SDK版本 3 的開發人員指南

AWS SDK for JavaScript

AWS SDK for JavaScript: SDK版本 3 的開發人員指南

Table of Contents

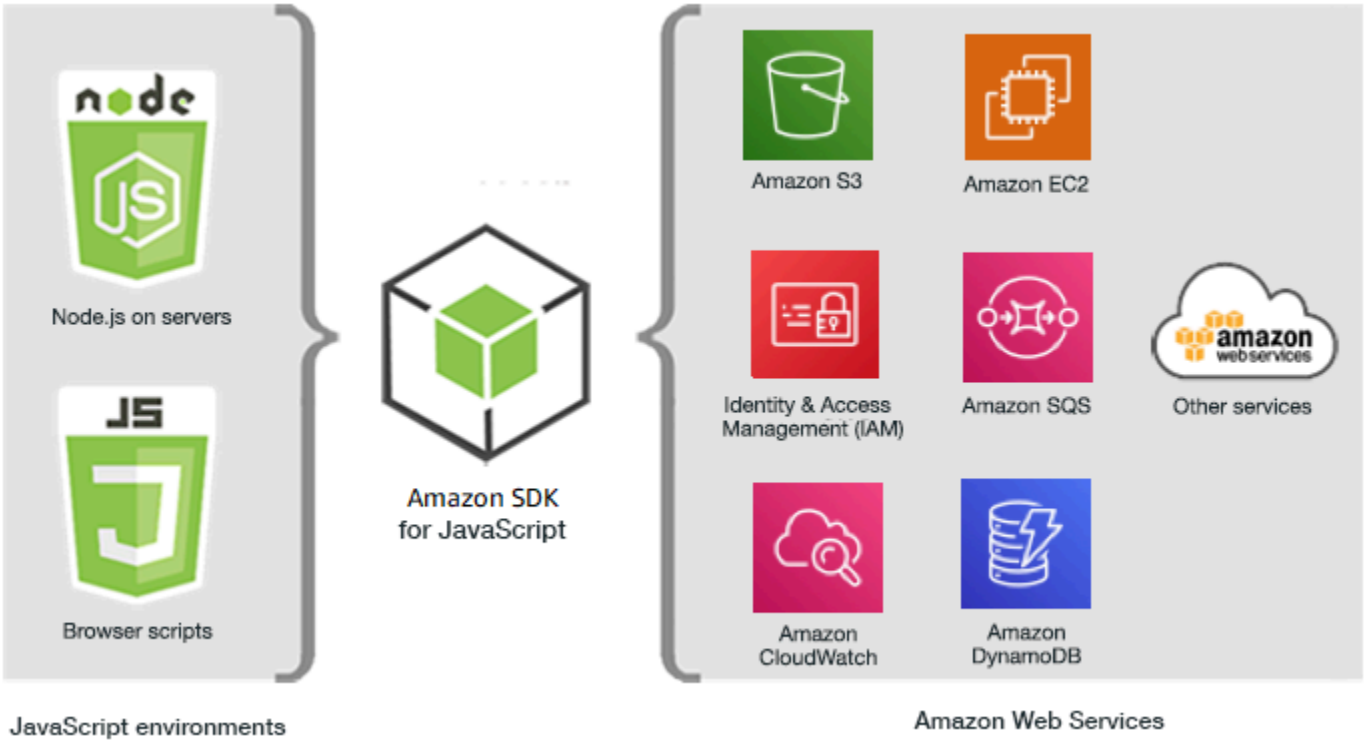
AWS SDK for JavaScript V3 參API考指南詳細描述了 AWS SDK for JavaScript 版本 3（V3）的所有API操作。

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。

什麼是 AWS SDK for JavaScript？

歡迎使用開 AWS SDK for JavaScript 發人員指南。本指南提供有關設定和設定 AWS SDK for JavaScript. 它也會引導您完成使用執行各種 AWS 服務的範例和教學課程 AWS SDK for JavaScript。

AWS SDK for JavaScript v3 API 參考指南提供了適用於 AWS 服務的 JavaScript API。您可以使用 JavaScript API 來建置 Node.js 或瀏覽器的程式庫或應用程式。



開始使用 SDK

如果您已準備好動手使用 SDK，請按照中的範例進行操作開始使用。

若要設定您的開發環境，請參閱設定下列項目的 SDK JavaScript。

如果您目前正在使用 SDK 的 2.x 版 JavaScript，請參閱遷移到 v3 以獲取具體指導。

如果您正在尋找的程式碼範例 AWS 服務，請參閱SDK對於 JavaScript （ v3 ）代碼示例。

開發套件主要版本的維護與支援

如需開發套件主要版本及其基礎相依性之維護與支援的相關資訊，請參閱《AWS 開發套件及工具參考指南》中的以下內容：

- AWS SDK 和工具維護政策
- AWS SDK 和工具版本支援對照表

使用軟體開發套件搭配 Node.js

Node.js 是執行伺服器端 JavaScript 應用程式的跨平台執行階段。您可以在亞馬遜彈性運算雲端 (Amazon EC2) 執行個體上設定 Node.js，以便在伺服器上執行。您也可以使用 Node.js 編寫按需 AWS Lambda 函數。

使用適用於 Node.js 的 SDK 與您在網頁瀏覽器 JavaScript 中使用它的方式不同。不同之處在於載入軟體開發套件及取得存取特定 web 服務所需登入資料的方式。當 Node.js 和瀏覽器之間的特定 API 的使用不同時，我們稱之為這些差異。

使用 SDK 搭配使用 AWS Cloud9

您也可以 JavaScript 在 AWS Cloud9 IDE 中使用的 SDK 來開發 Node.js 應用程式。如需與 SDK AWS Cloud9 搭配使用的詳細資訊 JavaScript，請參閱搭 AWS Cloud9 配使用 AWS SDK for JavaScript。

使用 SDK 搭配使用 AWS Amplify

對於以瀏覽器為基礎的 Web、行動裝置和混合式應用程式，您也可以在上 GitHub使用AWS Amplify 資料庫。它擴展了 SDK JavaScript，提供了一個聲明接口。

> **Note**
>
> 框架，如 Amplify 可能不會提供相同的瀏覽器支持作為 JavaScript SDK 的. 有關詳細信息，請參閱框架的文檔。

搭配網頁瀏覽器使用 SDK

所有主要的 Web 瀏覽器都支持執行 JavaScript. JavaScript 在 Web 瀏覽器中運行的代碼通常被稱為客戶端 JavaScript。

如需支援的瀏覽器清單 AWS SDK for JavaScript，請參閱支援網頁瀏覽器。

在網頁瀏覽器 JavaScript 中使用 SDK 的方式與您將其用於 Node.js 的方式不同。不同之處在於載入軟體開發套件及取得存取特定 web 服務所需登入資料的方式。當 Node.js 和瀏覽器之間的特定 API 的使用不同時，我們稱之為這些差異。

在 V3 中使用瀏覽器

V3 使您可以將所需 JavaScript 文件的 SDK 捆綁並包含在瀏覽器中，從而減少開銷。

要在 HTML 頁面 JavaScript 中使用 SDK 的 V3，您必須使用 Webpack 將所需的客戶端模塊和所有必需的 JavaScript 功能捆綁到單個 JavaScript 文件中，並將其添加到 HTML 頁面的<head>腳本標記中。例如：

```
<script src="./main.js"></script>
```

> **Note**
>
> 如需 Webpack 的詳細資訊，請參閱捆綁應用程序與網絡包。

若要使用 SDK 的 V2 JavaScript，您可以新增指向 V2 SDK 最新版本的指令碼標記。如需詳細資訊，請參閱 AWS SDK for JavaScript 開發人員指南 v2 中的範例。

常用案例

JavaScript 在瀏覽器腳本中使用 SDK 可以實現許多令人信服的用例。以下是您可以通過使用 SDK 訪問各種 Web 服務在瀏覽器應用程序中構建的幾個想法。 JavaScript

- 為 AWS 服務建置自訂主控台，讓您在其中存取並結合不同區域和服務的功能，以最佳符合您的組織或專案需求。
- 使用 Amazon Cognito 身分來啟用經過驗證的使用者存取您的瀏覽器應用程式和網站，包括使用來自 Facebook 和其他人的第三方身份驗證。
- 使用 Amazon Kinesis 即時處理點擊串流或其他行銷資料。
- 使用 Amazon DynamoDB 獲得無伺服器資料持續性，例如網站訪客或應用程式使用者的個別使用者偏好設定。
- 用 AWS Lambda 於封裝您可以從瀏覽器指令碼叫用的專有邏輯，而無需下載並向使用者顯示您的智慧財產權。

關於範例

您可以瀏覽 SDK 以取得AWS 程式碼 JavaScript 範例存放庫中的範例。

資源

除了本指南之外，下列線上資源可供 JavaScript 開發人員使用的 SDK：

- AWS SDK for JavaScript V3 API 參考指南
- AWS SDK 和工具參考指南：包含 SDK 中常見的設定、功能和其他基礎概念。 AWS
- JavaScript 開發者博客
- AWS JavaScript 論壇
- JavaScript AWS 程式碼目錄中的範例
- AWS 代碼示例存儲庫
- 吉特通道
- 堆疊溢位
- 堆棧溢出問題標籤。-sdk-js
- GitHub
  - 開發套件源
  - 文件來源

開始使用 AWS SDK for JavaScript

提 AWS SDK for JavaScript 供瀏覽器或 Node.js 環境中對 Web 服務的存取。本節提供入門練習，說明如何在這些 JavaScript 環境 JavaScript 中使用 SDK。

> **Note**
>
> 您可以 JavaScript 在 AWS Cloud9 IDE 中使用 SDK 來開發 Node.js 應用程式，以及 JavaScript 基於瀏覽器的應用程式。如需如何用 AWS Cloud9 於 Node.js 開發的範例，請參閱搭 AWS Cloud9 配使用 AWS SDK for JavaScript。

主題

使用 SDK 驗證功能 AWS

在開發 AWS 時，您必須建立程式碼的驗證方式。 AWS 服務您可以根據環境和您可用的存取權限，以不同的方式設定 AWS 資源的程式設計 AWS 存取。

若要選擇驗證方法並針對 SDK 進行設定，請參閱 SDK 和工具參考指南中AWS 的驗證和存取。

我們建議新用戶誰是在本地開發，並沒有給他們的雇主進行身份驗證的方法來設置 AWS IAM Identity Center。此方法包括安裝以便 AWS CLI 於設定，以及定期登入 AWS 存取入口網站。如果選擇此方法，則在 AWS SDK 和工具參考指南中完成 IAM 身分中心身份驗證的程序後，您的環境應包含以下元素：

- 您可以在 AWS CLI執行應用程式之前啟動 AWS 存取入口網站工作階段。
- 具有設定AWSconfig檔的共用檔案，其中包含一組可從 SDK 參考的設定值。`[default]`若要尋找此檔案的位置，請參閱 AWS SDK 和工具參考指南中的共用檔案位置。
- 共用`config`檔案會`region`設定設定。這會設 AWS 區域 定 SDK 用於 AWS 要求的預設值。此區域用於未指定與要使用的區域一起指定的 SDK 服務請求。

- SDK 會使用設定檔的 SSO 權杖提供者組態，在傳送要求之前取得認證 AWS。此sso_role_name值是連接至 IAM 身分中心權限集的 IAM 角色，可讓您存取應用程式中 AWS 服務 使用的角色。

  下列範例config檔案顯示使用 SSO 權杖提供者組態設定的預設設定檔。設定檔的sso_session設定是指已命名的sso-session區段。此sso-session區段包含用來啟動 AWS 存取入口網站工作階段的設定。

```
[default]
sso_session = my-sso
sso_account_id = 111122223333
sso_role_name = SampleRole
region = us-east-1
output = json

[sso-session my-sso]
sso_region = us-east-1
sso_start_url = https://provided-domain.awsapps.com/start
sso_registration_scopes = sso:account:access
```

AWS SDK for JavaScript v3 不需要將其他套件 (例如SSO和SSOOIDC) 新增至您的應用程式，即可使用 IAM 身分中心身份驗證。

有關明確使用此憑據提供程序的詳細信息，請參閱 npm（Node.js 包管理器）網站fromSSO()上的。

啟動 AWS 存取入口網站會話

在執行存取的應用程式之前 AWS 服務，您需要 SDK 的使用中存 AWS 取入口網站工作階段，才能使用 IAM 身分中心身分驗證來解析登入資料。根據您設定的工作階段長度，您的存取最終會過期，SDK 會遇到驗證錯誤。若要登入 AWS 存取入口網站，請在中執行下列命令 AWS CLI。

```
aws sso login
```

如果您遵循指引並具有預設設定檔設定，則不需要使用--profile選項呼叫指令。如果您的 SSO 權杖提供者組態使用已命名的設定檔，則命令為 `aws sso login --profile named-profile`。

若要選擇性地測試您是否已有作用中的工作階段，請執行下列 AWS CLI 命令。

```
aws sts get-caller-identity
```

如果您的工作階段處於作用中狀態，對此命令的回應會報告共用`config`檔案中設定的 IAM 身分中心帳戶和權限集。

> **Note**
>
> 如果您已經擁有作用中的 AWS 存取入口網站工作階段並執行`aws sso login`，則不需要提供認證。
>
> 登入程序可能會提示您允許 AWS CLI 存取您的資料。由於 AWS CLI 是建置在適用於 Python 的 SDK 之上，因此權限訊息可能會包含`botocore`名稱的變體。

更多認證資訊

人類使用者具有人類身分，是應用程式的相關人員、管理員、開發人員、操作員和消費者。他們必須具有身份才能訪問您的 AWS 環境和應用程序。屬於您組織成員的人類使用者 (也就是開發人員) 稱為員工身分識別。

存取時使用臨時認證 AWS。您可以為您的人類使用者使用身分識別提供者，藉由假設角色 (提供臨時認證) 來提供 AWS 帳戶的聯合存取權。對於集中式存取管理，我們建議您使用 AWS IAM Identity Center (IAM Identity Center) 來管理帳戶的存取權限和這些帳戶內的許可。有關更多替代方案，請參閱以下內容：

- 如需了解有關最佳實務的資訊，請參閱 IAM 使用者指南中的 IAM 安全最佳實務。
- 若要建立短期 AWS 登入資料，請參閱 IAM 使用者指南中的臨時安全登入資料。
- 若要瞭解其他 AWS SDK for JavaScript V3 認證提供者，請參閱 AWS SDK 和工具參考指南中的標準化認證提供者。

開始使用 Node.js 的使用者

本指南說明如何初始化 NPM 套件、將服務用戶端新增至套件，以及如何使用 JavaScript SDK 呼叫服務動作。

該方案

使用一個執行下列作業的主要檔案建立新的 NPM 套件：

- 創建一個 Amazon 簡單存儲服務桶
- 把一個對象在 Amazon S3 存儲桶

- 讀取 Amazon S3 存儲桶中的對象
- 確認使用者是否要刪除資源

必要條件

您必須先執行下列動作，才能執行範例：

- 設定您的 SDK 驗證。如需詳細資訊，請參閱 使用 SDK 驗證功能 AWS。
- 安裝 Node.js。

步驟 1：設定套件結構並安裝用戶端套件

若要設定套件結構並安裝用戶端套件：

1. 建立包含套件的新資料夾nodegetstarted。
2. 從指令行導覽至新資料夾。
3. 執行下列命令以建立預設package.json檔案：

```
npm init -y
```

4. 執行下列命令以安裝 Amazon S3 用戶端套件：

```
npm i @aws-sdk/client-s3
```

5. 新增"type": "module"至檔package.json案。這告訴 Node.js 使用現代的 ESM 語法。最終package.json應類似於以下內容：

```
{
  "name": "example-javascriptv3-get-started-node",
  "version": "1.0.0",
  "description": "This guide shows you how to initialize an NPM package, add a
 service client to your package, and use the JavaScript SDK to call a service
 action.",
  "main": "index.js",
  "scripts": {
"test": "vitest run **/*.unit.test.js"
  },
  "author": "Your Name",
```

```
  "license": "Apache-2.0",
  "dependencies": {
 "@aws-sdk/client-s3": "^3.420.0"
  },
  "type": "module"
}
```

第 2 步：添加必要的導入和 SDK 代碼

將以下代碼添加到文件nodegetstarted夾index.js中名為的文件。

```
// This is used for getting user input.
import { createInterface } from "readline/promises";

import {
  S3Client,
  PutObjectCommand,
  CreateBucketCommand,
  DeleteObjectCommand,
  DeleteBucketCommand,
  paginateListObjectsV2,
  GetObjectCommand,
} from "@aws-sdk/client-s3";

export async function main() {
  // A region and credentials can be declared explicitly. For example
  // `new S3Client({ region: 'us-east-1', credentials: {...} })` would
  //initialize the client with those settings. However, the SDK will
  // use your local configuration and credentials if those properties
  // are not defined here.
  const s3Client = new S3Client({});

  // Create an Amazon S3 bucket. The epoch timestamp is appended
  // to the name to make it unique.
  const bucketName = `test-bucket-${Date.now()}`;
  await s3Client.send(
    new CreateBucketCommand({
      Bucket: bucketName,
    })
  );
```

```
// Put an object into an Amazon S3 bucket.
await s3Client.send(
  new PutObjectCommand({
    Bucket: bucketName,
    Key: "my-first-object.txt",
    Body: "Hello JavaScript SDK!",
  })
);

// Read the object.
const { Body } = await s3Client.send(
  new GetObjectCommand({
    Bucket: bucketName,
    Key: "my-first-object.txt",
  })
);

console.log(await Body.transformToString());

// Confirm resource deletion.
const prompt = createInterface({
  input: process.stdin,
  output: process.stdout,
});

const result = await prompt.question("Empty and delete bucket? (y/n) ");
prompt.close();

if (result === "y") {
  // Create an async iterator over lists of objects in a bucket.
  const paginator = paginateListObjectsV2(
    { client: s3Client },
    { Bucket: bucketName }
  );
  for await (const page of paginator) {
    const objects = page.Contents;
    if (objects) {
      // For every object in each page, delete it.
      for (const object of objects) {
        await s3Client.send(
          new DeleteObjectCommand({ Bucket: bucketName, Key: object.Key })
        );
      }
    }
```

```
    }

    // Once all the objects are gone, the bucket can be deleted.
    await s3Client.send(new DeleteBucketCommand({ Bucket: bucketName }));
  }
}

// Call a function if this file was run directly. This allows the file
// to be runnable without running on import.
import { fileURLToPath } from "url";
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  main();
}
```

您可以在這裡找到範例程式碼 GitHub。

步驟 3：執行範例

> **Note**
>
> 記得登入！如果您使用 IAM 身分中心進行驗證，請記得使用 AWS CLI `aws sso login`命令登入。

1. 執行 `node index.js`。
2. 選擇是否要清空並刪除值區。
3. 如果您沒有刪除值區，請務必手動清空並在稍後刪除。

在瀏覽器中開始使用

本節將逐步說明如何 JavaScript 在瀏覽器中執行 SDK 的第 3 版 (V3) 範例。

> **Note**
>
> 在瀏覽器中運行 V3 與版本 2（V2）略有不同。如需詳細資訊，請參閱 在 V3 中使用瀏覽器。

如需使用 SDK (V3) 的其他範例 JavaScript，請參閱SDK對於 JavaScript （v3）代碼示例。

此 Web 應用程式範例顯示：

- 如何使用 Amazon Cognito 進行身份驗證訪問 AWS 服務。
- 如何使用 (IAM) 角色讀取亞馬遜簡單儲存服務 (Amazon S3) 儲存貯體中的物件清單。 AWS Identity and Access Management

> **Note**
>
> 此範例不會用 AWS IAM Identity Center 於驗證。

使用案例

Amazon S3 是一項物件儲存服務，提供領先業界的可擴展性、資料可用性、安全性和效能。您可以使用 Amazon S3 將資料作為物件存放在稱為儲存貯體的容器中。如需有關 Amazon S3 的詳細資訊，請參閱 Amazon S3 使用者指南。

此範例說明如何設定和執行假設 IAM 角色以從 Amazon S3 儲存貯體讀取的 Web 應用程式。該示例使用 React 前端庫和 Vite 前端工具來提供 JavaScript 開發環境。Web 應用程式使用 Amazon Cognito 身分集區來提供存取 AWS 服務所需的登入資料。隨附的程式碼範例會示範 JavaScript 在 Web 應用程式中載入和使用 SDK 的基本模式。

步驟 1：建立 Amazon Cognito 身分集區和 IAM 角色

在本練習中，您會建立並使用 Amazon Cognito 身分集區，為 Amazon S3 服務提供對 Web 應用程式的未經驗證存取。建立身分集區也會建立 AWS Identity and Access Management (IAM) 角色來支援未經驗證的來賓使用者。在此範例中，我們只會使用未經驗證的使用者角色來保持工作的焦點。您之後可以整合對身分提供者和已驗證使用者的支援。如需有關新增 Amazon Cognito 身分識別集區的詳細資訊，請參閱 Amazon Cognito 開發人員指南中的教學課程：建立身分集區。

若要建立 Amazon Cognito 身分集區和相關聯的 IAM 角色

1. 登錄到 AWS Management Console 並打開 Amazon Cognito 控制台 https://console.aws.amazon.com/cognito/.
2. 在左側導覽窗格中，選擇 [識別集區]。
3. 選擇 建立身分池。
4. 在 [設定身分識別集區信任] 中，選擇 [來賓存取] 進行使用
5. 在 [設定權限] 中，選擇 [建立新的 IAM 角色]，然後在 IAM 角色名稱中輸入名稱 (例如 getStartedRole)。

6. 在 [設定內容] 中，在 [識別集區名稱] 中輸入名稱 (例如 get StartedPool)。
7. 在 檢閱和建立 中，確認您為新身分池所做的選擇。選取 編輯 以返回精靈並變更任何設定。當您完成時，請選取 建立身分池。
8. 請記下身分集區 ID 和新建立的 Amazon Cognito 身分識別集區的區域。################## 步驟 4：設定瀏覽器程式碼

建立 Amazon Cognito 身分集區之後，您就可以為 Web 應用程式所需的 Amazon S3 新增許可。

步驟 2：將政策新增至建立的 IAM 角色

若要在 Web 應用程式中啟用對 Amazon S3 儲存貯體的存取權，請使用為 Amazon Cognito 身分集區建立的未經驗證 IAM 角色 (例如 get StartedRole StartedPool)。這需要您將 IAM 政策附加到該角色。如需有關修改 IAM 角色的詳細資訊，請參閱《IAM 使用者指南》中的修改角色許可政策。

將 Amazon S3 政策新增至與未驗證使用者相關聯的 IAM 角色

1. 登入 AWS Management Console 並開啟身分與存取權管理主控台，網址為 https://console.aws.amazon.com/iam/。
2. 在左側導覽窗格中，選擇 Roles (角色)。
3. 選擇您要修改的角色名稱 (例如，get StartedRole)，然後選擇 [權限] 索引標籤。
4. 選擇 [新增權限]，然後選擇 [附加原則]
5. 在此角色的 [新增權限] 頁面中，尋找並選取 AmazonS3 ReadOnly Access 的核取方塊。

> **Note**
>
> 您可以使用此過程來啟用對任何 AWS 服務的訪問。

6. 選擇新增許可。

在您建立 Amazon Cognito 身分集區，並將 Amazon S3 的許可新增至未經驗證使用者的 IAM 角色後，您就可以準備新增和設定 Amazon S3 儲存貯體。

步驟 3：添加一個 Amazon S3 存儲桶和對象

在此步驟中，您將為範例新增 Amazon S3 儲存貯體和物件。您也將為值區啟用跨來源資源共用 (CORS)。如需有關建立 Amazon S3 儲存貯體和物件的詳細資訊，請參閱 Amazon S3 使用者指南中的開始使用 Amazon S3。

使用 CORS 添加 Amazon S3 存儲桶和對象

1. 登入 AWS Management Console 並開啟 Amazon S3 主控台，網址為 https://console.aws.amazon.com/s3/。
2. 在左側導覽窗格中，選擇「值區」，然後選擇「建立值區」。
3. 輸入符合值區命名規則的值區名稱 (例如 getstartedbucket)，然後選擇「建立值區」。
4. 選擇您建立的值區，然後選擇 [物件] 索引標籤。然後選擇 Upload (上傳)。
5. 在檔案和資料夾下，選擇新增檔案。
6. 選擇要上傳的檔案，然後選擇 Open (開啟)。然後選擇「上傳」以完成將物件上傳至值區。
7. 接下來，選擇值區的 [權限] 索引標籤，然後在 [跨來源資源共用 (CORS)] 區段中選擇 [編輯]。輸入下列 JSON：

```
[
  {
    "AllowedHeaders": [
      "*"
    ],
    "AllowedMethods": [
      "GET"
    ],
    "AllowedOrigins": [
      "*"
    ],
    "ExposeHeaders": []
  }
]
```

8. 選擇儲存變更。

新增 Amazon S3 儲存貯體並新增物件之後，就可以設定瀏覽器程式碼了。

步驟 4：設定瀏覽器程式碼

範例應用程式包含單頁 React 應用程式。您可以在這裡找到此範例的檔案 GitHub。

若要設定範例應用程式

1. 安裝 Node.js。
2. 從命令行中，克隆AWS 代碼示例存儲庫：

```
git clone --depth 1 https://github.com/awsdocs/aws-doc-sdk-examples.git
```

3. 導覽至範例應用程式：

```
cd aws-doc-sdk-examples/javascriptv3/example_code/web/s3/list-objects/
```

4. 執行下列命令以安裝所需的套件：

```
npm install
```

5. 接下來，`src/App.tsx`在文本編輯器中打開並完成以下操作：

   - 將*########################## ID*。步驟 1：建立 Amazon Cognito 身分集區和 IAM 角色
   - 將區域的值取代為指派給 Amazon S3 儲存貯體和 Amazon Cognito 身分識別集區的區域。請注意，這兩個服務的區域必須相同 (例如 us-east- 2)。
   - 將值區*#########*立的值區名稱。步驟 3：添加一個 Amazon S3 存儲桶和對象

取代文字後，儲存檔`App.tsx`案。您現在已準備好執行 Web 應用程式。

步驟 5：執行範例

執行範例應用程式

1. 從命令列導覽至範例應用程式：

```
cd aws-doc-sdk-examples/javascriptv3/example_code/web/s3/list-objects/
```

2. 從命令列執行下列命令：

```
npm run dev
```

Vite 開發環境將運行並顯示以下消息：

```
VITE v4.3.9  ready in 280 ms

#  Local:   http://localhost:5173/
#  Network: use --host to expose
#  press h to show help
```

3. 在您的網頁瀏覽器中，瀏覽至上面顯示的網址 (例如 http://localhost:5173)。範例應用程式會顯示 Amazon S3 儲存貯體中的物件檔案名稱清單。

清除

若要清理您在本教學課程中建立的資源，請執行下列操作：

- 在 Amazon S3 主控台中，刪除任何物件和建立的任何儲存貯體 (例如，getStartted bucket)。
- 在 IAM 主控台中，刪除角色名稱 (例如 get StartedRole)。
- 在 Amazon Cognito 主控台中，刪除身分識別集區名稱 (例如 get StartedPool)。

設定下列項目的 SDK JavaScript

本節中的主題說明如何安裝和載入 SDK，以 JavaScript 便您可以存取 SDK 支援的 Web 服務。

> **Note**
>
> React 原生開發人員應該使用 AWS Amplify 在 AWS. 有關詳細信息，請參見aws-sdk-react-native存檔。

主題

必要條件

在您的伺服器上安裝 Node.js (如果尚未安裝)。

主題

設定一個 AWS Node.js 環境

若要設定您可以在其中執行應用程式的 AWS Node.js 環境，請使用下列任一方法：

- 選擇一個預先安裝 Node.js 的 Amazon 計算機映像（ AMI ）。然後使用該 AMI 創建一個 Amazon EC2 實例。建立您的亞馬遜 EC2 執行個體 AMI，請從 AWS Marketplace. 搜 AWS Marketplace 尋 Node.js 並選擇包含預先安裝的 Node.js 版本 (32 位元或 64 位元) 的 AMI 選項。
- 創建一個 Amazon EC2 實例並在其上安裝 Node.js。如需如何在 Amazon Linux 執行個體上安裝 Node.js 的詳細資訊，請參閱在 Amazon EC2 實例上設置 Node.js。
- 建立一個無伺服器環境，使 AWS Lambda 用將 Node.js 做為 Lambda 函數執行。如需在 Lambda 函數中使用 Node.js 的詳細資訊，請參閱AWS Lambda 開發人員指南中的程式設計模型 (Node.js)。

- 將您的 Node.js 應用程式部署到 AWS Elastic Beanstalk. 如需有關將 Node.js 與 Elastic Beanstalk 搭配使用的詳細資訊，請參閱AWS Elastic Beanstalk 開發人員指南 AWS Elastic Beanstalk中的將 Node.js 應用程式部署至。
- 使用建立 Node.js 應用程式伺服器 AWS OpsWorks。如需搭配使用 Node.js 的詳細資訊 AWS OpsWorks，請參閱《使用AWS OpsWorks 者指南》中的「建立您的第一個 Node.js 堆疊」。

支援網頁瀏覽器

AWS SDK for JavaScript 支援所有現代網頁瀏覽器。

在版本 3.183.0 或更新版本中，適 JavaScript 用的 SDK 使用 ES2020 加工品，支援下列最低版本。

瀏覽器	版本
Google Chrome	80.0+
Mozilla Firefox	80.0+
Opera	63.0+
Microsoft Edge	80.0+
Apple Safari	14.1+
Samsung Internet	12.0+

在版本 3.182.0 或更早版本中，的 SDK JavaScript 使用 ES5 加工品，支援以下最低版本。

瀏覽器	版本
Google Chrome	49.0+
Mozilla Firefox	45.0+
Opera	36.0+
Microsoft Edge	12.0+
Windows Internet Explorer	N/A

瀏覽器	版本
Apple Safari	9.0+
Android 瀏覽器	76.0+
UC 瀏覽器	12 歲以上
Samsung Internet	5.0 以上

> **Note**
>
> 諸如此類的框架 AWS Amplify 可能不會提供與 JavaScript. 如需詳細資訊，請參閱AWS Amplify 文件。

為下列項目安裝 SDK JavaScript

並非所有服務都可立即在 SDK 或所有 AWS 區域中使用。

要從 AWS SDK for JavaScript 使用 npm，Node.js 軟件包管理器安裝服務，請在命令提示符下輸入以下命令，其中 SERVICE 是#務的名稱，例如s3。

```
npm install @aws-sdk/client-SERVICE
```

如需 AWS SDK for JavaScript 服務用戶端套件的完整清單，請參閱 AWS SDK for JavaScript API 參考指南。

載入下列項目的 SDK JavaScript

安裝 SDK 之後，您可以使用import. 例如，若要載入 Amazon S3 用戶端和 Amazon S3 ListBuckets命令，請使用下列指令。

```
import { S3Client, ListBucketsCommand } from "@aws-sdk/client-s3";
```

配SDK置 JavaScript

使用 SDK for JavaScript 來叫用 Web 服務之前API，必須先設定SDK. 您至少必須設定：

- 您要求服務的 AWS 地區
- 您的代碼如何進行身份驗證 AWS

除了這些設定之外，您可能還必須設定 AWS 資源的權限。例如，您可以限制對 Amazon S3 儲存貯體的存取，或限制 Amazon DynamoDB 表格以進行唯讀存取。

《AWS SDKs和工具參考指南》還包含許多設定、功能和其他常見的基礎概念。 AWS SDKs

本節中的主題說明SDK為 JavaScript Node.js 設定並在網頁瀏覽器中 JavaScript 執行的方法。

主題

每項服務的組態

您可以將配置資訊傳遞給服務物件來配置。SDK

服務層級組態可提供對個別服務的重要控制，讓您在需求與預設組態不同時更新個別服務物件的組態。

> **Note**
>
> 在 2.x 版中， AWS SDK for JavaScript 服務配置可以傳遞給單個客戶端構造函數。不過，這些組態會先自動合併到全域SDK組態的副本中`AWS.config`。
> 此外，在進行更新呼叫之後，`AWS.config.update({/* params */})`僅呼叫實例化服務用戶端的更新配置，而不是任何現有的客戶端。
> 這種行為是經常造成混淆的來源，因此很難將設定新增至只會以前向相容方式影響服務用戶端子集的全域物件。在版本 3 中，不再有由SDK. 配置必須傳遞給實例化的每個服務客戶端。您仍然可以在多個用戶端之間共用相同的組態，但該組態不會自動與全域狀態合併。

設定每個服務的組態

您在中使用的每個服務都可透過屬SDK於 JavaScript 該服務之一部分的API服務物件存取。例如，若要存取 Amazon S3 服務，您需要建立 Amazon S3 服務物件。您可以指定組態設定，該設定是專屬於該服務物件之建構子的服務。

