`startWorkflowExecution` 傳回的執行物件提供執行 ID，此值可用來識別工作流程執行 Amazon SWF歷史記錄中的此特定工作流程執行。

+ 注意：執行 ID 是由 產生 Amazon SWF，與您在啟動工作流程執行時傳入的工作流程執行名稱不同。

## 建置範例

若要使用 Maven 建置範例專案，請前往 `helloswf`目錄並輸入：

```
mvn package
```

產生的 `helloswf-1.0.jar`會在 `target`目錄中產生。

## 執行範例

此範例包含四個獨立的可執行檔類別，彼此獨立執行。

> **Note**
>
> 如果您使用的是 Linux、macOS 或 Unix 系統，您可以在單一終端機視窗中逐一執行所有系統。如果您執行 Windows，您應該開啟兩個額外的命令列執行個體，並導覽至每個執行個體中的`helloswf`目錄。

設定 Java classpath

雖然 Maven 已為您處理相依性，但若要執行範例，您需要在 Java classpath 上提供 AWS SDK 程式庫及其相依性。您可以將`CLASSPATH`環境變數設定為 AWS SDK 程式庫的位置和 SDK 中的`third-party/lib`目錄，其中包括必要的相依性：

```
export CLASSPATH='target/helloswf-1.0.jar:/path/to/sdk/lib/*:/path/to/sdk/third-party/
lib/*'
java example.swf.hello.HelloTypes
```

或使用 **`java`** 命令`-cp`的選項，在執行每個應用程式時設定 classpath。

```
java -cp target/helloswf-1.0.jar:/path/to/sdk/lib/*:/path/to/sdk/third-party/lib/* \
  example.swf.hello.HelloTypes
```

您使用的樣式由您決定。如果您在建置程式碼時沒有問題，則嘗試執行範例並取得一系列的 "NoClassDefFound" 錯誤，這可能是因為 classpath 設定不正確。

註冊網域、工作流程和活動類型

在執行工作者和工作流程啟動者之前，您需要註冊網域以及工作流程和活動類型。要執行此操作的程式碼是在註冊網域工作流程和活動類型中實作。

建置之後，如果您已設定 CLASSPATH，您可以執行 命令來執行註冊碼：

```
    echo 'Supply the name of one of the example classes as an argument.'
```

啟動活動和工作流程工作者

現在類型已註冊，您可以啟動活動和工作流程工作者。這些任務將繼續執行並輪詢任務，直到它們被刪除，因此您應該在單獨的終端視窗中執行它們，或者，如果您在 Linux、macOS 或 Unix 上執行，您可以使用 &運算子在執行時使它們產生單獨的程序。

```
    echo 'If there are arguments to the class, put them in quotes after the class
name.'
    exit 1
```

如果您在不同的視窗中執行這些命令，請省略每行的最終&運算子。

啟動工作流程執行

現在您的活動和工作流程工作者正在輪詢，您可以開始工作流程執行。此程序將執行，直到工作流程傳回完成的狀態。您應該在新的終端機視窗中執行它（除非您使用 &運算子，將工作者執行為新產生的程序）。

```
fi
```

> **Note**
>
> 如果您想要提供自己的輸入資料，這些資料會先傳遞至工作流程，然後再傳遞至活動，請將其新增至命令列。例如：
>
> ```
> echo "## Running $className..."
> ```

開始工作流程執行後，您應該會開始看到工作者和工作流程執行本身交付的輸出。當工作流程最終完成時，其輸出將列印到螢幕。

## 此範例的完整來源

您可以在 aws-java-developer-guide 儲存庫的 Github 上瀏覽此範例的完整來源。

## 如需詳細資訊

- 如果工作者在工作流程輪詢進行時關閉，則此處顯示的工作者可能會導致任務遺失。若要了解如何正常關閉工作者，請參閱關閉活動和工作流程工作者。
- 若要進一步了解 Amazon SWF，請造訪 Amazon SWF 首頁或檢視 Amazon SWF 開發人員指南。
- 您可以使用 AWS Flow Framework 適用於 Java 的 ，使用 註釋，以優雅 Java 樣式撰寫更複雜的工作流程。若要進一步了解，請參閱適用於 AWS Flow Framework Java 的 開發人員指南。

# Lambda 任務

做為活動或與其結合 Amazon SWF 使用的替代方案，您可以使用 Lambda 函數來代表工作流程中的工作單位，並以類似活動的方式進行排程。

本主題著重於如何使用 實作 Amazon SWF Lambda 任務 適用於 Java 的 AWS SDK。如需一般 Lambda 任務的詳細資訊，請參閱《 Amazon SWF 開發人員指南》中的AWS Lambda 任務。

## 設定跨服務 IAM 角色來執行 Lambda 函數

Amazon SWF 您必須先設定 IAM 角色，以授予代表您執行 Lambda 函數的 Amazon SWF 許可，才能執行 Lambda 函數。如需如何執行此操作的完整資訊，請參閱AWS Lambda 任務。

當您註冊將使用 Lambda 任務的工作流程時，將需要此 IAM 角色的 Amazon Resource Name (ARN)。

## 建立 Lambda 函數

您可以使用多種不同的語言撰寫 Lambda 函數，包括 Java。如需如何撰寫、部署和使用 Lambda 函數的完整資訊，請參閱 AWS Lambda 開發人員指南。

> **Note**
>
> 無論您使用哪種語言來撰寫 Lambda 函數，都可以排程和執行任何 Amazon SWF 工作流程，無論您的工作流程程式碼是使用哪種語言撰寫。 會 Amazon SWF 處理執行函數和將資料傳出和傳出的詳細資訊。

以下是一個簡單的 Lambda 函數，可用於取代建置簡單 Amazon SWF 應用程式中的活動。

- 此版本以 JavaScript 撰寫，可直接使用 輸入AWS 管理主控台：

```
exports.handler = function(event, context) {
    context.succeed("Hello, " + event.who + "!");
};
```

- 以下是以 Java 撰寫的相同函數，您也可以在 Lambda 上部署和執行：

```
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**
>
> 若要進一步了解如何將 Java 函數部署至 Lambda，請參閱《 AWS Lambda 開發人員指南》中的建立部署套件 (Java)。您也想要查看標題為 Java 中編寫 Lambda 函數程式設計模型的章節。

Lambda 函數會將事件或輸入物件做為第一個參數，並將內容物件做為第二個參數，提供執行 Lambda 函數之請求的相關資訊。此特定函數預期輸入會以 JSON 表示，who欄位設定為用來建立問候語的名稱。

## 註冊工作流程以搭配 Lambda 使用

若要讓工作流程排程 Lambda 函數，您必須提供 IAM 角色的名稱，該角色 Amazon SWF 提供叫用 Lambda 函數的許可。您可以使用 RegisterWorkflowTypeRequest 的 `withDefaultLambdaRole`或 `setDefaultLambdaRole`方法，在工作流程註冊期間設定此項目。

```
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) {
```

## 排程 Lambda 任務

排程 Lambda 任務與排程活動類似。您可以使用 `ScheduleLambdaFunction`DecisionType 和 ScheduleLambdaFunctionDecisionAttributes 提供決策。

```
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(
```

在 中ScheduleLambdaFuntionDecisionAttributes，您必須提供名稱，也就是要呼叫的 Lambda 函數的 ARN，以及 id，這是 Amazon SWF 用來識別歷史記錄日誌中 Lambda 函數的名稱。

您也可以為 Lambda 函數提供選用輸入，並設定其開始關閉逾時值，這是允許 Lambda 函數在產生LambdaFunctionTimedOut事件之前執行的秒數。

> **Note**
>
> 此程式碼使用 AWSLambdaClient 來擷取 Lambda 函數的 ARN，並指定函數名稱。您可以使用此技術來避免在程式碼中硬式編碼完整的 ARN （包括您的 AWS 帳戶 ID)。

## 在決策者中處理 Lambda 函數事件

Lambda 任務將產生許多事件，您可以在工作流程工作者中輪詢決策任務時對其採取動作，對應於 Lambda 任務的生命週期，並使用 EventType 值，例如 LambdaFunctionScheduled、LambdaFunctionStarted和 LambdaFunctionCompleted。如果 Lambda 函數失敗，或執行時間超過其設定的逾時值，您分別會收到 LambdaFunctionFailed或 LambdaFunctionTimedOut事件類型。

```
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;
```

### 從 Lambda 函數接收輸出

當您在 HistoryEvent LambdaFunctionCompleted`EventType, you can retrieve your 0 function's return value by first calling `getLambdaFunctionCompletedEventAttributes上收到 以取得 LambdaFunctionCompletedEventAttributes 物件，然後呼叫其getResult方法擷取 Lambda 函數的輸出：

```
 LambdaFunctionCompleted:
running_functions--;
```

### 此範例的完整來源

您可以在 aws-java-developer-guide 儲存庫的 Github 上瀏覽此範例的完整來源 ：github：`<awsdocs/aws-java-developer-guide/tree/master/doc_source/snippets/helloswf_lambda/>。 aws-java-developer-guide

## 正常關閉活動和工作流程工作者

建置簡易 Amazon SWF 應用程式主題提供簡單工作流程應用程式的完整實作，其中包含註冊應用程式、活動和工作流程工作者，以及工作流程入門。

工作者類別旨在持續執行，輪詢 Amazon SWF 傳送的任務，以執行活動或傳回決策。提出輪詢請求後， 會 Amazon SWF 記錄輪詢器，並嘗試指派任務給輪詢器。

如果工作流程工作者在長時間輪詢期間終止， Amazon SWF 可能仍會嘗試將任務傳送至已終止的工作者，導致任務遺失 （直到任務逾時）。

處理這種情況的一種方法是等待所有長輪詢請求傳回，再讓工作者終止。

在本主題中，我們將從 重寫活動工作者helloswf，使用 Java 的關閉勾點來嘗試正常關閉活動工作者。

以下是完整的程式碼：

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

```
}
```

在此版本中，原始版本中main函數中的輪詢程式碼已移至自己的方法 pollAndExecute。

main 函數現在使用 CountDownLatch 搭配關機掛鉤，讓執行緒在請求終止後等待最多 60 秒，然後讓執行緒關閉。

## 註冊網域

中的每個工作流程和活動Amazon SWF都需要網域才能執行。

1. 建立新的 RegisterDomainRequest 物件，提供至少網域名稱和工作流程執行保留期（這些參數都是必要的）。
2. 使用 RegisterDomainRequest 物件呼叫 AmazonSimpleWorkflowClient.registerDomain RegisterDomainRequest方法。
3. 如果您請求的網域已存在（在此情況下，通常不需要採取任何動作），請擷取 DomainAlreadyExistsException。

下列程式碼示範此程序：

```
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!");
    }
}
```

## 列出網域

您可以依註冊類型列出與您的帳戶和 AWS 區域相關聯的Amazon SWF網域。

1. 建立 ListDomainsRequest 物件，並指定您感興趣的網域註冊狀態，這是必要的。

2. 使用 ListDomainRequest 物件呼叫 AmazonSimpleWorkflowClient.listDomains。ListDomainRequest 結果會在 DomainInfos 物件中提供。
3. 在傳回的物件上呼叫 getDomainInfos，以取得 DomainInfo 物件的清單。
4. 在每個 DomainInfo 物件上呼叫 getName 以取得其名稱。

下列程式碼示範此程序：

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

# 軟體開發套件隨附的程式碼範例

適用於 Java 的 AWS SDK 隨附程式碼範例，可在可建置、可執行的程式中示範開發套件的許多功能。您可以使用 來研究或修改這些項目，以實作您自己的 AWS 解決方案 適用於 Java 的 AWS SDK。

## 如何取得範例

適用於 Java 的 AWS SDK 程式碼範例提供於 SDK 的範例目錄中。如果您使用設定 適用於 Java 的 AWS SDK 中的資訊下載並安裝 SDK，則表示您已在系統上擁有範例。

您也可以在 src/samples 目錄中檢視 適用於 Java 的 AWS SDK GitHub 儲存庫上的最新範例。 https://github.com/aws/aws-sdk-java/tree/master/src/samples

## 使用命令列建置和執行範例

這些範例包含 Ant 建置指令碼，讓您可以輕鬆地從命令列建置和執行指令碼。每個範例也包含 HTML 格式的 README 檔案，其中包含每個範例的特定資訊。

> **Note**
>
> 如果您在 GitHub 上瀏覽範本程式碼，請在檢視範例的 README.html 檔案時，按一下原始程式碼顯示中的原始按鈕。在原始模式中，HTML 會在瀏覽器中如預期呈現。

## 先決條件

在執行任何 適用於 Java 的 AWS SDK 範例之前，您需要在環境或使用 設定 AWS 登入資料 AWS CLI，如設定 AWS 登入資料和開發區域所指定。這些範例會盡可能使用預設登入資料提供者鏈結。因此，透過以這種方式設定您的登入資料，您可以避免將 AWS 登入資料插入來源碼目錄中的檔案中（在其中可能不小心簽入和公開共用）的風險實務。

## 執行範例

1. 變更為包含範例程式碼的目錄。例如，如果您位於 AWS SDK 下載的根目錄中，並想要執行AwsConsoleApp範例，您可以輸入：

```
cd samples/AwsConsoleApp
```

2. 使用 Ant 建置並執行範例。預設建置目標會執行兩個動作，因此您只需要輸入：

```
ant
```

範例會將資訊列印至標準輸出，例如：

```
===========================================

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.
```

# 使用 Eclipse IDE 建置和執行範例

如果您使用 AWS Toolkit for Eclipse，也可以根據 在 Eclipse 中啟動新專案， 適用於 Java 的 AWS SDK 或將 SDK 新增至現有的 Java 專案。

## 先決條件

安裝 之後 AWS Toolkit for Eclipse，建議您使用安全登入資料設定 Toolkit。您可以隨時從 Eclipse 的視窗選單中選擇偏好設定，然後選擇 AWS 工具組區段，以執行此操作。

## 執行範例

1. 開啟 Eclipse。
2. 建立新的 AWS Java 專案。在 Eclipse 的檔案功能表上，選擇新增，然後按一下專案。新專案精靈隨即開啟。
3. 展開 AWS 類別，然後選擇  AWS Java 專案。
4. 選擇 Next (下一步)。專案設定頁面隨即顯示。
5. 在專案名稱方塊中輸入名稱。 適用於 Java 的 AWS SDK 範例群組會顯示 SDK 中可用的範例，如前所述。
6. 選取每個核取方塊，以選取要包含在專案中的範例。
7. 輸入您的 AWS 登入資料。如果您已 AWS Toolkit for Eclipse 使用登入資料設定 ，則會自動填入。
8. 選擇 Finish (完成)。專案已建立並新增至 Project Explorer。
9. 選擇您要執行的範例`.java`檔案。例如，針對 Amazon S3 範例，選擇 `S3Sample.java`。
10. 從執行功能表中選擇執行。
11. 在 Project Explorer 中的專案上按一下滑鼠右鍵，指向建置路徑，然後選擇新增程式庫。
12. 選擇  AWS Java 開發套件，選擇下一步，然後遵循其餘的螢幕指示。

# 的安全性 適用於 Java 的 AWS SDK

雲端安全是 Amazon Web Services (AWS) 最重視的一環。身為 AWS 客戶的您，將能從資料中心和網路架構的建置中獲益，以滿足組織最為敏感的安全要求。安全性是 AWS 與您之間的共同責任。共同責任模型 將此描述為雲端本身的安全和雲端內部的安全。

雲端的安全性 – AWS 負責保護執行 AWS 雲端中提供的所有服務的基礎設施，並提供您可以安全使用的服務。我們的安全責任是 的最高優先順序 AWS，我們的安全有效性由第三方稽核人員定期測試和驗證，作為AWS 合規計劃的一部分。

雲端的安全性 – 您的責任取決於您使用 AWS 的服務，以及其他因素，包括資料的敏感度、組織的需求，以及適用的法律和法規。

此 AWS 產品或服務會透過其支援的特定 Amazon Web Services (AWS) 服務，遵循共同責任模型。如需 AWS 服務安全資訊，請參閱AWS 服務安全文件頁面AWS ，以及合規計劃在 AWS 合規工作範圍內的服務。

主題

- 適用於 Java 的 AWS SDK 1.x 中的資料保護
- 適用於 Java 的 AWS SDK 支援 TLS
- 身分和存取權管理
- 此 AWS 產品或服務的合規驗證
- 此 AWS 產品或服務的彈性
- 此 AWS 產品或服務的基礎設施安全
- Amazon S3 加密用戶端遷移

## 適用於 Java 的 AWS SDK 1.x 中的資料保護

共同責任模型適用於此 AWS 產品或服務中的資料保護。如此模型所述， AWS 負責保護執行所有 AWS 雲端的 全球基礎設施。您必須負責維護在此基礎設施上託管之內容的控制權。此內容包括您所使用 AWS 服務的安全組態和管理任務。如需有關資料隱私權的詳細資訊，請參閱資料隱私權常見問答集。如需歐洲資料保護的相關資訊，請參閱 AWS 安全部落格上的AWS 共同責任模型和 GDPR 部落格文章。

基於資料保護目的，我們建議您保護 AWS 帳戶 登入資料，並使用 AWS Identity and Access Management () 設定個別使用者帳戶IAM。如此一來，每個使用者都只會獲得授予完成其任務所必須的許可。我們也建議您採用下列方式保護資料：

- 每個帳戶均要使用多重要素驗證 (MFA)。
- 使用 SSL/TLS 與 AWS 資源通訊。
- 使用 設定 API 和使用者活動記錄 AWS CloudTrail。
- 使用 AWS 加密解決方案，搭配 AWS 服務中的所有預設安全控制。
- 使用進階受管安全服務，例如 Amazon Macie，以協助探索和保護存放在其中的個人資料 Amazon S3。
- 如果您在 AWS 透過命令列界面或 API 存取 時需要 FIPS 140-2 驗證的密碼編譯模組，請使用 FIPS 端點。如需有關 FIPS 和 FIPS 端點的更多相關資訊，請參閱聯邦資訊處理標準 (FIPS) 140-2 概觀。

我們強烈建議您絕對不要將客戶帳戶號碼等敏感的識別資訊，放在自由格式的欄位中，例如名稱欄位。這包括當您使用主控台 AWS CLI、API 或 AWS SDKs 使用此 AWS 產品或服務或其他 AWS 服務時。您輸入此 AWS 產品或服務或其他 服務的任何資料都可能被選入診斷日誌中。當您提供外部伺服器的 URL 時，請勿在驗證您對該伺服器請求的 URL 中包含登入資料資訊。

# 適用於 Java 的 AWS SDK 支援 TLS

以下資訊僅適用於 Java SSL 實作 ( 中的預設 SSL 實作 適用於 Java 的 AWS SDK)。如果您使用不同的 SSL 實作，請參閱特定的 SSL 實作，以了解如何強制執行 TLS 版本。

## 如何檢查 TLS 版本

請參閱 Java 虛擬機器 (JVM) 供應商的文件，以判斷您的平台支援哪些 TLS 版本。對於某些 JVMs，下列程式碼會列印支援的 SSL 版本。

```
System.out.println(Arrays.toString(SSLContext.getDefault().getSupportedSSLParameters().getProto
```

要查看運作中的 SSL 交握和使用的 TLS 版本，您可以使用系統屬性 javax.net.debug。

```
java app.jar -Djavax.net.debug=ssl
```

> **Note**
>
> TLS 1.3 與適用於 Java 的 SDK 版本 1.9.5 至 1.10.31 不相容。如需詳細資訊，請參閱下列部落格文章。
> https://aws.amazon.com/blogs/developer/tls-1-3-incompatibility-with-aws-sdk-for-java-versions-1-9-5-to-1-10-31/

### 強制執行最低 TLS 版本

軟體開發套件一律偏好平台和服務支援的最新 TLS 版本。如果您想要強制執行特定的最低 TLS 版本，請參閱 JVM 的文件。對於 OpenJDK 型 JVMs，您可以使用系統屬性 `jdk.tls.client.protocols`。

```
java app.jar -Djdk.tls.client.protocols=PROTOCOLS
```

如需 PROTOCOLS 支援的值，請參閱 JVM 的文件。

## 身分和存取權管理

AWS Identity and Access Management (IAM) 是一種 AWS 服務 ，可協助管理員安全地控制對 AWS 資源的存取。IAM 管理員可控制誰可以進行驗證 （登入） 和授權 （具有許可） 來使用 AWS 資源。IAM 是 AWS 服務 您可以免費使用的 。

主題

- 目標對象
- 使用身分驗證
- 使用政策管理存取權
- AWS 服務 如何使用 IAM
- 對 AWS 身分和存取進行故障診斷

### 目標對象

使用方式 AWS Identity and Access Management (IAM) 會有所不同，取決於您在 中執行的工作 AWS。

服務使用者 – 如果您使用 AWS 服務 執行任務，管理員會為您提供所需的登入資料和許可。當您使用更多 AWS 功能來執行工作時，您可能需要額外的許可。了解存取許可的管理方式可協助您向管理員請求正確的許可。如果您無法存取 中的功能 AWS，請參閱 對 AWS 身分和存取進行故障診斷或 AWS 服務 您正在使用的 使用者指南。

服務管理員 – 如果您負責公司 AWS 的資源，您可能擁有 的完整存取權 AWS。您的任務是判斷服務使用者應存取哪些 AWS 功能和資源。接著，您必須將請求提交給您的 IAM 管理員，來變更您服務使用者的許可。檢閱此頁面上的資訊，了解 IAM 的基本概念。若要進一步了解貴公司如何使用 IAM AWS，請參閱您正在使用的 使用者指南 AWS 服務 。

IAM 管理員：如果您是 IAM 管理員，建議您掌握如何撰寫政策以管理 AWS存取權的詳細資訊。若要檢視您可以在 IAM 中使用的以 AWS 身分為基礎的政策範例，請參閱 AWS 服務 您正在使用的 使用者指南。

## 使用身分驗證

身分驗證是您 AWS 使用身分憑證登入 的方式。您必須以 AWS 帳戶根使用者、IAM 使用者或擔任 IAM 角色身分進行身分驗證。

您可以使用身分來源的登入資料，例如 AWS IAM Identity Center (IAM Identity Center)、單一登入身分驗證或 Google/Facebook 登入資料，以聯合身分的形式登入。如需有關登入的詳細資訊，請參閱《AWS 登入 使用者指南》中的如何登入您的 AWS 帳戶。

對於程式設計存取， AWS 提供 SDK 和 CLI 以密碼編譯方式簽署請求。如需詳細資訊，請參閱《IAM 使用者指南》中的 API 請求的AWS 第 4 版簽署程序。

### AWS 帳戶 根使用者

當您建立 時 AWS 帳戶，您會從一個名為 AWS 帳戶 theroot 使用者的登入身分開始，該身分可完整存取所有 AWS 服務 和 資源。強烈建議不要使用根使用者來執行日常任務。有關需要根使用者憑證的任務，請參閱《IAM 使用者指南》中的需要根使用者憑證的任務。

### 聯合身分

最佳實務是要求人類使用者使用聯合身分提供者，以 AWS 服務 使用臨時憑證存取 。

聯合身分是您企業目錄、Web 身分提供者的使用者，或是 AWS 服務 使用身分來源的登入資料 Directory Service 存取的使用者。聯合身分會擔任角色，而該角色會提供臨時憑證。

若需集中化管理存取權限，建議使用 AWS IAM Identity Center。如需詳細資訊，請參閱 AWS IAM Identity Center 使用者指南中的什麼是 IAM Identity Center？。

### IAM 使用者和群組

IAM 使用者https://docs.aws.amazon.com/IAM/latest/UserGuide/id_users.html是一種身分具備單人或應用程式的特定許可權。建議以臨時憑證取代具備長期憑證的 IAM 使用者。如需詳細資訊，請參閱《IAM 使用者指南》中的要求人類使用者使用聯合身分提供者來 AWS 使用臨時憑證存取 。

IAM 群組會指定 IAM 使用者集合，使管理大量使用者的許可權更加輕鬆。如需詳細資訊，請參閱《IAM 使用者指南》中的 IAM 使用者的使用案例。

### IAM 角色

IAM 角色https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles.html的身分具有特定許可權，其可以提供臨時憑證。您可以透過從使用者切換到 IAM 角色 （主控台） 或呼叫 AWS CLI 或 AWS API 操作來擔任角色。如需詳細資訊，請參閱《IAM 使用者指南》中的擔任角色的方法。

IAM 角色適用於聯合身分使用者存取、臨時 IAM 使用者許可、跨帳戶存取權與跨服務存取，以及在 Amazon EC2 執行的應用程式。如需詳細資訊，請參閱《IAM 使用者指南》中的 IAM 中的快帳戶資源存取。

## 使用政策管理存取權

您可以透過建立政策並將其連接到身分或資源 AWS 來控制 AWS 中的存取。政策定義與身分或資源相關聯的許可。當委託人提出請求時 AWS ， 會評估這些政策。大多數政策會以 JSON 文件 AWS 形式存放在 中。如需進一步了解 JSON 政策文件，請參閱《IAM 使用者指南》中的 JSON 政策概觀。

管理員會使用政策，透過定義哪些主體可在哪些條件下對哪些資源執行動作，以指定可存取的範圍。

預設情況下，使用者和角色沒有許可。IAM 管理員會建立 IAM 政策並將其新增至角色，供使用者後續擔任。IAM 政策定義動作的許可，無論採用何種方式執行。

### 身分型政策

身分型政策是附加至身分 (使用者、使用者群組或角色) 的 JSON 許可政策文件。這類政策控制身分可對哪些資源執行哪些動作，以及適用的條件。如需了解如何建立身分型政策，請參閱《IAM 使用者指南》中的透過客戶管理政策定義自訂 IAM 許可。

身分型政策可分為內嵌政策 (直接內嵌於單一身分) 與受管政策 (可附加至多個身分的獨立政策)。如需了解如何在受管政策及內嵌政策之間做選擇，請參閱《IAM 使用者指南》中的在受管政策與內嵌政策之間選擇。

### 資源型政策

資源型政策是附加到資源的 JSON 政策文件。範例包括 IAM 角色信任政策與 Amazon S3 儲存貯體政策。在支援資源型政策的服務中，服務管理員可以使用它們來控制對特定資源的存取權限。您必須在資源型政策中指定主體。

資源型政策是位於該服務中的內嵌政策。您無法在資源型政策中使用來自 IAM 的 AWS 受管政策。

### 存取控制清單 (ACL)

存取控制清單 (ACL) 可控制哪些主體 (帳戶成員、使用者或角色) 擁有存取某資源的許可。ACL 類似於資源型政策，但它們不使用 JSON 政策文件格式。

Amazon S3 AWS WAF和 Amazon VPC 是支援 ACLs的服務範例。如需進一步了解 ACL，請參閱《Amazon Simple Storage Service 開發人員指南》中的存取控制清單 (ACL) 概觀。

### 其他政策類型

AWS 支援其他政策類型，可設定更多常見政策類型授予的最大許可：

- 許可界限 — 設定身分型政策可授與 IAM 實體的最大許可。如需詳細資訊，請參閱《 IAM 使用者指南》中的 IAM 實體許可界限。
- 服務控制政策 (SCP) — 為 AWS Organizations中的組織或組織單位指定最大許可。如需詳細資訊，請參閱《AWS Organizations 使用者指南》中的服務控制政策。
- 資源控制政策 (RCP) — 設定您帳戶中資源可用許可的上限。如需詳細資訊，請參閱《AWS Organizations 使用者指南》中的資源控制政策 (RCP)。
- 工作階段政策 — 在以程式設計方式為角色或聯合身分使用者建立臨時工作階段時，以參數形式傳遞的進階政策。如需詳細資訊，請參《IAM 使用者指南》中的工作階段政策。

### 多種政策類型

當多種類型的政策套用到請求時，產生的許可會更複雜而無法理解。若要了解如何 AWS 在涉及多個政策類型時決定是否允許請求，請參閱《IAM 使用者指南》中的政策評估邏輯。

## AWS 服務 如何使用 IAM

若要深入了解 如何使用 AWS 服務 大多數 IAM 功能，請參閱《IAM 使用者指南》中的與 AWS IAM 搭配使用的 服務。

若要了解如何 AWS 服務 搭配 IAM 使用特定 ，請參閱相關服務使用者指南的安全章節。

# 對 AWS 身分和存取進行故障診斷

使用以下資訊來協助您診斷和修正使用 AWS 和 IAM 時可能遇到的常見問題。

主題


- 我無權在 中執行動作 AWS
- 我未獲得執行 iam:PassRole 的授權
- 我想要允許 以外的人員 AWS 帳戶 存取我的 AWS 資源


## 我無權在 中執行動作 AWS

如果您收到錯誤，告知您未獲授權執行動作，您的政策必須更新，允許您執行動作。

下列範例錯誤會在mateojackson IAM 使用者嘗試使用主控台檢視一個虛構 *my-example-widget* 資源的詳細資訊，但卻無虛構 awes:*GetWidget* 許可時發生。

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 awes:GetWidget on resource: my-example-widget
```