例如，如果您需要存取多個區 AWS 域中的 Amazon EC2 物件，請為每個區域建立 Amazon EC2 服務物件，然後相應地設定每個服務物件的區域組態。

```
var ec2_regionA = new EC2({region: 'ap-southeast-2', maxAttempts: 15});
var ec2_regionB = new EC2({region: 'us-west-2', maxAttempts: 15});
```

設定 AWS 區域

一個 AWS 區域是同一地理區域中的一組具名 AWS 資源。區域的範例為`us-east-1`美國東部 (維吉尼亞北部) 區域。您可以在中建立服務用戶端時指定區域， JavaScript 以便SDK存取該區域中的服務。SDK某些 服務僅在特定區域提供。

for SDK 預設 JavaScript 不會選取 [地區]。但是，您可以使用環境變數或共用組態`config`檔案來設定 AWS Region。

在客戶端類構造函數中

實例化服務對象時，可以指定該資源的 AWS Region 作為客戶端類構造函數的一部分，如下所示。

```
const s3Client = new S3.S3Client({region: 'us-west-2'});
```

使用環境變數

您可以使用 `AWS_REGION` 環境變數來設定區域。如果您定義此變數，SDKfor 會 JavaScript 讀取並使用它。

使用共用設定檔

就像共享憑據文件允許您存儲憑據以供使用SDK，您可以SDK將 AWS Region 和其他配置設置保存在名`config`為以供使用的共享文件中。如果`AWS_SDK_LOAD_CONFIG`環境變數設定為真值，則在載入`config`檔案時，SDKfor JavaScript 會自動搜尋檔案。`config` 檔案的儲存位置取決於您的作業系統：

- Linux、macOS 系統或 Unix 用戶-`~/.aws/config`
- 視窗使用者-`C:\Users\USER_NAME\.aws\config`

如果您還沒有共用 `config` 檔案，您可以在指定的目錄中建立一個。在下列範例中，`config` 檔案會同時設定區域和輸出格式。

```
[default]
   region=us-west-2
   output=json
```

如需有關使用共用`config`和`credentials`檔案的詳細資訊，請參閱《工具參考指南》中的共用設定 AWS SDKs和認證檔案。

設定區域的優先順序

以下是「區域」設定的優先順序：

1. 如將某區域傳遞至用戶端類別建構子，則會使用該區域。
2. 如果在環境變數中設定了「區域」，則會使用該「區域」。
3. 否則，會使用共用設定檔中定義的「區域」。

設定認證

AWS 使用認證來識別誰正在呼叫服務，以及是否允許存取要求的資源。

無論是在網頁瀏覽器或 Node.js 伺服器中執行，您的 JavaScript 程式碼都必須取得有效的認證，才能透過API. 透過將認證直接傳遞給服務物件，即可為每個服務設定認證。

有幾種方法可以設置 Node.js 和 Web 瀏覽器之間不同 JavaScript的憑據。本節的主題說明如何在 Node.js 或 Web 瀏覽器設定登入資料。在每個案例中，選項會以建議的順序呈現。

認證的最佳做法

適當設定登入資料可確保您的應用程式或瀏覽器指令碼可以存取所需的服務與資源，同時將可能影響關鍵任務應用程式或洩漏敏感資料的安全性問題的接觸降到最低。

在特定登入資料時要套用的重要原則，即是一律授予任務所需的最低權限。提供資源的最低許可並視需要新增進一步許可是較安全的作法，而不是提供超過最低權限的許可，而造成您必須修復在稍後可能發

現的安全性問題。例如，除非您需要讀取和寫入個別資源 (例如 Amazon S3 儲存貯體或 DynamoDB 表格中的物件)，否則請將這些許可設定為唯讀。

如需有關授與最低權限的詳細資訊，請參閱《IAM使用指南》中的最佳作法主題的 < 授與最低權限 > 一節。

主題

- 在 Node.js 中設定認證
- 在網頁瀏覽器中設定認證

在 Node.js 中設定認證

我們建議新用戶誰是在本地開發，並沒有給他們的雇主進行身份驗證的方法來設置 AWS IAM Identity Center。如需詳細資訊，請參閱 使用 SDK 驗證功能 AWS。

在 Node.js 中將登入資料提供給軟體開發套件有幾種方法。有些方法比較安全，有些在開發應用程式時更方便。在 Node.js 中取得認證時，請小心仰賴多個來源，例如您載入的環境變數和 JSON 檔案。您可以變更程式碼執行所用的許可，而不需了解發生的變更。

AWS SDK for JavaScript V3 在 Node.js 中提供了預設的憑證提供者鏈結，因此您不需要明確提供憑證提供者。預設認證提供者鏈結會嘗試以指定優先順序從各種不同來源解析認證，直到從其中一個來源傳回認證為止。您可以在此處找到 JavaScript V3 適用於 SDK 的憑證提供者鏈。

憑證提供者鏈

所有 SDK 都有一系列位置（或來源），他們檢查這些位置（或來源），以獲取有效的憑據以用於向. AWS 服務找到有效的憑證後，系統就會停止搜尋。此系統搜尋稱為預設認證提供者鏈結。

對於鏈中的每個步驟，都有不同的方法可以設定值。直接在代碼中設置值始終具有優先級，然後設置為環境變量，然後在共享 AWS `config`文件中設置。如需詳細資訊，請參閱 AWS SDK 和工具參考指南中的設定優先順序。

AWS SDK 和工具參考指南包含所有 AWS SDK 和. AWS CLI要了解有關如何通過共 AWS `config`享文件配置 SDK 的更多信息，請參閱共享配置和憑據文件。若要深入瞭解如何透過設定環境變數來設定 SDK，請參閱環境變數支援。

若要使用進行驗證 AWS，會依照下表中列出的順序 AWS SDK for JavaScript 檢查認證提供者。

AWS SDK for JavaScript API 依優先順序參考憑證提供者方法	可用的憑證提供者	AWS SDK 和工具參考指南
fromEnv()	AWS 來自環境變量的訪問鍵	AWS 存取金鑰
fromSSO()	AWS IAM Identity Center。 在本指南中，請參閱使用 SDK 驗證功能 AWS。	IAM 身分中心憑證提供者
fromIni()	AWS 來自共用config和credentials 檔案的存取金鑰	AWS 存取金鑰
	信任的實體提供者 (例如AWS_ROLE_ARN)	假設存取權管理角色
	來自 AWS Security Token Service （ AWS STS ）的 Web 身份令牌	與網絡身份或 OpenID Connect 聯盟
	Amazon Elastic Container Service (Amazon ECS) 登入資料	容器認證提供者
	亞馬遜彈性運算雲端 (Amazon EC2) 執行個體設定檔登入資料 (IMDS 登入資料提供者)	IMDS 認證提供者
	程序認證提供者	程序認證提供者
	AWS IAM Identity Center 認證	IAM 身分中心憑證提供者
fromProcess()	程序認證提供者	程序認證提供者
fromTokenFile()	來自 AWS Security Token Service （ AWS STS ）的 Web 身份令牌	與網絡身份或 OpenID Connect 聯盟

AWS SDK for JavaScript API 依優先順序參考憑證提供者方法	可用的憑證提供者	AWS SDK 和工具參考指南
fromContainerMetadata()	Amazon Elastic Container Service (Amazon ECS) 登入資料	容器認證提供者
fromInstanceMetadata()	亞馬遜彈性運算雲端 (Amazon EC2) 執行個體設定檔登入資料 (IMDS 登入資料提供者)	IMDS 認證提供者

如果您遵循建議的方法讓新使用者開始使用，您可以在 [開始使用] 主題期間使用 SDK 驗證功能 AWS設定 AWS IAM Identity Center 驗證。其他驗證方法在不同的情況下很有用。為了避免安全風險，我們建議您始終使用短期憑證。有關其他身份驗證方法程序，請參閱 AWS SDK 和工具參考指南中的身份驗證和訪問。

本節的主題說明如何在 Node.js 中載入登入資料。

主題

- 從適用於 Amazon EC2 的 IAM 角色載入 Node.js 中的登入資料
- 載入 Node.js 函數的 Lambda 證

從適用於 Amazon EC2 的 IAM 角色載入 Node.js 中的登入資料

如果您在 Amazon EC2 執行個體上執行 Node.js 應用程式，您可以利用適用於 Amazon EC2 的 IAM 角色，自動為執行個體提供登入資料。如果您將執行個體設定為使用 IAM 角色，SDK 會自動為您的應用程式選取 IAM 登入資料，無需手動提供登入資料。

如需將 IAM 角色新增至 Amazon EC2 執行個體的詳細資訊，請參閱適用於 Amazon EC2 的 IAM 角色。

載入 Node.js 函數的 Lambda 證

建立 AWS Lambda 函數時，您必須建立具有執行函數之權限的特殊 IAM 角色。此角色稱為執行角色。設定 Lambda 函數時，您必須將建立的 IAM 角色指定為對應的執行角色。

執行角色為 Lambda 函數提供執行和叫用其他 Web 服務所需的認證。因此，您不需要為您在 Lambda 函數中撰寫的 Node.js 程式碼提供認證。

如需建立 Lambda 執行角色的詳細資訊，請參閱AWS Lambda 開發人員指南中的管理權限：使用 IAM 角色 (執行角色)。

在網頁瀏覽器中設定認證

透過瀏覽器指令碼將登入資料提供給軟體開發套件有幾種方法。有些方法比較安全，有些在開發指令碼時更方便。

以下是您可以按照建議順序提供憑證的方法：

1. 使用 Amazon Cognito 身分驗證使用者並提供登入資料
2. 使用 Web 聯合身分

> **Warning**
>
> 我們不建議您在指令碼中對 AWS 認證進行硬式編碼。將登入資料寫死會造成存取金鑰 ID 和私密存取金鑰遭暴露的風險。

主題

- 使用 Amazon Cognito 身份驗證用戶

使用 Amazon Cognito 身份驗證用戶

取得瀏覽器指令碼 AWS 登入資料的建議方式是使用 Amazon Cognito 身分登入資料用戶端`CognitoIdentityClient`。Amazon Cognito 透過第三方身分供應商啟用使用者身分驗證。

若要使用 Amazon Cognito 身分識別，您必須先在 Amazon Cognito 主控台中建立身分集區。身分集區代表應用程式提供給使用者的身分群組。提供給使用者的身分可唯一識別每個使用者帳戶。Amazon Cognito 身分不是憑證。它們會使用 AWS Security Token Service (AWS STS) 中的 Web 身分同盟支援來交換憑證。

Amazon Cognito 可協助您管理多個身分識別供應商之間的身分抽象化。接著，會將載入的身分交換為在 AWS STS中的登入資料。

設定 Amazon Cognito 身分登入資料物件

如果您尚未建立身分集區，請在設定 Amazon Cognito 用戶端之前，先在 Amazon Cognito 主控台中建立要與瀏覽器指令碼搭配使用的身分集區。為您的身分集區建立並關聯已驗證和未驗證的 IAM 角色。如需詳細資訊，請參閱 Amazon Cognito 開發人員指南中的教學課程：建立身分集區。

未經身份驗證的用戶沒有驗證其身份，使此角色適用於您的應用程序的來賓用戶，或者在用戶是否已驗證其身份時無關緊要。已驗證使用者透過驗證其身分的第三方身分提供者來登入應用程式。請務必適當地限制資源許可範圍，以避免從未經授權的使用者授與資源的存取權。

設定身分識別集區後，請使用中的`fromCognitoIdentityPool@aws-sdk/credential-providers`方法從識別集區擷取認證。在下列建立 Amazon S3 用戶端的範例中，請將 *AW_REGION ####*，並以身分集區識別碼取代 *Identity_POOL_* ID。

```
// Import required AWS SDK clients and command for Node.js
import {S3Client} from "@aws-sdk/client-s3";
import {fromCognitoIdentityPool} from "@aws-sdk/credential-providers";

const REGION = AWS_REGION;

const s3Client = new S3Client({
  region: REGION,
  credentials: fromCognitoIdentityPool({
    clientConfig: { region: REGION }, // Configure the underlying
 CognitoIdentityClient.
    identityPoolId: 'IDENTITY_POOL_ID',
    logins: {
            // Optional tokens, used for authenticated login.
        },
  })
});
```

選用的 `logins` 屬性是身分提供者名稱與這些身分提供者的身分權杖的對應。您從身分提供者取得權杖的方式，取決於您使用的供應商。例如，如果您使用 Amazon Cognito 使用者集區做為身份驗證提供者，則可以使用類似以下方法的方法。

```
// Get the Amazon Cognito ID token for the user. 'getToken()' below.
let idToken = getToken();
let COGNITO_ID = "COGNITO_ID"; // 'COGNITO_ID' has the format 'cognito-
idp.REGION.amazonaws.com/COGNITO_USER_POOL_ID'
```

```
let loginData = {
  [COGNITO_ID]: idToken,
};
const s3Client = new S3Client({
    region: REGION,
    credentials: fromCognitoIdentityPool({
    clientConfig: { region: REGION }, // Configure the underlying
 CognitoIdentityClient.
    identityPoolId: 'IDENTITY_POOL_ID',
    logins: loginData
  })
});

// Strips the token ID from the URL after authentication.
window.getToken = function () {
  var idtoken = window.location.href;
  var idtoken1 = idtoken.split("=")[1];
  var idtoken2 = idtoken1.split("&")[0];
  var idtoken3 = idtoken2.split("&")[0];
  return idtoken3;
};
```

將未經驗證的使用者切換至已驗證

Amazon Cognito 同時支援已驗證和未經驗證的使用者。未驗證的使用者即使沒有以任何身分提供者登入，也能存取您的資源。這個程度的存取能在使用者登入前就顯示內容，非常有用。每個未經驗證的使用者在 Amazon Cognito 中都有一個唯一的身分，即使他們尚未經個別登入和驗證。

最初未驗證的使用者

使用者通常會以未經驗證的角色開始，您會為該角色設定組態物件的登入資料屬性，而不使用 `logins` 屬性。在此情況下，您的預設認證可能如下所示：

```
// Import the required AWS SDK for JavaScript v3 modules.
import {fromCognitoIdentityPool} from "@aws-sdk/credential-providers";
// Set the default credentials.
const creds = fromCognitoIdentityPool({
  identityPoolId: 'IDENTITY_POOL_ID',
  clientConfig: { region: REGION } // Configure the underlying CognitoIdentityClient.
});
```

切換到已驗證使用者

當未驗證的使用者登入身分識別提供者且您有權杖時，您可以呼叫會更新認證物件並新增權杖的自訂函數，將使用者從未驗證切換為已驗證。`logins`

```
// Called when an identity provider has a token for a logged in user
function userLoggedIn(providerName, token) {
  creds.params.Logins = creds.params.logins || {};
  creds.params.Logins[providerName] = token;

  // Expire credentials to refresh them on the next request
  creds.expired = true;
}
```

Node.js 考量事項

雖然 Node.js 程式碼是 JavaScript，使用 Node.js AWS SDK for JavaScript 中的程式碼可能與在瀏覽器指令碼SDK中使用不同。某些API方法在 Node.js 中工作，但不適用於瀏覽器腳本以及其他方式。成功使用一些APIs取決於您對常見 Node.js 編碼模式的熟悉程度，例如導入和使用其他 Node.js 模塊（如`File System (fs)`模塊）。

使用內建的 Node.js 模組

Node.js 會提供您可以使用而不需安裝的內建模組集合。若要使用這些模組，請使用 `require` 方法建立物件來指定模組名稱。例如，若要包含內建HTTP模組，請使用下列指令。

```
import http from 'http';
```

呼叫模組方法 (就好像是該物件的方法)。例如，這裡是讀取HTML文件的代碼。

```
// include File System module
import fs from "fs";
// Invoke readFile method
fs.readFile('index.html', function(err, data) {
  if (err) {
    throw err;
  } else {
    // Successful file read
  }
});
```

如需 Node.js 提供的所有內建模組的完整清單，請參閱 Node.js 網站上的 Node.js 文件。

使用 npm 套件

除了內建模組之外，您還可以從 npm Node.js 套件管理員中包含和合併第三方程式碼。這是開放原始碼 Node.js 套件的儲存庫以及用來安裝那些套件的命令列界面。如需目前可用套件的詳細資訊npm和清單，請參閱 https://www.npmjs.com。您也可以在此處瞭解其他 Node.js 套件 GitHub。

maxSockets 在 Node.js 中進行設定

您可以在 Node.js 中，依來源設定連線數量上限。如果設 `maxSockets`定，低階用HTTP戶端會排入佇列要求，並在通訊端可用時將它們指派給通訊端。

這可讓您隨時為指定來源的並行請求數設定上限。降低此值可能會減少調節量或接收的逾時錯誤數。然而，這也可能增加記憶體使用量，因為請求在通訊端可用前都會排在佇列中。

下列範例顯示如何為 DynamoDB `maxSockets` 用戶端設定。

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { NodeHttpHandler } from "@smithy/node-http-handler";
import https from "https";
let agent = new https.Agent({
  maxSockets: 25
});

let dynamodbClient = new DynamoDBClient({
  requestHandler: new NodeHttpHandler({
    requestTimeout: 3_000,
    httpsAgent: agent
  });
});
```

如果您未提供`maxSockets`值或`Agent`物件，for JavaScript 會使用值 50。SDK如果你提供一個`Agent`對象，它的`maxSockets`值將被使用。如需 Node.js `maxSockets` 中設定的詳細資訊，請參閱 Node.js 說明文件。

從的 v3.521.0 開始 AWS SDK for JavaScript，您可以使用下列速記語法來設定。`requestHandler`

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({
  requestHandler: {
```

```
    requestTimeout: 3_000,
    httpsAgent: { maxSockets: 25 },
  },
});
```

在 Node.js 中重複使用保持活動狀態的連接

預設的 Node.jsHTTP/HTTPS代理程式會為每個新要求建立新TCP連線。為了避免建立新連線的成本，預設會 AWS SDK for JavaScript 重複使用TCP連線。

對於短期操作 (例如 Amazon DynamoDB 查詢)，設定TCP連線的延遲開銷可能會大於作業本身。此外，由於靜態 DynamoDB 加密已與整合 AWS KMS，因此您可能會遇到資料庫的延遲，必須為每個作業重新建立新的 AWS KMS 快取項目。

如果您不想重複使用TCP連線，您可以在每個服務用戶端`keepAlive`上停用這些連線，如下列 DynamoDB 用戶端範例所示。

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { NodeHttpHandler } from "@smithy/node-http-handler";
import { Agent } from "https";

const dynamodbClient = new DynamoDBClient({
    requestHandler: new NodeHttpHandler({
        httpsAgent: new Agent({ keepAlive: false })
    })
});
```

如果啟`keepAlive`用，您也可以設定 TCP Keep-Alive 封包的初始延遲`keepAliveMsecs`，預設為 1000 ms。請參閱 Node.js 文件了解詳細資訊。

設定 Node.js 的代理伺服器

如果您無法直接連線到網際網路，則SDK支 JavaScript 援透過第三方代理HTTP程式使用HTTP或HTTPS Proxy。

若要尋找第三方HTTP代理程式，請在 npm 搜尋「HTTP代理」。

若要安裝協力廠商HTTP代理程式 Proxy，請在命令提示字元中輸入下列命令，其中 *PROXY* 是npm套件的名稱。

```
npm install PROXY --save
```

若要在應用程式中使用 Proxy，請針對 DynamoDB 用戶端使用 `httpAgent` and `httpsAgent` 屬性，如下列範例所示。

```
import { DynamoDBClient } from '@aws-sdk/client-dynamodb';
import { NodeHttpHandler } from "@smithy/node-http-handler";
import { HttpsProxyAgent } from "hpagent";
const agent = new HttpsProxyAgent({ proxy: "http://internal.proxy.com" });
const dynamodbClient = new DynamoDBClient({
    requestHandler: new NodeHttpHandler({
        httpAgent: agent,
        httpsAgent: agent
    }),
});
```

> **Note**
>
> `httpAgent`與不相同`httpsAgent`，並且由於來自客戶端的大多數呼叫都應該設置。`https`

在 Node.js 中註冊憑證組合

Node.js 的預設信任存放區包括存取 AWS 服務所需的憑證。在某些案例中，建議您僅包含特定一組憑證。

在此範例中，磁碟上特定憑證會用來建立拒絕連線的 `https.Agent` (除非提供指定的憑證)。然後，DynamoDB 用戶端會使用新建立`https.Agent`的。

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { NodeHttpHandler } from "@smithy/node-http-handler";
import { Agent } from "https";
import { readFileSync } from "fs";
const certs = [readFileSync("/path/to/cert.pem")];
const agent = new Agent({
  rejectUnauthorized: true,
  ca: certs
});
const dynamodbClient = new DynamoDBClient({
  requestHandler: new NodeHttpHandler({
    httpAgent: agent,
    httpsAgent: agent
  })
```

```
});
```

瀏覽器指令碼考量

下列主題說明 AWS SDK for JavaScript 在瀏覽器指令碼中使用的特殊考量事項。

主題

- SDK為瀏覽器構建
- 跨來源資源共享 (CORS)
- 捆綁應用程序與網絡包

SDK為瀏覽器構建

與SDK JavaScript 版本 2 (V2) 不同，V3 不會作為包含預設服務集支援的 JavaScript 檔案提供。相反，V3 使您可以將所需 JavaScript 文件捆綁並包含在瀏覽器中，從而減少開銷。SDK我們建議您使用 Webpack 將所SDK需的檔案以及您需要的任何其他第三方套 JavaScript 件捆綁到單一`Javascript`檔案中，然後使用`<script>`標籤將其載入瀏覽器指令碼中。如需 Webpack 的詳細資訊，請參閱捆綁應用程序與網絡包。如需使用 Webpack 將 V3 載 JavaScript 入至瀏覽器的範SDK例，請參閱建置應用程式以將資料提交至 DynamoDB。

如果您在瀏覽器CORS中強制執行的環境SDK外部工作，並且想要存取由 SDK for 提供的所有服務 JavaScript，則可以複製存放庫並執行建置預設託管版本的相同建置工具的本SDK機自訂副本。SDK以下各節描述了SDK使用額外服務和API版本構建的步驟。

使用SDK生成器來SDK構建 JavaScript

> **Note**
>
> Amazon Web Services 版本 3（V3）不再支持瀏覽器生成器。為了盡量減少瀏覽器應用程式的頻寬使用量，建議您匯入具名模組，然後將它們捆綁起來以減少大小。如需有關捆綁的更多資訊，請參閱捆綁應用程序與網絡包。

跨來源資源共享 (CORS)

跨來源資源分享 (CORS) 是現代 Web 瀏覽器的一項安全功能，其可讓 Web 瀏覽器協調能請求外部網站或服務的網域。

由於系統會將大多數資源請求傳送至外部網域 (如 Web 服務的端點)，當您使用 AWS SDK for JavaScript 開發瀏覽器應用程式時，CORS 是項重要的考量條件。如果您的 JavaScript 環境強制執行 CORS 安全性，則必須使用服務配置 CORS。

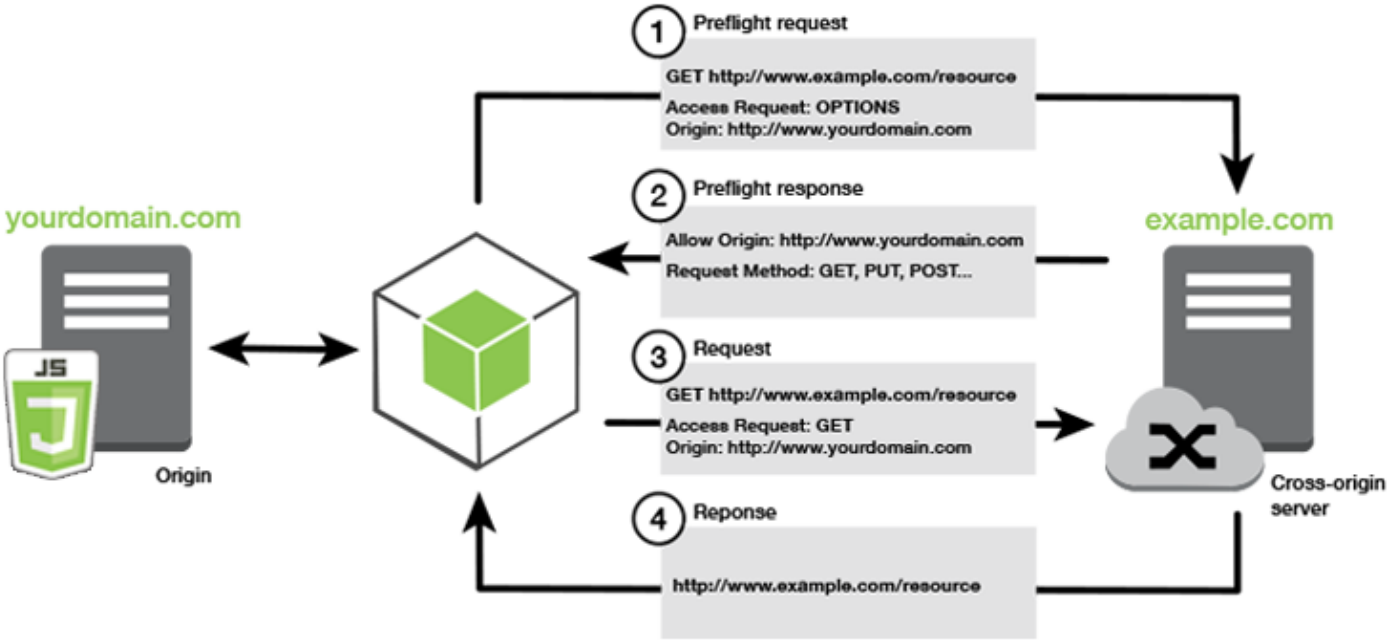
CORS 會根據下列項目判斷是否允許跨來源要求中的資源共用：

- 提出請求的特定網域
- 提出的 HTTP 請求類型 (GET、PUT、POST、DELETE 等)

CORS 的工作原理

在最簡單的案例中，瀏覽器指令碼會從另一個網域中的伺服器發出資源 GET 請求。依據該伺服器的 CORS 組態而定，如果請求是來自有權提交 GET 請求的網域，則跨來源伺服器會透過傳回請求的資源來予以回應。

若發出請求的網域或 HTTP 請求類型皆未經授權，該請求即會遭拒。然而，CORS 能夠在實際提交請求前進行預檢。此案例中，會發出預檢請求，此請求中會一併傳送 OPTIONS 存取請求操作。如果跨來源伺服器的 CORS 組態將存取權限授予給提出請求的網域，伺服器就會傳回預檢回應，其中列出提出請求的網域能對請求資源所進行的 HTTP 請求類型。



是否需要 CORS 設定？

Amazon S3 儲存貯體需要 CORS 組態，才能對它們執行操作。在某些 JavaScript 環境中，CORS 可能不會強制執行，因此不需要配置 CORS。例如，如果您從 Amazon S3 儲存貯體託管應用程式，並從`*.s3.amazonaws.com`或某些其他特定端點存取資源，則您的請求將無法存取外部網域。因此，這個組態就不需要 CORS。在這種情況下，CORS 仍然用於 Amazon S3 以外的服務。

為 Amazon S3 儲存貯體設定 CORS

您可以將 Amazon S3 儲存貯體設定為在 Amazon S3 主控台中使用 CORS。

如果您要在 AWS Web 服務管理主控台中設定 CORS，則必須使用 JSON 來建立 CORS 組態。新的 AWS Web 服務管理主控台僅支援 JSON CORS 設定。

> **Important**
>
> 在新的 AWS Web 服務管理主控台中，CORS 組態必須是 JSON。

1. 在 AWS Web 服務管理主控台中，開啟 Amazon S3 主控台，找到您要設定的儲存貯體，然後選取其核取方塊。
2. 在開啟的窗格中，選擇「權限」。
3. 在 [權限] 索引標籤上，選擇 [CORS 設定]。
4. 在 CORS 設定編輯器中輸入您的 CORS 設定，然後選擇 [儲存]。

CORS 組態是一種 XML 檔案，包含 `<CORSRule>` 內的一系列規則。一個組態最多可以擁有 100 條規則，而規則是由下列其中一個標籤所定義：

- `<AllowedOrigin>`— 指定您允許發出跨網域要求的網域來源。
- `<AllowedMethod>`— 指定跨網域要求中允許的要求類型 (GET、PUT、POST、刪除、HEAD)。
- `<AllowedHeader>`— 指定預檢請求中允許的標頭。

如需設定範例，請參閱如何在儲存貯體上設定 CORS？ 在 Amazon 簡單存儲服務用戶指南。

CORS 組態範例

下列 CORS 組態範例可讓使用者從網域example.org檢視、新增、移除或更新值區內的物件。不過，我們建議您<AllowedOrigin>將網站的網域範圍設定為範圍。若要允許任何來源，則可指定"*"。

> **Important**
>
> 在新的 S3 主控台中，CORS 組態必須為 JSON 格式。

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<CORSConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <CORSRule>
    <AllowedOrigin>https://example.org</AllowedOrigin>
    <AllowedMethod>HEAD</AllowedMethod>
    <AllowedMethod>GET</AllowedMethod>
    <AllowedMethod>PUT</AllowedMethod>
    <AllowedMethod>POST</AllowedMethod>
    <AllowedMethod>DELETE</AllowedMethod>
    <AllowedHeader>*</AllowedHeader>
    <ExposeHeader>ETag</ExposeHeader>
    <ExposeHeader>x-amz-meta-custom-header</ExposeHeader>
  </CORSRule>
</CORSConfiguration>
```

JSON

```
[
    {
        "AllowedHeaders": [
            "*"
        ],
        "AllowedMethods": [
            "HEAD",
            "GET",
            "PUT",
            "POST",
            "DELETE"
        ],
```

```
        "AllowedOrigins": [
            "https://www.example.org"
        ],
        "ExposeHeaders": [
             "ETag",
             "x-amz-meta-custom-header"]
    }
]
```

這個組態並不會授權使用者對儲存貯體執行任何動作，它使瀏覽器的安全模型允許向 Amazon S3 發出請求。必須透過儲存貯體許可或 IAM 角色許可設定許可。

您可以使用ExposeHeader來讓開發套件讀取從 Amazon S3 傳回的回應標頭。例如，從PUT或多部分上傳中讀取標ETag題，您需要在配置中包含ExposeHeader標籤，如前面的範例所示。軟體開發套件僅能存取透過 CORS 組態公開的標頭。若您在物件上設定中繼資料，則傳回的值即為具有字首 x-amz-meta- 的標頭 (如 x-amz-meta-my-custom-header)；請務必以相同方式公開該標頭。

捆綁應用程序與網絡包

在瀏覽器腳本或 Node.js 中使用 Web 應用程序的代碼模塊創建依賴關係。這些程式碼模組可能會擁有自己的依存項目，成為一組您的應用程式運作所需的互連模組。要管理依賴關係，您可以使用類似webpack的模塊捆綁器。

webpack模塊捆綁器解析您的應用程序代碼，搜索import或require語句，以創建包含應用程序需要的所有資產的包。這樣可以通過網頁輕鬆提供資產。的 SDK JavaScript 可webpack作為要包含在輸出服務包中的其中一個依賴項。

如需詳細資訊webpack，請參閱上的 webpack 模組捆綁器。 GitHub

安裝網絡包

要安裝webpack模塊捆綁器，您必須先安裝 npm，Node.js 包管理器。鍵入以下命令以安裝 webpack CLI 和 JavaScript 模塊。

```
npm install --save-dev webpack
```

要使用該path模塊來處理文件和目錄路徑，這是與 webpack 自動安裝，你可能需要安裝 Node.js path-browserify 包。

```
npm install --save-dev path-browserify
```

配置網絡包

默認情況下，Webpack 會搜索項目根目錄`webpack.config.js`中名為的 JavaScript 文件。此檔案能夠指定組態選項。以下是 5.0.0 版及更新 WebPack 版本的`webpack.config.js`組態檔範例。

> **Note**
>
> Webpack 的配置需求根據您安裝的 Webpack 的版本而有所不同。如需詳細資訊，請參閱 Webpack 文件。

```
// Import path for resolving file paths
var path = require("path");
module.exports = {
  // Specify the entry point for our app.
  entry: [path.join(__dirname, "browser.js")],
  // Specify the output file containing our bundled code.
  output: {
    path: __dirname,
    filename: 'bundle.js'
  },
   // Enable WebPack to use the 'path' package.
   resolve:{
  fallback: { path: require.resolve("path-browserify")}
  }
  /**
  * In Webpack version v2.0.0 and earlier, you must tell
  * webpack how to use "json-loader" to load 'json' files.
  * To do this Enter 'npm --save-dev install json-loader' at the
  * command line to install the "json-loader' package, and include the
  * following entry in your webpack.config.js.
  * module: {
    rules: [{test: /\.json$/, use: use: "json-loader"}]
  }
  **/
};
```

在此範例中`browser.js`，指定為入口點。進入點是`webpack`用來開始搜尋匯入模組的檔案。系統會指定輸出檔案名稱為 `bundle.js`，該輸出文件將包含所有需要運行 JavaScript 的應用程序。如果進入點中指定的程式碼匯入或需要其他模組 (例如 SDK) JavaScript，則該程式碼會隨附而不需要在組態中指定。

運行網絡包

要構建要使用的應用程序webpack，請將以下內容添加到package.json文scripts件中的對象中。