在此情況下，必須更新 mateojackson 使用者的政策，允許使用 awes:*GetWidget* 動作存取 *my-example-widget* 資源。

如果您需要協助，請聯絡您的 AWS 管理員。您的管理員提供您的簽署憑證。

## 我未獲得執行 iam:PassRole 的授權

如果您收到錯誤，告知您未獲授權執行 iam:PassRole 動作，您的政策必須更新，允許您將角色傳遞給 AWS。

有些 AWS 服務 可讓您將現有角色傳遞給該服務，而不是建立新的服務角色或服務連結角色。如需執行此作業，您必須擁有將角色傳遞至該服務的許可。

名為 marymajor 的 IAM 使用者嘗試使用主控台在 AWS中執行動作時，發生下列範例錯誤。但是，動作要求服務具備服務角色授予的許可。Mary 沒有將角色傳遞給服務的許可。

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

在這種情況下，Mary 的政策必須更新，允許她執行 `iam:PassRole` 動作。

如果您需要協助，請聯絡您的 AWS 管理員。您的管理員提供您的簽署憑證。

### 我想要允許 以外的人員 AWS 帳戶 存取我的 AWS 資源

您可以建立一個角色，讓其他帳戶中的使用者或您組織外部的人員存取您的資源。您可以指定要允許哪些信任物件取得該角色。針對支援基於資源的政策或存取控制清單 (ACL) 的服務，您可以使用那些政策來授予人員存取您的資源的許可。

如需進一步了解，請參閱以下內容：

- 若要了解 是否 AWS 支援這些功能，請參閱 AWS 服務 如何使用 IAM。
- 若要了解如何提供您擁有 AWS 帳戶 的資源存取權，請參閱《IAM 使用者指南》中的在您擁有 AWS 帳戶 的另一個 中為 IAM 使用者提供存取權。
- 若要了解如何將資源的存取權提供給第三方 AWS 帳戶，請參閱《IAM 使用者指南》中的將存取權提供給第三方 AWS 帳戶 擁有。
- 如需了解如何透過聯合身分提供存取權，請參閱《IAM 使用者指南》中的將存取權提供給在外部進行身分驗證的使用者 (聯合身分)。
- 如需了解使用角色和資源型政策進行跨帳戶存取之間的差異，請參閱《IAM 使用者指南》中的 IAM 中的跨帳戶資源存取。

## 此 AWS 產品或服務的合規驗證

我們的新 AWS 註冊體驗並非針對受管制的工作負載而設計。如果您正在使用我們新的 AWS 註冊體驗，但想要將 用於受管制 AWS 的工作負載，您可以註冊 AWS （進階） 或為您的 AWS 環境啟用進階功能。

若要了解 AWS 服務 是否在特定合規計劃範圍內，請參閱AWS 服務 合規計劃範圍內然後選擇您感興趣的合規計劃。如需一般資訊，請參閱AWS 合規計劃。

您可以使用 下載第三方稽核報告 AWS Artifact。如需詳細資訊，請參閱下載報告 in AWS Artifact

您使用 時的合規責任 AWS 服務 取決於資料的敏感度、您公司的合規目標，以及適用的法律和法規。如需使用 時合規責任的詳細資訊 AWS 服務，請參閱 AWS 安全文件。

此 AWS 產品或服務會透過其支援的特定 Amazon Web Services (AWS) 服務，遵循共同責任模型。如需 AWS 服務安全資訊，請參閱AWS 服務安全文件頁面AWS ，以及合規計劃在 AWS 合規工作範圍內的服務。

# 此 AWS 產品或服務的彈性

AWS 全球基礎設施是以 AWS 區域 和可用區域為基礎建置。

AWS 區域 提供多個實體隔離和隔離的可用區域，這些可用區域與低延遲、高輸送量和高備援聯網連接。

透過可用區域，您可以設計與操作的應用程式和資料庫，在可用區域之間自動容錯移轉而不會發生中斷。可用區域的可用性、容錯能力和擴展能力，均較單一或多個資料中心的傳統基礎設施還高。

如需 AWS 區域和可用區域的詳細資訊，請參閱 AWS 全球基礎設施。

此 AWS 產品或服務會透過其支援的特定 Amazon Web Services (AWS) 服務，遵循共同責任模型。如需 AWS 服務安全資訊，請參閱AWS 服務安全文件頁面AWS ，以及合規計劃在 AWS 合規工作範圍內的服務。

# 此 AWS 產品或服務的基礎設施安全

此 AWS 產品或服務使用 受管服務，因此受到 全球網路安全的 AWS 保護。如需 AWS 安全服務以及如何 AWS 保護基礎設施的相關資訊，請參閱AWS 雲端安全。若要使用基礎設施安全的最佳實務設計您的 AWS 環境，請參閱安全支柱 AWS Well-Architected Framework 中的基礎設施保護。

您可以使用 AWS 已發佈的 API 呼叫，透過網路存取此 AWS 產品或服務。使用者端必須支援下列專案：

- Transport Layer Security (TLS)。我們需要 TLS 1.2 並建議使用 TLS 1.3。
- 具備完美轉送私密(PFS)的密碼套件，例如 DHE (Ephemeral Diffie-Hellman)或 ECDHE (Elliptic Curve Ephemeral Diffie-Hellman)。現代系統(如 Java 7 和更新版本)大多會支援這些模式。

此外，請求必須使用存取金鑰 ID 和與 IAM 主體相關聯的私密存取金鑰來簽署。或者，您可以透過 AWS Security Token Service (AWS STS) 來產生暫時安全憑證來簽署請求。

此 AWS 產品或服務會透過其支援的特定 Amazon Web Services (AWS) 服務，遵循共同責任模型。如需 AWS 服務安全資訊，請參閱AWS 服務安全文件頁面，以及AWS 合規計劃在 AWS 合規工作範圍內的服務。

# Amazon S3 加密用戶端遷移

本主題說明如何將應用程式從第 1 版 (V1) 的 Amazon Simple Storage Service (Amazon S3) 加密用戶端遷移到第 2 版 (V2)，並確保整個遷移過程中的應用程式可用性。

## 先決條件

Amazon S3 用戶端加密需要下列項目：

- 您的應用程式環境中已安裝 Java 8 或更新版本。 適用於 Java 的 AWS SDK 適用於 Oracle Java SE 開發套件和 Open Java 開發套件 (OpenJDK) 的發行版本，例如 Amazon Corretto、Red Hat OpenJDK 和 AdoptOpenJDK。
- Bouncy Castle Crypto 套件。您可以在應用程式環境的 classpath 上放置 Bouncy Castle .jar 檔案，或將對 artifactId `bcprov-ext-jdk15on` ( 具有 的 groupId`org.bouncycastle`) 的相依性新增至 Maven `pom.xml` 檔案。

## 遷移概觀

此遷移分為兩個階段：

1. 更新現有用戶端以讀取新格式。更新您的應用程式以使用 1.11.837 版或更新版本， 適用於 Java 的 AWS SDK 並重新部署應用程式。這可讓應用程式中的 Amazon S3 用戶端加密服務用戶端解密 V2 服務用戶端建立的物件。如果您的應用程式使用多個 AWS SDKs，您必須分別更新每個 SDK。
2. 將加密和解密用戶端遷移至 V2。一旦所有 V1 加密用戶端都可以讀取 V2 加密格式，請在應用程式程式碼中更新 Amazon S3 用戶端加密和解密用戶端，以使用其 V2 對等項目。

## 更新現有用戶端以讀取新格式

V2 加密用戶端使用舊版 適用於 Java 的 AWS SDK 不支援的加密演算法。

遷移的第一步是更新您的 V1 加密用戶端，以使用 1.11.837 版或更新版本。 適用於 Java 的 AWS SDK ( 我們建議您更新至最新版本，您可以在 Java API 參考 1.x 版中找到此版本。) 若要這樣做，請在專案組態中更新相依性。更新專案組態後，請重建專案並重新部署。

完成這些步驟後，您應用程式的 V1 加密用戶端將能夠讀取 V2 加密用戶端寫入的物件。

### 更新專案組態中的相依性

修改您的專案組態檔案（例如 pom.xml 或 build.gradle），以使用 1.11.837 版或更新版本 適用於 Java 的 AWS SDK。然後，重建您的專案並重新部署。

在部署新的應用程式程式碼之前完成此步驟，有助於確保加密和解密操作在遷移程序期間在整個機群中保持一致。

使用 Maven 的範例

pom.xml 檔案的程式碼片段：

```
<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>
```

使用 Gradle 的範例

build.gradle 檔案的程式碼片段：

```
dependencies {
  implementation platform('com.amazonaws:aws-java-sdk-bom:1.11.837')
  implementation 'com.amazonaws:aws-java-sdk-s3'
}
```

## 將加密和解密用戶端遷移至 V2

使用最新的 SDK 版本更新專案後，您可以修改應用程式程式碼以使用 V2 用戶端。若要這樣做，請先更新您的程式碼以使用新的服務用戶端建置器。然後，使用已重新命名的建置器方法提供加密資料，並視需要進一步設定您的服務用戶端。

這些程式碼片段示範如何搭配 使用用戶端加密 適用於 Java 的 AWS SDK，並提供 V1 和 V2 加密用戶端之間的比較。

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();
```

上述範例會將 cryptoMode設定為 AuthenticatedEncryption。這是允許 V2 加密用戶端讀取 V1 加密用戶端所寫入物件的設定。如果您的用戶端不需要讀取 V1 用戶端所寫入物件的功能，建議您StrictAuthenticatedEncryption改用 的預設設定。

## 建構 V2 加密用戶端

您可以透過呼叫 AmazonS3EncryptionClientV2.encryptionBuilder() 來建構 V2 加密用戶端。AmazonS3EncryptionClientV2.encryptionBuilder().

您可以使用 V1加密用戶端取代所有現有的 V21 加密用戶端。只要您允許 V2 加密用戶端使用 `AuthenticatedEncryption`cryptoMode，V2 加密用戶端一律可以讀取 V1 加密用戶端寫入的任何物件。

建立新的 V2 加密用戶端與建立 V1 加密用戶端的方式非常相似。不過，有幾個差異：

- 您將使用 CryptoConfigurationV2 物件來設定用戶端，而非CryptoConfiguration物件。此為必要參數。
- V2 加密用戶端cryptoMode的預設設定為 StrictAuthenticatedEncryption。對於 V1 加密用戶端，它是 EncryptionOnly。

- 加密用戶端建置器上withEncryptionMaterials() 的方法已重新命名為 withEncryptionMaterialsProvider()。這只是一種外觀變更，可更準確地反映引數類型。設定服務用戶端時，您必須使用新的 方法。

> **Note**
>
> 使用 AES-GCM 解密時，請先將整個物件讀到結尾，再開始使用解密的資料。這是為了驗證物件在加密後尚未修改。

## 使用加密資料提供者

您可以繼續使用已搭配 V1 加密用戶端使用的相同加密資料提供者和加密資料物件。這些類別負責提供加密用戶端用來保護資料的金鑰。它們可以與 V2 和 V1 加密用戶端互換使用。

## 設定 V2 加密用戶端

V2 加密用戶端已設定 `CryptoConfigurationV2` 物件。您可以透過呼叫其預設建構函數，然後視需要從預設值修改其屬性來建構此物件。

的預設值`CryptoConfigurationV2`為：

- `cryptoMode` = `CryptoMode.StrictAuthenticatedEncryption`
- `storageMode` = `CryptoStorageMode.ObjectMetadata`
- `secureRandom` = 的執行個體 `SecureRandom`
- `rangeGetMode` = `CryptoRangeGetMode.DISABLED`
- `unsafeUndecryptableObjectPassthrough` = `false`

請注意，`cryptoMode`V2 加密用戶端不支援 EncryptionOnly。V2 加密用戶端一律會使用已驗證的加密來加密內容，並使用 V2 `KeyWrap` 物件來保護內容加密金鑰 (CEKs)。

下列範例示範如何在 V1 中指定加密組態，以及如何執行個體化要傳遞給 V2 CryptoConfigurationV22 物件。 V2

V1

```
CryptoConfiguration cryptoConfiguration = new CryptoConfiguration()
```

```
        .withCryptoMode(CryptoMode.StrictAuthenticatedEncryption);
```

V2

```
CryptoConfigurationV2 cryptoConfiguration = new CryptoConfigurationV2()
        .withCryptoMode(CryptoMode.StrictAuthenticatedEncryption);
```

## 其他範例

下列範例示範如何解決與從 V1 遷移至 V2 相關的特定使用案例。

### 設定服務用戶端以讀取 V1 加密用戶端建立的物件

若要讀取先前使用 V1 加密用戶端寫入的物件，請將 `cryptoMode`設定為 `AuthenticatedEncryption`。下列程式碼片段示範如何使用此設定建構組態物件。