```
"build": "webpack"
```

以下是示範新增的範例package.json檔案webpack。

```
{
  "name": "aws-webpack",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "build": "webpack"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "@aws-sdk/client-iam": "^3.32.0",
    "@aws-sdk/client-s3": "^3.32.0"
  },
  "devDependencies": {
    "webpack": "^5.0.0"
  }
}
```

若要建立您的應用程式，請輸入下列命令。

```
npm run build
```

然後，webpack模塊捆綁器生成您在項目根目錄中指定的 JavaScript 文件。

使用網絡包包

若要在瀏覽器指令碼中使用套裝軟體，您可以使用<script>標籤來合併套裝軟體，如下列範例所示。

```
<!DOCTYPE html>
<html>
    <head>
        <title>Amazon SDK with webpack</title>
```

```
    </head>
    <body>
        <div id="list"></div>
        <script src="bundle.js"></script>
    </body>
</html>
```

Node.js 的套件組合

您可webpack以在組態中指定node為目標，來產生在 Node.js 中執行的套裝軟體。

```
target: "node"
```

當您在磁碟空間有限的環境中執行 Node.js 應用程式時，這個做法十分實用。下方為 `webpack.config.js` 組態範例，而該組態會將 Node.js 指定為輸出目標。

```
// Import path for resolving file paths
var path = require("path");
module.exports = {
  // Specify the entry point for our app.
  entry: [path.join(__dirname, "browser.js")],
  // Specify the output file containing our bundled code.
  output: {
    path: __dirname,
    filename: 'bundle.js'
  },
  // Let webpack know to generate a Node.js bundle.
  target: "node",
   // Enable WebPack to use the 'path' package.
   resolve:{
  fallback: { path: require.resolve("path-browserify")}
   /**
   * In Webpack version v2.0.0 and earlier, you must tell
   * webpack how to use "json-loader" to load 'json' files.
   * To do this Enter 'npm --save-dev install json-loader' at the
   * command line to install the "json-loader' package, and include the
   * following entry in your webpack.config.js.
   module: {
    rules: [{test: /\.json$/, use: use: "json-loader"}]
  }
  **/
};
```

使用下列項目中的 AWS SDK服務 JavaScript

AWS SDK for JavaScript v3 通過客戶端類集合提供對它支持的服務的訪問。您可以使用這些用戶端類別來建立服務界面物件，其通常稱為服務物件。每個支援的 AWS 服務都有一或多個用戶端類別，這些用戶端類別提供低階APIs服務功能和資源的使用。例如，Amazon DynamoDB 可透過該APIsDynamoDB類別取得。

透過 of 公開的服務會SDK JavaScript 遵循要求-回應模式，以便與呼叫應用程式交換訊息。在此模式中，叫用服務的程式碼會將HTTP/HTTPS要求提交至服務的端點。為成功叫用所呼叫的特定功能，該請求包含所有必要參數。接著，叫用的服務會產生要傳回請求程式的回應。如果操作成功，該回應會包含相關資料；如果操作失敗，回應便會內含錯誤資訊。

叫用 AWS 服務包括服務物件上作業的完整要求和回應生命週期，包括嘗試的任何重試。請求包含零個或多個屬性作為JSON參數。回應會封裝在與作業相關的物件中，並透過數種技術之一 (例如回呼函式或 promise) 傳回給要求程式 JavaScript 。

主題

創建和調用服務對象

JavaScript API支持大多數可用的 AWS 服務。中的每個服務都會 JavaScriptAPI提供用戶端類別，其中包含您用來叫用每個服務支援API的`send`方法。如需有關中的服務類別、作業和參數的詳細資訊 JavaScript API，請參閱《參API考》。

SDK在 Node.js 中使用時，您可以將您需要的每個服務的SDK套件新增至應用程式使用`import`，以提供目前所有服務的支援。下列範例會在該`us-west-1`區域中建立 Amazon S3 服務物件。

```
// Import the Amazon S3 service client
import { S3Client } from "@aws-sdk/client-s3";
// Create an S3 client in the us-west-1 Region
```

```
const s3Client = new S3Client({
    region: "us-west-1"
});
```

指定服務物件參數

呼叫服務物件的方法時，請依需JSON要將參數傳入API。例如，在 Amazon S3 中，若要取得指定儲存貯體和金鑰的物件，請將下列參`GetObjectCommand`數從`S3Client`. 如需有關傳遞JSON參數的詳細資訊，請參閱使用 JSON。

```
s3Client.send(new GetObjectCommand({Bucket: 'bucketName', Key: 'keyName'}));
```

如需有關 Amazon S3 參數的詳細資訊，請參閱參考資料中的 @aws-sdk/客戶端s3。API

異步呼叫服務

透過提出的所有要求SDK都是非同步的。編寫瀏覽器腳本時，請記住這一點很重要。 JavaScript 在 Web 瀏覽器中運行通常只有一個執行線程。在對 AWS 服務進行非同步呼叫之後，瀏覽器指令碼會繼續執行，而且在處理序中可以嘗試在傳回之前執行依賴於該非同步結果的程式碼。

對 AWS 服務進行非同步調用包括管理這些調用，以便您的代碼不會在數據可用之前嘗試使用數據。本節中的主題會說明管理非同步呼叫的重要性，並詳細解說可用來管理這些呼叫的不同技術。

雖然您可以使用這些技術中的任何一種來管理非同步呼叫，但我們建議您針對所有新程式碼使用 async/await。

異步/等待

我們建議您使用此技術，因為它是 V3 中的預設行為。

諾言

在不支援異步/等待的瀏覽器中使用此技巧。

回調

避免使用回調，除非在非常簡單的情況下。但是，您可能會發現它對遷移案例很有用。

主題

- 管理非同步呼叫
- 使用異步/等待

- 使用 JavaScript 承諾
- 使用匿名回調函數

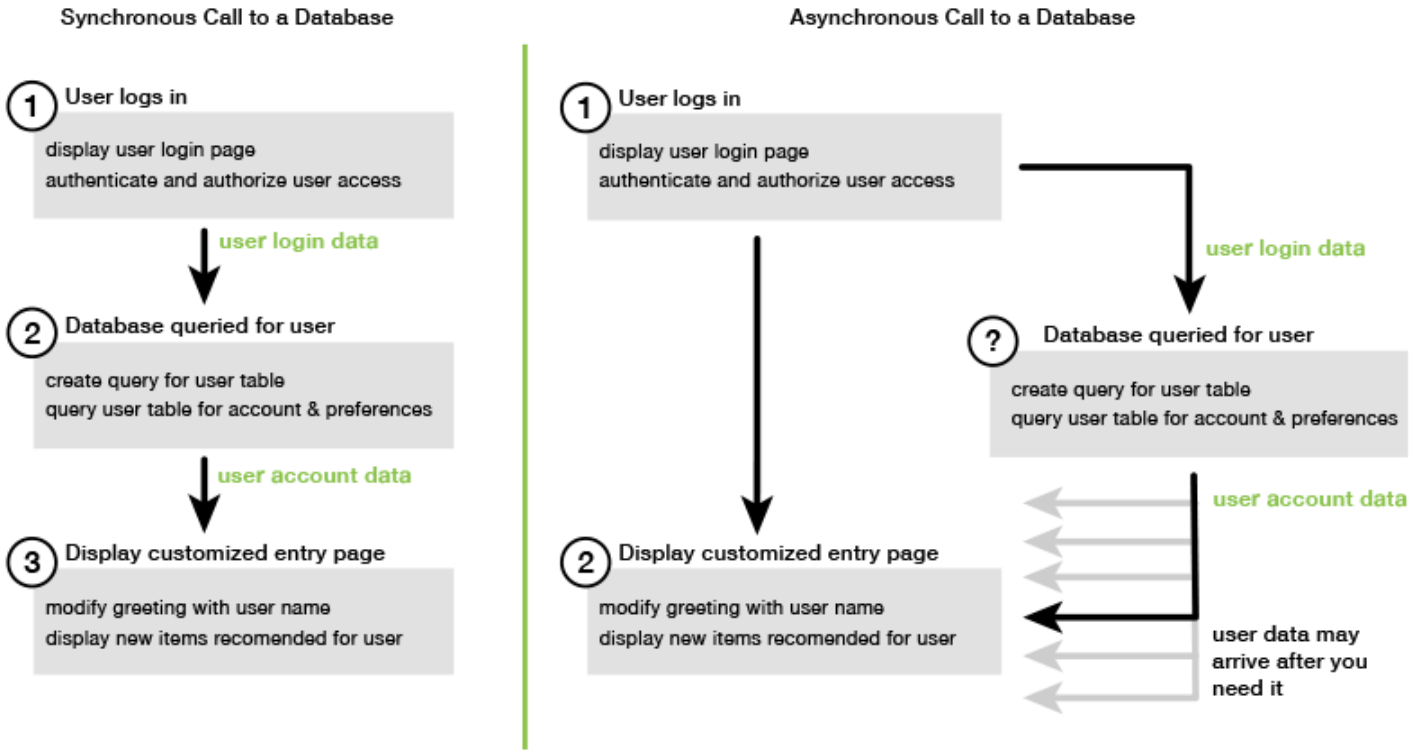
管理非同步呼叫

舉例來說，回流的客戶能經由電子商務網站的首頁進行登入。對登入的客戶而言，這項功能的部分優點是網站會在客戶登入後，根據其特定偏好設定來自訂本身版面。為實現此目標，必須滿足以下條件：

1. 客戶必須登入並使用其登入認證進行驗證。
2. 系統可從客戶資料庫請求客戶的偏好設定。
3. 資料庫需提供客戶的偏好設定，以便系統在載入網頁前使用該設定自訂網站。

如果您是同步執行這些任務，則每個任務必須在下一個任務開始之前完成。在客戶偏好設定從資料庫傳回之前，網頁將無法完成載入。但是，當系統將資料庫查詢傳送至伺服器後，網路瓶頸、異常高的資料庫流量，或是行動裝置連線品質不佳，都可能造成客戶資料接收延遲，甚至失敗。

若要防止網站在這些情況下凍結，請以非同步方式呼叫資料庫。開始執行資料庫呼叫後，您能夠傳送非同步請求，讓程式碼能繼續正常運作。如果您沒有適當管理非同步呼叫的回應，程式碼就有可能在資料尚不可用的情況下，嘗試使用資料庫原先應回傳的相關資訊。



使用異步/等待

你應考慮使用 async/await，而非 Promise。Async 函數比使用 Promise 更簡單，採用的樣板更少。Await 僅可在 async 函數中使用，以非同步方式等待值。

下列範例會使用異步/等待來列出中的所有 Amazon DynamoDB 表格。 `us-west-2`

> **Note**
>
> 對於此示例運行：
>
> - 在專案的命令列`npm install @aws-sdk/client-dynamodb`中輸入以安裝 AWS SDK for JavaScript DynamoDB 用戶端。
> - 請確定您已正確設定 AWS 認證。如需詳細資訊，請參閱 設定認證。

```
import {
  DynamoDBClient,
  ListTablesCommand
} from "@aws-sdk/client-dynamodb";
(async function () {
  const dbClient = new DynamoDBClient({ region: "us-west-2" });
  const command = new ListTablesCommand({});

  try {
    const results = await dbClient.send(command);
    console.log(results.TableNames.join('\n'));
  } catch (err) {
    console.error(err)
  }
})();
```

> **Note**
>
> 並非所有瀏覽器都支持異步/等待。有關具有異步/等待支持的瀏覽器列表，請參閱異步函數。

使用 JavaScript 承諾

使用服務客戶端的 AWS SDK for JavaScript v3 方法 (`ListTablesCommand`) 進行服務調用和管理異步流程，而不是使用回調。下列範例示範如何在中取得 Amazon DynamoDB 表格的名稱。`us-west-2`

```
import {
  DynamoDBClient,
  ListTablesCommand
} from "@aws-sdk/client-dynamodb";
const dbClient = new DynamoDBClient({ region: 'us-west-2' });

dbClient.listtables(new ListTablesCommand({}))
  .then(response => {
    console.log(response.TableNames.join('\n'));
  })
  .catch((error) => {
    console.error(error);
  });
```

協調多個承諾

在某些情況下，程式碼必須發出多個非同步呼叫，且唯有在這些呼叫全部成功回傳時，才需要採取動作。如果您要透過 promise 來管理個別非同步方法呼叫，則可建立採用 `all` 方法的額外 promise。

您傳遞至方法的 promise 陣列都達成時，此方法即達成這個全域 promise。promise 傳遞至 `all` 方法的陣列值，會傳遞至回呼函數。

在下列範例中， AWS Lambda 函數必須對 Amazon DynamoDB 進行三個非同步呼叫，但只有在完成每個呼叫的承諾後才能完成。

```
const values = await Promise.all([firstPromise, secondPromise, thirdPromise]);

console.log("Value 0 is " + values[0].toString);
console.log("Value 1 is " + values[1].toString);
console.log("Value 2 is " + values[2].toString);

return values;
```

瀏覽器和 Node.js 支持承諾

對本機 JavaScript 承諾（ECMAScript2015）的 Support 取決於代碼執行的 JavaScript引擎和版本。若要協助判斷程式碼需要執行之每個環境中 JavaScript promise 的支援，請參閱上的ECMAScript相容性表格 GitHub。

使用匿名回調函數

每個服務對象方法都可以接受匿名回調函數作為最後一個參數。這個回調函數的簽名如下。

```
function(error, data) {
    // callback handling code
};
```

當系統傳回成功回應或錯誤資料時，這類回呼函數即會開始執行。如果方法呼叫成功，回應內容便可供 `data` 參數中的回呼函數使用；如果呼叫不成功，則 `error` 參數會提供失敗的詳細資訊。

回呼函數內部的程式碼通常會測試錯誤，並處理傳回的錯誤。若沒有傳回錯誤，該程式碼就會擷取來自 `data` 參數的回應資料。回呼函數的基本形式如下方範例所示。

```
function(error, data) {
    if (error) {
        // error handling code
        console.log(error);
    } else {
        // data handling code
        console.log(data);
    }
};
```

在上述範例中，錯誤或所傳回資料的詳細資訊都會記錄到主控台。在接下來的範例中，系統會將傳遞的回呼函數做為呼叫服務物件方法的一部分。

```
ec2.describeInstances(function(error, data) {
  if (error) {
    console.log(error); // an error occurred
  } else {
    console.log(data); // request succeeded
  }
});
```

建立服務用戶端要求

向 AWS 服務客戶提出請求非常簡單。的版本 3 (V3) JavaScript 可讓您傳送要求。SDK

> **Note**
>
> 您也可以在使用的 V3 時，使用版本 2 (V2) 命令執行作SDK業 JavaScript。如需詳細資訊，請參閱 使用 v2 命令。

若要傳送請求：

1. 使用所需的配置（例如特定 AWS 區域）初始化客戶端對象。
2. (選擇性) 建立包含請求值的請求JSON物件，例如特定 Amazon S3 儲存貯體的名稱。您可以查看介面的「參API考」主題，以及與用戶端方法相關聯的名稱，以檢查要求的參數。例如，如果您使用 *AbcCommand* 客戶端方法，請求接口是 *AbcInput*.
3. 以要求物件作為輸入，選擇性地初始化服務命令。
4. 使用命令物件作為輸入呼send叫用戶端。

例如，若要在中列出您的 Amazon DynamoDB 表格us-west-2，您可以使用異步/等待來執行此操作。

```
import {
  DynamoDBClient,
  ListTablesCommand
} from "@aws-sdk/client-dynamodb";

(async function () {
  const dbClient = new DynamoDBClient({ region: 'us-west-2' });
  const command = new ListTablesCommand({});

  try {
    const results = await dbClient.send(command);
    console.log(results.TableNames.join('\n'));
  } catch (err) {
    console.error(err);
  }
})();
```

處理服務用戶端回應

一個服務客戶端方法已被調用後，它返回與客戶端方法相關聯的名稱接口的響應對象實例。例如，如果您使用 *AbcCommand* 客戶端方法，響應對象是 *AbcResponse* （接口）類型。

訪問響應中返回的數據

響應對象包含數據，作為屬性，由服務請求返回。

在中建立服務用戶端要求，命 `ListTablesCommand`令會傳回回應`TableNames`屬性中的資料表名稱。

存取錯誤資訊

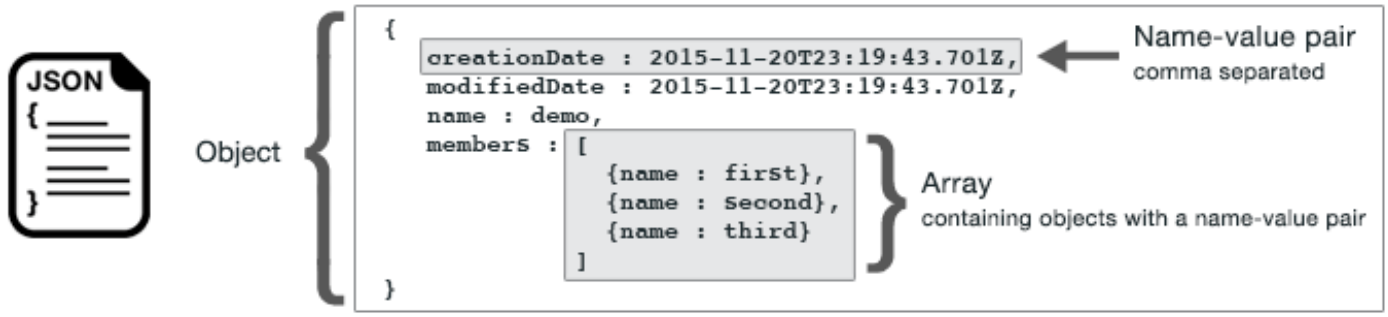
如果一個命令失敗，它拋出一個異常。下面的代碼片段顯示了一種處理服務異常的方法。

```
try {
  await client.send(someCommand);
} catch (e) {
  if (e.name === "InvalidSignatureException") {
    // Handle InvalidSignatureException
  } else if (e.name === "ResourceNotFoundException") {
    // Handle ResourceNotFoundException
  } else if (e.name === "FooServiceException") {
    // Handle all other server-side exceptions from Foo service
  } else {
    // Handle errors from SDK
  }
}
```

使用 JSON

JSON是一種人類可讀和機器可讀的數據交換格式。雖然名稱JSON是「JavaScript 物件符號」的首字母縮寫，但的格式獨立JSON於任何程式設計語言。

AWS SDK for JavaScript 用於在發JSON出請求時將數據發送到服務對象，並以下式從服務對象接收數據JSON。如需有關的詳細資訊JSON，請參閱 json.org。



JSON以兩種方式表示資料：

- 作為一個對象，它是名稱-值對的無序集合。系統會在左 ({) 和右 (}) 括號內定義物件。每個名稱/值對皆以名稱開始，接著是冒號，然後是值。名稱/值對則是以逗號分隔。
- 作為一個數組，它是值的有序集合。系統會在左 ([) 和右 (]) 括號內定義陣列。陣列中的項目皆是以逗號分隔。

下面是一個JSON對象的例子，其中包含一個對象的數組，其中對象代表紙牌遊戲中的卡片。每張卡都由兩個名稱-值對定義，一個指定用於識別該卡的唯一值，另一個URL指定指向相應卡片圖像的唯一值。

```
var cards = [
  {"CardID":"defaultname", "Image":"defaulturl"},
  {"CardID":"defaultname", "Image":"defaulturl"},
  {"CardID":"defaultname", "Image":"defaulturl"},
  {"CardID":"defaultname", "Image":"defaulturl"},
  {"CardID":"defaultname", "Image":"defaulturl"}
];
```

JSON做為服務物件參數

下面是一個簡單的例子，JSON用於定義調用 AWS Lambda 服務對象的參數。

```
const params = {
   FunctionName : funcName,
   Payload : JSON.stringify(payload),
   LogType : LogType.Tail,
};
```

params 物件是由三個名稱/值對所定義，系統會在左右括號內以逗號分隔該物件。提供參數給服務物件方法呼叫時，名稱會依欲呼叫之服務物件方法的參數名稱而定。當調用 Lambda 函數時，FunctionNamePayload、和 LogType是用於調用 Lambda 服務對象上的invoke方法的參數。

將參數傳遞至服務物件方法呼叫時，請將JSON物件提供給方法呼叫，如下列叫用 Lambda 函數的範例所示。

```
const invoke = async (funcName, payload) => {
  const client = new LambdaClient({});
  const command = new InvokeCommand({
    FunctionName: funcName,
    Payload: JSON.stringify(payload),
    LogType: LogType.Tail,
  });

  const { Payload, LogResult } = await client.send(command);
  const result = Buffer.from(Payload).toString();
  const logs = Buffer.from(LogResult, "base64").toString();
  return { logs, result };
};
```

SDK代 JavaScript 碼示例

本節中的主題包含如何使用各APIs種服務來執行一般工作的範例。 AWS SDK for JavaScript

在上的程式碼範例儲存庫中尋找這些範例和其他範例的原始AWS 程式碼 GitHub。若要提出 AWS 文件小組考慮產生的新程式碼範例，請建立要求。團隊希望產生涵蓋更廣泛案例和使用案例的程式碼範例，而不是只涵蓋個別API呼叫的簡單程式碼片段。如需指示，請參閱的提供指南中的〈撰寫程式碼〉一節GitHub。

> **Important**
>
> 這些範例使用ECMAScript6匯入/匯出語法。
>
> - 這需要 Node.js 版本 14.17 或更高版本。要下載並安裝最新版本的 Node.js，請參閱 Node.js 下載。 。
> - 如果您偏好使用 CommonJS 語法，請參閱以JavaScript ES6/共同語法取得轉換準則。

主題

- JavaScript ES6/共同語法
- Amazon DynamoDB 範例
- AWS Elemental MediaConvert 範例
- AWS Lambda 範例
- 亞馬遜萊克斯例
- Amazon Polly 的例子
- Amazon Redshift 示例
- Amazon 簡單電子郵件服務
- 亞馬遜簡單通知服務示例
- Amazon Transcribe 示例
- 在 Amazon EC2 實例上設置 Node.js
- 建置應用程式以將資料提交至 DynamoDB
- 使用 API Gateway 叫用 Lambda
- 建立排程事件以執行AWS Lambda功能
- 建立 Amazon Lex 聊天機器人
- 建立範例訊息應用程式

JavaScript ES6/共同語法

代AWS SDK for JavaScript碼示例是寫在印刷稿 6（ES6）。ES6 帶來了新的語法和新功能，使您的代碼更加現代化和可讀性，並做更多。

ES6 要求您使用 Node.js 版本 13.x 或更高版本。若要下載和安裝下載和安裝下載和安裝下載和安裝 Node.js 的最新版本。Node.js 但您若喜歡，可使用下面的準則將我們的任何範例轉換成下面的準則：

- `"type" : "module"`從專案環境`package.json`中移除。
- 將所有 ES6`import` 語句轉換為普通 JS`require` 語句。例如，轉換：

```
import { CreateBucketCommand } from "@aws-sdk/client-s3";
import { s3 } from "./libs/s3Client.js";
```

對於它的共同 JS 等價物：

```
const { CreateBucketCommand } = require("@aws-sdk/client-s3");
const { s3 } = require("./libs/s3Client.js");
```

- 將所有 ES6`export` 語句轉換為普通 JS`module.exports` 語句。例如，轉換：

```
export {s3}
```

對於它的共同 JS 等價物：

```
module.exports = {s3}
```

下列範例示範如何在 ES6 和 CommonJS 中建立 Amazon S3 儲存貯體的程式碼範例。

ES6

libs/s3Client.js

```
// Create service client module using ES6 syntax.
import { S3Client } from "@aws-sdk/client-s3";
// Set the AWS region
const REGION = "eu-west-1"; //e.g. "us-east-1"
// Create Amazon S3 service object.
const s3 = new S3Client({ region: REGION });
// Export 's3' constant.
export {s3};
```

s3_createbucket.js

```
// Get service clients module and commands using ES6 syntax.
 import { CreateBucketCommand } from "@aws-sdk/client-s3";
 import { s3 } from "./libs/s3Client.js";

// Get service clients module and commands using CommonJS syntax.
// const { CreateBucketCommand } = require("@aws-sdk/client-s3");
// const { s3 } = require("./libs/s3Client.js");

// Set the bucket parameters
```

```
const bucketParams = { Bucket: "BUCKET_NAME" };

// Create the Amazon S3 bucket.
const run = async () => {
  try {
    const data = await s3.send(new CreateBucketCommand(bucketParams));
    console.log("Success", data.Location);
    return data;
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

CommonJS

libs/s3Client.js

```
// Create service client module using CommonJS syntax.
 const { S3Client } = require("@aws-sdk/client-s3");
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
 // Create Amazon S3 service object.
const s3 = new S3Client({ region: REGION });
// Export 's3' constant.
 module.exports ={s3};
```

s3_createbucket.js

```
// Get service clients module and commands using CommonJS syntax.
const { CreateBucketCommand } = require("@aws-sdk/client-s3");
const { s3 } = require("./libs/s3Client.js");

// Set the bucket parameters
const bucketParams = { Bucket: "BUCKET_NAME" };

// Create the Amazon S3 bucket.
```

```
const run = async () => {
  try {
    const data = await s3.send(new CreateBucketCommand(bucketParams));
    console.log("Success", data.Location);
    return data;
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

Amazon DynamoDB 範例

Amazon DynamoDB 是全受管的無SQL雲端資料庫，可同時支援文件和金鑰值存放區模型。您建立適用於資料的結構描述資料表，不需佈建或維護專屬的資料庫伺服器。

JavaScript API適用於 DynamoDB 的會透過DynamoDB、和DynamoDB.DocumentClient用戶端類別DynamoDBStreams公開。如需有關使用 DynamoDB 用戶端類別的詳細資訊，請參閱參考資料中的類別：DynamoDB、類別：D ynamoDBStreams 和類別：DynamoDB 公用程式。API

主題

- 在 DynamoDB 中建立和使用資料表
- 在 DynamoDB 中讀取和寫入單一項目
- 在 DynamoDB 中批次讀取和寫入項目
- 查詢和掃描 DynamoDB 資料表
- 使用 DynamoDB 用戶端

在 DynamoDB 中建立和使用資料表

這個 Node.js 程式碼範例會說明：

- 如何建立和管理用於從 DynamoDB 儲存和擷取資料的表格。

該方案

與其他資料庫系統類似，DynamoDB 會將資料儲存在表格中。DynamoDB 資料表是資料的集合，這些資料組織成類似於列的項目。若要在 DynamoDB 中儲存或存取資料，您可以建立和使用資料表。

在此範例中，您會使用一系列 Node.js 模組，透過 DynamoDB 表格執行基本作業。程式碼會使用 SDK for JavaScript 來建立並使用用DynamoDB戶端類別的下列方法來處理資料表：

- `CreateTableCommand`
- `ListTablesCommand`
- `DescribeTableCommand`
- `DeleteTableCommand`

必要工作

若要設定和執行此範例，請先完成這些任務：

- 設置項目環境以運行這些 Node.js 示例，並安裝所需 AWS SDK for JavaScript 的第三方模塊。按照上的說明進行操作 GitHub。
- SDK為用戶端 JavaScript DynamoDB 安裝。如需詳細資訊，請參閱 第 3 版中的新功能。
- 透過使用者登入資料建立共用組態檔。如需有關提供共用認證檔案的詳細資訊，請參閱《工具參考指南》中的共用設定AWS SDKs和認證檔案。

> **Important**
>
> 這些範例使用 ECMAScript6 (ES6)。這需要 Node.js 版本 13.x 或更高版本。要下載並安裝最新版本的 Node.js，請參閱 Node.js 下載。 。

但是，如果您更喜歡使用 CommonJS 語法，請參閱JavaScript ES6/共同語法。

> **Note**
>
> 如需這些範例中使用之資料類型的詳細資訊，請參閱 Amazon DynamoDB 中支援的資料類型和命名規則。

建立資料表

以檔名 `create-table.js` 建立一個 Node.js 模組。請務必SDK如先前所示設定，包括下載所需的用戶端和套件。若要存取 DynamoDB，請建立DynamoDB用戶端服務物件。建立包含建立表格所需參數的JSON物件，在此範例中包括每個屬性的名稱和資料類型、索引鍵綱要、表格名稱以及要佈建的輸送量單位。呼叫 DynamoDB 服務物件的`CreateTableCommand`方法。

```
import { CreateTableCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new CreateTableCommand({
    TableName: "EspressoDrinks",
    // For more information about data types,
    // see https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
HowItWorks.NamingRulesDataTypes.html#HowItWorks.DataTypes and
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Programming.LowLevelAPI.html#Programming.LowLevelAPI.DataTypeDescriptors
    AttributeDefinitions: [
      {
        AttributeName: "DrinkName",
        AttributeType: "S",
      },
    ],
    KeySchema: [
      {
        AttributeName: "DrinkName",
        KeyType: "HASH",
      },
    ],
    ProvisionedThroughput: {
```

```
      ReadCapacityUnits: 1,
      WriteCapacityUnits: 1,
    },
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

若要執行範例，請在命令提示字元中輸入下列命令。

```
node create-table.js
```

您可以在這裡找到這個範例程式碼 GitHub。

列出您的表

以檔名 `list-tables.js` 建立一個 Node.js 模組。請務必SDK如先前所示設定，包括下載所需的用戶端和套件。若要存取 DynamoDB，請建立DynamoDB用戶端服務物件。建立一個JSON物件，其中包含列出表格所需的參數，在此範例中，此範例會將列出的表格數目限制為 10 個。呼叫 DynamoDB 服務物件的`ListTablesCommand`方法。

```
import { ListTablesCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new ListTablesCommand({});

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

若要執行範例，請在命令提示字元中輸入下列命令。

```
node list-tables.js
```

您可以在這裡找到這個範例程式碼 GitHub。

說明資料表

以檔名 describe-table.js 建立一個 Node.js 模組。請務必SDK如先前所示設定，包括下載所需的用戶端和套件。若要存取 DynamoDB，請建立DynamoDB用戶端服務物件。建立JSON物件，其中包含描述 DynamoDB 服務物件DescribeTableCommand方法所需的參數。

```
import { DescribeTableCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new DescribeTableCommand({
    TableName: "Pastries",
  });

  const response = await client.send(command);
  console.log(`TABLE NAME: ${response.Table.TableName}`);
  console.log(`TABLE ITEM COUNT: ${response.Table.ItemCount}`);
  return response;
};
```

若要執行範例，請在命令提示字元中輸入下列命令。

```
node describe-table.js
```

您可以在這裡找到這個範例程式碼 GitHub。

刪除資料表

以檔名 delete-table.js 建立一個 Node.js 模組。請務必SDK如先前所示設定，包括下載所需的用戶端和套件。若要存取 DynamoDB，請建立DynamoDB用戶端服務物件。建立包含刪除資料表所需之參數的JSON物件，在此範例中包含以命令列參數形式提供的資料表名稱。呼叫 DynamoDB 服務物件的DeleteTableCommand方法。

```
import { DeleteTableCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new DeleteTableCommand({
    TableName: "DecafCoffees",
```

```
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

若要執行範例，請在命令提示字元中輸入下列命令。

```
node delete-table.js
```

您可以在這裡找到這個範例程式碼 GitHub。

在 DynamoDB 中讀取和寫入單一項目

這個 Node.js 程式碼範例會說明：

- 如何在 DynamoDB 資料表中新增項目。
- 如何擷取 DynamoDB 資料表中的項目。
- 如何刪除 DynamoDB 資料表中的項目。

該方案

在此範例中，您可以使用一系列 Node.js 模組來讀取和寫入 DynamoDB 表格中的一個項目，方法是使用用DynamoDB戶端類別的下列方法：

- PutItemCommand
- UpdateItemCommand
- GetItemCommand
- DeleteItemCommand

必要工作

若要設定和執行此範例，請先完成這些任務：

- 設置項目環境以運行這些 Node.js 示例，並安裝所需 AWS SDK for JavaScript 的第三方模塊。按照上的說明進行操作 GitHub。
- 透過使用者登入資料建立共用組態檔。如需有關提供共用認證檔案的詳細資訊，請參閱《工具參考指南》中的共用設定AWS SDKs和認證檔案。
- 建立一個您可以存取其項目的 DynamoDB 表格。如需有關建立 DynamoDB 資料表的詳細資訊，請參閱。在 DynamoDB 中建立和使用資料表

> **Important**
>
> 這些範例使用 ECMAScript6 (ES6)。這需要 Node.js 版本 13.x 或更高版本。要下載並安裝最新版本的 Node.js，請參閱 Node.js 下載。 。
> 但是，如果您更喜歡使用 CommonJS 語法，請參閱JavaScript ES6/共同語法。