```
CryptoConfigurationV2 cryptoConfiguration = new CryptoConfigurationV2()
        .withCryptoMode(CryptoMode.AuthenticatedEncryption);
```

### 設定服務用戶端以取得物件的位元組範圍

若要能夠從加密的 S3 物件`get`範圍位元組，請啟用新的組態設定 `rangeGetMode`。預設會在 V2 加密用戶端上停用此設定。請注意，即使啟用 ，範圍`get`僅適用於使用 用戶端`cryptoMode`設定所支援演算法加密的物件。如需詳細資訊，請參閱 適用於 Java 的 AWS SDK API 參考中的 CryptoRangeGetMode。

如果您打算使用 Amazon S3 TransferManager 執行使用 V2 加密用戶端的加密 Amazon S3 物件分段下載，則必須先在 V2 加密用戶端上啟用 `rangeGetMode`設定。

下列程式碼片段示範如何設定 V2 用戶端以執行範圍 `get`。

```
// Allows range gets using AES/CTR, for V2 encrypted objects only
CryptoConfigurationV2 cryptoConfiguration = new CryptoConfigurationV2()
       .withRangeGetMode(CryptoRangeGetMode.ALL);

// Allows range gets using AES/CTR and AES/CBC, for V1 and V2 objects
CryptoConfigurationV2 cryptoConfiguration = new CryptoConfigurationV2()
       .withCryptoMode(CryptoMode.AuthenticatedEncryption)
       .withRangeGetMode(CryptoRangeGetMode.ALL);
```

# 的 OpenPGP 金鑰 適用於 Java 的 AWS SDK

所有公開可用的 Maven 成品 適用於 Java 的 AWS SDK 都會使用 OpenPGP 標準簽署。下節提供驗證成品簽章所需的公有金鑰。

## 目前金鑰

下表顯示適用於 Java 的 SDK 1x 和適用於 Java 的 SDK 2.x 目前版本的 OpenPGP 金鑰資訊。

| | |
|---|---|
| 金鑰 ID | 0xAC107B386692DADD |
| 類型 | RSA |
| 大小 | 4096/4096 |
| 已建立 | 2016-06-30 |
| 到期 | 2026-09-27 |
| 使用者 ID | AWS SDKs和工具 <aws-dr-tools@amazon.com> |
| 金鑰指紋 | FEB9 209F 2F2F 3F46 6484 1E55 AC10 7B38 6692 DADD |

若要將適用於 Java 的 SDK 的下列 OpenPGP 公有金鑰複製到剪貼簿，請選取右上角的「複製」圖示。

```
-----BEGIN PGP PUBLIC KEY BLOCK-----
Comment: Hostname:
Version: Hockeypuck 2.2

xsFNBFd1gAUBEACqbmmFbxdJgz1lD7wrlskQA1LLuSAC4p8ny9u/D2zLR8Ynk3Yz
mzJuQ+Kfjne2t+xTDex6MPJlMYpOviSWsX2psgvdmeyUpW9ap0lrThNYkc+W5fRc
buFehfbi9LSATZGJi8RG0sCCr5FsYVz0gEk85M2+PeM24cXhQIOZtQUjswX/pdk/
KduGtZASqNAYLKROmRODzUuaokLPo24pfm9bnr1RnRtwt5ktPAA5bM9ZZaGKriej
kT2lPffbBjp8F5AZvmGLtNm2Cmg4FKBvI04SQjy2jjrQ3wBzi5Lc9HTxDuHK/rtV
u6PewUe2WPlnxlXenhMZU1UK4YoSB9E9StQ2VxQiySLHSdxR7Ma4WgYdVLn9bOie
nj3QxLuQ1ZUKF79ES6JaM4tOz1gGcQeU1+UklgjFLuKwmzWRdEIFfxMyvH6qgKnd
```

U+DioH5mcUwhwffAAsuIJyAdMIEUYh7IfzJJXQf+fF+XfOCl6byOJFWrIGQkAzMu
CEvaCfwtHC2Lpzo33/WRFeMAuzzd0QJ4uz4xFFvaSOSZHMLHWI9YV/+Pea3X99Ms
0Nlek/LolAJh67MynHeVBOHKrq+fluorWepQivctzN6Y1NOkx5naTPGGaKWK7G2q
TbcY5SMnkIWfLFSougj0Fvmjczq8iZRwYxWA+i+LQvsR9WEXEiQffIWRoQARAQAB
zSxBV1MgU0RLcyBhbmQgVG9vbHMgPGF3cy1kci10b29sc0BhbWF6b24uY29tPsLB
lAQTAQoAPgIbAwULCQgHAwUVCgkICwUWAgMBAAIeAQIXgBYhBP65IJ8vLz9GZIQe
VawQezhmktrdBQJo12ZrBQkTQxnmAAoJEKwQezhmktrdi18P/A3De83MBx8bdcWJ
Fot71Vk1TyBQFErgrtcytSU0czEHx3tGbzgQLbMlyzjirOT03usxEkOeqTVK+RU+
5uFXNZYQLwMJlHJ6S8tnfLe/ExM5WQ2KPwIUPfZs1GDDRQB2dIKSc+qYrP1O1vf4
O4iPgfLHMW2bFh3zjjxcaHCJyqc7Cau33eZFBAsRni1jOUo7MeyX0hlXfW8pd48Q
wZllQVZ/6KmDiFWA0CZ+2svJ5cL0tgPoh1OQjoz0nHpNfuDILMrZ+e7tx2VTlkGH
UGeNSydnrK8v9ztFn34KtU/k7NEWoVSyEi+5ICZL18FBwPqTwdVWXwXrqZCKiIpr
8ZdJWDz2sJfgDFNCC6rKgCQ6FrmaD9G76dYWkQ4AbZqABlUzU3q36W1K0r3iOAb5
G4tdOt4yqXHTe1x+ZUNaeW7gaCmtXAxLw0OfeJrcq/44b/SQP+qJ8sSOv76Yg2oF
BsF5DWOVUFghbTyokHAoVROyhBR4dUUisY39AqLSL8+Lp9Pr3wNuGl9GLrMD5701
piUb88B3Gwe1EiKV1gaKrvZ3mECDUiSMVO0Z5iG8E4QDpNmVbJbV1uT821ubvtOv
2Ko10Fa0uwCYGssdRGqEXNy6jz/Er8LAC3+nmGINDJQzrF+loYoSSkI2Nu7lhMuL
7iWwUPF7OhDXoVSAn4X3x6q2rGK0wsGUBBMBCgA+AhsDBQsJCAcDBRUKCQgLBRYC
AwEAAh4BAheAFiEE/rkgny8vP0ZkhB5VrBB7OGaS2t0FAmjXZTsFCRNDGLYACgkQ
rBB7OGaS2t0/Ow//YIv51vHtD+kwMmIvk3zpizDHY0zW2dOezAo+C/DsSyC7wDll
Dixw34EQ1yLXH5xLR8CH1zupl3JmmEp1ucdQggoefbidxDl8Fld7tJOD1y3GGnTD
0jAl2ZC+W65Oh+wS1mD1FlaKjMGGkvJf0dA7RtU2T8dv3vt8dsxg76FMFS3+fqlC
FNOAsNTn9zWR1SqBIfkMJK83aq6s/rcEV9VrAYgDgqex58fygB5EuTf842/IF7WZ
Q9gd6fupB0mMZP5YWd2uj/vsBTYakG+mgQwDxZuKPeEzAqnqqS7biSQOUO6Wozlq
Yy4fSczE9GkBAvg0pGmbko+zHvpnjvX/h1CUpC6odvFyOAhZp6zyhs0QWz9thfqV
lU8WlbgJ2atFDn5GUSxF/fe0Yzovlbbs6sbYXuvMG9RiEOuJ1mBbZR3aIdZ1U6Do
BHc/vjc5mWcV7JQSP7i4W/8W7X3UAuN9LdxB+IvF3Cwrgtlw2BWvA5Alco5Tnz8t
P/CIVmBjk+sLme8W4kfLK3IWEbwClOdNnErI/MHRm65A2Y5EMIhwjrOi07SU1Pxa
nPpg3OYJCdvjzdB8QE3/DBiMfOl4dISfKDVEWnfK8mZaYd/BeRm2gUAa9UrqSFCG
BlA7Lg+eLI3US0FvwWJ4j5bBJqgLu+y7crIkiUOPAQuLk3lO+5uYU/I3DuLCwZQE
EwEKAD4CGwMFCwkIBwMFFQoJCAsFFgIDAQACHgECF4AWIQT+uSCfLy8/RmSEHlWs
EHs4ZpLa3QUCZwAXCAUJEWvKgwAKCRCsEHs4ZpLa3ZdTEACMBLg2q9zk8ZH02nDz
Sg5zc8Wlqq8WdxU0Pj8qx4UOrrMca7wyiUvrgoxPW5lh1RVNUeMkDRfu9pSXcOVI
V9LvmYE/WnwKROubgGbsC4T7M/LqV0/AulXil4d7IXcO6l4toa8LTNWtD5bODgrN
gvay1AzCU8kq1Qw1cKZ2gAfvA3Ba7PWyLeUN4HTlGrXcw73G+0CofY1L8wqWxHCJ
29XqQzeTEc6MDEeIlNlVdUcy8Qr5uwkEsl34H9AxS5F1opJ4TqvXiDZsrSRRv57R
XYmRZDWeYT+9PZaMsHXza5qgej7BfATxhYfICsNaY6MK3x6b+nDSKkoZgO+i09zh
1YjpahhQe6G336v/3mRj0dKGCRQ6znQ9ghUaB5z9zfvgH5AOEkTe3l8MqM+j5A6P
VjSBBJAHKejxr7+wKJKIA6P+DqpsYAunzftwUzrLVqb+BZQ+DcTmVrE7OPcMYJD5
QglX/Le+WmWZHI154NXgpWWUOUgZUbUge4DKrT+zCJ9iecPLKTW7OcULyXO+rjb8
8BGrD5GPlHB3dOUXXTlMKCqg3qy1Bu2KnZTQiaEEdZgSIGQbrW0JTMmmXJkKjokd
JMA4vYeg5en51G9nRQjScPngx77IxvByNyFWTJdG1ENpJpsK9TtmENcpyUJtJZTJ
ZSOIRVPP5RzR5vInuXWq6VV0BMLBlAQTAQoAPgIbAwULCQgHAwUVCgkICwUWAgMB
AAIeAQIXgBYhBP65IJ8vLz9GZIQeVawQezhmktrdBQJlJEoiBQkPj/2dAAoJEKwQ

```
ezhmktrdx1YP/0vvym3jgX/pwnR7K1rafZMb1iKQBr0ISG8cdbaf4pqX5vuUZnyj
w9Cl/oONn7jJjnQxOIIzuBoxne2WN28ftM2w0nVXm85mAmz2fwQz/fdKDyonXcOh
pfD2iMqn7gESjhEgRE7wMDYMDuLdqHI7OKWGVfgrh7xEmKapLh45h7cnumo2VjL9
uDYY1aOBHz993T7oE41y43rhk+6kKbGFd2uuo7h5j1ZF8Lj6sYfcEzXOU1OhRlD0
nyBjDy9MYWu0YNouc70WgMceGx6hjvCAM/5fxP7SZFecZ7ePeB0GpvVA24hSNENE
0r3tUekuOf1I0FunMnMnbh7ZO9rPYqWvWDNIpU3S4CjFhY82L+IeKnmLy8N6ASRk
HsPiNCOHSK8C/0ynrd9xLhX8Jsk/TGiQYaleoHhWkNLlZsL86QHL8SKEqkqZCQf5
AEqghDP6NEGS7lnOenA7JjIrA9KLlT7fnNWZOwFi5X+o/CymE2ytEMS0Yf3nmY4U
n9x56Wgn6J2zqB5nqOXf6NxGdAIgOBm098YEnKCIFzk+yhoDlprVpHcnd2b5f6Oq
uh8KYOEbKgpMJ3zZuWSL5kwGF1nNoYiAkonMaz9H3pOQnOMVYCUeUTDRsiO/prrd
UhNlry4TAsBMpeXnFhdLVM3vFQZVpByadGOJNmnaN/Wavw2a00UGBFa4wsF9BBMB
CgAnAhsDBQsJCAcDBRUKCQgLBRYCAwEAAh4BAheABQJhMqGaBQkLnlUVAAoJEKwQ
ezhmktrd2sQP/3YHM+U+BbOy1nSEAfykZ71+uCM2hkHMLdxQYWB/rBWkmg/pbu+d
r4t45RsTASrNjRcZOntlPMQRIq973ymHfpmeS+noFwvTGH7zDv1BRBR9wPrd1XUz
iSuEUHGi/fqxUVXQ5mbonzfThX8tuXeuiQmeToqoB00FYlZm6xsNnEHcjVl66mC4
IPoJLWnZJs4rOCeoRf5XvDTgX6xt5/kLYRZf79qaWGFvaZpsc1CH+rQJUdVa/D4T
7pI7hX6zy0S91z4iuC5HZUiOTF+y5auEZHGTdTWNS1kvOvfcCTi0XK/GkGL82SZu
7X2VGnpCeUnFyViRGlk+KaDG1sVyDY+lcBPg6ilr45M6MQV0iHS5OF04QNXSKt5+
UnzJH7lldgNsR6ibRMyNV3k5v3fyUcSBvIYyLORTTBiVEjQDSbk1QNqbrQlX9CWz
+EJWn16BFTmMFvxBSWPm640GncHP5J3/0MbMw3Cm90x7k8UfNANIemcrJrSxIDwm
g9cVAg3a+D+wxjrVe8jGg0ejvECpm+0yswigj5x6Lqj09A4UgdjEauN+/pn0nhBo
Gv7DzMXtM/LoDtgp6wn93qZVN2TsuHnkEk4UyntB6eWJbBdXHWUr47exiWh0dvQN
tpwCWPT6I7ZTPtA5K/zx+q9m6797BLgAkTYc6gloQL3vs1Z1S3m/hZNawsF9BBMB
CgAnAhsDBQsJCAcDBRUKCQgLBRYCAwEAAh4BAheABQJgmrz2BQkLBnBxAAoJEKwQ
ezhmktrd36oP/2rB2EkwSOCKC4m0heWSfDWi6OBKoEbbDtFtc6/HwqBW8SPsiKlq
zV0e3qBY/LVju04+ktJEK+EGXLnC3iC36MegrQ8zt391kEx/Zv9LIuVOCX90QIAX
dL8MVUkkjRLCFFH8pTgRy1cJYWk1X4dYdXWYc29fCwNVartNdNBhsb2ht3VJeKDE
kUivBHmkjuISDPEnI1coY7Lj0ZtY5cHdRF2eZpB0RkTBpsIt18rCYyHkERZrhmvb
j3rOyPyvOa+1/dQS8/hv5pEmbKx8cy8RdJkmbUHYatPBsjHkJSWr7O7G9VFW4GoN
9CRAI4KkbDSEDjCL5dv2pq0Sew1MkLuWJGULAMgiIUlWcOs5SZZGFSksNQrtSFV9
Z/wGocecMGkGQNXQ06JV/Fry/TvyphBlmylEqL+NLqEcEjnlz90IVu+ZA+M09J96
UlHO7V5GvBgM+QK/q/dJeMHPWrNlo1gA6Nwl/HBdM0DqzdZ2jEPvsQSABvZrPMty
+BAqEar4wqY1AH4X5ccEjO7nJQoBQSDRSki1fkBsc1nx44N/m0kHdIa0Z/Y+Mw4v
WiZhREkOospG1I4lBa3CNTVAhSs9msGsYfkqvFJGHL7sZY8XSv82GBBvA0nUNrsJ
bLBwo2FaQG9eoatRAGkqp4b/OtNtBuGeiQoNwFGbfUZTAaStj5/zZj0swsF9BBMB
CgAnAhsDBQsJCAcDBRUKCQgLBRYCAwEAAh4BAheABQJe+9bwBQkJZ4plAAoJEKwQ
ezhmktrd+ScP/RoaUKriVVAgLHOGs+/mnfKtnfTlClzi5dsdI9/6H0vLpmSWK/Cl
2cT6gary45VMgAeVK+H1lQXafYj+FY++I5kYoe2GrSvIXhpjaFAJyNf/dKleTsqR
Tm371i8b3FDYs5kvy2CnTbmHB8MsOGxck8/YHd1x+g8WpO2IgF89yYCSF3CAdxC3
6bHbs6Z3C3lcM/3SoWF+Yie2P8XeBMPCGp/BcjQzUcHF6G06TwDDYhixucUi6vEY
EH5JtOwVVQ7bubT8OFeOoJwVxlzYz4UoqxjKDWymarTzu03AUIT0PXPece94bJAK
mSh68ItQe3H8tSPMubERWz2tEV3lVkChDGXcC7BYQmxHseolxz/qzCtJ0iX9BvZR
dniZNeNJ/Cu8M2pDp47zdNFXzf/Q/sQ9pQlws22G2g119rWDneBku9n1vTP80/er
SB+VLTBjDiArlCY5y9+BG8wbscExJySoQxkB9j/nlMzPY5rgk0SyxsNj9GbqH+hr
```