> **Note**
>
> 如需這些範例中使用之資料類型的詳細資訊，請參閱 Amazon DynamoDB 中支援的資料類型和命名規則。

寫入項目

以檔名 `put-item.js` 建立一個 Node.js 模組。請務必SDK如先前所示設定，包括下載所需的用戶端和套件。若要存取 DynamoDB，請建立DynamoDB用戶端服務物件。建立包含新增項目所需參數的JSON物件，在此範例中包含表格的名稱和定義要設定之屬性和每個屬性值的對映。呼叫 DynamoDB用戶端服務物件的`PutItemCommand`方法。

```
import { PutItemCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new PutItemCommand({
    TableName: "Cookies",
    // For more information about data types,
    // see https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
HowItWorks.NamingRulesDataTypes.html#HowItWorks.DataTypes and
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Programming.LowLevelAPI.html#Programming.LowLevelAPI.DataTypeDescriptors
```

```
    Item: {
      Flavor: { S: "Chocolate Chip" },
      Variants: { SS: ["White Chocolate Chip", "Chocolate Chunk"] },
    },
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

若要執行範例，請在命令提示字元中輸入下列命令。

```
node put-item.js
```

您可以在這裡找到這個範例程式碼 GitHub。

更新項目

以檔名 `update-item.js` 建立一個 Node.js 模組。請務必SDK如先前所示設定，包括下載所需的用戶端和套件。若要存取 DynamoDB，請建立DynamoDB用戶端服務物件。建立包含新增項目所需參數的JSON物件，在此範例中包括資料表名稱、要更新的索引鍵、對應新屬性名稱的日期運算式，以及每個新屬性的值。呼叫 DynamoDB 用戶端服務物件的`UpdateItemCommand`方法。

```
import { UpdateItemCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new UpdateItemCommand({
    TableName: "IceCreams",
    // For more information about data types,
    // see https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
HowItWorks.NamingRulesDataTypes.html#HowItWorks.DataTypes and
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Programming.LowLevelAPI.html#Programming.LowLevelAPI.DataTypeDescriptors
    Key: {
      Flavor: { S: "Vanilla" },
    },
    UpdateExpression: "set HasChunks = :chunks",
    ExpressionAttributeValues: {
      ":chunks": { BOOL: "false" },
```

```
    },
    ReturnValues: "ALL_NEW",
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

若要執行範例，請在命令提示字元中輸入下列命令。

```
node update-item.js
```

您可以在這裡找到這個範例程式碼 GitHub。

取得項目

以檔名 `get-item.js` 建立一個 Node.js 模組。請務必SDK如先前所示設定，包括下載所需的用戶端和套件。若要存取 DynamoDB，請建立DynamoDB用戶端服務物件。若要辨識要取得的項目，您必須為資料表中的項目提供主索引鍵值。根據預設，`GetItemCommand` 方法會傳回為該項目定義的所有屬性值。若只要取得部分得可能屬性值，請使用投射表達式。

建立一個JSON物件，其中包含取得項目所需的參數，在此範例中包括資料表名稱、取得項目的金鑰名稱和金鑰值，以及可識別您要擷取之項目屬性的投影運算式。呼叫 DynamoDB 用戶端服務物件的`GetItemCommand`方法。

下列程式碼範例會從資料表擷取項目，其主索引鍵僅由分割索引鍵組成，而不是同時包含分割區索引鍵和排序索引鍵。如果資料表具有由分割索引鍵和排序索引鍵組成的主索引鍵，您也必須指定排序索引鍵名稱和屬性。

```
import { GetItemCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new GetItemCommand({
    TableName: "CafeTreats",
    // For more information about data types,
    // see https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
HowItWorks.NamingRulesDataTypes.html#HowItWorks.DataTypes and
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Programming.LowLevelAPI.html#Programming.LowLevelAPI.DataTypeDescriptors
```

```
    Key: {
      TreatId: { N: "101" },
    },
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

若要執行範例，請在命令提示字元中輸入下列命令。

```
node get-item.js
```

您可以在這裡找到這個範例程式碼 GitHub。

刪除項目

以檔名 `delete-item.js` 建立一個 Node.js 模組。請務必SDK如先前所示設定，包括下載所需的用戶端和套件。若要存取 DynamoDB，請建立DynamoDB用戶端服務物件。建立一個JSON物件，其中包含刪除項目所需的參數，在此範例中包含資料表的名稱，以及要刪除之項目的索引鍵名稱和值。呼叫 DynamoDB 用戶端服務物件的`DeleteItemCommand`方法。

```
import { DeleteItemCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new DeleteItemCommand({
    TableName: "Drinks",
    // For more information about data types,
    // see https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
HowItWorks.NamingRulesDataTypes.html#HowItWorks.DataTypes and
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Programming.LowLevelAPI.html#Programming.LowLevelAPI.DataTypeDescriptors
    Key: {
      Name: { S: "Pumpkin Spice Latte" },
    },
  });

  const response = await client.send(command);
  console.log(response);
```

```
  return response;
};
```

若要執行範例，請在命令提示字元中輸入下列命令。

```
node delete-item.js
```

您可以在這裡找到這個範例程式碼 GitHub。

在 DynamoDB 中批次讀取和寫入項目

這個 Node.js 程式碼範例會說明：

- 如何讀取和寫入 DynamoDB 表中的批次項目。

該方案

在此範例中，您會使用一系列 Node.js 模組，將一批項目放入 DynamoDB 表格中，並讀取一批項目。程式碼會使用 SDK for JavaScript 來執行使用 DynamoDB 用戶端類別的下列方法來執行批次讀取和寫入作業：

- BatchGetItemCommand
- BatchWriteItemCommand

必要工作

若要設定和執行此範例，請先完成這些任務：

- 設置項目環境以運行這些節點 TypeScript 示例，並安裝所需 AWS SDK for JavaScript 的第三方模塊。按照上的說明進行操作 GitHub。
- 透過使用者登入資料建立共用組態檔。如需有關提供共用認證檔案的詳細資訊，請參閱《工具參考指南》中的共用設定AWS SDKs和認證檔案。
- 建立一個您可以存取其項目的 DynamoDB 表格。如需有關建立 DynamoDB 資料表的詳細資訊，請參閱。在 DynamoDB 中建立和使用資料表

Important

這些範例使用 ECMAScript6 (ES6)。這需要 Node.js 版本 13.x 或更高版本。要下載並安裝最新版本的 Node.js，請參閱 Node.js 下載。 。
但是，如果您更喜歡使用 CommonJS 語法，請參閱JavaScript ES6/共同語法。

Note

如需這些範例中使用之資料類型的詳細資訊，請參閱 Amazon DynamoDB 中支援的資料類型和命名規則。

讀取批次中的項目

以檔名 `batch-get-item.js` 建立一個 Node.js 模組。請務必SDK如先前所示設定，包括下載所需的用戶端和套件。若要存取 DynamoDB，請建立DynamoDB用戶端服務物件。建立一個JSON物件，其中包含取得批次項目所需的參數，在此範例中包含要讀取的一或多個資料表的名稱、要在每個資料表中讀取的索引鍵值，以及指定要傳回之屬性的投影運算式。呼叫 DynamoDB 服務物件的`BatchGetItemCommand`方法。

```
import { BatchGetItemCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new BatchGetItemCommand({
    RequestItems: {
      // Each key in this object is the name of a table. This example refers
      // to a PageAnalytics table.
      PageAnalytics: {
        // Each entry in Keys is an object that specifies a primary key.
        Keys: [
          {
            // "PageName" is the partition key (simple primary key).
            // "S" specifies a string as the data type for the value "Home".
            // For more information about data types,
            // see https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
HowItWorks.NamingRulesDataTypes.html#HowItWorks.DataTypes and
            // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Programming.LowLevelAPI.html#Programming.LowLevelAPI.DataTypeDescriptors
```

```
            PageName: { S: "Home" },
          },
          {
            PageName: { S: "About" },
          },
        ],
        // Only return the "PageName" and "PageViews" attributes.
        ProjectionExpression: "PageName, PageViews",
      },
    },
  });

  const response = await client.send(command);
  console.log(response.Responses["PageAnalytics"]);
  return response;
};
```

若要執行範例，請在命令提示字元中輸入下列命令。

```
node batch-get-item.js
```

您可以在這裡找到這個範例程式碼 GitHub。

在批次中寫入項目

以檔名 `batch-write-item.js` 建立一個 Node.js 模組。請務必SDK如先前所示設定，包括下載所需的用戶端和套件。若要存取 DynamoDB，請建立DynamoDB用戶端服務物件。建立一個JSON物件，其中包含取得一批項目所需的參數，在此範例中包含您要寫入項目的資料表、您要為每個項目撰寫的索引鍵，以及屬性及其值。呼叫 DynamoDB 服務物件的`BatchWriteItemCommand`方法。

```
import {
  BatchWriteItemCommand,
  DynamoDBClient,
} from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new BatchWriteItemCommand({
    RequestItems: {
      // Each key in this object is the name of a table. This example refers
      // to a Coffees table.
```

```
      Coffees: [
        // Each entry in Coffees is an object that defines either a PutRequest or
 DeleteRequest.
        {
          // Each PutRequest object defines one item to be inserted into the table.
          PutRequest: {
            // The keys of Item are attribute names. Each attribute value is an object
 with a data type and value.
            // For more information about data types,
            // see https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
HowItWorks.NamingRulesDataTypes.html#HowItWorks.DataTypes
            Item: {
              Name: { S: "Donkey Kick" },
              Process: { S: "Wet-Hulled" },
              Flavors: { SS: ["Earth", "Syrup", "Spice"] },
            },
          },
        },
        {
          PutRequest: {
            Item: {
              Name: { S: "Flora Ethiopia" },
              Process: { S: "Washed" },
              Flavors: { SS: ["Stone Fruit", "Toasted Almond", "Delicate"] },
            },
          },
        },
      ],
    },
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

若要執行範例，請在命令提示字元中輸入下列命令。

```
node batch-write-item.js
```

您可以在這裡找到這個範例程式碼 GitHub。

查詢和掃描 DynamoDB 資料表

這個 Node.js 程式碼範例會說明：

- 如何查詢和掃描 DynamoDB 資料表中的項目。

該方案

查詢只使用主索引鍵屬性值在資料表或次要索引中尋找項目。您必須提供要搜尋的分區索引鍵名稱與值。您也可以提供排序索引鍵名稱和值，並使用比較運算子縮小搜尋結果。掃描會透過檢查指定資料表中的每個項目來尋找項目。

在此範例中，您可以使用一系列 Node.js 模組來識別您要從 DynamoDB 表格擷取的一或多個項目。程式碼會使用 SDK for JavaScript 來查詢和掃描使用 DynamoDB 用戶端類別的下列方法的資料表：

- QueryCommand
- ScanCommand

必要工作

若要設定和執行此範例，請先完成這些任務：

- 設置項目環境以運行這些 Node.js 示例，並安裝所需 AWS SDK for JavaScript 的第三方模塊。按照上的說明進行操作 GitHub。
- 透過使用者登入資料建立共用組態檔。如需有關提供共用認證檔案的詳細資訊，請參閱《工具參考指南》中的共用設定AWS SDKs和認證檔案。
- 建立一個您可以存取其項目的 DynamoDB 表格。如需有關建立 DynamoDB 資料表的詳細資訊，請參閱。在 DynamoDB 中建立和使用資料表

> **Important**
>
> 這些範例使用 ECMAScript6 (ES6)。這需要 Node.js 版本 13.x 或更高版本。要下載並安裝最新版本的 Node.js，請參閱 Node.js 下載。 。

但是，如果您更喜歡使用 CommonJS 語法，請參閱JavaScript ES6/共同語法。

> **Note**
>
> 如需這些範例中使用之資料類型的詳細資訊，請參閱 Amazon DynamoDB 中支援的資料類型和命名規則。

查詢資料表

以檔名 `query.js` 建立一個 Node.js 模組。請務必SDK如先前所示設定，包括下載所需的用戶端和套件。若要存取 DynamoDB，請建立DynamoDB用戶端服務物件。建立一個JSON物件，其中包含查詢資料表所需的參數，在此範例中包含資料表名稱、查詢`ExpressionAttributeValues`所需的資料表名稱、使用`KeyConditionExpression`這些值來定義查詢傳回的項目，以及每個項目所要傳回的屬性值名稱。呼叫 DynamoDB 服務物件的`QueryCommand`方法。

```
import { DynamoDBClient, QueryCommand } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new QueryCommand({
    KeyConditionExpression: "Flavor = :flavor",
    // For more information about data types,
    // see https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
HowItWorks.NamingRulesDataTypes.html#HowItWorks.DataTypes and
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Programming.LowLevelAPI.html#Programming.LowLevelAPI.DataTypeDescriptors
    ExpressionAttributeValues: {
      ":flavor": { S: "Key Lime" },
      ":searchKey": { S: "no coloring" },
    },
    FilterExpression: "contains (Description, :searchKey)",
    ProjectionExpression: "Flavor, CrustType, Description",
    TableName: "Pies",
  });

  const response = await client.send(command);
  response.Items.forEach(function (pie) {
    console.log(`${pie.Flavor.S} - ${pie.Description.S}\n`);
  });
```

```
  return response;
};
```

若要執行範例，請在命令提示字元中輸入下列命令。

```
node query.js
```

您可以在這裡找到這個範例程式碼 GitHub。

掃描資料表

以檔名 `scan.js` 建立一個 Node.js 模組。請務必SDK如先前所示設定，包括下載所需的用戶端和套件。若要存取 DynamoDB，請建立DynamoDB用戶端服務物件。建立一個JSON物件，其中包含掃描表格中項目所需的參數，在此範例中包括表格名稱、為每個相符項目傳回的屬性值清單，以及用來篩選結果集以尋找包含指定片語之項目的運算式。呼叫 DynamoDB 服務物件的`ScanCommand`方法。

```
import { DynamoDBClient, ScanCommand } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new ScanCommand({
    FilterExpression: "CrustType = :crustType",
    // For more information about data types,
    // see https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
HowItWorks.NamingRulesDataTypes.html#HowItWorks.DataTypes and
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Programming.LowLevelAPI.html#Programming.LowLevelAPI.DataTypeDescriptors
    ExpressionAttributeValues: {
      ":crustType": { S: "Graham Cracker" },
    },
    ProjectionExpression: "Flavor, CrustType, Description",
    TableName: "Pies",
  });

  const response = await client.send(command);
  response.Items.forEach(function (pie) {
    console.log(`${pie.Flavor.S} - ${pie.Description.S}\n`);
  });
  return response;
};
```

若要執行範例，請在命令提示字元中輸入下列命令。

```
node scan.js
```

您可以在這裡找到這個範例程式碼 GitHub。

使用 DynamoDB 用戶端

這個 Node.js 程式碼範例會說明：

- 如何使用 DynamoDB 公用程式存取 DynamoDB 資料表。

使用案例

DynamoDB 文件用戶端透過抽象屬性值的概念，簡化了使用項目的工作。這種抽象註釋作為輸入參數提供的本機 JavaScript 類型，並將帶註釋的響應數據轉換為本地類型。 JavaScript

如需有關文件用戶端的詳細資訊，請參閱上的 @aws-README sdk GitHub 如需有關使用 Amazon DynamoDB 進行程式設計的詳細資訊，請參閱 Amazon DynamoDB 開發人員指南 JavaScript中的使用程式設計。

在此範例中，您可以使用一系列 Node.js 模組，使用 DynamoDB 公用程式在 DynamoDB 表上執行基本操作。程式碼會使用 SDK for JavaScript 來查詢和掃描使用 DynamoDB 文件用戶端類別的下列方法的資料表：

- GetCommand
- PutCommand
- UpdateCommand
- QueryCommand
- DeleteCommand

如需有關設定 DynamoDB 文件用戶端的詳細資訊，請參閱 @aws-sdk

先決條件任務

若要設定和執行此範例，請先完成這些任務：

- 設置項目環境以運行這些 Node.js 示例，並安裝所需 AWS SDK for JavaScript 的第三方模塊。按照上的說明進行操作 GitHub。
- 透過使用者登入資料建立共用組態檔。如需有關提供共用認證檔案的詳細資訊，請參閱《工具參考指南》中的共用設定AWS SDKs和認證檔案。
- 建立一個您可以存取其項目的 DynamoDB 表格。如需有關使用的建立 DynamoDB 資料表的詳細資訊 JavaScript，SDK請參閱。在 DynamoDB 中建立和使用資料表您也可以使用 DynamoDB 主控台來建立資料表。

> **Important**
>
> 這些範例使用 ECMAScript6 (ES6)。這需要 Node.js 版本 13.x 或更高版本。要下載並安裝最新版本的 Node.js，請參閱 Node.js 下載。 。
> 但是，如果您更喜歡使用 CommonJS 語法，請參閱JavaScript ES6/共同語法。

> **Note**
>
> 如需這些範例中使用之資料類型的詳細資訊，請參閱 Amazon DynamoDB 中支援的資料類型和命名規則。

從資料表取得項目

以檔名 `get.js` 建立一個 Node.js 模組。請務必SDK如先前所示設定，包括安裝所需的用戶端和套件。這包括`@aws-sdk/lib-dynamodb`提供文件用戶端功能的程式庫套件`@aws-sdk/client-dynamodb`。接下來，在創建文檔客戶端期間設置配置，如下所示用於編組和解組-作為可選的第二個參數。接下來，建立用戶端。現在創建一個包含所需參數的JSON對象從表中獲取一個項目，在這個例子中包括表的名稱，該表中的哈希鍵的名稱，以及要獲取的項目的哈希鍵的值。呼叫 DynamoDB 文件用戶端的`GetCommand`方法。

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { DynamoDBDocumentClient, GetCommand } from "@aws-sdk/lib-dynamodb";
```

```
const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new GetCommand({
    TableName: "AngryAnimals",
    Key: {
      CommonName: "Shoebill",
    },
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

若要執行範例，請在命令提示字元中輸入下列命令。

```
node get.js
```

您可以在這裡找到這個範例程式碼 GitHub。

將項目放置在資料表中

以檔名 put.js 建立一個 Node.js 模組。請務必SDK如先前所示設定，包括安裝所需的用戶端和套件。這包括@aws-sdk/lib-dynamodb提供文件用戶端功能的程式庫套件@aws-sdk/client-dynamodb。接下來，在創建文檔客戶端期間設置配置，如下所示用於編組和解組-作為可選的第二個參數。接下來，建立用戶端。建立JSON物件，其中包含將項目寫入資料表所需的參數，在此範例中包含資料表的名稱，以及要新增或更新之項目的描述，其中包括雜湊鍵和 value，以及要在項目上設定之屬性的名稱和值。呼叫 DynamoDB 文件用戶端的PutCommand方法。

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { PutCommand, DynamoDBDocumentClient } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new PutCommand({
    TableName: "HappyAnimals",
    Item: {
      CommonName: "Shiba Inu",
```

```
    },
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

若要執行範例，請在命令提示字元中輸入下列命令。

```
node put.js
```

您可以在這裡找到這個範例程式碼 GitHub。

在資料表中更新項目

以檔名 update.js 建立一個 Node.js 模組。請務必SDK如先前所示設定，包括安裝所需的用戶端和套件。這包括@aws-sdk/lib-dynamodb提供文件用戶端功能的程式庫套件@aws-sdk/client-dynamodb。接下來，在創建文檔客戶端期間設置配置，如下所示用於編組和解組-作為可選的第二個參數。接下來，建立用戶端。建立一個JSON物件，其中包含將項目寫入表格所需的參數，在此範例中包括表格名稱、要更新的項目的金鑰、一組定義UpdateExpressions要使用在ExpressionAttributeValues參數中指派值的權杖更新項目的屬性。呼叫 DynamoDB Document Client 的UpdateCommand方法。

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { DynamoDBDocumentClient, UpdateCommand } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new UpdateCommand({
    TableName: "Dogs",
    Key: {
      Breed: "Labrador",
    },
    UpdateExpression: "set Color = :color",
    ExpressionAttributeValues: {
      ":color": "black",
    },
    ReturnValues: "ALL_NEW",
```

```
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

若要執行範例，請在命令提示字元中輸入下列命令。

```
node update.js
```

您可以在這裡找到這個範例程式碼 GitHub。

查詢資料表

以檔名 query.js 建立一個 Node.js 模組。請務必SDK如先前所示設定，包括安裝所需的用戶端和套件。這包括@aws-sdk/lib-dynamodb提供文件用戶端功能的程式庫套件@aws-sdk/client-dynamodb。建立一個包含查詢資料表所需之參數的JSON物件，在此範例中包含資料表名稱、查詢ExpressionAttributeValues所需的資料表名稱，以及使用KeyConditionExpression這些值來定義查詢傳回的項目。呼叫 DynamoDB 文件用戶端的QueryCommand方法。

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { QueryCommand, DynamoDBDocumentClient } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new QueryCommand({
    TableName: "CoffeeCrop",
    KeyConditionExpression:
      "OriginCountry = :originCountry AND RoastDate > :roastDate",
    ExpressionAttributeValues: {
      ":originCountry": "Ethiopia",
      ":roastDate": "2023-05-01",
    },
    ConsistentRead: true,
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
```

```
};
```

若要執行範例，請在命令提示字元中輸入下列命令。

```
node query.js
```

您可以在這裡找到這個範例程式碼 GitHub。

從資料表中刪除項目

以檔名 `delete.js` 建立一個 Node.js 模組。請務必SDK如先前所示設定，包括安裝所需的用戶端和套件。這包括`@aws-sdk/lib-dynamodb`提供文件用戶端功能的程式庫套件`@aws-sdk/client-dynamodb`。接下來，在創建文檔客戶端期間設置配置，如下所示用於編組和解組-作為可選的第二個參數。接下來，建立用戶端。若要存取 DynamoDB，請建立一個DynamoDB物件。創建一個JSON對象，其中包含刪除表中的項目所需的參數，在此示例中包括表的名稱以及要刪除的項目的 hashkey 的名稱和值。呼叫 DynamoDB 文件用戶端的`DeleteCommand`方法。

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { DynamoDBDocumentClient, DeleteCommand } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new DeleteCommand({
    TableName: "Sodas",
    Key: {
      Flavor: "Cola",
    },
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

若要執行範例，請在命令提示字元中輸入下列命令。

```
node delete.js
```

您可以在這裡找到這個範例程式碼 GitHub。

AWS Elemental MediaConvert 範例

AWS Elemental MediaConvert 是一款檔案型視訊轉碼服務，具備廣播級功能。您可以使用它來建立用於廣播和 video-on-demand (VOD) 在網際網路上傳送的資產。如需詳細資訊，請參閱 AWS Elemental MediaConvert 使用者指南。

的 JavaScript API 會透過`MediaConvert`用戶端類別公開。 MediaConvert 有關更多信息，請參閱 API 參考 MediaConvert中的類：。

主題

- 獲取特定於區域的端點 MediaConvert
- 在以下位置建立和管理轉碼工作 MediaConvert
- 使用工作範本 MediaConvert

獲取特定於區域的端點 MediaConvert

這個 Node.js 程式碼範例會說明：

- 如何從中檢索特定於區域的端點。 MediaConvert

該方案

在此範例中，您可以使用 Node.js 模組呼叫 MediaConvert 和擷取區域特定端點。您可以從服務默認端點檢索端點 URL，因此尚不需要特定於區域的端點。程式碼會使用 SDK JavaScript 來擷取此端點，使用用 MediaConvert 戶端類別的這個方法：

- `DescribeEndpointsCommand`

必要工作

若要設定和執行此範例，請先完成這些任務：

- 設置項目環境以運行這些節點 TypeScript 示例，並安裝所需AWS SDK for JavaScript的第三方模塊。按照上的說明進行操作 GitHub。

- 透過使用者登入資料建立共用組態檔。有關提供共用認證檔案的詳細資訊，請參閱 AWSSDK 和工具參考指南中的共用設定和認證檔案。
- 建立可存 MediaConvert 取輸入檔案和存放輸出檔案的 Amazon S3 儲存貯體的 IAM 角色。如需詳細資訊，請參閱AWS Elemental MediaConvert使用指南中的設定 IAM 許可。

> **Important**
>
> 此範例使用電子郵件密碼 6 (ES6)。這需要 Node.js 版本 13.x 或更高版本。要下載並安裝最新版本的 Node.js，請參閱 Node.js 下載。 。
> 但是，如果您更喜歡使用 CommonJS 語法，請參閱JavaScript ES6/共同語法。

取得您的端點網址

創建一個libs目錄，並使用文件名創建一個 Node.js 模塊emcClientGet.js。將下面的代碼複製並粘貼到其中，以創建 MediaConvert 客戶端對象。以您的##取代「AWS地區」。

```
import { MediaConvertClient } from "@aws-sdk/client-mediaconvert";
// Set the AWS Region.
const REGION = "REGION";
//Set the MediaConvert Service Object
const emcClientGet = new MediaConvertClient({ region: REGION });
export { emcClientGet };
```

您可以在這裡找到這個範例程式碼 GitHub。

以檔名 emc_getendpoint.js 建立一個 Node.js 模組。請務必如先前所示設定 SDK，包括安裝所需的用戶端和套件。

創建一個對象，為 MediaConvert 客戶端類的DescribeEndpointsCommand方法傳遞空的請求參數。然後呼叫 DescribeEndpointsCommand 方法。

```
// Import required AWS-SDK clients and commands for Node.js
import { DescribeEndpointsCommand } from "@aws-sdk/client-mediaconvert";
import { emcClientGet } from "./libs/emcClientGet.js";

//set the parameters.
const params = { MaxResults: 0 };
```

```
const run = async () => {
  try {
    // Create a new service object and set MediaConvert to customer endpoint
    const data = await emcClientGet.send(new DescribeEndpointsCommand(params));
    console.log("Your MediaConvert endpoint is ", data.Endpoints);
    return data;
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

若要執行範例，請在命令提示字元中輸入下列命令。

```
node emc_getendpoint.js
```

您可以在這裡找到這個範例程式碼 GitHub。

在以下位置建立和管理轉碼工作 MediaConvert

這個 Node.js 程式碼範例會說明：

- 如何指定要搭配使用的區域特定端點。 MediaConvert
- 如何在 MediaConvert.
- 如何取消轉碼任務。
- 如何擷取已完成轉碼任務的 JSON。
- 如何擷取高達 20 個最近建立任務的 JSON 陣列。

該方案

在此範例中，您可以使用 Node.js 模組來呼叫 MediaConvert 以建立和管理轉碼工作。程式碼會使用 SDK JavaScript 來執行這項作業，方法是使用用 MediaConvert 戶端類別的下列方法：

- `CreateJobCommand`

- `CancelJobCommand`
- `GetJobCommand`
- `ListJobsCommand`

必要工作

若要設定和執行此範例，請先完成這些任務：

- 設置項目環境以運行這些節點 TypeScript 示例，並安裝所需AWS SDK for JavaScript的第三方模塊。按照上的說明進行操作 GitHub。
- 透過使用者登入資料建立共用組態檔。有關提供共用認證檔案的詳細資訊，請參閱 AWSSDK 和工具參考指南中的共用設定和認證檔案。
- 建立和設定 Amazon S3 儲存貯體，為任務輸入檔案和輸出檔案提供儲存。如需詳細資訊，請參閱《AWS Elemental MediaConvert使用指南》中的為檔案建立儲存空間
- 將輸入影片上傳到您佈建用於輸入儲存的 Amazon S3 儲存貯體。如需支援的輸入視訊轉碼器和容器清單，請參閱AWS Elemental MediaConvert使用指南中支援的輸入轉碼器和容器。
- 建立可存 MediaConvert 取輸入檔案和存放輸出檔案的 Amazon S3 儲存貯體的 IAM 角色。如需詳細資訊，請參閱AWS Elemental MediaConvert使用指南中的設定 IAM 許可。

> **Important**
>
> 此範例使用電子郵件密碼 6 (ES6)。這需要 Node.js 版本 13.x 或更高版本。要下載並安裝最新版本的 Node.js，請參閱 Node.js 下載。 。
> 但是，如果您更喜歡使用 CommonJS 語法，請參閱JavaScript ES6/共同語法。

設定軟體開發套件

如先前所示設定 SDK，包括下載所需的用戶端和套件。由於每個帳戶都 MediaConvert 使用自訂端點，因此您還必須將用戶`MediaConvert`端類別設定為使用區域特定端點。若要這麼做，請在 `mediaconvert(endpoint)` 上設定 `endpoint` 參數。

```
// Import required AWS-SDK clients and commands for Node.js
import { CreateJobCommand } from "@aws-sdk/client-mediaconvert";
import { emcClient } from "./libs/emcClient.js";
```

定義簡單的轉碼工作

創建一個libs目錄，並使用文件名創建一個 Node.js 模塊emcClient.js。將下面的代碼複製並粘貼到其中，以創建 MediaConvert 客戶端對象。以您的*##*取代「AWS地區」。將 END *POINT* 取代為您的 MediaConvert 帳戶端點，您可以在主控台的 [帳戶] 頁面上使用該 MediaConvert 端點。

```
import { MediaConvertClient } from "@aws-sdk/client-mediaconvert";
// Set the account end point.
const ENDPOINT = {
  endpoint: "https://ENDPOINT_UNIQUE_STRING.mediaconvert.REGION.amazonaws.com",
};
// Set the MediaConvert Service Object
const emcClient = new MediaConvertClient(ENDPOINT);
export { emcClient };
```

您可以在這裡找到這個範例程式碼 GitHub。

以檔名 emc_createjob.js 建立一個 Node.js 模組。請務必如先前所示設定 SDK，包括安裝所需的用戶端和套件。建立 JSON，定義轉碼任務參數。

這些參數相當詳細。您可以使用AWS Elemental MediaConvert主控台產生 JSON Job 參數，方法是在主控台中選擇工作設定，然後選擇 [工作] 區段底部的 [顯示工作 JSON]。此範例顯示簡單任務的 JSON。

> **Note**
>
> *######## MediaConvert ###### IAM_ROLE_ARN ### IAM ########## (ARN)### _ BUCKET_NAME #########-###"s3://OUTPUT_BUCKET_NAME/################## ######-###" s3://INPUT_BUCKET/FILE_NAME##*

```
const params = {
  Queue: "JOB_QUEUE_ARN", //JOB_QUEUE_ARN
  UserMetadata: {
    Customer: "Amazon",
  },
  Role: "IAM_ROLE_ARN", //IAM_ROLE_ARN
  Settings: {
    OutputGroups: [
      {
        Name: "File Group",
```

```
        OutputGroupSettings: {
          Type: "FILE_GROUP_SETTINGS",
          FileGroupSettings: {
            Destination: "OUTPUT_BUCKET_NAME", //OUTPUT_BUCKET_NAME, e.g., "s3://
BUCKET_NAME/"
          },
        },
        Outputs: [
          {
            VideoDescription: {
              ScalingBehavior: "DEFAULT",
              TimecodeInsertion: "DISABLED",
              AntiAlias: "ENABLED",
              Sharpness: 50,
              CodecSettings: {
                Codec: "H_264",
                H264Settings: {
                  InterlaceMode: "PROGRESSIVE",
                  NumberReferenceFrames: 3,
                  Syntax: "DEFAULT",
                  Softness: 0,
                  GopClosedCadence: 1,
                  GopSize: 90,
                  Slices: 1,
                  GopBReference: "DISABLED",
                  SlowPal: "DISABLED",
                  SpatialAdaptiveQuantization: "ENABLED",
                  TemporalAdaptiveQuantization: "ENABLED",
                  FlickerAdaptiveQuantization: "DISABLED",
                  EntropyEncoding: "CABAC",
                  Bitrate: 5000000,
                  FramerateControl: "SPECIFIED",
                  RateControlMode: "CBR",
                  CodecProfile: "MAIN",
                  Telecine: "NONE",
                  MinIInterval: 0,
                  AdaptiveQuantization: "HIGH",
                  CodecLevel: "AUTO",
                  FieldEncoding: "PAFF",
                  SceneChangeDetect: "ENABLED",
                  QualityTuningLevel: "SINGLE_PASS",
                  FramerateConversionAlgorithm: "DUPLICATE_DROP",
                  UnregisteredSeiTimecode: "DISABLED",
                  GopSizeUnits: "FRAMES",
```

```
                ParControl: "SPECIFIED",
                NumberBFramesBetweenReferenceFrames: 2,
                RepeatPps: "DISABLED",
                FramerateNumerator: 30,
                FramerateDenominator: 1,
                ParNumerator: 1,
                ParDenominator: 1,
              },
            },
            AfdSignaling: "NONE",
            DropFrameTimecode: "ENABLED",
            RespondToAfd: "NONE",
            ColorMetadata: "INSERT",
          },
          AudioDescriptions: [
            {
              AudioTypeControl: "FOLLOW_INPUT",
              CodecSettings: {
                Codec: "AAC",
                AacSettings: {
                  AudioDescriptionBroadcasterMix: "NORMAL",
                  RateControlMode: "CBR",
                  CodecProfile: "LC",
                  CodingMode: "CODING_MODE_2_0",
                  RawFormat: "NONE",
                  SampleRate: 48000,
                  Specification: "MPEG4",
                  Bitrate: 64000,
                },
              },
              LanguageCodeControl: "FOLLOW_INPUT",
              AudioSourceName: "Audio Selector 1",
            },
          ],
          ContainerSettings: {
            Container: "MP4",
            Mp4Settings: {
              CslgAtom: "INCLUDE",
              FreeSpaceBox: "EXCLUDE",
              MoovPlacement: "PROGRESSIVE_DOWNLOAD",
            },
          },
          NameModifier: "_1",
        },
```

```
        ],
      },
    ],
    AdAvailOffset: 0,
    Inputs: [
      {
        AudioSelectors: {
          "Audio Selector 1": {
            Offset: 0,
            DefaultSelection: "NOT_DEFAULT",
            ProgramSelection: 1,
            SelectorType: "TRACK",
            Tracks: [1],
          },
        },
        VideoSelector: {
          ColorSpace: "FOLLOW",
        },
        FilterEnable: "AUTO",
        PsiControl: "USE_PSI",
        FilterStrength: 0,
        DeblockFilter: "DISABLED",
        DenoiseFilter: "DISABLED",
        TimecodeSource: "EMBEDDED",
        FileInput: "INPUT_BUCKET_AND_FILENAME", //INPUT_BUCKET_AND_FILENAME, e.g.,
 "s3://BUCKET_NAME/FILE_NAME"
      },
    ],
    TimecodeConfig: {
      Source: "EMBEDDED",
    },
  },
};
```