```
EjS3/uacNwSLxGcOT2E9Teot5pfTEO6fQVq+35QhfAlP8c8jze01W/+u+wXu1Ui9
azRSzYtCHanGyyet6U1mlBpAkqkZzH6t3CA5czc9i6FbzjvFVZnbRUZIRzfISYew
lF5WqgTn2iYVdxagPRvLF5kjd696brGW9d5HwirCVGaK04VsXWlAblB9wsF9BBMB
CgAnBQJXdYAFAhsDBQkHhh+ABQsJCAcDBRUKCQgLBRYCAwEAAh4BAheAAAoJEKwQ
ezhmktrdWigP/3QWl7aO81BUWyby4HEhN4SdAoWGY/FLqO4mCtuplcnMgRUCSiL9
l2BSCTMCtUcdSWtYwOgSChN2mMsdi1U2FNR5HvNunYR/pFdqjfQurf1ZmKVeG5/4
uuKaOxMw9e8pK5uYAfs+O7gr8gu/f6/Drp7NZk3/yVKpf4WCY9oX9TA1q9O/11nN
cwS45U/d7YP+N1YM9cBXa1DnDcdfm0BlykzouAF0qd1Lwi/tmLENvybD3+2c2WsE
rlFZGSa5ZafO0tTIWXh5k6wh5FdRRycrnSyRK3B9N9+yaXfMQ0XpOypa8dqQEnCi
IsngDCJPxtTrhMWKhBFRUMzK/WZTDboTQSQDK+YVRrE4K8MtoZSKwZLV2r9O3TpX
kpbKsPVYmexerfdMeZfjZMF1bC7BmEs7jciH6JjbqAoAPnHzN0481aeNarINSViX
PQWr2mp9qShei2/RavLtx2ZNrvmGW72ZKpF8E3WWUDpBJqFVeGNRvOm3aZj8o/Hl
ewtNjcT4ouJfqlfKiULv+g7ANEMDLQTFDTg5twRdvmZlB7oTBsavf+LwxPIXhH32
IR7TX7VeicMMxmZnmZK2ANT/QBi3laf+ojVHvB+f6D74eLNq0Zqjfi/3UFNYsYjg
E+YgCqEUBpHbl61nOHwGOSsQwfap2uKKlzukD/KxH5SPBC3DYGBI+KCbzsFNBFd1
gAUBEAC8zNArPWb3dPMThL2xAY+fS60vXdB1SkOtYJpDWpFgvo0d+VQ+hV6XulGA
HAS6xG1WHysPT9KejIRSgLG+e9CaM5yhsxNa1WFGUM4Q9ESo3t+a75Go7xHIxgFj
C046/O6Vh3g9N/PREeuG8zkZ3H2v5fmD+ejyPgk4W9sFL00zjRiZD0FKVYR/j9ue
nEC/2NBcLuFy3q6cDfmCoDEOO62kXMnaGz3knzEK/X1SkcjsxRDq7zaQlQ1Kou+3
dICwy4x5SJQ8jl+eeeEvF2C2/dXmDohb57tqUwioohMUQkmCtvZgEHjypUwgp0MT
o25gWxkvJlSJKUOb6b1786WNySIzF2gxqlkkEmBl4RAssQkeXjrSmGwsMDyHNqyJ
eYFusl8sPaSpo+V2n0z+2B070Uq+wmf1S5A5FpegH0PZzzoNZo8I6QxaZje9YSZU
ijGmZIdEBleRVt3Svhi8MYlnasd4bW2RK1sr7plkBf8QRe6biiQRF3KDOSn5CbmX
pAcHJ1ZHzRRdkXZDNQC6vCJxsy13O0TrhJtAV1Yq347uyUbVi291ISVgroUVtprs
mHoEk5GoOTHbg9SCSt+xi/FiJQC+ubWmIGXoFKMR3UmhDnnzobKcbnbs/Hd981Fd
VghYYvq//gTAkJk0WxfGqO30wtXRndPOA0T+qhP3TE+LtGRJ+wARAQABwsF8BBgB
CgAmAhsMFiEE/rkgny8vP0ZkhB5VrBB7OGaS2t0FAmjXZm4FCRNDGegACgkQrBB7
OGaS2t3y5g/7BFXp/fdanzuQPToJTPen7AVwhLloKaiYhG3GjdXfMPLvu6UtaaGm
qynLolUNNooobptFqc1G9BKoAghQrta7CsDHtsQF2xyc3Mfu0gmpL/7X5a7sFIeJ
j08UjfweHx4DSG4LEZgNaAoWFjZltp4+8cqijkAHXt+r+1ayQG4VVHOWyXXqmSH4
9HqtbPcPyRzxdoVLeshZC9jmhHhhKqw/LwGyipWSOUKQDjWarBwdyhNmWCaLvxH1
ndMp4tq8DPGC3G4T9tYAbANrn7nKfZgHebMSzMw9kSp0L6QvwwTDjJyIWz85WyeH
WHeBysDaBOit3XDlehUew27y7N6a9hQSYjnXuwvre5mjDIOqJon/31R6ui2Z1y9P
a+bC11hbLXXh9tLCXRuoOt6thh9Cq5X1a76PPpEv30o3bpsb6l2hbrut1OKezwvK
l7txito/jfMiWfsZHA9O4SoM+8GnmVingHtZ805n1T4RddJvT/vaqplfI6zf7jmf
a69lALP420riFOQcwntNUM5tVmFUZsnFp2YRd4Ls7MiXVjtABahlSbb94l5WSVc0
jrOLDf94edvzk4R8i2Ob8CfVZNqEsTR6bHz8dT7Q+xQzEdjUujyyZY1UUl157Qeb
OsHjhCtuZYCI04X9hZ37nKnZXSxRlRDCnt5BEiyFu2WD1RscUe6PcVDCwXwEGAEK
ACYCGwwWIQT+uSCfLy8/RmSEHlWsEHs4ZpLa3QUCaNdlPQUJE0MYuAAKCRCsEHs4
ZpLa3XCpD/42DrcveE+q2ulrAIYPDlUlHiwIMejqBDRm6zmr1KSAeb4E6/MFcP4s
rXSSscMlrqG6NVynjNCXjD2YzWii68EwoXLJkgoD3r2ifzkV62EX2MIEeNZAVwuy
KNxorzmy6bhuWltRYNK/hITs2AG5orOk9ADEJ8PixKymrWlhesPaWX6Yhp9/tWaC
RHOSRiLbRVaJ+7sqT88urLmkV9Hqx949Zxv4+cgBVUGL6WXKsfWhHjbDMNJnozWB
SZaIJznLApOM8z+1DNrqUYyfR8SkF4IOVmg6HDzoyuseJJ8JvMAlkvT6F9VBq/iE
```

```
yeDYdEEQxwHwozKrEx5Ybxl5mntbqwCXy6kHSx2+/3RZWpZQ8K29YP9QEk0KeGF8
9Vap3jjNrX4u3cuRNQpeblQc4uFn3Nzaj+cVV4YzcRw94NifecXpujSvk8XU2ytJ
/JgMBxPIBKglN4eEMet9b4FRB5XeBdPAm19/LXyb4IlIipGNXlgNz/HCuBzidzHT
QQdqfA9rZVx1hwFr7AJCVqWaXVsx1oEAhKqpTtsLMyj594DvnRuwKw5Vse+1eydW
MIHYdbxmJccsTGIt/hsOpc8zfm+QYk5752jshhOKEBy+Ey3QZI1WbO547NOb2Hwr
Pgt7fw2NCKMPElSu98zmneFPhqNHf7L5urBe5gADj81E8lm6t/oVxcLBfAQYAQoA
JgIbDBYhBP65IJ8vLz9GZIQeVawQezhmktrdBQJnABcLBQkRa8qFAAoJEKwQezhm
ktrde3MP/13CLWp99XvRROrzD/bWOfWjAenT2PE/tYdOY9YcTQFbnIUhaVUDWAo3
pibR3D4u9LlY4olpGfJ7BTIHFa9myfpaVvmrNjueYI4omli24JQ/CKqNdY8Qzxz+
/QyiNK7Aw5cEBWIu84WGB1SsefWWT3rZe9YBb77gNcWHZ15pXTXrcgUxGY4808MC
I9YFWq8EA0iHawtFnmB3UFfClWt37Hy3PKvr1is3uG60+ULI8RQz3/+ZwSG8U+xt
b+I7H9+gITc1eFCb+tIwp5xWflyxcFXYk6UzOL7y3Fg2tIEuSNtIHUC9NDVobf6c
I0KAzZcMvKiPQiuBnVOjgDLmCZM5H6axj9x+gi4oVh6ea3HLqMzyjm5JkeCGgKWv
H0gD3yGEZDvcbavkQOle5T+4JefndKzCPrluX0iyx+oQiiOL8WieSSkSB6BsZcUN
SeuGJwM79Y7Oqld/YVrQNBZj5Vz+m3nZ+0EWDDMI0hRgMpSEIc+dnTC0u103Z+Rc
c2IJq8INmU653sUcfCZE12ParW4rF7ib6kViYrABT8f4e2TP0aOyP5kp51ied9qL
azaBA6tt/C9X1V2EJZK4srXtmcZO2Im45RAiVXyfpBAmmiF3eZWCbKe7qBC4rDRh
LZG4RQW/S86Da0BID7gQz9IFSkaG504MsDhvnA7iAqaHUHUepCsiwsF8BBgBCgAm
AhsMFiEE/rkgny8vP0ZkhB5VrBB7OGaS2t0FAmUkSiQFCQ+P/Z8ACgkQrBB7OGaS
2t3AwA/9GkXKUgvjKGCxwE4SdDt7c2jw6to2TTP9iFJ3Xbk3+5BURT3gkZCuu9D7
gt+97aVo/B4EM7Xz8DQKyY7Ic9VAwDRra/Hwi1V0hw1zyIWQ/gAnX3baU6qLRWHR
vVR5meV8r35C+rg9DaWFYmvS7PIv9LfxESwBPUjbmx8k4/5EJpHUwf12bzkTnot5
7q5lHxKQa6IvqQak+Hp9ZM2KPdsgKO2HWJJIIvYcI5byW9zBKV0O7YR8gtRAJKp9
IbtsXx0WT6cqHOFVc5SSzdcaMt0gLFl7BTnJyvKK2l9GABGBmzYDjeCyF2J+Ippf
oqxqfTe6EoOsuEMc2PbLTs9SsWjyCC2VGlX8+uUH9SoKwL0VQ6LFsP6fhkVKqi/a
rB6UuPR/iZnrKIuxMNQ4U+t2Q6UdMlmXsAXTNdkwzoK9oJRokIrH0ZV1KtH4sjjA
tCic+tOddq+GQLiKe2WpJfxlA0uESCB0TxjAwQmfn1H+dUhPeLlbNimHlH0/hXPd
ifuNGozzADIRseQDyzjl8xGL1qRZLD3cfmda6RyZ+S3dQRuaRrcFCDccpY/pO+F8
jbx64zyqqNs+KV+SkQGOcKFhWTZGCfQ/zMDtDmQKjb3eTAkv1zdEOMw9zEjjmS0q
8FNl+2wO3VnvXwvBbtDdVCIaIq+jVcsy5XtnnV+bJ19Q9yue/XvCwWUEGAEKAA8C
GwwFAmEyoZoFCQueVRUACgkQrBB7OGaS2t1uHBAAhOYVvrtchRmzCvdNER1DtkIs
bgQPJ9OxbyfvmvoD06qxH7PrycLZKbt7yYpAUU/CMc86GwaEe0I5Nm1CTs6NvDIv
g3e7EPIS859tyQflbM56NlwbsopCuoCJYknuroIf/M6dW6vJKNXLMmnL/AtalUBw
X+5pblmGUUJep49oTOxQEnvnuqyvaGjXgFXix5PVFJD2ed5NnQeFpvfCpc/ioNOj
z7ORO82j1ht5nWqPraXX5AYhQFM/kwR1cK4LV7gVDd/q+dfGYHzpxQ/HtyX/Lasi
N6I52QqA95SM1ZZLPFLaNh6EvnB7uC9pLCYS8nvilX7/cez5PFff1e1gXCOT0jv3
mJ2exLmXV0BbfKgjccFCxhrdRLtukfiDfJkySy1zdscnpfng8wJ3xKRv43cUTz7M
Z24OYNMqK26aJZVXEQUYjCwsBylY/F5wjYAwgwZ8yF5RFix28P/K8JsIHb3QrAJK
sNWQAb03ZWis3N3spR5M9Mw3VuDZ3WUXq7mxB5M3kpVoZ3vETU5cwTbADYNPf4Sw
BDK2uIVtxabezxSBtz0FcyYoF+OW8q7r4WvoyC9/+3GfnozZLJcEIVDk4W2pMW4A
UhG/6drKTm3HkSDWIDu7d1sHWMffLEYfUHtN5DKkDkGoPfHvZvu9teR5yLfUrPTf
ktihPn/JMrmwa9pwi8LCwWUEGAEKAA8CGwwFAmCavPcFCQsGcHIACgkQrBB7OGaS
2t0uaA//UWRaRiHEAKeRqBG/T2ak+XZJNu7QHfNgoUEAub9Zru8oPPXx2AJLcHEN
KWmeFlLxADdWOZs4Bm9oOew3VQnR/dBqjnXfob9Rc+eYUjA3rXazM/QrqcU8Syi3
```