建立轉碼工作

建立工作參數 JSON 之後，呼叫非同步run方法來叫用用MediaConvert戶端服務物件，並傳遞參數。回應 data 中會傳回所建立任務的 ID。

```
const run = async () => {
  try {
    const data = await emcClient.send(new CreateJobCommand(params));
```

```
    console.log("Job created!", data);
    return data;
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

若要執行範例，請在命令提示字元中輸入下列命令。

```
node emc_createjob.js
```

這個完整的示例代碼可以在這裡找到 GitHub。

取消轉碼工作

創建一個libs目錄，並使用文件名創建一個 Node.js 模塊emcClient.js。將下面的代碼複製並粘貼到其中，以創建 MediaConvert 客戶端對象。以您的*##*取代「AWS地區」。將 END *POINT* 取代為您的 MediaConvert 帳戶端點，您可以在主控台的 [帳戶] 頁面上使用該 MediaConvert 端點。

```
import { MediaConvertClient } from "@aws-sdk/client-mediaconvert";
// Set the account end point.
const ENDPOINT = {
  endpoint: "https://ENDPOINT_UNIQUE_STRING.mediaconvert.REGION.amazonaws.com",
};
// Set the MediaConvert Service Object
const emcClient = new MediaConvertClient(ENDPOINT);
export { emcClient };
```

您可以在這裡找到這個範例程式碼 GitHub。

以檔名 emc_canceljob.js 建立一個 Node.js 模組。請務必如先前所示設定 SDK，包括下載所需的用戶端和套件。建立包含要取消任務 ID 的 JSON。然後通過創建一個 promise 來調用MediaConvert客戶端服務對象並傳遞參數來調用該CancelJobCommand方法。在 promise 回呼中處理回應。

> **Note**
>
> 將 *JOB_ID* 取代為要取消之工作的識別碼。

```
// Import required AWS-SDK clients and commands for Node.js
import { CancelJobCommand } from "@aws-sdk/client-mediaconvert";
import { emcClient } from "./libs/emcClient.js";

// Set the parameters
const params = { Id: "JOB_ID" }; //JOB_ID

const run = async () => {
  try {
    const data = await emcClient.send(new CancelJobCommand(params));
    console.log("Job  " + params.Id + " is canceled");
    return data;
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

若要執行範例，請在命令提示字元中輸入下列命令。

```
node ec2_canceljob.js
```

您可以在這裡找到這個範例程式碼 GitHub。

列出最近的轉碼工作

創建一個`libs`目錄，並使用文件名創建一個 Node.js 模塊`emcClient.js`。將下面的代碼複製並粘貼到其中，以創建 MediaConvert 客戶端對象。以您的*##*取代「AWS地區」。將 END *POINT* 取代為您的 MediaConvert 帳戶端點，您可以在主控台的 [帳戶] 頁面上使用該 MediaConvert 端點。

```
import { MediaConvertClient } from "@aws-sdk/client-mediaconvert";
// Set the account end point.
const ENDPOINT = {
  endpoint: "https://ENDPOINT_UNIQUE_STRING.mediaconvert.REGION.amazonaws.com",
};
// Set the MediaConvert Service Object
const emcClient = new MediaConvertClient(ENDPOINT);
export { emcClient };
```

您可以在這裡找到這個範例程式碼 GitHub。

以檔名 emc_listjobs.js 建立一個 Node.js 模組。請務必如先前所示設定 SDK，包括安裝所需的用戶端和套件。

建立參數 JSON，包括用於指定要檢查之任務佇列的 ASCENDING Amazon 資源名稱 (ARN) 清DESCENDING單排序或排序清單的值，以及要包含的任務狀態。然後通過創建一個 promise 來調用MediaConvert客戶端服務對象並傳遞參數來調用該ListJobsCommand方法。

> **Note**
>
> 將 *QUEUE_ARN* 取代為要檢查的任務佇列的亞馬遜資源名稱 (ARN)，並將狀態取代為佇列的###。

```
// Import required AWS-SDK clients and commands for Node.js
import { ListJobsCommand } from "@aws-sdk/client-mediaconvert";
import { emcClient } from "./libs/emcClient.js";

// Set the parameters
const params = {
  MaxResults: 10,
  Order: "ASCENDING",
  Queue: "QUEUE_ARN",
  Status: "SUBMITTED", // e.g., "SUBMITTED"
};

const run = async () => {
  try {
    const data = await emcClient.send(new ListJobsCommand(params));
    console.log("Success. Jobs: ", data.Jobs);
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

若要執行範例，請在命令提示字元中輸入下列命令。

```
node emc_listjobs.js
```

您可以在這裡找到這個範例程式碼 GitHub。

使用工作範本 MediaConvert

這個 Node.js 程式碼範例會說明：

- 如何建立 AWS Elemental MediaConvert任務範本。
- 如何使用任務範本來建立轉譯任務。
- 如何列出所有任務範本。
- 如何刪除任務範本。

該方案

中建立轉碼工作所需的 JSON 詳細資訊，其中 MediaConvert 包含大量的設定。您可以在您在建立後續任務能用的任務範本中儲存已知良好的設定，來大幅簡化任務的建立作業。在此範例中，您可以使用 Node.js 模組 MediaConvert 來呼叫建立、使用和管理工作範本。程式碼會使用 SDK JavaScript 來執行這項作業，方法是使用用 MediaConvert戶端類別的下列方法：

- `CreateJobTemplateCommand`
- `CreateJobCommand`
- `DeleteJobTemplateCommand`
- `ListJobTemplatesCommand`

必要工作

若要設定和執行此範例，請先完成這些任務：

- 設置項目環境以運行這些節點 TypeScript 示例，並安裝所需AWS SDK for JavaScript的第三方模塊。按照上的說明進行操作 GitHub。
- 透過使用者登入資料建立共用組態檔。有關提供共用認證檔案的詳細資訊，請參閱 AWSSDK 和工具參考指南中的共用設定和認證檔案。
- 建立可存 MediaConvert 取輸入檔案和存放輸出檔案的 Amazon S3 儲存貯體的 IAM 角色。如需詳細資訊，請參閱AWS Elemental MediaConvert使用指南中的設定 IAM 許可。

> **Important**
>
> 這些示例使用電子信息密碼 6（ES6）。這需要 Node.js 版本 13.x 或更高版本。要下載並安裝最新版本的 Node.js，請參閱 Node.js 下載。 。
> 但是，如果您更喜歡使用 CommonJS 語法，請參閱JavaScript ES6/共同語法。

建立工作樣板

創建一個`libs`目錄，並使用文件名創建一個 Node.js 模塊`emcClient.js`。將下面的代碼複製並粘貼到其中，以創建 MediaConvert 客戶端對象。以您的*##*取代「AWS地區」。將 END *POINT* 取代為您的 MediaConvert 帳戶端點，您可以在主控台的 [帳戶] 頁面上使用該 MediaConvert 端點。

```
import { MediaConvertClient } from "@aws-sdk/client-mediaconvert";
// Set the account end point.
const ENDPOINT = {
  endpoint: "https://ENDPOINT_UNIQUE_STRING.mediaconvert.REGION.amazonaws.com",
};
// Set the MediaConvert Service Object
const emcClient = new MediaConvertClient(ENDPOINT);
export { emcClient };
```

您可以在這裡找到這個範例程式碼 GitHub。

以檔名 `emc_create_jobtemplate.js` 建立一個 Node.js 模組。請務必如先前所示設定 SDK，包括安裝所需的用戶端和套件。

為範本建立指定 JSON 參數。您可以使用來自先前成功任務的多數 JSON 參數，來在範本中指定 `Settings` 值。此範例使用來自 在以下位置建立和管理轉碼工作 MediaConvert的任務設定。

通過創建一個承諾調用`MediaConvert`客戶端服務對象，並傳遞參數調用該`CreateJobTemplateCommand`方法。

> **Note**
>
> 將 *JOB_QUEE_ARN* 取代為要檢查的任務佇列的亞馬遜資源名稱 (ARN)，並以目的地 Amazon S3 儲存貯體的名稱取代 *BUCKET_NAME*-例如，"s3://BUCKET_NAME/」。

```
// Import required AWS-SDK clients and commands for Node.js
```

```
import { CreateJobTemplateCommand } from "@aws-sdk/client-mediaconvert";
import { emcClient } from "./libs/emcClient.js";

const params = {
  Category: "YouTube Jobs",
  Description: "Final production transcode",
  Name: "DemoTemplate",
  Queue: "JOB_QUEUE_ARN", //JOB_QUEUE_ARN
  Settings: {
    OutputGroups: [
      {
        Name: "File Group",
        OutputGroupSettings: {
          Type: "FILE_GROUP_SETTINGS",
          FileGroupSettings: {
            Destination: "BUCKET_NAME", // BUCKET_NAME e.g., "s3://BUCKET_NAME/"
          },
        },
        Outputs: [
          {
            VideoDescription: {
              ScalingBehavior: "DEFAULT",
              TimecodeInsertion: "DISABLED",
              AntiAlias: "ENABLED",
              Sharpness: 50,
              CodecSettings: {
                Codec: "H_264",
                H264Settings: {
                  InterlaceMode: "PROGRESSIVE",
                  NumberReferenceFrames: 3,
                  Syntax: "DEFAULT",
                  Softness: 0,
                  GopClosedCadence: 1,
                  GopSize: 90,
                  Slices: 1,
                  GopBReference: "DISABLED",
                  SlowPal: "DISABLED",
                  SpatialAdaptiveQuantization: "ENABLED",
                  TemporalAdaptiveQuantization: "ENABLED",
                  FlickerAdaptiveQuantization: "DISABLED",
                  EntropyEncoding: "CABAC",
                  Bitrate: 5000000,
                  FramerateControl: "SPECIFIED",
                  RateControlMode: "CBR",
```

```
            CodecProfile: "MAIN",
            Telecine: "NONE",
            MinIInterval: 0,
            AdaptiveQuantization: "HIGH",
            CodecLevel: "AUTO",
            FieldEncoding: "PAFF",
            SceneChangeDetect: "ENABLED",
            QualityTuningLevel: "SINGLE_PASS",
            FramerateConversionAlgorithm: "DUPLICATE_DROP",
            UnregisteredSeiTimecode: "DISABLED",
            GopSizeUnits: "FRAMES",
            ParControl: "SPECIFIED",
            NumberBFramesBetweenReferenceFrames: 2,
            RepeatPps: "DISABLED",
            FramerateNumerator: 30,
            FramerateDenominator: 1,
            ParNumerator: 1,
            ParDenominator: 1,
          },
        },
        AfdSignaling: "NONE",
        DropFrameTimecode: "ENABLED",
        RespondToAfd: "NONE",
        ColorMetadata: "INSERT",
      },
      AudioDescriptions: [
        {
          AudioTypeControl: "FOLLOW_INPUT",
          CodecSettings: {
            Codec: "AAC",
            AacSettings: {
              AudioDescriptionBroadcasterMix: "NORMAL",
              RateControlMode: "CBR",
              CodecProfile: "LC",
              CodingMode: "CODING_MODE_2_0",
              RawFormat: "NONE",
              SampleRate: 48000,
              Specification: "MPEG4",
              Bitrate: 64000,
            },
          },
          LanguageCodeControl: "FOLLOW_INPUT",
          AudioSourceName: "Audio Selector 1",
        },
```

```
            ],
            ContainerSettings: {
              Container: "MP4",
              Mp4Settings: {
                CslgAtom: "INCLUDE",
                FreeSpaceBox: "EXCLUDE",
                MoovPlacement: "PROGRESSIVE_DOWNLOAD",
              },
            },
            NameModifier: "_1",
          },
        ],
      },
    ],
    AdAvailOffset: 0,
    Inputs: [
      {
        AudioSelectors: {
          "Audio Selector 1": {
            Offset: 0,
            DefaultSelection: "NOT_DEFAULT",
            ProgramSelection: 1,
            SelectorType: "TRACK",
            Tracks: [1],
          },
        },
        VideoSelector: {
          ColorSpace: "FOLLOW",
        },
        FilterEnable: "AUTO",
        PsiControl: "USE_PSI",
        FilterStrength: 0,
        DeblockFilter: "DISABLED",
        DenoiseFilter: "DISABLED",
        TimecodeSource: "EMBEDDED",
      },
    ],
    TimecodeConfig: {
      Source: "EMBEDDED",
    },
  },
};

const run = async () => {
```

```
  try {
    // Create a promise on a MediaConvert object
    const data = await emcClient.send(new CreateJobTemplateCommand(params));
    console.log("Success!", data);
    return data;
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

若要執行範例，請在命令提示字元中輸入下列命令。

```
node emc_create_jobtemplate.js
```

您可以在這裡找到這個範例程式碼 GitHub。

從工作範本建立轉碼工作

創建一個libs目錄，並使用文件名創建一個 Node.js 模塊emcClient.js。將下面的代碼複製並粘貼到其中，以創建 MediaConvert 客戶端對象。以您的*##*取代「AWS地區」。將 END *POINT* 取代為您的 MediaConvert 帳戶端點，您可以在主控台的 [帳戶] 頁面上使用該 MediaConvert 端點。

```
import { MediaConvertClient } from "@aws-sdk/client-mediaconvert";
// Set the account end point.
const ENDPOINT = {
  endpoint: "https://ENDPOINT_UNIQUE_STRING.mediaconvert.REGION.amazonaws.com",
};
// Set the MediaConvert Service Object
const emcClient = new MediaConvertClient(ENDPOINT);
export { emcClient };
```

您可以在這裡找到這個範例程式碼 GitHub。

以檔名 emc_template_createjob.js 建立一個 Node.js 模組。請務必如先前所示設定 SDK，包括安裝所需的用戶端和套件。

建立任務建立參數 JSON，其中包含要用的任務範本名稱，以及專屬於您在建立之任務的要使用Settings。然後通過創建一個 promise 來調用MediaConvert客戶端服務對象並傳遞參數來調用該CreateJobsCommand方法。

> **Note**
>
> *# JOB_QUEE_ARN ######################ARN### KEY_PAIR_NAME ######## ###########ARN################ARN##########-####s3://BUCKET_NAME/FILE_NAME##*

```
// Import required AWS-SDK clients and commands for Node.js
import { CreateJobCommand } from "@aws-sdk/client-mediaconvert";
import { emcClient } from "./libs/emcClient.js";

const params = {
  Queue: "QUEUE_ARN", //QUEUE_ARN
  JobTemplate: "TEMPLATE_NAME", //TEMPLATE_NAME
  Role: "ROLE_ARN", //ROLE_ARN
  Settings: {
    Inputs: [
      {
        AudioSelectors: {
          "Audio Selector 1": {
            Offset: 0,
            DefaultSelection: "NOT_DEFAULT",
            ProgramSelection: 1,
            SelectorType: "TRACK",
            Tracks: [1],
          },
        },
        VideoSelector: {
          ColorSpace: "FOLLOW",
        },
        FilterEnable: "AUTO",
        PsiControl: "USE_PSI",
        FilterStrength: 0,
        DeblockFilter: "DISABLED",
        DenoiseFilter: "DISABLED",
        TimecodeSource: "EMBEDDED",
        FileInput: "INPUT_BUCKET_AND_FILENAME", //INPUT_BUCKET_AND_FILENAME, e.g.,
 "s3://BUCKET_NAME/FILE_NAME"
      },
    ],
  },
};
```

```
const run = async () => {
  try {
    const data = await emcClient.send(new CreateJobCommand(params));
    console.log("Success! ", data);
    return data;
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

若要執行範例，請在命令提示字元中輸入下列命令。

```
node emc_template_createjob.js
```

您可以在這裡找到這個範例程式碼 GitHub。

列出您的工作模板

創建一個libs目錄，並使用文件名創建一個 Node.js 模塊emcClient.js。將下面的代碼複製並粘貼到其中，以創建 MediaConvert 客戶端對象。以您的*##*取代「AWS地區」。將 END *POINT* 取代為您的 MediaConvert 帳戶端點，您可以在主控台的 [帳戶] 頁面上使用該 MediaConvert 端點。

```
import { MediaConvertClient } from "@aws-sdk/client-mediaconvert";
// Set the account end point.
const ENDPOINT = {
  endpoint: "https://ENDPOINT_UNIQUE_STRING.mediaconvert.REGION.amazonaws.com",
};
// Set the MediaConvert Service Object
const emcClient = new MediaConvertClient(ENDPOINT);
export { emcClient };
```

您可以在這裡找到這個範例程式碼 GitHub。

以檔名 emc_listtemplates.js 建立一個 Node.js 模組。請務必如先前所示設定 SDK，包括安裝所需的用戶端和套件。

建立物件，傳遞 MediaConvert 用戶端類別的 listTemplates 方法的空請求參數。包含值來判斷要列出哪些範本 (NAME, CREATION_DATE, SYSTEM)、要列出多少以及這些範本的排序。要調用該ListTemplatesCommand方法，請創建一個承諾調用 MediaConvert客戶端服務對象，並傳遞參數。

```
// Import required AWS-SDK clients and commands for Node.js
import { ListJobTemplatesCommand } from "@aws-sdk/client-mediaconvert";
import { emcClient } from "./libs/emcClient.js";

const params = {
  ListBy: "NAME",
  MaxResults: 10,
  Order: "ASCENDING",
};

const run = async () => {
  try {
    const data = await emcClient.send(new ListJobTemplatesCommand(params));
    console.log("Success ", data.JobTemplates);
    return data;
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

若要執行範例，請在命令提示字元中輸入下列命令。

```
node emc_listtemplates.js
```

您可以在這裡找到這個範例程式碼 GitHub。

刪除工作樣板

創建一個libs目錄，並使用文件名創建一個 Node.js 模塊emcClient.js。將下面的代碼複製並粘貼到其中，以創建 MediaConvert 客戶端對象。以您的*##*取代「AWS地區」。將 END *POINT* 取代為您的 MediaConvert 帳戶端點，您可以在主控台的 [帳戶] 頁面上使用該 MediaConvert 端點。

```
import { MediaConvertClient } from "@aws-sdk/client-mediaconvert";
// Set the account end point.
const ENDPOINT = {
  endpoint: "https://ENDPOINT_UNIQUE_STRING.mediaconvert.REGION.amazonaws.com",
};
// Set the MediaConvert Service Object
const emcClient = new MediaConvertClient(ENDPOINT);
```

```
export { emcClient };
```

您可以在這裡找到這個範例程式碼 GitHub。

以檔名 emc_deletetemplate.js 建立一個 Node.js 模組。請務必如先前所示設定 SDK，包括安裝所需的用戶端和套件。

建立物件，為 MediaConvert 用戶端類別的 DeleteJobTemplateCommand 方法傳遞您要刪除做為參數的任務範本名稱。要調用該DeleteJobTemplateCommand方法，請創建一個承諾調用 MediaConvert 客戶端服務對象，並傳遞參數。

```
// Import required AWS-SDK clients and commands for Node.js
import { DeleteJobTemplateCommand } from "@aws-sdk/client-mediaconvert";
import { emcClient } from "./libs/emcClient.js";

// Set the parameters
const params = { Name: "test" }; //TEMPLATE_NAME

const run = async () => {
  try {
    const data = await emcClient.send(new DeleteJobTemplateCommand(params));
    console.log(
      "Success, template deleted! Request ID:",
      data.$metadata.requestId,
    );
    return data;
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

若要執行範例，請在命令提示字元中輸入下列命令。

```
node emc_deletetemplate.js
```

您可以在這裡找到這個範例程式碼 GitHub。

AWS Lambda 範例

AWS Lambda是一種無伺服器運算服務，可讓您在不佈建或管理伺服器的情況下執行程式碼、建立可感知工作負載的叢集擴展邏輯、維護事件整合或管理執行階段。

的 JavaScript API 會透過LambdaService用戶端類別公開。AWS Lambda

以下是示範如何透過 AWS SDK for JavaScript v3 建立和使用 Lambda 函數的範例清單：

- 使用 API Gateway 叫用 Lambda
- 建立排程事件以執行AWS Lambda功能

亞馬遜萊克斯例

Amazon Lex 是一種使用語音和文字在應用程式中建立交談界面的AWS服務。

亞馬遜 Lex 的 JavaScript API 是透過 L ex 執行階段服務用戶端類別公開。

- 建立 Amazon Lex 聊天機器人

Amazon Polly 的例子

這個 Node.js 程式碼範例會說明：

- 將使用 Amazon Polly 錄製的音頻上傳到 Amazon S3

該方案

在此範例中，使用一系列 Node.js 模組，使用 Amazon S3 用戶端類別的下列方法，將使用 Amazon Polly 錄製的音訊自動上傳到 Amazon S3：

- `StartSpeechSynthesisTaskCommand`

必要工作

若要設定和執行此範例，您必須先完成這些任務：

- 按照上的說明設置專案環境以執行 Node JavaScript 範例 GitHub。

- 透過使用者登入資料建立共用組態檔。有關提供共用認證檔案的詳細資訊，請參閱 AWS SDK 和工具參考指南中的共用設定和認證檔案。
- 建立 AWS Identity and Access Management (IAM) 未經驗證的 Amazon Cognito 使用者角色波莉：SynthesizeSpeech 許可，以及附加 IAM 角色的 Amazon Cognito 身分集區。以下使用建立 AWS 資源 AWS CloudFormation章節說明如何建立這些資源。

> **Note**
>
> 此範例使用 Amazon Cognito，但如果您不使用 Amazon Cognito，則您的使用 AWS 者必須具有遵循 IAM 許可政策

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "mobileanalytics:PutEvents",
        "cognito-sync:*"
      ],
      "Resource": "*",
      "Effect": "Allow"
    },
    {
      "Action": "polly:SynthesizeSpeech",
      "Resource": "*",
      "Effect": "Allow"
    }
  ]
}
```

使用建立 AWS 資源 AWS CloudFormation

AWS CloudFormation 可讓您以預測和重複的方式建立和佈建 AWS 基礎結構部署。若要取得有關的更多資訊 AWS CloudFormation，請參閱AWS CloudFormation 使用者指南。

若要建立 AWS CloudFormation 堆疊：

1. 安裝並設定 AWS CLI 以下AWS CLI 使用者指南中的指示。

2. 在項目文件夾的根目錄`setup.yaml`中創建一個名為的文件，然後將此處的內容複製 GitHub到其中。

> **Note**
>
> AWS CloudFormation 範本是使用中 AWS CDK 提供的來產生的 GitHub。如需有關的詳細資訊 AWS CDK，請參閱開AWS Cloud Development Kit (AWS CDK) 發人員指南。

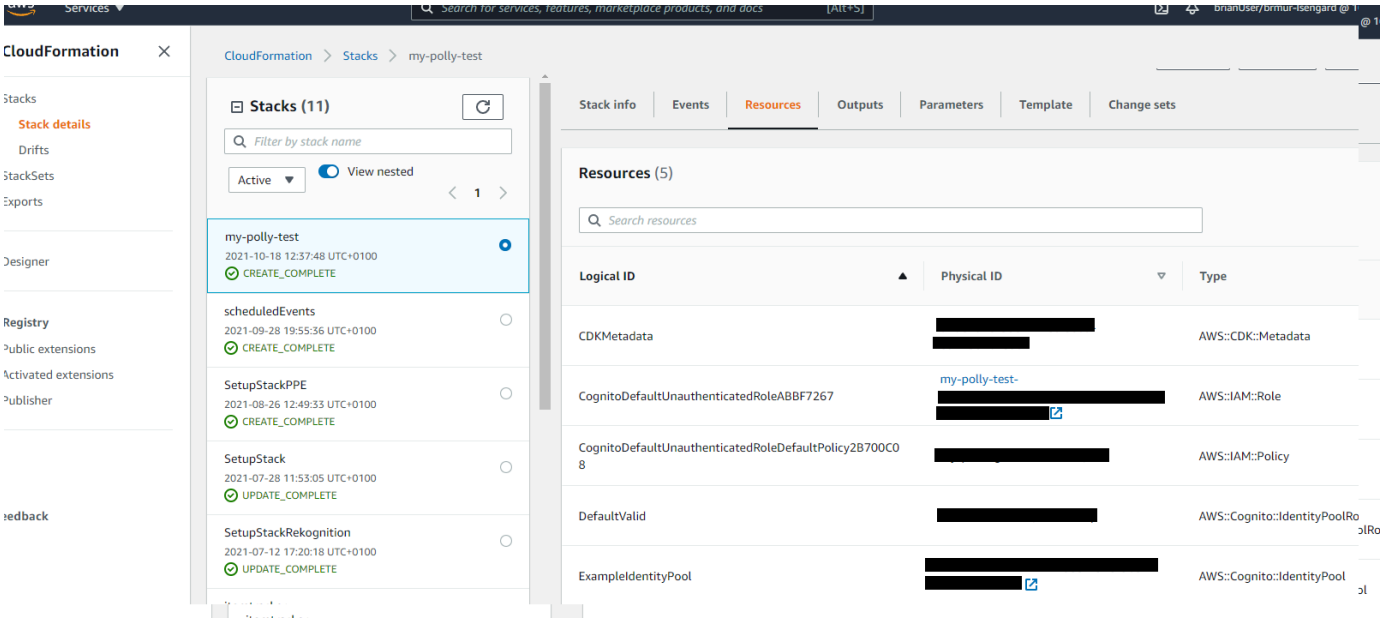
3. 從命令列執行下列命令，並以*######### STACK_NAME*。

> **Important**
>
> 堆疊名稱在 AWS 區域和 AWS 帳戶中必須是唯一的。您最多可以指定 128 個字元，且允許使用數字和連字號。

```
aws cloudformation create-stack --stack-name STACK_NAME --template-body file://
setup.yaml --capabilities CAPABILITY_IAM
```

若要取得有關指`create-stack`令參數的更多資訊，請參閱《指AWS CLI 令參考指南》和《AWS CloudFormation 使用指南》。

4. 導覽至 AWS CloudFormation 管理主控台，選擇 [堆疊]，選擇堆疊名稱，然後選擇 [資源] 索引標籤以檢視已建立資源的清單。



將使用 Amazon Polly 錄製的音頻上傳到 Amazon S3

以檔名 `polly_synthesize_to_s3.js` 建立一個 Node.js 模組。請務必如先前所示設定 SDK，包括安裝所需的用戶端和套件。`##################`要訪問 Amazon Polly，請創建一個`Polly`客戶端服務對象。將 *`#IDENTITY_POOL_ID#`* 取代為您為此`IdentityPoolId`範例建立之亞馬遜認可身分集區的範例頁面。這也傳遞給每個客戶端對象。

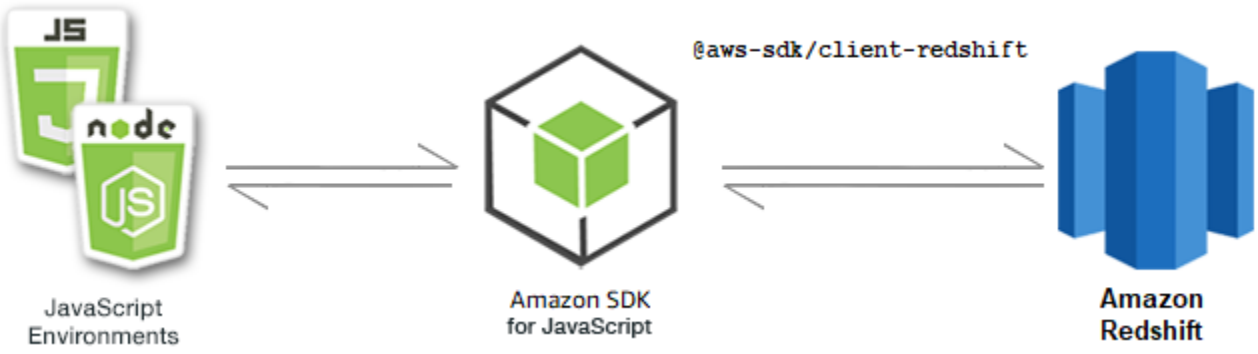
呼叫 Amazon Polly 用戶端服務物件的`StartSpeechSynthesisCommand`方法，以合成語音訊息並將其上傳到 Amazon S3 儲存貯體。

```
import { StartSpeechSynthesisTaskCommand } from "@aws-sdk/client-polly";
import { pollyClient } from "./libs/pollyClient.js";

// Create the parameters
var params = {
  OutputFormat: "mp3",
  OutputS3BucketName: "videoanalyzerbucket",
  Text: "Hello David, How are you?",
  TextType: "text",
  VoiceId: "Joanna",
  SampleRate: "22050",
};

const run = async () => {
  try {
    await pollyClient.send(new StartSpeechSynthesisTaskCommand(params));
    console.log("Success, audio file added to " + params.OutputS3BucketName);
  } catch (err) {
    console.log("Error putting object", err);
  }
};
run();
```

您可以在這裡找到此範例程式碼 GitHub。

Amazon Redshift 示例

Amazon Redshift 是一種在雲端中完全受管的 PB 級資料倉儲服務。Amazon Redshift 資料倉儲是稱為節點的運算資源集合，這些資源被組織成一個稱為叢集的群組。每個叢集皆執行 Amazon Redshift 引擎並包含一或多個資料庫。



Amazon Redshift 的 JavaScript API 是通過 Amazon Redshift 客戶端類公開。

主題

- Amazon Redshift 示例

Amazon Redshift 示例

在此範例中，使用一系列 Node.js 模組來建立、修改、描述的參數，然後使用下列用Redshift戶端類別的方法刪除 Amazon Redshift 叢集：

- `CreateClusterCommand`
- `ModifyClusterCommand`
- `DescribeClustersCommand`
- `DeleteClusterCommand`

如需有關 Amazon Redshift 使用者的詳細資訊，請參閱 Amazon Redshift 入門指南。

必要工作

若要設定和執行此範例，您必須先完成這些任務：

- 設置項目環境以運行這些節點 TypeScript 示例，並安裝所需AWS SDK for JavaScript的第三方模塊。按照上的說明進行操作 GitHub。
- 透過使用者登入資料建立共用組態檔。有關提供共用認證檔案的詳細資訊，請參閱 AWSSDK 和工具參考指南中的共用設定和認證檔案。

> **Important**
>
> 這些範例示範如何使用 ECMAScript6 (ES6) 匯入/匯出用戶端服務物件和指令。
>
> - 這需要 Node.js 版本 13.x 或更高版本。要下載並安裝最新版本的 Node.js，請參閱 Node.js 下載。 。
> - 如果您偏好使用 CommonJS 語法，請參閱 JavaScript ES6/共同語法

創建一個 Amazon Redshift 集群

此範例示範 Amazon Redshift 使用AWS SDK for JavaScript. 如需詳細資訊，請參閱CreateCluster。

> **Important**
>
> 您即將建立的叢集是即時的 (而不是在沙箱中執行)。您必須支付叢集的標準 Amazon Redshift 使用費，直到您刪除叢集為止。如果您在與建立叢集時相同的位置中刪除叢集，則總費用將最低。

創建一個`libs`目錄，並使用文件名創建一個 Node.js 模塊`redshiftClient.js`。將下面的代碼複製並粘貼到其中，以創建 Amazon Redshift 客戶端對象。以您的*##*取代「AWS地區」。

```
import  { RedshiftClient }  from  "@aws-sdk/client-redshift";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create Redshift service object.
const redshiftClient = new RedshiftClient({ region: REGION });
export { redshiftClient };
```

您可以在這裡找到此範例程式碼 GitHub。

以檔名 `redshift-create-cluster.js` 建立一個 Node.js 模組。請務必如先前所示設定 SDK，包括安裝所需的用戶端和套件。建立 Parameters 物件、指定要佈建的節點類型，以及在叢集中自動建立之資料庫執行個體的主要登入認證，最後是叢集類型。

> **Note**
>
> 以*#####*取代叢集名稱。若為*####*，請指定要佈建的節點類型，例如「dc2.large」。*MASTER #####*和 *MASTER_USER_PASSW* ORD 是叢集中資料庫執行個體主要使用者的登入認證。在*##*