```
MjNGUmjdL5aQF+IppAMgOBLG1TEnM7C5/PvrGJuYpGEnkKEwMK/GYhqg2V60pHEV
Pvs66mefJpCzbZSy56qtknSt6yBNWc14XgDX6VTn2kW4CV/3vVJUuvjvYs9SPyY8
mKEXa6QvUd3PcXv6RiWk4lGYuT1+jh2VkcFQ+JnUwv9TbKFB9b5jq1bvW9+LMDEl
YXux7pBP5RPk+OLpyiExIRFWhi3x7aMWOzQ+I9yuNTeYkTHiEAQRUhs/1Fh4oLgI
v9QZgC0mRSN3zm8plQdivs1ZlAosAqqkA9BQwqsgosQe7P92irYIJqay0si9wGCD
wSMsmeXdIF6wW3/UMJZl66aarPeiZApGX0QdTZwjMh/QK/8gTKyeZulKmNkNfwWq
O17OirWqLKssVHtg3VUM8EIdh+oNqDDXSeWtYUmpPpWp+yWZ0x1MFFZhUQHQZTGu
TIj4A92LQzbrfj/jXRvWm2SrJMivUoiDUn+qxKIpVwFlI5gVb+uyTFhw89PCkphr
JwRi052RLoU9yd6Ek46UH4XfZZWrZuzY+zzB7oqGONphLgi/h3DCwWUEGAEKAA8C
GwwFAl771b8FCQlniTUACgkQrBB7OGaS2t2/MxAAjoEGPdzavhsOlXdPCRd1D5QJ
r8T/NSEV2z1cp8ZvdrkjNF09TBP4qsBnKJiuvY1Iw7OGX9W2okvXxgJizE45v9MH
WEMz4hmIjmAfRwcqENgpOc1IY/T0/+kkCW8dB6d30J1kT0n2PCRzN9L5vPqZXGTG
mLvd9MOjH1256w4uxLb+e1HMDTCqEN1ppq9G+EAR/29q8JZWs1marbZZWxSWcg/E
1YYbNafzklgjq4CLh/j8AEWSvLr39zRy9uvQ/yqAKZ4K4aZfh/SPupGDvsD6ZK54
EPHxErQ7aiXTbUHtvwhxWLOP6WmxFA3Shr6L6YUb6jq+0PVliFC517g3mxFHJtwt
yXGNIKhmzmr0l9OlsHafulJ/9QPfK3Ce32SkPhW/11MYA8HzduMv5Arp7cBczXSP
EUTmNIVKv3gTjSQrzRhwhHmMuqyDZ/rXQQ1jl2sxIDjO4MUMvVjYKF+OCNm42gVs
8ca3/wN9ZNU6hyFWeKQDuCAqPPbT5GO/DKseFEwB+07wwyH1RXbyl0v4fneg605X
S7lqhNtw2p1hDL0HYHDiV+aPZ+LoOmX6+dmnqE6bQJaIlVb922KWmliO7F3DkqP7
0jFlhoE1gfiXWkxP4Gy8wOobNfEMgvz02djkGQy+oQqeNdIcZFZgzPTGKB/nVgpt
9CcRDWjPltFCd2e1FBbCwWUEGAEKAA8FAld1gAUCGwwFCQeGH4AACgkQrBB7OGaS
2t1PIQ//Qc5VYfBCxpaMysaPQ44wXPEZSjxIGZhhMGzb1UzzAEYOw+RgKN5nNTXq
L2KoOkOrGnKqZOKByMdXwIPH/rGwwEsbbIpopnibf5ic5B/+xCTIK+qLIwX2ZLuk
NhbL6Y+E+7DxMMh+KqBWHONKkgwVY+rFWOfoops839ABKvc9/Ry4/qqkcb40AzpD
l1iQJ5vK/DMuaDWxWeKXqJLIl3WMGPcPfheuBZL1u7LEEHYKMgzvpbF81WIn3MBo
8jvxf2/o+kMafSSDqgvOu6yu8GOhmScpCbRJn7jV/HrG+tM+zy48TN6/MkGWSR7q
TD34pqBjyatVfVl6dGD6xj/i/Emt5hZB6qXruCDH7AWMoNx+FkDubs4sc4PKysZU
Itya6KdQFo2UeYsNwZhdn6QwKhd85um4JUHJCY0mARvjsQgWXH/5MR40cow77bbE
vVq0XNd+QRVlyT42CEtnIUOFLeDVuZrum5Tuvvna6ImMDoi/z6QcNeL79XsY2m6I
QVRiHr1BDb/8JLkfnWiwL8GRv169Kf8unx0y5u1YBpcMYkyDD2+pnnk3TY0rR+8X
8goecaS8fbyu/Q48K85ZMD8wKW/bzLQ+tK9y8xed24u2QERftMhIw9b6f45Nrrf/
PhgV8RnuwUusSbdDe8kw3eYTmLdzD4kZc9K7SdO2CqT+hm//9JI=
=uGHC
-----END PGP PUBLIC KEY BLOCK-----
```

# 上一個索引鍵

> **Important**
>
> 新的金鑰會在先前的金鑰過期之前建立。因此，在任何給定時間，一個以上的金鑰可能有效。金鑰是從建立成品當天開始用來簽署成品，因此當金鑰的有效性重疊時，請使用最近發行的金鑰。

過期日期：2025-10-04

| | |
|---|---|
| 金鑰 ID | 0xAC107B386692DADD |
| 類型 | RSA |
| 大小 | 4096/4096 |
| 已建立 | 2016-06-30 |
| 過期日期 | 2025-10-04 |
| 使用者 ID | AWS SDKs和工具 <aws-dr-tools@amazon.com> |
| 金鑰指紋 | FEB9 209F 2F2F 3F46 6484 1E55 AC10 7B38 6692 DADD |

若要將適用於 Java 的 SDK 的下列 OpenPGP 公有金鑰複製到剪貼簿，請選取右上角的「複製」圖示。

```
-----BEGIN PGP PUBLIC KEY BLOCK-----
Comment: Hostname:
Version: Hockeypuck 2.2

xsFNBFd1gAUBEACqbmmFbxdJgz1lD7wrlskQA1LLuSAC4p8ny9u/D2zLR8Ynk3Yz
mzJuQ+Kfjne2t+xTDex6MPJlMYpOviSWsX2psgvdmeyUpW9ap0lrThNYkc+W5fRc
buFehfbi9LSATZGJi8RG0sCCr5FsYVz0gEk85M2+PeM24cXhQIOZtQUjswX/pdk/
KduGtZASqNAYLKROmRODzUuaokLPo24pfm9bnr1RnRtwt5ktPAA5bM9ZZaGKriej
kT2lPffbBjp8F5AZvmGLtNm2Cmg4FKBvI04SQjy2jjrQ3wBzi5Lc9HTxDuHK/rtV
```

```
u6PewUe2WPlnxlXenhMZU1UK4YoSB9E9StQ2VxQiySLHSdxR7Ma4WgYdVLn9bOie
nj3QxLuQ1ZUKF79ES6JaM4tOz1gGcQeU1+UklgjFLuKwmzWRdEIFfxMyvH6qgKnd
U+DioH5mcUwhwffAAsuIJyAdMIEUYh7IfzJJXQf+fF+XfOCl6byOJFWrIGQkAzMu
CEvaCfwtHC2Lpzo33/WRFeMAuzzd0QJ4uz4xFFvaSOSZHMLHWI9YV/+Pea3X99Ms
0Nlek/LolAJh67MynHeVBOHKrq+fluorWepQivctzN6Y1NOkx5naTPGGaKWK7G2q
TbcY5SMnkIWfLFSougj0Fvmjczq8iZRwYxWA+i+LQvsR9WEXEiQffIWRoQARAQAB
zSxBV1MgU0RLcyBhbmQgVG9vbHMgPGF3cy1kci10b29sc0BhbWF6b24uY29tPsLB
lAQTAQoAPgIbAwULCQgHAwUVCgkICwUWAgMBAAIeAQIXgBYhBP65IJ8vLz9GZIQe
VawQezhmktrdBQJnABcIBQkRa8qDAAoJEKwQezhmktrdl1MQAIwEuDar3OTxkfTa
cPNKDnNzxaWqrxZ3FTQ+PyrHhQ6usxxrvDKJS+uCjE9bmWHVFU1R4yQNF+72lJdw
5UhX0u+ZgT9afApE65uAZuwLhPsz8upXT8C6VeKXh3shdw7qXi2hrwtM1a0Pls4O
Cs2C9rLUDMJTySrVDDVwpnaAB+8DcFrs9bIt5Q3gdOUatdzDvcb7QKh9jUvzCpbE
cInb1epDN5MRzowMR4iU2VV1RzLxCvm7CQSyXfgf0DFLkXWiknhOq9eINmytJFG/
ntFdiZFkNZ5hP709loywdfNrmqB6PsF8BPGFh8gKw1pjowrfHpv6cNIqShmA76LT
3OHViOlqGFB7obffq//eZGPR0oYJFDrOdD2CFRoHnP3N++AfkA4SRN7eXwyoz6Pk
Do9WNIEEkAcp6PGvv7AokogDo/4OqmxgC6fN+3BTOstWpv4FlD4NxOZWsTs49wxg
kPlCCVf8t75aZZkcjXng1eClZZQ5SBlRtSB7gMqtP7MIn2J5w8spNbs5xQvJc76u
NvzwEasPkY+UcHd05RddOUwoKqDerLUG7YqdlNCJoQR1mBIgZButbQlMyaZcmQqO
iR0kwDi9h6Dl6fnUb2dFCNJw+eDHvsjG8HI3IVZMl0bUQ2kmmwr1O2YQ1ynJQm0l
lMllI4hFU8/lHNHm8ie5darpVXQEwsGUBBMBCgA+AhsDBQsJCAcDBRUKCQgLBRYC
AwEAAh4BAheAFiEE/rkgny8vP0ZkhB5VrBB7OGaS2t0FAmUkSiIFCQ+P/Z0ACgkQ
rBB7OGaS2t3HVg//S+/KbeOBf+nCdHsrWtp9kxvWIpAGvQhIbxx1tp/impfm+5Rm
fKPD0KX+g42fuMmOdDE4gjO4GjGd7ZY3bx+0zbDSdVebzmYCbPZ/BDP990oPKidd
w6Gl8PaIyqfuARKOESBETvAwNgwO4t2ocjs4pYZV+CuHvESYpqkuHjmHtye6ajZW
Mv24NhjVo4EfP33dPugTjXLjeuGT7qQpsYV3a66juHmPVkXwuPqxh9wTNc5TU6FG
UPSfIGMPL0xha7Rg2i5zvRaAxx4bHqGO8IAz/l/E/tJkV5xnt494HQam9UDbiFI0
Q0TSve1R6S45/UjQW6cycyduHtk72s9ipa9YM0ilTdLgKMWFjzYv4h4qeYvLw3oB
JGQew+I0I4dIrwL/TKet33EuFfwmyT9MaJBhqV6geFaQ0uVmwvzpAcvxIoSqSpkJ
B/kASqCEM/o0QZLuWc56cDsmMisD0ouVPt+c1Zk7AWLlf6j8LKYTbK0QxLRh/eeZ
jhSf3HnpaCfonbOoHmeo5d/o3EZ0AiA4GbT3xgScoIgXOT7KGgOWmtWkdyd3Zvl/
o6q6Hwpg4RsqCkwnfNm5ZIvmTAYXWc2hiICSicxrP0fek5Cc4xVgJR5RMNGyI7+m
ut1SE2WvLhMCwEyl5ecWF0tUze8VBlWkHJp0Y4k2ado39Zq/DZrTRQYEVrjCwX0E
EwEKACcCGwMFCwkIBwMFFQoJCAsFFgIDAQACHgECF4AFAmEyoZoFCQueVRUACgkQ
rBB7OGaS2t3axA//dgcz5T4Fs7LWdIQB/KRnvX64IzaGQcwt3FBhYH+sFaSaD+lu
752vi3jlGxMBKs2NFxk6e2U8xBEir3vfKYd+mZ5L6egXC9MYfvMO/UFEFH3A+t3V
dTOJK4RQcaL9+rFRVdDmZuifN9OFfy25d66JCZ5OiqgHTQViVmbrGw2cQdyNWXrq
YLgg+gktadkmzis4J6hF/le8NOBfrG3n+QthFl/v2ppYYW9pmmxzUIf6tAlR1Vr8
PhPukjuFfrPLRL3XPiK4LkdlSI5MX7Llq4RkcZN1NY1LWS8699wJOLRcr8aQYvzZ
Jm7tfZUaekJ5ScXJWJEaWT4poMbWxXINj6VwE+DqKWvjkzoxBXSIdLk4XThA1dIq
3n5SfMkfuWV2A2xHqJtEzI1XeTm/d/JRxIG8hjIs5FNMGJUSNANJuTVA2putCVf0
JbP4QlafXoEVOYwW/EFJY+brjQadwc/knf/QxszDcKb3THuTxR80A0h6ZysmtLEg
PCaD1xUCDdr4P7DGOtV7yMaDR6O8QKmb7TKzCKCPnHouqPT0DhSB2MRq437+mfSe
EGga/sPMxe0z8ugO2CnrCf3eplU3ZOy4eeQSThTKe0Hp5YlsF1cdZSvjt7GJaHR2
9A22nAJY9PojtlM+0Dkr/PH6r2brv3sEuACRNhzqCWhAve+zVnVLeb+Fk1rCwX0E
```