#####入叢集類型。如果您指定single-node，則不需要NumberOfNodes參數。其餘的參數是可選的。

```
// Import required AWS SDK clients and commands for Node.js
import { CreateClusterCommand } from "@aws-sdk/client-redshift";
import { redshiftClient } from "./libs/redshiftClient.js";

const params = {
  ClusterIdentifier: "CLUSTER_NAME", // Required
  NodeType: "NODE_TYPE", //Required
  MasterUsername: "MASTER_USER_NAME", // Required - must be lowercase
  MasterUserPassword: "MASTER_USER_PASSWORD", // Required - must contain at least one
 uppercase letter, and one number
  ClusterType: "CLUSTER_TYPE", // Required
  IAMRoleARN: "IAM_ROLE_ARN", // Optional - the ARN of an IAM role with permissions
 your cluster needs to access other AWS services on your behalf, such as Amazon S3.
  ClusterSubnetGroupName: "CLUSTER_SUBNET_GROUPNAME", //Optional - the name of a
 cluster subnet group to be associated with this cluster. Defaults to 'default' if not
 specified.
  DBName: "DATABASE_NAME", // Optional - defaults to 'dev' if not specified
  Port: "PORT_NUMBER", // Optional - defaults to '5439' if not specified
};

const run = async () => {
  try {
    const data = await redshiftClient.send(new CreateClusterCommand(params));
    console.log(
      "Cluster " + data.Cluster.ClusterIdentifier + " successfully created",
    );
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

若要執行範例，請在命令提示字元中輸入下列命令。

```
node redshift-create-cluster.js
```

您可以在這裡找到此範例程式碼 GitHub。

修改亞 Amazon Redshift 叢集

此範例說明如何使用修改 Amazon Redshift 叢集的主要使用者密碼。AWS SDK for JavaScript若要取得有關可修改哪些其他設定的更多資訊，請參閱〈〉ModifyCluster。

創建一個libs目錄，並使用文件名創建一個 Node.js 模塊redshiftClient.js。將下面的代碼複製並粘貼到其中，以創建 Amazon Redshift 客戶端對象。以您的##取代「AWS地區」。

```
import  { RedshiftClient }  from  "@aws-sdk/client-redshift";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create Redshift service object.
const redshiftClient = new RedshiftClient({ region: REGION });
export { redshiftClient };
```

您可以在這裡找到此範例程式碼 GitHub。

以檔名 `redshift-modify-cluster.js` 建立一個 Node.js 模組。請務必如先前所示設定 SDK，包括安裝所需的用戶端和套件。指定 [AWS區域]、要修改的叢集名稱以及新的主要使用者密碼。

> **Note**
>
> 將##########，並以新的主要使用者密碼取代#######。

```
// Import required AWS SDK clients and commands for Node.js
import { ModifyClusterCommand } from "@aws-sdk/client-redshift";
import { redshiftClient } from "./libs/redshiftClient.js";

// Set the parameters
const params = {
  ClusterIdentifier: "CLUSTER_NAME",
  MasterUserPassword: "NEW_MASTER_USER_PASSWORD",
};

const run = async () => {
  try {
    const data = await redshiftClient.send(new ModifyClusterCommand(params));
    console.log("Success was modified.", data);
    return data; // For unit tests.
  } catch (err) {
```

```
    console.log("Error", err);
  }
};
run();
```

若要執行範例，請在命令提示字元中輸入下列命令。

```
node redshift-modify-cluster.js
```

您可以在這裡找到此範例程式碼 GitHub。

檢視 Amazon Redshift 叢集的詳細資訊

此範例顯示 Amazon Redshift 用AWS SDK for JavaScript. 如需選用的詳細資訊，請參閱DescribeClusters。

創建一個libs目錄，並使用文件名創建一個 Node.js 模塊redshiftClient.js。將下面的代碼複製並粘貼到其中，以創建 Amazon Redshift 客戶端對象。以您的##取代「AWS地區」。

```
import  { RedshiftClient }  from  "@aws-sdk/client-redshift";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create Redshift service object.
const redshiftClient = new RedshiftClient({ region: REGION });
export { redshiftClient };
```

您可以在這裡找到此範例程式碼 GitHub。

以檔名 redshift-describe-clusters.js 建立一個 Node.js 模組。請務必如先前所示設定 SDK，包括安裝所需的用戶端和套件。指定 [AWS區域]、要修改的叢集名稱以及新的主要使用者密碼。

> **Note**
>
> 以#####取代叢集名稱。

```
// Import required AWS SDK clients and commands for Node.js
import { DescribeClustersCommand } from "@aws-sdk/client-redshift";
import { redshiftClient } from "./libs/redshiftClient.js";
```

```
const params = {
  ClusterIdentifier: "CLUSTER_NAME",
};

const run = async () => {
  try {
    const data = await redshiftClient.send(new DescribeClustersCommand(params));
    console.log("Success", data);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

若要執行範例，請在命令提示字元中輸入下列命令。

```
node redshift-describe-clusters.js
```

您可以在這裡找到此範例程式碼 GitHub。

刪除亞 Amazon Redshift 集群

此範例顯示 Amazon Redshift 用AWS SDK for JavaScript. 若要取得有關可修改哪些其他設定的更多資訊，請參閱〈〉DeleteCluster。

創建一個libs目錄，並使用文件名創建一個 Node.js 模塊redshiftClient.js。將下面的代碼複製並粘貼到其中，以創建 Amazon Redshift 客戶端對象。以您的##取代「AWS地區」。

```
import  { RedshiftClient }  from  "@aws-sdk/client-redshift";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create Redshift service object.
const redshiftClient = new RedshiftClient({ region: REGION });
export { redshiftClient };
```

您可以在這裡找到此範例程式碼 GitHub。

使用名為的檔案建立 Node.js 模組redshift-delete-clusters.js。請務必如先前所示設定 SDK，包括安裝所需的用戶端和套件。指定 [AWS區域]、要修改的叢集名稱以及新的主要使用者密碼。指定是否要在刪除之前儲存叢集的最終快照，如果存在，則指定快照的 ID。

> **Note**
>
> 以*#####*取代叢集名稱。對於 *SkipFinalClusterSnapshot*，請指定是否在刪除叢集之前建立叢集的最終快照。如果您指定「假」，請在叢集 _SNAPSHO *T* _ID 中指定最終叢集快照的識別碼。您可以按一下 [叢集] 儀表板上叢集之 [快照] 資料欄中的連結，然後向下捲動至 [快照] 窗格，以取得此 ID。請注意，幹不`rs:`是快照 ID 的一部分。

```
// Import required AWS SDK clients and commands for Node.js
import { DeleteClusterCommand } from "@aws-sdk/client-redshift";
import { redshiftClient } from "./libs/redshiftClient.js";

const params = {
  ClusterIdentifier: "CLUSTER_NAME",
  SkipFinalClusterSnapshot: false,
  FinalClusterSnapshotIdentifier: "CLUSTER_SNAPSHOT_ID",
};

const run = async () => {
  try {
    const data = await redshiftClient.send(new DeleteClusterCommand(params));
    console.log("Success, cluster deleted. ", data);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

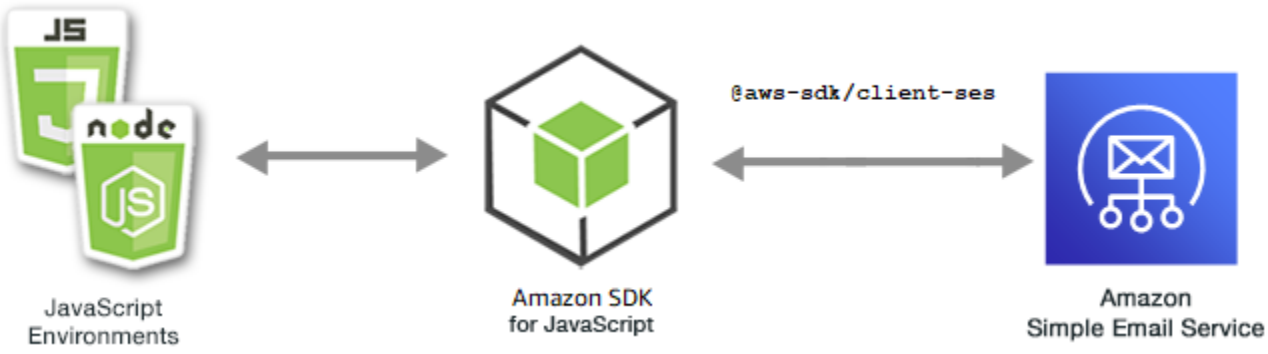
若要執行範例，請在命令提示字元中輸入下列命令。

```
node redshift-delete-cluster.js
```

您可以在這裡找到此範例程式碼 GitHub。

Amazon 簡單電子郵件服務

Amazon 簡易電子郵件服務 (AmazonSES) 是一種雲端式電子郵件傳送服務，專為協助數位行銷人員和應用程式開發人員傳送行銷、通知和交易電子郵件。這是一個可靠且經濟實惠的服務，適合透過電子郵件與客戶保持聯繫的所有規模公司使用。



對 JavaScript API於 Amazon SES 是通過SES客戶端類暴露。如需有關使用 Amazon 用SES戶端類別的詳細資訊，請參閱API參考資料SES中的 Class:。

主題

- 管理 Amazon SES 身分
- 在 Amazon 中使用電子郵件模板 SES
- 使用 Amazon 發送電子郵件 SES

管理 Amazon SES 身分

這個 Node.js 程式碼範例會說明：

- 如何驗證與 Amazon 搭配使用的電子郵件地址和網域SES。
- 如何為您的 Amazon SES 身份分配 AWS Identity and Access Management （ IAM ）政策。
- 如何列出您的 AWS 帳戶的所有 Amazon SES 身份。
- 如何刪除與 Amazon 一起使用的身份SES。

Amazon SES 身分是 Amazon SES 用來傳送電子郵件的電子郵件地址或網域。Amazon SES 要求您驗證您的電子郵件身分，確認您擁有這些身分並防止其他人使用它們。

有關如何在 Amazon 中驗證電子郵件地址和網域的詳細資訊SES，請參閱 Amazon 簡單電子郵件服務開發人員指南SES中的驗證 Amazon 中的電子郵件地址和網域。如需在 Amazon 中傳送授權的相關資訊SES，請參閱 Amazon SES 傳送授權概觀。

該方案

在此範例中，您會使用一系列 Node.js 模組來驗證和管理 Amazon SES 身分。Node.js 模組會使SDK用 JavaScript 來驗證電子郵件地址和網域，使用用SES戶端類別的下列方法：

- `ListIdentitiesCommand`
- `DeleteIdentityCommand`
- `VerifyEmailIdentityCommand`
- `VerifyDomainIdentityCommand`

必要工作

若要設定和執行此範例，您必須先完成這些任務：

- 設置項目環境以運行這些節點 TypeScript 示例，並安裝所需 AWS SDK for JavaScript 的第三方模塊。按照上的說明進行操作 GitHub。
- 透過使用者登入資料建立共用組態檔。如需有關提供共用認證檔案的詳細資訊，請參閱《工具參考指南》中的共用設定AWS SDKs和認證檔案。

> **Important**
>
> 這些範例示範如何使用 ECMAScript6 (ES6) 匯入/匯出用戶端服務物件和指令。
>
> - 這需要 Node.js 版本 13.x 或更高版本。要下載並安裝最新版本的 Node.js，請參閱 Node.js 下載。 。
> - 如果您偏好使用 CommonJS 語法，請參閱JavaScript ES6/共同語法。

列出您的身份

在此範例中，使用 Node.js 模組列出要搭配 Amazon 使用的電子郵件地址和網域SES。

創建一個`libs`目錄，並使用文件名創建一個 Node.js 模塊`sesClient.js`。將下面的代碼複製並粘貼到其中，以創建 Amazon SES 客戶端對象。Replace (取代) *`REGION`* 與您的 AWS 區域。

```
import { SESClient } from "@aws-sdk/client-ses";
// Set the AWS Region.
const REGION = "us-east-1";
```

```
// Create SES service object.
const sesClient = new SESClient({ region: REGION });
export { sesClient };
```

您可以在這裡找到這個範例程式碼 GitHub。

以檔名 ses_listidentities.js 建立一個 Node.js 模組。SDK如前所示設定，包括安裝所需的用戶端和套件。

建立物件，以傳遞 SES 用戶端類別的 ListIdentitiesCommand 方法之 IdentityType 和其他參數。要調用該ListIdentitiesCommand方法，請調用 Amazon SES 服務對象，並傳遞參數對象。

返data回的包含由參數指定的域標識IdentityType數組。

> **Note**
>
> Replace (取代) *IdentityType* 具有身份類型，可以是「EmailAddress」或「域」。

```
import { ListIdentitiesCommand } from "@aws-sdk/client-ses";
import { sesClient } from "./libs/sesClient.js";

const createListIdentitiesCommand = () =>
  new ListIdentitiesCommand({ IdentityType: "EmailAddress", MaxItems: 10 });

const run = async () => {
  const listIdentitiesCommand = createListIdentitiesCommand();

  try {
    return await sesClient.send(listIdentitiesCommand);
  } catch (err) {
    console.log("Failed to list identities.", err);
    return err;
  }
};
```

若要執行範例，請在命令提示字元中輸入下列命令。

```
node ses_listidentities.js
```

您可以在這裡找到這個範例程式碼 GitHub。

驗證電子郵件地址身分

在此範例中，請使用 Node.js 模組來驗證要與 Amazon 搭配使用的電子郵件寄件者SES。

創建一個libs目錄，並使用文件名創建一個 Node.js 模塊sesClient.js。將下面的代碼複製並粘貼到其中，以創建 Amazon SES 客戶端對象。Replace (取代) *REGION* 與您的 AWS 區域。

```
import { SESClient } from "@aws-sdk/client-ses";
// Set the AWS Region.
const REGION = "us-east-1";
// Create SES service object.
const sesClient = new SESClient({ region: REGION });
export { sesClient };
```

您可以在這裡找到這個範例程式碼 GitHub。

以檔名 ses_verifyemailidentity.js 建立一個 Node.js 模組。SDK如先前所示設定，包括下載所需的用戶端和套件。

建立物件以傳遞 SES 用戶端類別的 VerifyEmailIdentityCommand 方法之 EmailAddress 參數。要調用該VerifyEmailIdentityCommand方法，請調用 Amazon SES 客戶端服務對象，並傳遞參數。

> **Note**
>
> Replace (取代) *EMAIL_ADDRESS* 使用電子郵件地址，例如 name@example.com。

```
// Import required AWS SDK clients and commands for Node.js
import { VerifyEmailIdentityCommand } from "@aws-sdk/client-ses";
import { sesClient } from "./libs/sesClient.js";

const EMAIL_ADDRESS = "name@example.com";

const createVerifyEmailIdentityCommand = (emailAddress) => {
  return new VerifyEmailIdentityCommand({ EmailAddress: emailAddress });
};

const run = async () => {
  const verifyEmailIdentityCommand =
    createVerifyEmailIdentityCommand(EMAIL_ADDRESS);
  try {
```

```
    return await sesClient.send(verifyEmailIdentityCommand);
  } catch (err) {
    console.log("Failed to verify email identity.", err);
    return err;
  }
};
```

若要執行範例，請在命令提示字元中輸入下列命令。該域名已添加到 Amazon SES 進行驗證。

```
node ses_verifyemailidentity.js
```

您可以在這裡找到這個範例程式碼 GitHub。

驗證網域身分

在此範例中，請使用 Node.js 模組來驗證要與 Amazon 搭配使用的電子郵件網域SES。

創建一個libs目錄，並使用文件名創建一個 Node.js 模塊sesClient.js。將下面的代碼複製並粘貼到其中，以創建 Amazon SES 客戶端對象。Replace (取代) *REGION* 與您的 AWS 區域。

```
import { SESClient } from "@aws-sdk/client-ses";
// Set the AWS Region.
const REGION = "us-east-1";
// Create SES service object.
const sesClient = new SESClient({ region: REGION });
export { sesClient };
```

您可以在這裡找到這個範例程式碼 GitHub。

以檔名 ses_verifydomainidentity.js 建立一個 Node.js 模組。SDK如前所示設定，包括安裝所需的用戶端和套件。

建立物件以傳遞 SES 用戶端類別的 VerifyDomainIdentityCommand 方法之 Domain 參數。要調用該VerifyDomainIdentityCommand方法，請調用 Amazon SES 客戶端服務對象，並傳遞參數對象。

> **Note**
>
> 此範例會匯入並使用所需的 AWS Service V3 套件用戶端、V3 命令，並以異步/等待模式使用該send方法。您可以使用 V2 命令來創建此示例，而不是通過進行一些小的更改。如需詳細資訊，請參閱 使用 v3 命令。

> **Note**
>
> Replace (取代) *DOMAIN_NAME* 與域名。

```
import { VerifyDomainIdentityCommand } from "@aws-sdk/client-ses";
import {
  getUniqueName,
  postfix,
} from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { sesClient } from "./libs/sesClient.js";

/**
 * You must have access to the domain's DNS settings to complete the
 * domain verification process.
 */
const DOMAIN_NAME = postfix(getUniqueName("Domain"), ".example.com");

const createVerifyDomainIdentityCommand = () => {
  return new VerifyDomainIdentityCommand({ Domain: DOMAIN_NAME });
};

const run = async () => {
  const VerifyDomainIdentityCommand = createVerifyDomainIdentityCommand();

  try {
    return await sesClient.send(VerifyDomainIdentityCommand);
  } catch (err) {
    console.log("Failed to verify domain.", err);
    return err;
  }
};
```

若要執行範例，請在命令提示字元中輸入下列命令。該域名已添加到 Amazon SES 進行驗證。

```
node ses_verifydomainidentity.js
```

您可以在這裡找到這個範例程式碼 GitHub。

刪除身份

在此範例中，使用 Node.js 模組刪除與 Amazon 搭配使用的電子郵件地址或網域SES。

創建一個libs目錄，並使用文件名創建一個 Node.js 模塊sesClient.js。將下面的代碼複製並粘貼到其中，以創建 Amazon SES 客戶端對象。Replace (取代) *REGION* 與您的 AWS 區域。

```
import { SESClient } from "@aws-sdk/client-ses";
// Set the AWS Region.
const REGION = "us-east-1";
// Create SES service object.
const sesClient = new SESClient({ region: REGION });
export { sesClient };
```

您可以在這裡找到這個範例程式碼 GitHub。

以檔名 ses_deleteidentity.js 建立一個 Node.js 模組。SDK如前所示設定，包括安裝所需的用戶端和套件。

建立物件以傳遞 SES 用戶端類別的 DeleteIdentityCommand 方法之 Identity 參數。要調用該DeleteIdentityCommand方法，請創建一個用request於調用 Amazon SES 客戶端服務對象，並傳遞參數。

> **Note**
>
> 此範例會匯入並使用所需的 AWS Service V3 套件用戶端、V3 命令，並以異步/等待模式使用該send方法。您可以使用 V2 命令來創建此示例，而不是通過進行一些小的更改。如需詳細資訊，請參閱 使用 v3 命令。

> **Note**
>
> Replace (取代) *IDENTITY_EMAIL* 與要刪除的身份的電子郵件。

```
import { DeleteIdentityCommand } from "@aws-sdk/client-ses";
import { sesClient } from "./libs/sesClient.js";

const IDENTITY_EMAIL = "fake@example.com";

const createDeleteIdentityCommand = (identityName) => {
  return new DeleteIdentityCommand({
    Identity: identityName,
  });
```

```
};

const run = async () => {
  const deleteIdentityCommand = createDeleteIdentityCommand(IDENTITY_EMAIL);

  try {
    return await sesClient.send(deleteIdentityCommand);
  } catch (err) {
    console.log("Failed to delete identity.", err);
    return err;
  }
};
```

若要執行範例，請在命令提示字元中輸入下列命令。

```
node ses_deleteidentity.js
```

您可以在這裡找到這個範例程式碼 GitHub。

在 Amazon 中使用電子郵件模板 SES

這個 Node.js 程式碼範例會說明：

- 如何獲取所有電子郵件模板的列表。
- 如何檢索和更新電子郵件模板。
- 如何建立和刪除電子郵件範本。

Amazon SES 可讓您使用電子郵件範本傳送個人化的電子郵件訊息。有關如何在 Amazon 中建立和使用電子郵件範本的詳細資訊SES，請參閱 Amazon 簡單電子郵件服務開發人員指南SESAPI中的使用 Amazon 傳送個人化電子郵件。

該方案

在此範例中，您會使用一系列的 Node.js 模組以使用電子郵件範本。Node.js 模組會使SDK用 JavaScript 來建立和使用用SES戶端類別下列方法的電子郵件範本：

- ListTemplatesCommand
- CreateTemplateCommand
- GetTemplateCommand
- DeleteTemplateCommand
- UpdateTemplateCommand

必要工作

若要設定和執行此範例，您必須先完成這些任務：

- 設置項目環境以運行這些節點 TypeScript 示例，並安裝所需 AWS SDK for JavaScript 的第三方模塊。按照上的說明進行操作 GitHub。

- 透過使用者登入資料建立共用組態檔。如需有關提供共用認證檔案的詳細資訊，請參閱《工具參考指南》中的共用設定AWS SDKs和認證檔案。

> **Important**
>
> 這些範例示範如何使用 ECMAScript6 (ES6) 匯入/匯出用戶端服務物件和指令。
>
> - 這需要 Node.js 版本 13.x 或更高版本。要下載並安裝最新版本的 Node.js，請參閱 Node.js 下載。 。
> - 如果您偏好使用 CommonJS 語法，請參閱JavaScript ES6/共同語法。

列出您的電子郵件範

在此範例中，請使用 Node.js 模組建立要搭配 Amazon 使用的電子郵件範本SES。

創建一個`libs`目錄，並使用文件名創建一個 Node.js 模塊`sesClient.js`。將下面的代碼複製並粘貼到其中，以創建 Amazon SES 客戶端對象。Replace (取代) *`REGION`* 與您的 AWS 區域。

```
import { SESClient } from "@aws-sdk/client-ses";
// Set the AWS Region.
const REGION = "us-east-1";
// Create SES service object.
const sesClient = new SESClient({ region: REGION });
```

```
export { sesClient };
```

您可以在這裡找到這個範例程式碼 GitHub。

以檔名 ses_listtemplates.js 建立一個 Node.js 模組。SDK如前所示設定，包括安裝所需的用戶端和套件。

建立物件以傳遞 SES 用戶端類別的 ListTemplatesCommand 方法之參數。要調用該ListTemplatesCommand方法，請調用 Amazon SES 客戶端服務對象，並傳遞參數。

> **Note**
>
> 此範例會匯入並使用所需的 AWS Service V3 套件用戶端、V3 命令，並以異步/等待模式使用該send方法。您可以使用 V2 命令來創建此示例，而不是通過進行一些小的更改。如需詳細資訊，請參閱 使用 v3 命令。

```
import { ListTemplatesCommand } from "@aws-sdk/client-ses";
import { sesClient } from "./libs/sesClient.js";

const createListTemplatesCommand = (maxItems) =>
  new ListTemplatesCommand({ MaxItems: maxItems });

const run = async () => {
  const listTemplatesCommand = createListTemplatesCommand(10);

  try {
    return await sesClient.send(listTemplatesCommand);
  } catch (err) {
    console.log("Failed to list templates.", err);
    return err;
  }
};
```

若要執行範例，請在命令提示字元中輸入下列命令。Amazon SES 返回模板列表。

```
node ses_listtemplates.js
```

您可以在這裡找到這個範例程式碼 GitHub。

取得電子郵件範本

在此示例中，使用 Node.js 模塊獲取與 Amazon 一起使用的電子郵件模板SES。

創建一個libs目錄，並使用文件名創建一個 Node.js 模塊sesClient.js。將下面的代碼複製並粘貼到其中，以創建 Amazon SES 客戶端對象。Replace (取代) *REGION* 與您的 AWS 區域。

```
import { SESClient } from "@aws-sdk/client-ses";
// Set the AWS Region.
const REGION = "us-east-1";
// Create SES service object.
const sesClient = new SESClient({ region: REGION });
export { sesClient };
```

您可以在這裡找到這個範例程式碼 GitHub。

以檔名 ses_gettemplate.js 建立一個 Node.js 模組。SDK如前所示設定，包括安裝所需的用戶端和套件。

建立物件以傳遞 SES 用戶端類別的 GetTemplateCommand 方法之 TemplateName 參數。要調用該GetTemplateCommand方法，請調用 Amazon SES 客戶端服務對象，並傳遞參數。

> **Note**
>
> 此範例會匯入並使用所需的 AWS Service V3 套件用戶端、V3 命令，並以異步/等待模式使用該send方法。您可以使用 V2 命令來創建此示例，而不是通過進行一些小的更改。如需詳細資訊，請參閱 使用 v3 命令。

> **Note**
>
> Replace (取代) *TEMPLATE_NAME* 與要返回的模板的名稱。

```
import { GetTemplateCommand } from "@aws-sdk/client-ses";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { sesClient } from "./libs/sesClient.js";

const TEMPLATE_NAME = getUniqueName("TemplateName");
```

```
const createGetTemplateCommand = (templateName) =>
  new GetTemplateCommand({ TemplateName: templateName });

const run = async () => {
  const getTemplateCommand = createGetTemplateCommand(TEMPLATE_NAME);

  try {
    return await sesClient.send(getTemplateCommand);
  } catch (caught) {
    if (caught instanceof Error && caught.name === "MessageRejected") {
      /** @type { import('@aws-sdk/client-ses').MessageRejected} */
      const messageRejectedError = caught;
      return messageRejectedError;
    }
    throw caught;
  }
};
```

若要執行範例，請在命令提示字元中輸入下列命令。Amazon SES 返回模板的詳細信息。

```
node ses_gettemplate.js
```

您可以在這裡找到這個範例程式碼 GitHub。

建立電子郵件範本

在此範例中，請使用 Node.js 模組建立要搭配 Amazon 使用的電子郵件範本SES。

創建一個`libs`目錄，並使用文件名創建一個 Node.js 模塊`sesClient.js`。將下面的代碼複製並粘貼到其中，以創建 Amazon SES 客戶端對象。Replace (取代) *`REGION`* 與您的 AWS 區域。

```
import { SESClient } from "@aws-sdk/client-ses";
// Set the AWS Region.
const REGION = "us-east-1";
// Create SES service object.
const sesClient = new SESClient({ region: REGION });
export { sesClient };
```

您可以在這裡找到這個範例程式碼 GitHub。

以檔名 `ses_createtemplate.js` 建立一個 Node.js 模組。SDK如前所示設定，包括安裝所需的用戶端和套件。

建立物件以傳遞 SES 用戶端類別的 CreateTemplateCommand 方法 (包括 TemplateName、HtmlPart、SubjectPart 和 TextPart) 之參數。要調用該CreateTemplateCommand方法，請調用 Amazon SES 客戶端服務對象，並傳遞參數。

> **Note**
>
> 此範例會匯入並使用所需的 AWS Service V3 套件用戶端、V3 命令，並以異步/等待模式使用該send方法。您可以使用 V2 命令來創建此示例，而不是通過進行一些小的更改。如需詳細資訊，請參閱 使用 v3 命令。

> **Note**
>
> Replace (取代) *TEMPLATE_NAME* 使用新範本的名稱，*HtmlPart* 帶有標HTML籤的電子郵件內容，以及 *SubjectPart* 與電子郵件的主題。

```
import { CreateTemplateCommand } from "@aws-sdk/client-ses";
import { sesClient } from "./libs/sesClient.js";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";

const TEMPLATE_NAME = getUniqueName("TestTemplateName");

const createCreateTemplateCommand = () => {
  return new CreateTemplateCommand({
    /**
     * The template feature in Amazon SES is based on the Handlebars template system.
     */
    Template: {
      /**
       * The name of an existing template in Amazon SES.
       */
      TemplateName: TEMPLATE_NAME,
      HtmlPart: `
        <h1>Hello, {{contact.firstName}}!</h1>
        <p>
        Did you know Amazon has a mascot named Peccy?
        </p>
      `,
      SubjectPart: "Amazon Tip",
```

```
    },
  });
};

const run = async () => {
  const createTemplateCommand = createCreateTemplateCommand();

  try {
    return await sesClient.send(createTemplateCommand);
  } catch (err) {
    console.log("Failed to create template.", err);
    return err;
  }
};
```

若要執行範例，請在命令提示字元中輸入下列命令。該模板被添加到 Amazon SES。

```
node ses_createtemplate.js
```

您可以在這裡找到這個範例程式碼 GitHub。

更新電子郵件範本

在此範例中，請使用 Node.js 模組建立要搭配 Amazon 使用的電子郵件範本SES。

創建一個`libs`目錄，並使用文件名創建一個 Node.js 模塊`sesClient.js`。將下面的代碼複製並粘貼到其中，以創建 Amazon SES 客戶端對象。Replace (取代) *`REGION`* 與您的 AWS 區域。

```
import { SESClient } from "@aws-sdk/client-ses";
// Set the AWS Region.
const REGION = "us-east-1";
// Create SES service object.
const sesClient = new SESClient({ region: REGION });
export { sesClient };
```

您可以在這裡找到這個範例程式碼 GitHub。

以檔名 `ses_updatetemplate.js` 建立一個 Node.js 模組。SDK如前所示設定，包括安裝所需的用戶端和套件。

建立一個物件，並搭配需要的 TemplateName 參數 (傳遞至 SES 用戶端類別的 UpdateTemplateCommand 方法之參數)，以傳遞您要在範本中更新的 Template 參數值。要調用該UpdateTemplateCommand方法，請調用 Amazon SES 服務對象，並傳遞參數。

> **Note**
>
> 此範例會匯入並使用所需的 AWS Service V3 套件用戶端、V3 命令，並以異步/等待模式使用該send方法。您可以使用 V2 命令來創建此示例，而不是通過進行一些小的更改。如需詳細資訊，請參閱 使用 v3 命令。

> **Note**
>
> Replace (取代) *TEMPLATE_NAME* 與模板的名稱和 *HTML_PART* 帶有標HTML籤的電子郵件內容。

```
import { UpdateTemplateCommand } from "@aws-sdk/client-ses";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { sesClient } from "./libs/sesClient.js";

const TEMPLATE_NAME = getUniqueName("TemplateName");
const HTML_PART = "<h1>Hello, World!</h1>";

const createUpdateTemplateCommand = () => {
  return new UpdateTemplateCommand({
    Template: {
      TemplateName: TEMPLATE_NAME,
      HtmlPart: HTML_PART,
      SubjectPart: "Example",
      TextPart: "Updated template text.",
    },
  });
};

const run = async () => {
  const updateTemplateCommand = createUpdateTemplateCommand();

  try {
    return await sesClient.send(updateTemplateCommand);
  } catch (err) {
```

```
    console.log("Failed to update template.", err);
    return err;
  }
};
```

若要執行範例，請在命令提示字元中輸入下列命令。Amazon SES 返回模板的詳細信息。

```
node ses_updatetemplate.js
```

您可以在這裡找到這個範例程式碼 GitHub。

刪除電子郵件範本

在此範例中，請使用 Node.js 模組建立要搭配 Amazon 使用的電子郵件範本SES。

創建一個libs目錄，並使用文件名創建一個 Node.js 模塊sesClient.js。將下面的代碼複製並粘貼到其中，以創建 Amazon SES 客戶端對象。Replace (取代) *REGION* 與您的 AWS 區域。

```
import { SESClient } from "@aws-sdk/client-ses";
// Set the AWS Region.
const REGION = "us-east-1";
// Create SES service object.
const sesClient = new SESClient({ region: REGION });
export { sesClient };
```

您可以在這裡找到這個範例程式碼 GitHub。

以檔名 ses_deletetemplate.js 建立一個 Node.js 模組。SDK如前所示設定，包括安裝所需的用戶端和套件。

建立物件以傳遞需要的 TemplateName 參數至 SES 用戶端類別的 DeleteTemplateCommand 方法。要調用該DeleteTemplateCommand方法，請調用 Amazon SES 服務對象，並傳遞參數。

> **Note**
>
> 此範例會匯入並使用所需的 AWS Service V3 套件用戶端、V3 命令，並以異步/等待模式使用該send方法。您可以使用 V2 命令來創建此示例，而不是通過進行一些小的更改。如需詳細資訊，請參閱 使用 v3 命令。

> **Note**
>
> Replace (取代) *TEMPLATE_NAME* 與要刪除的模板的名稱。

```
import { DeleteTemplateCommand } from "@aws-sdk/client-ses";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { sesClient } from "./libs/sesClient.js";

const TEMPLATE_NAME = getUniqueName("TemplateName");

const createDeleteTemplateCommand = (templateName) =>
  new DeleteTemplateCommand({ TemplateName: templateName });

const run = async () => {
  const deleteTemplateCommand = createDeleteTemplateCommand(TEMPLATE_NAME);

  try {
    return await sesClient.send(deleteTemplateCommand);
  } catch (err) {
    console.log("Failed to delete template.", err);
    return err;
  }
};
```

若要執行範例，請在命令提示字元中輸入下列命令。Amazon SES 返回模板的詳細信息。