```
EwEKACcCGwMFCwkIBwMFFQoJCAsFFgIDAQACHgECF4AFAmCavPYFCQsGcHEACgkQ
rBB7OGaS2t3fqg//asHYSTBI4IoLibSF5ZJ8NaLo4EqgRtsO0W1zr8fCoFbxI+yI
qWrNXR7eoFj8tWO7Tj6S0kQr4QZcucLeILfox6CtDzO3f3WQTH9m/0si5U4Jf3RA
gBd0vwxVSSSNEsIUUfylOBHLVwlhaTVfh1h1dZhzb18LA1Vqu0100GGxvaG3dUl4
oMSRSK8EeaSO4hIM8ScjVyhjsuPRm1jlwd1EXZ5mkHRGRMGmwi3XysJjIeQRFmuG
a9uPes7I/K85r7X91BLz+G/mkSZsrHxzLxF0mSZtQdhq08GyMeQlJavs7sb1UVbg
ag30JEAjgqRsNIQOMIvl2/amrRJ7DUyQu5YkZQsAyCIhSVZw6zlJlkYVKSw1Cu1I
VX1n/Aahx5wwaQZA1dDTolX8WvL9O/KmEGWbKUSov40uoRwSOeXP3QhW75kD4zT0
n3pSUc7tXka8GAz5Ar+r90l4wc9as2WjWADo3CX8cF0zQOrN1naMQ++xBIAG9ms8
y3L4ECoRqvjCpjUAfhflxwSM7uclCgFBINFKSLV+QGxzWfHjg3+bSQd0hrRn9j4z
Di9aJmFESQ6iykbUjiUFrcI1NUCFKz2awaxh+Sq8UkYcvuxljxdK/zYYEG8DSdQ2
uwlssHCjYVpAb16hq1EAaSqnhv86020G4Z6JCg3AUZt9RlMBpK2Pn/NmPSzCwX0E
EwEKACcCGwMFCwkIBwMFFQoJCAsFFgIDAQACHgECF4AFAl771vAFCQlnimUACgkQ
rBB7OGaS2t35Jw/9GhpQquJVUCAsc4az7+ad8q2d9OUKXOLl2x0j3/ofS8umZJYr
8KXZxPqBqvLjlUyAB5Ur4fWVBdp9iP4Vj74jmRih7YatK8heGmNoUAnI1/90qV5O
ypFObfvWLxvcUNizmS/LYKdNuYcHwyw4bFyTz9gd3XH6Dxak7YiAXz3JgJIXcIB3
ELfpsduzpncLeVwz/dKhYX5iJ7Y/xd4Ew8Ian8FyNDNRwcXobTpPAMNiGLG5xSLq
8RgQfkm07BVVDtu5tPw4V46gnBXGXNjPhSirGMoNbKZqtPO7TcBQhPQ9c95x73hs
kAqZKHrwi1B7cfy1I8y5sRFbPa0RXeVWQKEMZdwLsFhCbEex6iXHP+rMK0nSJf0G
9lF2eJk140n8K7wzakOnjvN00VfN/9D+xD2lCXCzbYbaDXX2tYOd4GS72fW9M/zT
96tIH5UtMGMOICuUJjnL34EbzBuxwTEnJKhDGQH2P+eUzM9jmuCTRLLGw2P0Zuof
6GsSNLf+5pw3BIvEZw5PYT1N6i3ml9MQ7p9BWr7flCF8CU/xzyPN7TVb/677Be7V
SL1rNFLNi0IdqcbLJ63pTWaUGkCSqRnMfq3cIDlzNz2LoVvOO8VVmdtFRkhHN8hJ
h7CUXlaqBOfaJhV3FqA9G8sXmSN3r3pusZb13kfCKsJUZorThWxdaUBuUH3CwX0E
EwEKACcFAld1gAUCGwMFCQeGH4AFCwkIBwMFFQoJCAsFFgIDAQACHgECF4AACgkQ
rBB7OGaS2t1aKA//dBaXto7zUFRbJvLgcSE3hJ0ChYZj8Uuo7iYK26mVycyBFQJK
Iv2XYFIJMwK1Rx1Ja1jA6BIKE3aYyx2LVTYU1Hke826dhH+kV2qN9C6t/VmYpV4b
n/i64po7EzD17ykrm5gB+z47uCvyC79/r8Ouns1mTf/JUql/hYJj2hf1MDWr07/X
Wc1zBLjlT93tg/43Vgz1wFdrUOcNx1+bQGXKTOi4AXSp3UvCL+2YsQ2/JsPf7ZzZ
awSuUVkZJrllp87S1MhZeHmTrCHkV1FHJyudLJErcH0337Jpd8xDRek7Klrx2pAS
cKIiyeAMIk/G1OuExYqEEVFQzMr9ZlMNuhNBJAMr5hVGsTgrwy2hlIrBktXav07d
OleSlsqw9ViZ7F6t90x5l+NkwXVsLsGYSzuNyIfomNuoCgA+cfM3TjzVp41qsg1J
WJc9Bavaan2pKF6Lb9Fq8u3HZk2u+YZbvZkqkXwTdZZQOkEmoVV4Y1G86bdpmPyj
8eV7C02NxPii4l+qV8qJQu/6DsA0QwMtBMUNODm3BF2+ZmUHuhMGxq9/4vDE8heE
ffYhHtNftV6JwwzGZmeZkrYA1P9AGLeVp/6iNUe8H5/oPvh4s2rRmqN+L/dQU1ix
iOAT5iAKoRQGkduXrWc4fAY5KxDB9qna4oqXO6QP8rEflI8ELcNgYEj4oJvOwU0E
V3WABQEQALzM0Cs9Zvd08xOEvbEBj59LrS9d0HVKQ61gmkNakWC+jR35VD6FXpe6
UYAcBLrEbVYfKw9P0p6MhFKAsb570JoznKGzE1rVYUZQzhD0RKje35rvkajvEcjG
AWMLTjr87pWHeD0389ER64bzORncfa/l+YP56PI+CThb2wUvTTONGJkPQUpVhH+P
256cQL/Y0Fwu4XLerpwN+YKgMQ47raRcydobPeSfMQr9fVKRyOzFEOrvNpCVDUqi
77d0gLDLjHlIlDyOX5554S8XYLb91eYOiFvnu2pTCKiiExRCSYK29mAQePKlTCCn
QxOjbmBbGS8mVIkpQ5vpvXvzpY3JIjMXaDGqWSQSYGXhECyxCR5eOtKYbCwwPIc2
rIl5gW6yXyw9pKmj5XafTP7YHTvRSr7CZ/VLkDkWl6AfQ9nPOg1mjwjpDFpmN71h
JlSKMaZkh0QGV5FW3dK+GLwxiWdqx3htbZErWyvumWQF/xBF7puKJBEXcoM5KfkJ
```

```
uZekBwcnVkfNFF2RdkM1ALq8InGzLXc7ROuEm0BXVirfju7JRtWLb3UhJWCuhRW2
muyYegSTkag5MduD1IJK37GL8WIlAL65taYgZegUoxHdSaEOefOhspxuduz8d33z
UV1WCFhi+r/+BMCQmTRbF8ao7fTC1dGd084DRP6qE/dMT4u0ZEn7ABEBAAHCwXwE
GAEKACYCGwwWIQT+uSCfLy8/RmSEHlWsEHs4ZpLa3QUCZwAXCwUJEWvKhQAKCRCs
EHs4ZpLa3XtzD/9dwi1qffV70UTq8w/21jn1owHp09jxP7WHTmPWHE0BW5yFIWlV
A1gKN6Ym0dw+LvS5WOKJaRnyewUyBxWvZsn6Wlb5qzY7nmCOKJpYtuCUPwiqjXWP
EM8c/v0MojSuwMOXBAViLvOFhgdUrHn1lk962XvWAW++4DXFh2deaV0163IFMRmO
PNPDAiPWBVqvBANIh2sLRZ5gd1BXwpVrd+x8tzyr69YrN7hutPlCyPEUM9//mcEh
vFPsbW/iOx/foCE3NXhQm/rSMKecVn5csXBV2JOlMzi+8txYNrSBLkjbSB1AvTQ1
aG3+nCNCgM2XDLyoj0IrgZ1To4Ay5gmTOR+msY/cfoIuKFYenmtxy6jM8o5uSZHg
hoClrx9IA98hhGQ73G2r5EDpXuU/uCXn53Sswj65bl9IssfqEIoji/FonkkpEgeg
bGXFDUnrhicDO/WOzqpXf2Fa0DQWY+Vc/pt52ftBFgwzCNIUYDKUhCHPnZ0wtLtd
N2fkXHNiCavCDZlOud7FHHwmRNdj2q1uKxe4m+pFYmKwAU/H+Htkz9Gjsj+ZKedY
nnfai2s2gQOrbfwvV9VdhCWSuLK17ZnGTtiJuOUQIlV8n6QQJpohd3mVgmynu6gQ
uKw0YS2RuEUFv0vOg2tASA+4EM/SBUpGhudODLA4b5wO4gKmh1B1HqQrIsLBfAQY
AQoAJgIbDBYhBP65IJ8vLz9GZIQeVawQezhmktrdBQJlJEokBQkPj/2fAAoJEKwQ
ezhmktrdwMAP/RpFylIL4yhgscBOEnQ7e3No8OraNk0z/YhSd125N/uQVEU94JGQ
rrvQ+4Lfve2laPweBDO18/A0CsmOyHPVQMA0a2vx8ItVdIcNc8iFkP4AJ1922lOq
i0Vh0b1UeZnlfK9+Qvq4PQ2lhWJr0uzyL/S38REsAT1I25sfJOP+RCaR1MH9dm85
E56Lee6uZR8SkGuiL6kGpPh6fWTNij3bICjth1iSSCL2HCOW8lvcwSldDu2EfILU
QCSqfSG7bF8dFk+nKhzhVXOUks3XGjLdICxZewU5ycryitpfRgARgZs2A43gshdi
fiKaX6Ksan03uhKDrLhDHNj2y07PUrFo8ggtlRpV/PrlB/UqCsC9FUOixbD+n4ZF
Sqov2qwelLj0f4mZ6yiLsTDUOFPrdkOlHTJZl7AF0zXZMM6CvaCUaJCKx9GVdSrR
+LI4wLQonPrTnXavhkC4intlqSX8ZQNLhEggdE8YwMEJn59R/nVIT3i5WzYph5R9
P4Vz3Yn7jRqM8wAyEbHkA8s45fMRi9akWSw93H5nWukcmfkt3UEbmka3BQg3HKWP
6TvhfI28euM8qqjbPilfkpEBjnChYVk2Rgn0P8zA7Q5kCo293kwJL9c3RDjMPcxI
45ktKvBTZftsDt1Z718LwW7Q3VQiGiKvo1XLMuV7Z51fmydfUPcrnv17wsFlBBgB
CgAPAhsMBQJhMqGaBQkLnlUVAAoJEKwQezhmktrdbhwQAITmFb67XIUZswr3TREd
Q7ZCLG4EDyfTsW8n75r6A9OqsR+z68nC2Sm7e8mKQFFPwjHPOhsGhHtCOTZtQk7O
jbwyL4N3uxDyEvOfbckH5WzOejZcG7KKQrqAiWJJ7q6CH/zOnVurySjVyzJpy/wL
WpVAcF/uaW5ZhlFCXqePaEzsUBJ757qsr2ho14BV4seT1RSQ9nneTZ0Hhab3wqXP
4qDTo8+zkTvNo9YbeZ1qj62l1+QGIUBTP5MEdXCuC1e4FQ3f6vnXxmB86cUPx7cl
/y2rIjeiOdkKgPeUjNWWSzxS2jYehL5we7gvaSwmEvJ74pV+/3Hs+TxX39XtYFwj
k9I795idnsS5l1dAW3yoI3HBQsYa3US7bpH4g3yZMkstc3bHJ6X54PMCd8Skb+N3
FE8+zGduDmDTKitumiWVVxEFGIwsLAcpWPxecI2AMIMGfMheURYsdvD/yvCbCB29
0KwCSrDVkAG9N2VorNzd7KUeTPTMN1bg2d1lF6u5sQeTN5KVaGd7xE1OXME2wA2D
T3+EsAQytriFbcWm3s8Ugbc9BXMmKBfjlvKu6+Fr6Mgvf/txn56M2SyXBCFQ5OFt
qTFuAFIRv+nayk5tx5Eg1iA7u3dbB1jH3yxGH1B7TeQypA5BqD3x72b7vbXkeci3
1Kz035LYoT5/yTK5sGvacIvCwsFlBBgBCgAPAhsMBQJgmrz3BQkLBnByAAoJEKwQ
ezhmktrdLmgP/1FkWkYhxACnkagRv09mpPl2STbu0B3zYKFBALm/Wa7vKDz18dgC
S3BxDSlpnhZS8QA3VjmbOAZvaDnsN1UJ0f3Qao5136G/UXPnmFIwN612szP0K6nF
PEsotzIzRlJo3S+WkBfiKaQDIDgSxtUxJzOwufz76xibmKRhJ5ChMDCvxmIaoNle
tKRxFT77OupnnyaQs22UsueqrZJ0resgTVnNeF4A1+lU59pFuAlf971SVLr472LP
Uj8mPJihF2ukL1Hdz3F7+kYlpOJRmLk9fo4dlZHBUPiZ1ML/U2yhQfW+Y6tW71vf
```