```
node ses_deletetemplate.js
```

您可以在這裡找到這個範例程式碼 GitHub。

使用 Amazon 發送電子郵件 SES

這個 Node.js 程式碼範例會說明：

- 發送短信或HTML電子郵件。

- 根據電子郵件範本傳送電子郵件。
- 根據電子郵件範本傳送大量電子郵件。

Amazon SES API 提供了兩種不同的方式來發送電子郵件，具體取決於您希望對電子郵件消息組成的控制：格式化和原始。如需詳細資訊，請參閱使用 Amazon 傳送格式化的電子郵件SESAPI和使用 Amazon 傳送原始電子郵件SESAPI。

該方案

在此範例中，您會使用一系列的 Node.js 模組，以多種不同方式傳送電子郵件。Node.js 模組會使SDK用 JavaScript 來建立和使用用SES戶端類別下列方法的電子郵件範本：

- `SendEmailCommand`
- `SendTemplatedEmailCommand`
- `SendBulkTemplatedEmailCommand`

必要工作

若要設定和執行此範例，您必須先完成這些任務：

- 設置項目環境以運行這些節點 TypeScript 示例，並安裝所需 AWS SDK for JavaScript 的第三方模塊。按照上的說明進行操作 GitHub。

- 透過使用者登入資料建立共用組態檔。如需有關提供共用認證檔案的詳細資訊，請參閱《工具參考指南》中的共用設定AWS SDKs和認證檔案。

> **Important**
>
> 這些範例示範如何使用 ECMAScript6 (ES6) 匯入/匯出用戶端服務物件和指令。
>
> - 這需要 Node.js 版本 13.x 或更高版本。要下載並安裝最新版本的 Node.js，請參閱 Node.js 下載。 。
> - 如果您偏好使用 CommonJS 語法，請參閱JavaScript ES6/共同語法。

電子郵件傳送需求

Amazon SES 撰寫電子郵件訊息，並立即將其排入佇列以進行傳送。若要使用 SendEmailCommand 方法傳送電子郵件，您的訊息必須符合下列需求：

- 您必須從已驗證的電子郵件地址或網域傳送訊息。如果您要使用非驗證的地址或網域來傳送電子郵件，則該操作會導致 "Email address not verified" 錯誤。
- 如果您的帳戶仍在 Amazon SES 沙箱中，則只能傳送到經過驗證的地址或網域，或傳送至與 Amazon SES 信箱模擬器相關聯的電子郵件地址。如需詳細資訊，請參閱 Amazon 簡易電子郵件服務開發人員指南中的驗證電子郵件地址和網域。
- 訊息的總大小 (包括附件) 必須小於 10 MB。
- 該訊息至少必須含有一個收件人電子郵件地址。收件者地址可以是 [收件者:] 地址、[副本:地址] 或 [BCC:] 地址。如果收件者電子郵件地址無效 (也就是格式不是UserName@[SubDomain.]Domain.TopLevelDomain)，則整封郵件都會遭到拒絕，即使郵件包含其他有效收件者也一樣。
- 郵件在 [收件者:]、[副本:] 和 BCC [:] 欄位中不能包含 50 個以上的收件者。若您需要傳送電子郵件訊息給更多的收件人，則您必須將收件人清單分成 50 人或更少人的群組，然後再多次呼叫 sendEmail 方法以傳送訊息至個別群組。

發送電子郵件

在此範例中，使用 Node.js 模組透過 Amazon 傳送電子郵件SES。

創建一個libs目錄，並使用文件名創建一個 Node.js 模塊sesClient.js。將下面的代碼複製並粘貼到其中，以創建 Amazon SES 客戶端對象。Replace (取代) *REGION* 與您的 AWS 區域。

```
import { SESClient } from "@aws-sdk/client-ses";
// Set the AWS Region.
const REGION = "us-east-1";
// Create SES service object.
const sesClient = new SESClient({ region: REGION });
export { sesClient };
```

您可以在這裡找到這個範例程式碼 GitHub。

以檔名 ses_sendemail.js 建立一個 Node.js 模組。SDK如前所示設定，包括安裝所需的用戶端和套件。

建立物件，以便將定義要傳送之電子郵件的參數值，包括寄件者和接收者位址、主旨和電子郵件內文 (以純文字和HTML格式) 傳遞至SES用戶端類別的SendEmailCommand方法。要調用該SendEmailCommand方法，請調用 Amazon SES 服務對象，並傳遞參數。

> **Note**
>
> 此範例會匯入並使用所需的 AWS Service V3 套件用戶端、V3 命令，並以異步/等待模式使用該send方法。您可以使用 V2 命令來創建此示例，而不是通過進行一些小的更改。如需詳細資訊，請參閱 使用 v3 命令。

> **Note**
>
> Replace (取代) *toAddress* 與發送電子郵件的地址，*fromAddress* 與電子郵件地址發送電子郵件。

```
import { SendEmailCommand } from "@aws-sdk/client-ses";
import { sesClient } from "./libs/sesClient.js";

const createSendEmailCommand = (toAddress, fromAddress) => {
  return new SendEmailCommand({
    Destination: {
      /* required */
      CcAddresses: [
        /* more items */
      ],
      ToAddresses: [
        toAddress,
        /* more To-email addresses */
      ],
    },
    Message: {
      /* required */
      Body: {
        /* required */
        Html: {
          Charset: "UTF-8",
          Data: "HTML_FORMAT_BODY",
        },
        Text: {
```

```
          Charset: "UTF-8",
          Data: "TEXT_FORMAT_BODY",
        },
      },
      Subject: {
        Charset: "UTF-8",
        Data: "EMAIL_SUBJECT",
      },
    },
    Source: fromAddress,
    ReplyToAddresses: [
      /* more items */
    ],
  });
};

const run = async () => {
  const sendEmailCommand = createSendEmailCommand(
    "recipient@example.com",
    "sender@example.com",
  );

  try {
    return await sesClient.send(sendEmailCommand);
  } catch (caught) {
    if (caught instanceof Error && caught.name === "MessageRejected") {
      /** @type { import('@aws-sdk/client-ses').MessageRejected} */
      const messageRejectedError = caught;
      return messageRejectedError;
    }
    throw caught;
  }
};
```

若要執行範例，請在命令提示字元中輸入下列命令。電子郵件排隊由 Amazon SES 發送。

```
node ses_sendemail.js
```

您可以在這裡找到這個範例程式碼 GitHub。

使用範本傳送電子郵件

在此範例中，使用 Node.js 模組透過 Amazon 傳送電子郵件SES。以檔名 `ses_sendtemplatedemail.js` 建立一個 Node.js 模組。SDK如前所示設定，包括安裝所需的用戶端和套件。

創建一個對象，以將定義要發送的電子郵件的參數值，包括發送者和接收者地址，主題，電子郵件正文以純文本和HTML格式傳遞給SES客戶端類的`SendTemplatedEmailCommand`方法。要調用該`SendTemplatedEmailCommand`方法，請調用 Amazon SES 客戶端服務對象，並傳遞參數。

> **Note**
>
> 此範例會匯入並使用所需的 AWS Service V3 套件用戶端、V3 命令，並以異步/等待模式使用該`send`方法。您可以使用 V2 命令來創建此示例，而不是通過進行一些小的更改。如需詳細資訊，請參閱 使用 v3 命令。

> **Note**
>
> Replace (取代) *`REGION`* 與您的 AWS 地區，*`USER`* 包含發送電子郵件的名稱和電子郵件地址，*`VERIFIED_EMAIL`* 使用發送電子郵件的電子郵件地址，以及 *`TEMPLATE_NAME`* 與模板的名稱。

```
import { SendTemplatedEmailCommand } from "@aws-sdk/client-ses";
import {
  getUniqueName,
  postfix,
} from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { sesClient } from "./libs/sesClient.js";

/**
 * Replace this with the name of an existing template.
 */
const TEMPLATE_NAME = getUniqueName("ReminderTemplate");

/**
 * Replace these with existing verified emails.
 */
const VERIFIED_EMAIL = postfix(getUniqueName("Bilbo"), "@example.com");
```

```
const USER = { firstName: "Bilbo", emailAddress: VERIFIED_EMAIL };

/**
 *
 * @param { { emailAddress: string, firstName: string } } user
 * @param { string } templateName - The name of an existing template in Amazon SES.
 * @returns { SendTemplatedEmailCommand }
 */
const createReminderEmailCommand = (user, templateName) => {
  return new SendTemplatedEmailCommand({
    /**
     * Here's an example of how a template would be replaced with user data:
     * Template: <h1>Hello {{contact.firstName}},</h1><p>Don't forget about the party
 gifts!</p>
     * Destination: <h1>Hello Bilbo,</h1><p>Don't forget about the party gifts!</p>
     */
    Destination: { ToAddresses: [user.emailAddress] },
    TemplateData: JSON.stringify({ contact: { firstName: user.firstName } }),
    Source: VERIFIED_EMAIL,
    Template: templateName,
  });
};

const run = async () => {
  const sendReminderEmailCommand = createReminderEmailCommand(
    USER,
    TEMPLATE_NAME,
  );
  try {
    return await sesClient.send(sendReminderEmailCommand);
  } catch (caught) {
    if (caught instanceof Error && caught.name === "MessageRejected") {
      /** @type { import('@aws-sdk/client-ses').MessageRejected} */
      const messageRejectedError = caught;
      return messageRejectedError;
    }
    throw caught;
  }
};
```

若要執行範例，請在命令提示字元中輸入下列命令。電子郵件排隊由 Amazon SES 發送。

```
node ses_sendtemplatedemail.js
```

您可以在這裡找到這個範例程式碼 GitHub。

使用範本傳送大量電子郵件

在此範例中，使用 Node.js 模組透過 Amazon 傳送電子郵件SES。

創建一個libs目錄，並使用文件名創建一個 Node.js 模塊sesClient.js。將下面的代碼複製並粘貼到其中，以創建 Amazon SES 客戶端對象。Replace (取代) *REGION* 與您的 AWS 區域。

```
import { SESClient } from "@aws-sdk/client-ses";
// Set the AWS Region.
const REGION = "us-east-1";
// Create SES service object.
const sesClient = new SESClient({ region: REGION });
export { sesClient };
```

您可以在這裡找到這個範例程式碼 GitHub。

以檔名 ses_sendbulktemplatedemail.js 建立一個 Node.js 模組。SDK如前所示設定，包括安裝所需的用戶端和套件。

建立物件，以便將定義要傳送之電子郵件的參數值，包括寄件者和接收者位址、主旨和電子郵件內文 (以純文字和HTML格式) 傳遞至SES用戶端類別的SendBulkTemplatedEmailCommand方法。要調用該SendBulkTemplatedEmailCommand方法，請調用 Amazon SES 服務對象，並傳遞參數。

> **Note**
>
> 此範例會匯入並使用所需的 AWS Service V3 套件用戶端、V3 命令，並以異步/等待模式使用該send方法。您可以使用 V2 命令來創建此示例，而不是通過進行一些小的更改。如需詳細資訊，請參閱 使用 v3 命令。

> **Note**
>
> Replace (取代) *USERS* 包含發送電子郵件的姓名和電子郵件地址，*VERIFIED_EMAIL_1* 使用發送電子郵件的電子郵件地址，以及 *TEMPLATE_NAME* 與模板的名稱。

```
import { SendBulkTemplatedEmailCommand } from "@aws-sdk/client-ses";
import {
  getUniqueName,
  postfix,
} from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { sesClient } from "./libs/sesClient.js";

/**
 * Replace this with the name of an existing template.
 */
const TEMPLATE_NAME = getUniqueName("ReminderTemplate");

/**
 * Replace these with existing verified emails.
 */
const VERIFIED_EMAIL_1 = postfix(getUniqueName("Bilbo"), "@example.com");
const VERIFIED_EMAIL_2 = postfix(getUniqueName("Frodo"), "@example.com");

const USERS = [
  { firstName: "Bilbo", emailAddress: VERIFIED_EMAIL_1 },
  { firstName: "Frodo", emailAddress: VERIFIED_EMAIL_2 },
];

/**
 *
 * @param { { emailAddress: string, firstName: string }[] } users
 * @param { string } templateName the name of an existing template in SES
 * @returns { SendBulkTemplatedEmailCommand }
 */
const createBulkReminderEmailCommand = (users, templateName) => {
  return new SendBulkTemplatedEmailCommand({
    /**
     * Each 'Destination' uses a corresponding set of replacement data. We can map each
 user
     * to a 'Destination' and provide user specific replacement data to create
 personalized emails.
     *
     * Here's an example of how a template would be replaced with user data:
     * Template: <h1>Hello {{name}},</h1><p>Don't forget about the party gifts!</p>
     * Destination 1: <h1>Hello Bilbo,</h1><p>Don't forget about the party gifts!</p>
     * Destination 2: <h1>Hello Frodo,</h1><p>Don't forget about the party gifts!</p>
     */
    Destinations: users.map((user) => ({
```

```
      Destination: { ToAddresses: [user.emailAddress] },
      ReplacementTemplateData: JSON.stringify({ name: user.firstName }),
    })),
    DefaultTemplateData: JSON.stringify({ name: "Shireling" }),
    Source: VERIFIED_EMAIL_1,
    Template: templateName,
  });
};

const run = async () => {
  const sendBulkTemplateEmailCommand = createBulkReminderEmailCommand(
    USERS,
    TEMPLATE_NAME,
  );
  try {
    return await sesClient.send(sendBulkTemplateEmailCommand);
  } catch (caught) {
    if (caught instanceof Error && caught.name === "MessageRejected") {
      /** @type { import('@aws-sdk/client-ses').MessageRejected} */
      const messageRejectedError = caught;
      return messageRejectedError;
    }
    throw caught;
  }
};
```

若要執行範例，請在命令提示字元中輸入下列命令。電子郵件排隊由 Amazon SES 發送。

```
node ses_sendbulktemplatedemail.js
```

您可以在這裡找到這個範例程式碼 GitHub。

亞馬遜簡單通知服務示例

Amazon Simple Notification Service (Amazon SNS) 是一種 Web 服務，會協調和管理消息傳遞或發送到訂閱端點或客戶端。

在 Amazon SNS 中，有兩種類型的用戶端 — 發佈者和訂閱者 — 也稱為生產者和消費者。

發佈者透過製作並傳送訊息到主題 (其為邏輯存取點和通訊管道) 與訂閱者進行非同步的通訊。訂閱者 (Web 伺服器、電子郵件地址、Amazon SQS 佇列、AWS Lambda函數) 在訂閱主題時，透過其中一個支援的協定 (Amazon SQS、HTTP/S、電子郵件、SMSAWS Lambda) 消耗或接收訊息或通知。

Amazon SNS 的 JavaScript API 透過類別：SNS 公開。

主題

- 管理 Amazon SNS 中的主題
- 在 Amazon SNS 中發佈訊息
- 管理 Amazon SNS 中的訂閱
- 使用 Amazon SNS 發送短信

管理 Amazon SNS 中的主題

這個 Node.js 程式碼範例會說明：

- 如何在 Amazon SNS 中建立可以向其發佈通知的主題。
- 如何刪除在 Amazon SNS 創建的主題。
- 如何取得可用主題的清單。
- 如何取得和設定主題屬性。

使用案例

在此範例中，您使用一系列 Node.js 模組來建立、列出和刪除 Amazon SNS 主題，以及處理主題屬性。Node.js 模組會使用 SDK JavaScript 來管理使用用SNS戶端類別的下列方法的主題：

- `CreateTopicCommand`
- `ListTopicsCommand`
- `DeleteTopicCommand`
- `GetTopicAttributesCommand`
- `SetTopicAttributesCommand`

先決條件任務

若要設定和執行此範例，您必須先完成這些任務：

- 設置項目環境以運行這些節點 TypeScript 示例，並安裝所需AWS SDK for JavaScript的第三方模塊。按照上的說明進行操作 GitHub。
- 透過使用者登入資料建立共用組態檔。有關提供共用認證檔案的詳細資訊，請參閱 AWSSDK 和工具參考指南中的共用設定和認證檔案。

> **Important**
>
> 這些範例示範如何使用 ECMAScript6 (ES6) 匯入/匯出用戶端服務物件和指令。
>
> - 這需要 Node.js 版本 13.x 或更高版本。要下載並安裝最新版本的 Node.js，請參閱 Node.js 下載。 。
> - 如果您偏好使用 CommonJS 語法，請參閱JavaScript ES6/共同語法。

建立主題

在此範例中，使用 Node.js 模組建立 Amazon SNS 主題。

創建一個`libs`目錄，並使用文件名創建一個 Node.js 模塊`snsClient.js`。將下列程式碼複製並貼到其中，以建立 Amazon SNS 用戶端物件。以您的`##`取代「AWS地區」。

```
import { SNSClient } from "@aws-sdk/client-sns";
```

```
// The AWS Region can be provided here using the `region` property. If you leave it
 blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

您可以在這裡找到此範例程式碼 GitHub。

以檔名 `create-topic.js` 建立一個 Node.js 模組。如先前所示設定 SDK，包括安裝所需的用戶端和套件。

建立一個物件，藉此將新主題的 `Name` 傳遞至 SNS 用戶端類別的 `CreateTopicCommand` 方法。若要呼叫方`CreateTopicCommand`法，請建立叫用 Amazon SNS 服務物件的非同步函數，並傳遞參數物件。傳`data`回的包含主題的 ARN。

> **Note**
>
> 將####取代為主題名稱。

```
import { CreateTopicCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} topicName - The name of the topic to create.
 */
export const createTopic = async (topicName = "TOPIC_NAME") => {
  const response = await snsClient.send(
    new CreateTopicCommand({ Name: topicName }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '087b8ad2-4593-50c4-a496-d7e90b82cf3e',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   TopicArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxxx:TOPIC_NAME'
  // }
```

```
  return response;
};
```

若要執行範例，請在命令提示字元中輸入下列命令。

```
node create-topic.js
```

您可以在這裡找到此範例程式碼 GitHub。

列出 主題

在此範例中，使用 Node.js 模組列出所有 Amazon SNS 主題。

創建一個libs目錄，並使用文件名創建一個 Node.js 模塊snsClient.js。將下列程式碼複製並貼到其中，以建立 Amazon SNS 用戶端物件。以您的##取代「AWS地區」。

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
 blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

您可以在這裡找到此範例程式碼 GitHub。

以檔名 list-topics.js 建立一個 Node.js 模組。如先前所示設定 SDK，包括安裝所需的用戶端和套件。

建立空白物件，並將其傳遞至 SNS 用戶端類別的 ListTopicsCommand 方法。若要呼叫方ListTopicsCommand法，請建立叫用 Amazon SNS 服務物件的非同步函數，並傳遞參數物件。data返回的包含您的主題亞馬遜資源名稱（ARN）的數組。

```
import { ListTopicsCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

export const listTopics = async () => {
  const response = await snsClient.send(new ListTopicsCommand({}));
  console.log(response);
  // {
  //   '$metadata': {
```

```
  //     httpStatusCode: 200,
  //     requestId: '936bc5ad-83ca-53c2-b0b7-9891167b909e',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   Topics: [ { TopicArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxxx:mytopic' } ]
  // }
  return response;
};
```

若要執行範例，請在命令提示字元中輸入下列命令。

```
node list-topics.js
```

您可以在這裡找到此範例程式碼 GitHub。

刪除主題

在此範例中，使用 Node.js 模組刪除 Amazon SNS 主題。

創建一個libs目錄，並使用文件名創建一個 Node.js 模塊snsClient.js。將下列程式碼複製並貼到其中，以建立 Amazon SNS 用戶端物件。以您的##取代「AWS地區」。

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
 blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

您可以在這裡找到此範例程式碼 GitHub。

以檔名 delete-topic.js 建立一個 Node.js 模組。如先前所示設定 SDK，包括安裝所需的用戶端和套件。

建立包含欲刪除主題 TopicArn 的物件，並將其傳遞至 SNS 用戶端類別的 DeleteTopicCommand 方法。若要呼叫方DeleteTopicCommand法，請建立叫用 Amazon SNS 用戶端服務物件的非同步函數，並傳遞參數物件。

> **Note**
>
> 將 *TOPIC_ARN* 取代為您要刪除之主題的亞馬遜資源名稱 (ARN)。

```
import { DeleteTopicCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} topicArn - The ARN of the topic to delete.
 */
export const deleteTopic = async (topicArn = "TOPIC_ARN") => {
  const response = await snsClient.send(
    new DeleteTopicCommand({ TopicArn: topicArn }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'a10e2886-5a8f-5114-af36-75bd39498332',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   }
  // }
};
```

若要執行範例，請在命令提示字元中輸入下列命令。

```
node delete-topic.js
```

您可以在這裡找到此範例程式碼 GitHub。

取得主題屬性

在此範例中，使用 Node.js 模組擷取 Amazon SNS 主題的屬性。

創建一個`libs`目錄，並使用文件名創建一個 Node.js 模塊`snsClient.js`。將下列程式碼複製並貼到其中，以建立 Amazon SNS 用戶端物件。以您的`##`取代「AWS地區」。

```
import { SNSClient } from "@aws-sdk/client-sns";
```

```
// The AWS Region can be provided here using the `region` property. If you leave it
 blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

您可以在這裡找到此範例程式碼 GitHub。

以檔名 `get-topic-attributes.js` 建立一個 Node.js 模組。依前述內容設定軟體開發套件。

建立包含欲刪除主題 `TopicArn` 的物件，並將其傳遞至 SNS 用戶端類別的 `GetTopicAttributesCommand` 方法。若要呼叫方`GetTopicAttributesCommand`法，請呼叫 Amazon SNS 用戶端服務物件，並傳遞參數物件。

> **Note**
>
> 以*### ARN* 取代主題。

```
import { GetTopicAttributesCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} topicArn - The ARN of the topic to retrieve attributes for.
 */
export const getTopicAttributes = async (topicArn = "TOPIC_ARN") => {
  const response = await snsClient.send(
    new GetTopicAttributesCommand({
      TopicArn: topicArn,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '36b6a24e-5473-5d4e-ac32-ff72d9a73d94',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   Attributes: {
```

```
  //     Policy: '{...}',
  //     Owner: 'xxxxxxxxxxx',
  //     SubscriptionsPending: '1',
  //     TopicArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxx:mytopic',
  //     TracingConfig: 'PassThrough',
  //     EffectiveDeliveryPolicy: '{"http":{"defaultHealthyRetryPolicy":
{"minDelayTarget":20,"maxDelayTarget":20,"numRetries":3,"numMaxDelayRetries":0,"numNoDelayRetri
{"headerContentType":"text/plain; charset=UTF-8"}}}',
  //     SubscriptionsConfirmed: '0',
  //     DisplayName: '',
  //     SubscriptionsDeleted: '1'
  //   }
  // }
  return response;
};
```

若要執行範例，請在命令提示字元中輸入下列命令。

```
node get-topic-attributes.js
```

您可以在這裡找到此範例程式碼 GitHub。

設定主題屬性

在此範例中，使用 Node.js 模組來設定 Amazon SNS 主題的可變屬性。

創建一個libs目錄，並使用文件名創建一個 Node.js 模塊snsClient.js。將下列程式碼複製並貼到其中，以建立 Amazon SNS 用戶端物件。以您的##取代「AWS地區」。

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
 blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

您可以在這裡找到此範例程式碼 GitHub。

以檔名 set-topic-attributes.js 建立一個 Node.js 模組。依前述內容設定軟體開發套件。

建立一個物件，其中包含用來更新屬性的參數，包括要設定屬性的主題 TopicArn、要設定的屬性名稱，以及該屬性的新數值。您只能設定 Policy、DisplayName 和 DeliveryPolicy 屬性。將參數傳遞至 SNS 用戶端類別的 SetTopicAttributesCommand 方法。若要呼叫

方SetTopicAttributesCommand法，請建立叫用 Amazon SNS 用戶端服務物件的非同步函數，並傳遞參數物件。

> **Note**
>
> 將*#################### TOPIC_ARN ###################### (ARN)######### ## NEW _ATERTE* _VALUE。

```
import { SetTopicAttributesCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

export const setTopicAttributes = async (
  topicArn = "TOPIC_ARN",
  attributeName = "DisplayName",
  attributeValue = "Test Topic",
) => {
  const response = await snsClient.send(
    new SetTopicAttributesCommand({
      AttributeName: attributeName,
      AttributeValue: attributeValue,
      TopicArn: topicArn,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'd1b08d0e-e9a4-54c3-b8b1-d03238d2b935',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   }
  // }
  return response;
};
```

若要執行範例，請在命令提示字元中輸入下列命令。

```
node set-topic-attributes.js
```

您可以在這裡找到此範例程式碼 GitHub。

在 Amazon SNS 中發佈訊息

這個 Node.js 程式碼範例會說明：

- 如何將訊息發佈到 Amazon SNS 主題。

使用案例

在此範例中，您使用一系列 Node.js 模組將來自 Amazon SNS 的訊息發佈到主題端點、電子郵件或電話號碼。Node.js 模組會使用 SDK 來使用用SNS戶端類別的這個方法 JavaScript 來傳送訊息：

- `PublishCommand`

先決條件任務

若要設定和執行此範例，您必須先完成這些任務：

- 設置項目環境以運行這些節點 TypeScript 示例，並安裝所需AWS SDK for JavaScript的第三方模塊。按照上的說明進行操作 GitHub。
- 透過使用者登入資料建立共用組態檔。有關提供共用認證檔案的詳細資訊，請參閱 AWSSDK 和工具參考指南中的共用設定和認證檔案。

> **Important**
>
> 這些範例示範如何使用 ECMAScript6 (ES6) 匯入/匯出用戶端服務物件和指令。
>
> - 這需要 Node.js 版本 13.x 或更高版本。要下載並安裝最新版本的 Node.js，請參閱 Node.js 下載。 。
> - 如果您偏好使用 CommonJS 語法，請參閱JavaScript ES6/共同語法。

發佈訊息至 SNS 主題

在此範例中，使用 Node.js 模組將訊息發佈到 Amazon SNS 主題。

創建一個libs目錄，並使用文件名創建一個 Node.js 模塊snsClient.js。將下列程式碼複製並貼到其中，以建立 Amazon SNS 用戶端物件。以您的##取代「AWS地區」。

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
 blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

您可以在這裡找到此範例程式碼 GitHub。

以檔名 publish-topic.js 建立一個 Node.js 模組。依前述內容設定軟體開發套件。

建立一個物件，其中包含用於發佈訊息的參數，包括訊息文字和 Amazon SnStoPic 的亞馬遜資源名稱 (ARN)。如需可用簡訊屬性的詳細資訊，請參閱 SetSMSAttributes。

將參數傳遞給用SNS戶端類別的PublishCommand方法。建立叫用 Amazon SNS 用戶端服務物件的非同步函數，並傳遞參數物件。

> **Note**
>
> 將訊###取代為訊息文字，並將主#### *SNS ### ARN*。

```
import { PublishCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string | Record<string, any>} message - The message to send. Can be a plain
 string or an object
 *                                                 if you are using the `json`
 `MessageStructure`.
 * @param {string} topicArn - The ARN of the topic to which you would like to publish.
 */
export const publish = async (
  message = "Hello from SNS!",
```

```
  topicArn = "TOPIC_ARN",
) => {
  const response = await snsClient.send(
    new PublishCommand({
      Message: message,
      TopicArn: topicArn,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'e7f77526-e295-5325-9ee4-281a43ad1f05',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   MessageId: 'xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx'
  // }
  return response;
};
```

若要執行範例，請在命令提示字元中輸入下列命令。

```
node publish-topic.js
```

您可以在這裡找到此範例程式碼 GitHub。

管理 Amazon SNS 中的訂閱

這個 Node.js 程式碼範例會說明：

- 如何列出 Amazon SNS 主題的所有訂閱。
- 如何訂閱電子郵件地址、應用程式端點或 Amazon SNS 主題的AWS Lambda函數。
- 如何退訂 Amazon SNS 主題。

使用案例

在此範例中，您會使用一系列 Node.js 模組將通知訊息發佈到 Amazon SNS 主題。Node.js 模組會使用 SDK JavaScript 來管理使用用SNS戶端類別的下列方法的主題：

- ListSubscriptionsByTopicCommand
- SubscribeCommand
- ConfirmSubscriptionCommand
- UnsubscribeCommand

先決條件任務

若要設定和執行此範例，您必須先完成這些任務：

- 設置項目環境以運行這些節點 TypeScript 示例，並安裝所需AWS SDK for JavaScript的第三方模塊。按照上的說明進行操作 GitHub。
- 透過使用者登入資料建立共用組態檔。有關提供共用認證檔案的詳細資訊，請參閱 AWSSDK 和工具參考指南中的共用設定和認證檔案。

> **Important**
>
> 這些範例示範如何使用 ECMAScript6 (ES6) 匯入/匯出用戶端服務物件和指令。
>
> - 這需要 Node.js 版本 13.x 或更高版本。要下載並安裝最新版本的 Node.js，請參閱 Node.js 下載。 。
> - 如果您偏好使用 CommonJS 語法，請參閱JavaScript ES6/共同語法。

列出主題的訂閱

在此範例中，使用 Node.js 模組列出 Amazon SNS 主題的所有訂閱。

創建一個`libs`目錄，並使用文件名創建一個 Node.js 模塊`snsClient.js`。將下列程式碼複製並貼到其中，以建立 Amazon SNS 用戶端物件。以您的`##`取代「AWS地區」。

```
import { SNSClient } from "@aws-sdk/client-sns";
```

```
// The AWS Region can be provided here using the `region` property. If you leave it
 blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

您可以在這裡找到此範例程式碼 GitHub。

以檔名 `list-subscriptions-by-topic.js` 建立一個 Node.js 模組。依前述內容設定軟體開發套件。

建立一個物件，其中包含要列出訂閱的主題 `TopicArn` 參數。將參數傳遞至 SNS 用戶端類別的 `ListSubscriptionsByTopicCommand` 方法。若要呼叫方`ListSubscriptionsByTopicCommand`法，請建立叫用 Amazon SNS 用戶端服務物件的非同步函數，然後傳遞參數物件。

> **Note**
>
> 將 *`TOPIC_ARN`* 取代為您要列出其訂閱的主題的亞馬遜資源名稱 (ARN)。

```
import { ListSubscriptionsByTopicCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} topicArn - The ARN of the topic for which you wish to list
 subscriptions.
 */
export const listSubscriptionsByTopic = async (topicArn = "TOPIC_ARN") => {
  const response = await snsClient.send(
    new ListSubscriptionsByTopicCommand({ TopicArn: topicArn }),
  );

  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '0934fedf-0c4b-572e-9ed2-a3e38fadb0c8',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
```

```
  //   },
  //   Subscriptions: [
  //     {
  //       SubscriptionArn: 'PendingConfirmation',
  //       Owner: '901487484989',
  //       Protocol: 'email',
  //       Endpoint: 'corepyle@amazon.com',
  //       TopicArn: 'arn:aws:sns:us-east-1:901487484989:mytopic'
  //     }
  //   ]
  // }
  return response;
};
```

若要執行範例，請在命令提示字元中輸入下列命令。

```
node list-subscriptions-by-topic.js
```

您可以在這裡找到此範例程式碼 GitHub。

使用電子郵件地址訂閱主題

在此範例中，使用 Node.js 模組訂閱電子郵件地址，以便接收來自 Amazon SNS 主題的 SMTP 電子郵件訊息。

創建一個`libs`目錄，並使用文件名創建一個 Node.js 模塊`snsClient.js`。將下列程式碼複製並貼到其中，以建立 Amazon SNS 用戶端物件。以您的`##`取代「AWS地區」。

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
 blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

您可以在這裡找到此範例程式碼 GitHub。

以檔名 `subscribe-email.js` 建立一個 Node.js 模組。依前述內容設定軟體開發套件。

建立包含 `Protocol` 參數的物件，以便指定 `email` 通訊協定、要訂閱的主題 `TopicArn`，以及要做為訊息 `Endpoint` 的電子郵件地址。將參數傳遞至 SNS 用戶端類別的 `SubscribeCommand` 方法。您可

以使用該subscribe方法將多個不同的端點訂閱 Amazon SNS 主題，具體取決於用於傳遞參數的值，如本主題中的其他範例所示。

若要呼叫方SubscribeCommand法，請建立叫用 Amazon SNS 用戶端服務物件的非同步函數，然後傳遞參數物件。

> **Note**
>
> 將 *TOPIC_ARN* 取代為該主題的亞馬遜資源名稱 (ARN)，將電子郵件地址取代為要訂閱的*#####*。

```
import { SubscribeCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} topicArn - The ARN of the topic for which you wish to confirm a
 subscription.
 * @param {string} emailAddress - The email address that is subscribed to the topic.
 */
export const subscribeEmail = async (
  topicArn = "TOPIC_ARN",
  emailAddress = "usern@me.com",
) => {
  const response = await snsClient.send(
    new SubscribeCommand({
      Protocol: "email",
      TopicArn: topicArn,
      Endpoint: emailAddress,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'c8e35bcd-b3c0-5940-9f66-06f6fcc108f0',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
```

```
  //   SubscriptionArn: 'pending confirmation'
  // }
};
```