```
izAxJWF7se6QT+UT5Pji6cohMSERVoYt8e2jFjs0PiPcrjU3mJEx4hAEEVIbP9RY
eKC4CL/UGYAtJkUjd85vKZUHYr7NWZQKLAKqpAPQUMKrIKLEHuz/doq2CCamstLI
vcBgg8EjLJnl3SBesFt/1DCWZeummqz3omQKRl9EHU2cIzIf0Cv/IEysnmbpSpjZ
DX8Fqjtezoq1qiyrLFR7YN1VDPBCHYfqDagw10nlrWFJqT6VqfslmdMdTBRWYVEB
0GUxrkyI+APdi0M2634/410b1ptkqyTIr1KIg1J/qsSiKVcBZSOYFW/rskxYcPPT
wpKYaycEYtOdkS6FPcnehJOOlB+F32WVq2bs2Ps8we6KhjjaYS4Iv4dwwsFlBBgB
CgAPAhsMBQJe+9W/BQkJZ4k1AAoJEKwQezhmktrdvzMQAI6BBj3c2r4bDpV3TwkX
dQ+UCa/E/zUhFds9XKfGb3a5IzRdPUwT+KrAZyiYrr2NSMOzhl/VtqJL18YCYsxO
Ob/TB1hDM+IZiI5gH0cHKhDYKTnNSGP09P/pJAlvHQend9CdZE9J9jwkczfS+bz6
mVxkxpi73fTDox9duesOLsS2/ntRzA0wqhDdaaavRvhAEf9vavCWVrNZmq22WVsU
lnIPxNWGGzWn85JYI6uAi4f4/ABFkry69/c0cvbr0P8qgCmeCuGmX4f0j7qRg77A
+mSueBDx8RK0O2ol021B7b8IcVizj+lpsRQN0oa+i+mFG+o6vtD1ZYhQude4N5sR
RybcLclxjSCoZs5q9JfTpbB2n7pSf/UD3ytwnt9kpD4Vv9dTGAPB83bjL+QK6e3A
XM10jxFE5jSFSr94E40kK80YcIR5jLqsg2f610ENY5drMSA4zuDFDL1Y2ChfjgjZ
uNoFbPHGt/8DfWTVOochVnikA7ggKjz20+RjvwyrHhRMAftO8MMh9UV28pdL+H53
oOtOV0u5aoTbcNqdYQy9B2Bw4lfmj2fi6Dpl+vnZp6hOm0CWiJVW/dtilppYjuxd
w5Kj+9IxZYaBNYH4l1pMT+BsvMDqGzXxDIL89NnY5BkMvqEKnjXSHGRWYMz0xigf
51YKbfQnEQ1oz5bRQndntRQWwsFlBBgBCgAPBQJXdYAFAhsMBQkHhh+AAAoJEKwQ
ezhmktrdTyEP/0HOVWHwQsaWjMrGj0OOMFzxGUo8SBmYYTBs29VM8wBGDsPkYCje
ZzU16i9iqDpDqxpyqmTigcjHV8CDx/6xsMBLG2yKaKZ4m3+YnOQf/sQkyCvqiyMF
9mS7pDYWy+mPhPuw8TDIfiqgVhzjSpIMFWPqxVjn6KKbPN/QASr3Pf0cuP6qpHG+
NAM6Q5dYkCebyvwzLmg1sVnil6iSyJd1jBj3D34XrgWS9buyxBB2CjIM76WxfNVi
J9zAaPI78X9v6PpDGn0kg6oLzrusrvBjoZknKQm0SZ+41fx6xvrTPs8uPEzevzJB
lkke6kw9+KagY8mrVX1ZenRg+sY/4vxJreYWQeql67ggx+wFjKDcfhZA7m7OLHOD
ysrGVCLcmuinUBaNlHmLDcGYXZ+kMCoXfObpuCVByQmNJgEb47EIFlx/+TEeNHKM
O+22xL1atFzXfkEVZck+NghLZyFDhS3g1bma7puU7r752uiJjA6Iv8+kHDXi+/V7
GNpuiEFUYh69QQ2//CS5H51osC/Bkb9evSn/Lp8dMubtWAaXDGJMgw9vqZ55N02N
K0fvF/IKHnGkvH28rv0OPCvOWTA/MClv28y0PrSvcvMXnduLtkBEX7TISMPW+n+O
Ta63/z4YFfEZ7sFLrEm3Q3vJMN3mE5i3cw+JGXPSu0nTtgqk/oZv//SS
=bboB
-----END PGP PUBLIC KEY BLOCK-----
```

## 過期日期：2024-10-08

| | |
|---|---|
| 金鑰 ID | 0xAC107B386692DADD |
| 類型 | RSA |
| 大小 | 4096/4096 |
| 已建立 | 2016-06-30 |

| | |
|---|---|
| 過期日期 | 2024-10-08 |
| 使用者 ID | AWS SDKs和工具 <aws-dr-tools@amazon.com> |
| 金鑰指紋 | FEB9 209F 2F2F 3F46 6484 1E55 AC10 7B38 6692 DADD |

若要將適用於 Java 的 SDK 的下列 OpenPGP 公有金鑰複製到剪貼簿，請選取右上角的「複製」圖示。

```
-----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-----
```

# 文件歷史記錄

此頁面列出 適用於 Java 的 AWS SDK 開發人員指南在其歷史記錄過程中的重要變更。

本指南發佈日期：2025 年 10 月 1 日。

2025 年 10 月 1 日

新增將於 2026-09-27 過期的新 PGP 金鑰。

2024 年 10 月 5 日

更新目前的 OpenPGP 金鑰資訊。

2024 年 9 月 4 日

新增 DynamoDB AWS 帳戶型端點的相關資訊。請參閱 the section called “使用帳戶 AWS 型端點”。

2024 年 5 月 21 日

使用 java 命令列系統屬性移除設定`networkaddress.cache.ttl`安全屬性的指示。請參閱 如何設定 JVM TTL。

2024 年 1 月 12 日

新增宣告 v1 適用於 Java 的 AWS SDK .x 支援結束的橫幅。

2023 年 12 月 6 日

- 提供目前的 OpenPGP 金鑰。

2023 年 3 月 14 日

- 更新了指南以符合 IAM 最佳實務。如需更多詳細資訊，請參閱 IAM 中的安全最佳實務。

2022 年 7 月 28 日

- 新增提醒，指出 EC2-Classic 將於 2022 年 8 月 15 日淘汰。

2018 年 3 月 22 日

- 移除管理 DynamoDB 範例中的 Tomcat 工作階段，因為不再支援該工具。

2017 年 11 月 2 日

- 新增 Amazon S3 加密用戶端的密碼編譯範例，包括新主題：使用 Amazon S3 用戶端加密和Amazon S3 用戶端加密搭配 AWS KMS 受管金鑰，以及Amazon S3 用戶端加密搭配用戶端主金鑰。

2017 年 4 月 14 日

- 對範例進行多次更新 Amazon S3 使用 適用於 Java 的 AWS SDK區段，包括新主題：管理儲存貯體和物件的 Amazon S3 存取許可，以及將 Amazon S3 儲存貯體設定為網站。

2017 年 4 月 4 日

- 啟用 指標的新主題 適用於 Java 的 AWS SDK說明如何為 產生應用程式和 SDK 效能指標 適用於 Java 的 AWS SDK。

2017 年 4 月 3 日

- 使用 章節將新的 CloudWatch 範例新增至 CloudWatch 範例：從 CloudWatch 取得指標、發佈自訂指標資料、使用 CloudWatch 警示、在 CloudWatch 中使用警示動作，以及將事件傳送至 CloudWatch CloudWatch 適用於 Java 的 AWS SDK

2017 年 3 月 27 日

- 新增更多 Amazon EC2 範例至Amazon EC2 範例 使用 適用於 Java 的 AWS SDK 區段：管理 Amazon EC2 執行個體、在 中使用彈性 IP 地址 Amazon EC2、使用區域和可用區域、使用 Amazon EC2 金鑰對，以及在 中使用安全群組 Amazon EC2。

2017 年 3 月 21 日

- 使用 區段將一組新的 IAM 範例新增至 IAM 範例 適用於 Java 的 AWS SDK：管理 IAM 存取金鑰、管理 IAM 使用者、使用 IAM 帳戶別名、使用 IAM 政策，以及使用 IAM 伺服器憑證

2017 年 3 月 13 日

- 將三個新主題新增至 Amazon SQS 區段：啟用 Amazon SQS 訊息佇列的長輪詢、在 中設定可見性逾時 Amazon SQS，以及在 中使用無效字母佇列 Amazon SQS。

2017 年 1 月 26 日

- 新增了使用 TransferManager for Amazon S3 Operations 的新 Amazon S3 主題，以及使用 適用於 Java 的 AWS SDK 一節中的AWS 主題開發的新最佳實務 適用於 Java 的 AWS SDK。

2017 年 1 月 16 日

- 新增了新 Amazon S3 主題、使用儲存貯體政策管理對 Amazon S3 儲存貯體的存取，以及使用 Amazon SQS 訊息佇列和傳送接收和刪除 Amazon SQS 訊息的兩個新 Amazon SQS 主題。

2016 年 12 月 16 日

- 新增了 的新範例主題 DynamoDB：在 中使用資料表 DynamoDB和在 中使用項目 DynamoDB。

2016 年 9 月 26 日

- 進階區段中的主題已移至使用 適用於 Java 的 AWS SDK，因為它們確實是使用 SDK 的核心。

2016 年 8 月 25 日

- 新主題建立服務用戶端已新增至使用 適用於 Java 的 AWS SDK，示範如何使用用戶端建置器來簡化 AWS 服務 用戶端的建立。

  適用於 Java 的 AWS SDK 程式碼範例區段已更新為 S3 的新範例，這些範例由 GitHub 上包含完整範例程式碼的儲存庫支援。

2016 年 5 月 2 日

- 新主題非同步程式設計已新增至使用 適用於 Java 的 AWS SDK一節，說明如何使用傳回`Future`物件或需要 的非同步用戶端方法`AsyncHandler`。

2016 年 4 月 26 日

- SSL 憑證需求主題已移除，因為它不再相關。2015 年已棄用對 SHA-1 簽署憑證的支援，並且已移除存放測試指令碼的網站。

2016 年 3 月 14 日

- 已將新主題新增至 Amazon SWF 章節：Lambda 任務，說明如何實作 Amazon SWF 工作流程，將 Lambda 函數呼叫為任務，以替代使用傳統 Amazon SWF 活動。

2016 年 3 月 4 日

- Amazon SWF 使用 區段的範例 適用於 Java 的 AWS SDK已更新為新內容：
  - Amazon SWF 基本知識 - 提供如何在專案中包含 SWF 的基本資訊。
  - 建置簡易 Amazon SWF 應用程式 - 新的教學課程，提供 Java 開發人員新手的step-by-step指引 Amazon SWF。
  - 關閉活動和工作流程工作者 - 描述如何使用 Java 的並行類別正常關閉 Amazon SWF 工作者類別。

2016 年 2 月 23 日

- 適用於 Java 的 AWS SDK 開發人員指南的來源已移至 aws-java-developer-guide。

2015 年 12 月 28 日

- the section called “設定 DNS 名稱查詢的 JVM TTL” 已從進階移至使用 適用於 Java 的 AWS SDK，且為了清楚起見已重新撰寫。

  使用 SDK 搭配 Apache Maven 已更新，其中包含有關如何在專案中包含 SDK 物料清單 (BOM) 的資訊。

2015 年 8 月 4 日

- SSL 憑證需求是入門章節的新主題，說明 AWS「移至 SSL 連線的 SHA256-signed憑證，以及如何修正早期 1.6 和先前的 Java 環境以使用這些憑證，這是 2015 年 9 月 30 日後存取的必要項目 AWS 。

> **Note**
>
> Java 1.7+ 已能夠使用 SHA256-signed憑證。

2014 年 5 月 14 日

- 簡介和入門資料已大幅修訂，以支援新的指南結構，現在包含如何設定 AWS 登入資料和開發區域的指引。

  程式碼範例的討論已移至其他文件和資源區段中自己的主題。

  有關如何檢視 SDK 修訂歷史記錄的資訊已移至簡介。

2014 年 5 月 9 日

- 文件的整體結構 適用於 Java 的 AWS SDK 已簡化，並已更新入門和其他文件和資源主題。

  已新增新主題：

  - 使用 AWS 登入資料 - 討論您可以指定登入資料以搭配 使用的各種方式 適用於 Java 的 AWS SDK。
  - 使用 IAM 角色授予 上 AWS 資源的存取權 Amazon EC2，提供有關如何安全地為在 EC2 執行個體上執行的應用程式指定登入資料的資訊。

2013 年 9 月 9 日

- 本主題文件歷史記錄會追蹤 適用於 Java 的 AWS SDK 開發人員指南的變更。其為版本備註歷史記錄的相關文件。