若要執行範例，請在命令提示字元中輸入下列命令。

```
node subscribe-email.js
```

您可以在這裡找到此範例程式碼 GitHub。

確認訂閱

在此範例中，透過驗證先前的訂閱動作傳送至端點的 Token，使用 Node.js 模組來驗證端點擁有者接收電子郵件的意圖。

創建一個libs目錄，並使用文件名創建一個 Node.js 模塊snsClient.js。將下列程式碼複製並貼到其中，以建立 Amazon SNS 用戶端物件。以您的##取代「AWS地區」。

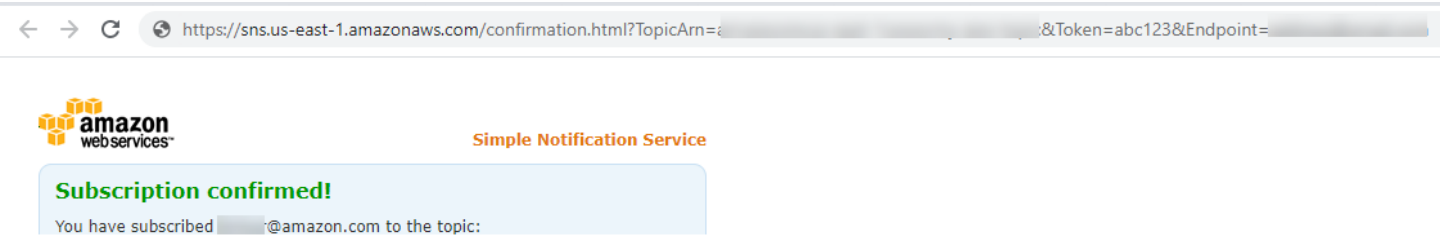
```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
 blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

您可以在這裡找到此範例程式碼 GitHub。

以檔名 confirm-subscription.js 建立一個 Node.js 模組。如先前所示設定 SDK，包括安裝所需的用戶端和套件。

定義參數，包括TOPIC_ARN和TOKEN，並定義TRUE或FALSE的值AuthenticateOnUnsubscribe。

令牌是在上一個SUBSCRIBE操作期間發送給端點所有者的短期令牌。例如，對於電子郵件端點，TOKEN則位於傳送給電子郵件擁有者的「確認訂閱」電子郵件的 URL 中。例如，abc123是下列 URL 中的權杖。



若要呼叫方ConfirmSubscriptionCommand法，請建立叫用 Amazon SNS 用戶端服務物件的非同步函數，並傳遞參數物件。

> **Note**
>
> 將 *TOPIC_ARN* 取代為主題的 Amazon 資源名稱 (ARN)，將 TO *KEN* 取代為上一個Subscribe動作中傳送至端點擁有者之 URL 的權杖值，並定義 *AuthenticateOnUnsubscribe*. 值為或。TRUE FALSE

```
import { ConfirmSubscriptionCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} token - This token is sent the subscriber. Only subscribers
 *                         that are not AWS services (HTTP/S, email) need to be
 confirmed.
 * @param {string} topicArn - The ARN of the topic for which you wish to confirm a
 subscription.
 */
export const confirmSubscription = async (
  token = "TOKEN",
  topicArn = "TOPIC_ARN",
) => {
  const response = await snsClient.send(
    // A subscription only needs to be confirmed if the endpoint type is
    // HTTP/S, email, or in another AWS account.
    new ConfirmSubscriptionCommand({
      Token: token,
      TopicArn: topicArn,
      // If this is true, the subscriber cannot unsubscribe while unauthenticated.
      AuthenticateOnUnsubscribe: "false",
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '4bb5bce9-805a-5517-8333-e1d2cface90b',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
```

```
  //     totalRetryDelay: 0
  //   },
  //   SubscriptionArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxxx:TOPIC_NAME:xxxxxxxx-xxxx-
xxxx-xxxx-xxxxxxxxxxxx'
  // }
  return response;
};
```

若要執行範例，請在命令提示字元中輸入下列命令。

```
node confirm-subscription.js
```

您可以在這裡找到此範例程式碼 GitHub。

使用應用程式端點訂閱主題

在此範例中，使用 Node.js 模組訂閱行動應用程式端點，以便接收來自 Amazon SNS 主題的通知。

創建一個libs目錄，並使用文件名創建一個 Node.js 模塊snsClient.js。將下列程式碼複製並貼到其中，以建立 Amazon SNS 用戶端物件。以您的##取代「AWS地區」。

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
 blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

您可以在這裡找到此範例程式碼 GitHub。

以檔名 subscribe-app.js 建立一個 Node.js 模組。如先前所示設定 SDK，包括安裝所需的模組和套件。

建立一個物件，其中包含用TopicArn於指定application通訊協定的Protocol參數、要訂閱的主題，以及Endpoint參數行動應用程式端點的 Amazon 資源名稱 (ARN)。將參數傳遞至 SNS 用戶端類別的 SubscribeCommand 方法。

若要呼叫方SubscribeCommand法，請建立叫用 Amazon SNS 服務物件的非同步函數，並傳遞參數物件。

> **Note**
>
> 將主####取代為亞馬遜資源名稱 (ARN)，將 *MOBILE_ENDPOINT _ARN ############*。

```
import { SubscribeCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} topicArn - The ARN of the topic the subscriber is subscribing to.
 * @param {string} endpoint - The Endpoint ARN of an application. This endpoint is
 created
 *                            when an application registers for notifications.
 */
export const subscribeApp = async (
  topicArn = "TOPIC_ARN",
  endpoint = "ENDPOINT",
) => {
  const response = await snsClient.send(
    new SubscribeCommand({
      Protocol: "application",
      TopicArn: topicArn,
      Endpoint: endpoint,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'c8e35bcd-b3c0-5940-9f66-06f6fcc108f0',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   SubscriptionArn: 'pending confirmation'
  // }
  return response;
};
```

若要執行範例，請在命令提示字元中輸入下列命令。

```
node subscribe-app.js
```

您可以在這裡找到此範例程式碼 GitHub。

訂閱 Lambda 函數至主題

在此範例中，使用 Node.js 模組來訂閱AWS Lambda函數，以便接收來自 Amazon SNS 主題的通知。

創建一個libs目錄，並使用文件名創建一個 Node.js 模塊snsClient.js。將下列程式碼複製並貼到其中，以建立 Amazon SNS 用戶端物件。以您的##取代「AWS地區」。

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
 blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

您可以在這裡找到此範例程式碼 GitHub。

以檔名 subscribe-lambda.js 建立一個 Node.js 模組。依前述內容設定軟體開發套件。

建立包含Protocol參數的物件、指定lambda通訊協定TopicArn、要訂閱的主題，以及AWS Lambda函數的 Amazon 資源名稱 (ARN) 做為Endpoint參數。將參數傳遞至 SNS 用戶端類別的 SubscribeCommand 方法。

若要呼叫方SubscribeCommand法，請建立叫用 Amazon SNS 用戶端服務物件的非同步函數，並傳遞參數物件。

> **Note**
>
> 用主*##########ARN*）替換主題，將 Lambda 函數的亞馬遜資源名稱（ARN）替換為 *Lambda ###########ARN*）。

```
import { SubscribeCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} topicArn - The ARN of the topic the subscriber is subscribing to.
```

```
 * @param {string} endpoint - The Endpoint ARN of and AWS Lambda function.
 */
export const subscribeLambda = async (
  topicArn = "TOPIC_ARN",
  endpoint = "ENDPOINT",
) => {
  const response = await snsClient.send(
    new SubscribeCommand({
      Protocol: "lambda",
      TopicArn: topicArn,
      Endpoint: endpoint,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'c8e35bcd-b3c0-5940-9f66-06f6fcc108f0',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   SubscriptionArn: 'pending confirmation'
  // }
  return response;
};
```

若要執行範例，請在命令提示字元中輸入下列命令。

```
node subscribe-lambda.js
```

您可以在這裡找到此範例程式碼 GitHub。

取消訂閱主題

在此範例中，使用 Node.js 模組取消訂閱 Amazon SNS 主題訂閱。

創建一個`libs`目錄，並使用文件名創建一個 Node.js 模塊`snsClient.js`。將下列程式碼複製並貼到其中，以建立 Amazon SNS 用戶端物件。以您的`##`取代「AWS地區」。

```
import { SNSClient } from "@aws-sdk/client-sns";
```

```
// The AWS Region can be provided here using the `region` property. If you leave it
 blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

您可以在這裡找到此範例程式碼 GitHub。

以檔名 unsubscribe.js 建立一個 Node.js 模組。如先前所示設定 SDK，包括安裝所需的用戶端和套件。

建立包含SubscriptionArn參數的物件，並指定要取消訂閱的 Amazon 資源名稱 (ARN)。將參數傳遞至 SNS 用戶端類別的 UnsubscribeCommand 方法。

若要呼叫方UnsubscribeCommand法，請建立叫用 Amazon SNS 用戶端服務物件的非同步函數，並傳遞參數物件。

> **Note**
>
> 將########### *(ARN)* 取代主題訂閱，以取消訂閱。

```
import { UnsubscribeCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} subscriptionArn - The ARN of the subscription to cancel.
 */
const unsubscribe = async (
  subscriptionArn = "arn:aws:sns:us-east-1:xxxxxxxxxxxx:mytopic:xxxxxxxx-xxxx-xxxx-
xxxx-xxxxxxxxxxxx",
) => {
  const response = await snsClient.send(
    new UnsubscribeCommand({
      SubscriptionArn: subscriptionArn,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '0178259a-9204-507c-b620-78a7570a44c6',
  //     extendedRequestId: undefined,
```

```
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   }
  // }
  return response;
};
```

若要執行範例，請在命令提示字元中輸入下列命令。

```
node unsubscribe.js
```

您可以在這裡找到此範例程式碼 GitHub。

使用 Amazon SNS 發送短信

這個 Node.js 程式碼範例會說明：

- 如何取得和設定 Amazon SNS 的簡訊喜好設定。
- 如何檢查電話號碼是否選擇不要接收簡訊。
- 如何取得選擇不要接收簡訊的電話號碼清單。
- 如何傳送簡訊。

使用案例

您可以使用 Amazon SNS 傳送文字訊息或簡訊至啟用簡訊功能的裝置。您可以直接傳送訊息至一組電話號碼，或一次傳送一則訊息至多組電話號碼，只要訂閱那些電話號碼到主題並且傳送您的訊息到該主題即可。

在此範例中，您可以使用一系列 Node.js 模組將來自 Amazon SNS 的簡訊文字訊息發佈到啟用 SMS 的裝置。Node.js 模組會使用 SDK JavaScript 來發佈 SMS 訊息，使用用SNS戶端類別的下列方法：

- GetSMSAttributesCommand
- SetSMSAttributesCommand

- CheckIfPhoneNumberIsOptedOutCommand
- ListPhoneNumbersOptedOutCommand
- PublishCommand

先決條件任務

若要設定和執行此範例，您必須先完成這些任務：

- 設置項目環境以運行這些節點 TypeScript 示例，並安裝所需AWS SDK for JavaScript的第三方模塊。按照上的說明進行操作 GitHub。
- 透過使用者登入資料建立共用組態檔。有關提供共用認證檔案的詳細資訊，請參閱 AWSSDK 和工具參考指南中的共用設定和認證檔案。

> **Important**
>
> 這些範例示範如何使用 ECMAScript6 (ES6) 匯入/匯出用戶端服務物件和指令。
>
> - 這需要 Node.js 版本 13.x 或更高版本。要下載並安裝最新版本的 Node.js，請參閱 Node.js 下載。 。
> - 如果您偏好使用 CommonJS 語法，請參閱JavaScript ES6/共同語法。

取得簡訊屬性

使用 Amazon SNS 指定 SMS 簡訊的偏好設定，例如如何優化交付 (基於成本或可靠交付)、每月支出限制、訊息傳遞的記錄方式，以及是否訂閱每日 SMS 使用量報告。系統會擷取這些偏好設定，並將其設定為 Amazon SNS 的 SMS 屬性。

在此範例中，使用 Node.js 模組取得 Amazon SNS 中目前的 SMS 屬性。

創建一個`libs`目錄，並使用文件名創建一個 Node.js 模塊`snsClient.js`。將下列程式碼複製並貼到其中，以建立 Amazon SNS 用戶端物件。以您的`##`取代「AWS地區」。

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
 blank
// the SDK will default to the region set in your AWS config.
```

```
export const snsClient = new SNSClient({});
```

您可以在這裡找到此範例程式碼 GitHub。

以檔名 `get-sms-attributes.js` 建立一個 Node.js 模組。

如先前所示設定 SDK，包括下載所需的用戶端和套件。建立一個物件，其中包含用來取得簡訊屬性的參數，包括要擷取的個別屬性名稱。如需有關可用 SMS 屬性的詳細資訊，請參閱 Amazon 簡單通知服務 API 參考中的 SetSMSAttributes。

此範例會取得 `DefaultSMSType` 屬性，其可控制簡訊的傳送方式；`Promotional` 會將訊息交付最佳化以降低成本，而 `Transactional` 則會將訊息交付最佳化以達到最高的可靠性。將參數傳遞至 SNS 用戶端類別的 `SetTopicAttributesCommand` 方法。若要呼叫方`SetSMSAttributesCommand`法，請建立叫用 Amazon SNS 用戶端服務物件的非同步函數，並傳遞參數物件。

> **Note**
>
> 將####取代為屬性的名稱。

```
import { GetSMSAttributesCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

export const getSmsAttributes = async () => {
  const response = await snsClient.send(
    // If you have not modified the account-level mobile settings of SNS,
    // the DefaultSMSType is undefined. For this example, it was set to
    // Transactional.
    new GetSMSAttributesCommand({ attributes: ["DefaultSMSType"] }),
  );

  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '67ad8386-4169-58f1-bdb9-debd281d48d5',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
```

```
  //   },
  //   attributes: { DefaultSMSType: 'Transactional' }
  // }
  return response;
};
```

若要執行範例，請在命令提示字元中輸入下列命令。

```
node get-sms-attributes.js
```

您可以在這裡找到此範例程式碼 GitHub。

設定簡訊屬性

在此範例中，使用 Node.js 模組取得 Amazon SNS 中目前的 SMS 屬性。

創建一個libs目錄，並使用文件名創建一個 Node.js 模塊snsClient.js。將下列程式碼複製並貼到其中，以建立 Amazon SNS 用戶端物件。以您的##取代「AWS地區」。

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
 blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

您可以在這裡找到此範例程式碼 GitHub。

以檔名 set-sms-attribute-type.js 建立一個 Node.js 模組。如先前所示設定 SDK，包括安裝所需的用戶端和套件。建立一個物件，其中包含用來設定簡訊屬性的參數，包括要設定的個別屬性名稱，以及要為每個屬性設定的值。如需有關可用 SMS 屬性的詳細資訊，請參閱 Amazon 簡單通知服務 API 參考中的 SetSMSAttributes。

此範例將 DefaultSMSType 屬性設為 Transactional，藉此將訊息交付最佳化為達成最高的可靠性。將參數傳遞至 SNS 用戶端類別的 SetTopicAttributesCommand 方法。若要呼叫方SetSMSAttributesCommand法，請建立叫用 Amazon SNS 用戶端服務物件的非同步函數，並傳遞參數物件。

```
import { SetSMSAttributesCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";
```

```
/**
 * @param {"Transactional" | "Promotional"} defaultSmsType
 */
export const setSmsType = async (defaultSmsType = "Transactional") => {
  const response = await snsClient.send(
    new SetSMSAttributesCommand({
      attributes: {
        // Promotional – (Default) Noncritical messages, such as marketing messages.
        // Transactional – Critical messages that support customer transactions,
        // such as one-time passcodes for multi-factor authentication.
        DefaultSMSType: defaultSmsType,
      },
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '1885b977-2d7e-535e-8214-e44be727e265',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   }
  // }
  return response;
};
```

若要執行範例，請在命令提示字元中輸入下列命令。

```
node set-sms-attribute-type.js
```

您可以在這裡找到此範例程式碼 GitHub。

檢查電話號碼是否已停止接收

在此範例中，您可以使用 Node.js 模組來檢查電話號碼是否選擇不要接收簡訊。

創建一個`libs`目錄，並使用文件名創建一個 Node.js 模塊`snsClient.js`。將下列程式碼複製並貼到其中，以建立 Amazon SNS 用戶端物件。以您的`##`取代「AWS地區」。

```
import { SNSClient } from "@aws-sdk/client-sns";
```

```
// The AWS Region can be provided here using the `region` property. If you leave it
 blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

您可以在這裡找到此範例程式碼 GitHub。

以檔名 check-if-phone-number-is-opted-out.js 建立一個 Node.js 模組。依前述內容設定軟體開發套件。建立一個物件，其中包含要以參數形式檢查的電話號碼。

此範例會設定 PhoneNumber 參數，藉此指定要檢查的電話號碼。接著，將物件傳遞至 SNS 用戶端類別的 CheckIfPhoneNumberIsOptedOutCommand 方法。若要呼叫方CheckIfPhoneNumberIsOptedOutCommand法，請建立叫用 Amazon SNS 用戶端服務物件的非同步函數，並傳遞參數物件。

> **Note**
>
> 1.
>
> 用電###取代電話號碼。

```
import { CheckIfPhoneNumberIsOptedOutCommand } from "@aws-sdk/client-sns";

import { snsClient } from "../libs/snsClient.js";

export const checkIfPhoneNumberIsOptedOut = async (
  phoneNumber = "5555555555",
) => {
  const command = new CheckIfPhoneNumberIsOptedOutCommand({
    phoneNumber,
  });

  const response = await snsClient.send(command);
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '3341c28a-cdc8-5b39-a3ee-9fb0ee125732',
  //     extendedRequestId: undefined,
```

```
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   isOptedOut: false
  // }
  return response;
};
```

若要執行範例，請在命令提示字元中輸入下列命令。

```
node check-if-phone-number-is-opted-out.js
```

您可以在這裡找到此範例程式碼 GitHub。

列出已停止接收的電話號碼

在此範例中，您可以使用 Node.js 模組來取得選擇不要接收簡訊的電話號碼清單。

創建一個libs目錄，並使用文件名創建一個 Node.js 模塊snsClient.js。將下列程式碼複製並貼到其中，以建立 Amazon SNS 用戶端物件。以您的##取代「AWS地區」。

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
 blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

您可以在這裡找到此範例程式碼 GitHub。

以檔名 `list-phone-numbers-opted-out.js` 建立一個 Node.js 模組。依前述內容設定軟體開發套件。建立空白物件做為參數。

接著，將物件傳遞至 SNS 用戶端類別的 `ListPhoneNumbersOptedOutCommand` 方法。若要呼叫方ListPhoneNumbersOptedOutCommand法，請建立叫用 Amazon SNS 用戶端服務物件的非同步函數，並傳遞參數物件。

```
import { ListPhoneNumbersOptedOutCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

export const listPhoneNumbersOptedOut = async () => {
```

```
  const response = await snsClient.send(
    new ListPhoneNumbersOptedOutCommand({}),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '44ff72fd-1037-5042-ad96-2fc16601df42',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   phoneNumbers: ['+15555550100']
  // }
  return response;
};
```

若要執行範例，請在命令提示字元中輸入下列命令。

```
node list-phone-numbers-opted-out.js
```

您可以在這裡找到此範例程式碼 GitHub。

發佈簡訊

在此範例中，Node.js 模組可用來傳送簡訊至電話號碼。

創建一個libs目錄，並使用文件名創建一個 Node.js 模塊snsClient.js。將下列程式碼複製並貼到其中，以建立 Amazon SNS 用戶端物件。以您的##取代「AWS地區」。

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
 blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

您可以在這裡找到此範例程式碼 GitHub。

以檔名 publish-sms.js 建立一個 Node.js 模組。如先前所示設定 SDK，包括安裝所需的用戶端和套件。建立包含 Message 和 PhoneNumber 參數的物件。

當您傳送簡訊時，請指定使用 E.164 格式的電話號碼。E。164 是電話號碼結構的標準，用於國際電信通訊。遵照此格式的電話號碼可以有 15 位數的上限限制，前面加上加號 (+) 字元和國碼。例如，E.164 格式的美國電話號碼顯示為 +1001XXX5550100。

此範例會設定 PhoneNumber 參數，藉此指定要傳送訊息的電話號碼。接著，將物件傳遞至 SNS 用戶端類別的 PublishCommand 方法。若要呼叫方PublishCommand法，請建立叫用 Amazon SNS 服務物件的非同步函數，並傳遞參數物件。

> **Note**
>
> 用文###替換文本消息，並用電話號碼替換####。

```
import { PublishCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string | Record<string, any>} message - The message to send. Can be a plain
 string or an object
 *                                                  if you are using the `json`
 `MessageStructure`.
 * @param {*} phoneNumber - The phone number to send the message to.
 */
export const publish = async (
  message = "Hello from SNS!",
  phoneNumber = "+15555555555",
) => {
  const response = await snsClient.send(
    new PublishCommand({
      Message: message,
      // One of PhoneNumber, TopicArn, or TargetArn must be specified.
      PhoneNumber: phoneNumber,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '7410094f-efc7-5f52-af03-54737569ab77',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
```

```
  //     totalRetryDelay: 0
  //   },
  //   MessageId: 'xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx'
  // }
  return response;
};
```

若要執行範例，請在命令提示字元中輸入下列命令。

```
node publish-sms.js
```

您可以在這裡找到此範例程式碼 GitHub。

Amazon Transcribe 示例

Amazon Transcribe 可讓開發人員輕鬆將語音新增至其應用程式的文字功能。

Amazon Transcribe 的 JavaScript API 是通過TranscribeService客戶端類公開。

主題

- Amazon Transcribe 示例
- Amazon Transcribe 醫學示例

Amazon Transcribe 示例

在此範例中，會使用一系列 Node.js 模組來建立、列出及刪除轉錄工作，使用用TranscribeService戶端類別的下列方法：

- StartTranscriptionJobCommand

- ListTranscriptionJobsCommand
- DeleteTranscriptionJobCommand

如需 Amazon Transcribe 使用者的詳細資訊，請參閱 Amazon Transcri be 開發人員指南。

必要工作

若要設定和執行此範例，您必須先完成這些任務：

- 設置項目環境以運行這些節點 TypeScript 示例，並安裝所需 AWS SDK for JavaScript 的第三方模塊。按照上的說明進行操作 GitHub。
- 透過使用者登入資料建立共用組態檔。有關提供共用認證檔案的詳細資訊，請參閱 AWS SDK 和工具參考指南中的共用設定和認證檔案。

> **Important**
>
> 這些範例示範如何使用 ECMAScript6 (ES6) 匯入/匯出用戶端服務物件和指令。
>
> - 這需要 Node.js 版本 13.x 或更高版本。要下載並安裝最新版本的 Node.js，請參閱 Node.js 下載。 。
> - 如果您偏好使用 CommonJS 語法，請參閱 JavaScript ES6/共同語法

開始 Amazon Transcribe 工作

此範例示範如何使用啟動 Amazon Transcribe 轉錄工作。 AWS SDK for JavaScript如需詳細資訊，請參閱StartTranscriptionJobCommand。

創建一個`libs`目錄，並使用文件名創建一個 Node.js 模塊`transcribeClient.js`。將下面的代碼複製並粘貼到其中，以創建 Amazon Transcribe 客戶端對象。以您的##取代「 AWS 地區」。

```
import { TranscribeClient } from "@aws-sdk/client-transcribe";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create an Amazon Transcribe service client object.
const transcribeClient = new TranscribeClient({ region: REGION });
export { transcribeClient };
```

您可以在這裡找到這個範例程式碼 GitHub。

以檔名 `transcribe-create-job.js` 建立一個 Node.js 模組。請務必如先前所示設定 SDK，包括安裝所需的用戶端和套件。創建一個參數對象，指定所需的參數。使用StartMedicalTranscriptionJobCommand指令啟動工作。

> **Note**
>
> 將##############稱。若為##############出的 Amazon S3 儲存貯體。若為「####」，請指定工作類型。對於############的位置。若是來##########入媒體檔案的位置。

```
// Import the required AWS SDK clients and commands for Node.js
import { StartTranscriptionJobCommand } from "@aws-sdk/client-transcribe";
import { transcribeClient } from "./libs/transcribeClient.js";

// Set the parameters
export const params = {
  TranscriptionJobName: "JOB_NAME",
  LanguageCode: "LANGUAGE_CODE", // For example, 'en-US'
  MediaFormat: "SOURCE_FILE_FORMAT", // For example, 'wav'
  Media: {
    MediaFileUri: "SOURCE_LOCATION",
    // For example, "https://transcribe-demo.s3-REGION.amazonaws.com/hello_world.wav"
  },
  OutputBucketName: "OUTPUT_BUCKET_NAME"
};

export const run = async () => {
  try {
    const data = await transcribeClient.send(
      new StartTranscriptionJobCommand(params)
    );
    console.log("Success - put", data);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

若要執行範例，請在命令提示字元中輸入下列命令。

```
node transcribe-create-job.js
```

您可以在這裡找到此範例程式碼 GitHub。

列出 Amazon Transcribe 工作

此範例顯示如何使用列出 Amazon 轉錄轉錄任務。 AWS SDK for JavaScript若要取得有關可修改哪些其他設定的更多資訊，請參閱〈〉`ListTranscriptionJobCommand`。

創建一個`libs`目錄，並使用文件名創建一個 Node.js 模塊`transcribeClient.js`。將下面的代碼複製並粘貼到其中，以創建 Amazon Transcribe 客戶端對象。以您的`##`取代「 AWS 地區」。

```
import { TranscribeClient } from "@aws-sdk/client-transcribe";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create an Amazon Transcribe service client object.
const transcribeClient = new TranscribeClient({ region: REGION });
export { transcribeClient };
```

您可以在這裡找到這個範例程式碼 GitHub。

以檔名 `transcribe-list-jobs.js` 建立一個 Node.js 模組。請務必如先前所示設定 SDK，包括安裝所需的用戶端和套件。建立具有所需參數的參數物件。

> **Note**
>
> 以傳回的作業名稱必須包含的關鍵字取代 *KEY_WORD*。

```
// Import the required AWS SDK clients and commands for Node.js

import { ListTranscriptionJobsCommand } from "@aws-sdk/client-transcribe";
import { transcribeClient } from "./libs/transcribeClient.js";

// Set the parameters
export const params = {
  JobNameContains: "KEYWORD", // Not required. Returns only transcription
  // job names containing this string
};

export const run = async () => {
```

```
  try {
    const data = await transcribeClient.send(
      new ListTranscriptionJobsCommand(params)
    );
    console.log("Success", data.TranscriptionJobSummaries);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

若要執行範例，請在命令提示字元中輸入下列命令。

```
node transcribe-list-jobs.js
```

您可以在這裡找到此範例程式碼 GitHub。

刪除 Amazon Transcribe 任務

此範例顯示如何使用刪除 Amazon Transcribe 轉錄工作。 AWS SDK for JavaScript如需選用的詳細資訊，請參閱DeleteTranscriptionJobCommand。

創建一個libs目錄，並使用文件名創建一個 Node.js 模塊transcribeClient.js。將下面的代碼複製並粘貼到其中，以創建 Amazon Transcribe 客戶端對象。以您的##取代「 AWS 地區」。

```
import  { TranscribeClient }  from  "@aws-sdk/client-transcribe";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create Transcribe service object.
const transcribeClient = new TranscribeClient({ region: REGION });
export { transcribeClient };
```

您可以在這裡找到這個範例程式碼 GitHub。

以檔名 `transcribe-delete-job.js` 建立一個 Node.js 模組。請務必如先前所示設定 SDK，包括安裝所需的用戶端和套件。指定「 AWS 區域」，以及您要刪除的工作名稱。

> **Note**
>
> 將 *JOB_NAME* 取代為要刪除的工作名稱。

```
// Import the required AWS SDK clients and commands for Node.js
import { DeleteTranscriptionJobCommand } from "@aws-sdk/client-transcribe";
import { transcribeClient } from "./libs/transcribeClient.js";

// Set the parameters
export const params = {
  TranscriptionJobName: "JOB_NAME", // Required. For example, 'transciption_demo'
};

export const run = async () => {
  try {
    const data = await transcribeClient.send(
      new DeleteTranscriptionJobCommand(params)
    );
    console.log("Success - deleted");
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

若要執行範例，請在命令提示字元中輸入下列命令。

```
node transcribe-delete-job.js
```

您可以在這裡找到此範例程式碼 GitHub。

Amazon Transcribe 醫學示例

在此範例中，使用一系列 Node.js 模組來建立、列出和刪除使用用TranscribeService戶端類別的下列方法的醫療轉錄工作：

- StartMedicalTranscriptionJobCommand
- ListMedicalTranscriptionJobsCommand
- DeleteMedicalTranscriptionJobCommand

如需 Amazon Transcribe 使用者的詳細資訊，請參閱 Amazon Transcri be 開發人員指南。

必要工作

若要設定和執行此範例，您必須先完成這些任務：

- 設置項目環境以運行這些節點 TypeScript 示例，並安裝所需 AWS SDK for JavaScript 的第三方模塊。按照上的說明進行操作 GitHub。
- 透過使用者登入資料建立共用組態檔。有關提供共用認證檔案的詳細資訊，請參閱 AWS SDK 和工具參考指南中的共用設定和認證檔案。

> **Important**
>
> 這些範例示範如何使用 ECMAScript6 (ES6) 匯入/匯出用戶端服務物件和指令。
>
> - 這需要 Node.js 版本 13.x 或更高版本。要下載並安裝最新版本的 Node.js，請參閱 Node.js 下載。 。
> - 如果您偏好使用 CommonJS 語法，請參閱 JavaScript ES6/共同語法

開始 Amazon Transcribe 醫療轉錄工作

此範例將示範如何開始使用 Amazon 轉錄醫療轉錄工作。 AWS SDK for JavaScript如需詳細資訊，請參閱啟動 MedicalTranscription Job。

創建一個`libs`目錄，並使用文件名創建一個 Node.js 模塊`transcribeClient.js`。將下面的代碼複製並粘貼到其中，以創建 Amazon Transcribe 客戶端對象。以您的`##`取代「 AWS 地區」。

```
import  { TranscribeClient }  from  "@aws-sdk/client-transcribe";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create Transcribe service object.
const transcribeClient = new TranscribeClient({ region: REGION });
export { transcribeClient };
```

您可以在這裡找到這個範例程式碼 GitHub。

以檔名 `transcribe-create-medical-job.js` 建立一個 Node.js 模組。請務必如先前所示設定 SDK，包括安裝所需的用戶端和套件。創建一個參數對象，指定所需的參數。使用`StartMedicalTranscriptionJobCommand`命令開始醫療工作。

> **Note**
>
> 用#########取代醫療工作名稱。若為############出的 Amazon S3 儲存貯體。若為「####」，請指定工作類型。對於############的位置。若是來##########入媒體檔案的位置。

```
// Import the required AWS SDK clients and commands for Node.js
import { StartMedicalTranscriptionJobCommand } from "@aws-sdk/client-transcribe";
import { transcribeClient } from "./libs/transcribeClient.js";

// Set the parameters
export const params = {
  MedicalTranscriptionJobName: "MEDICAL_JOB_NAME", // Required
  OutputBucketName: "OUTPUT_BUCKET_NAME", // Required
  Specialty: "PRIMARYCARE", // Required. Possible values are 'PRIMARYCARE'
  Type: "JOB_TYPE", // Required. Possible values are 'CONVERSATION' and 'DICTATION'
  LanguageCode: "LANGUAGE_CODE", // For example, 'en-US'
  MediaFormat: "SOURCE_FILE_FORMAT", // For example, 'wav'
  Media: {
    MediaFileUri: "SOURCE_FILE_LOCATION",
    // The S3 object location of the input media file. The URI must be in the same
 region
    // as the API endpoint that you are calling.For example,
    // "https://transcribe-demo.s3-REGION.amazonaws.com/hello_world.wav"
  },
};

export const run = async () => {
  try {
    const data = await transcribeClient.send(
      new StartMedicalTranscriptionJobCommand(params)
    );
    console.log("Success - put", data);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

若要執行範例，請在命令提示字元中輸入下列命令。

```
node transcribe-create-medical-job.js
```

您可以在這裡找到此範例程式碼 GitHub。

列出 Amazon Transcribe 醫療工作

此範例示範如何使用列出 Amazon 轉錄轉錄任務。 AWS SDK for JavaScript如需詳細資訊，請參閱ListTranscriptionMedicalJobs命令。

創建一個libs目錄，並使用文件名創建一個 Node.js 模塊transcribeClient.js。將下面的代碼複製並粘貼到其中，以創建 Amazon Transcribe 客戶端對象。以您的##取代「 AWS 地區」。

```
import { TranscribeClient } from "@aws-sdk/client-transcribe";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create an Amazon Transcribe service client object.
const transcribeClient = new TranscribeClient({ region: REGION });
export { transcribeClient };
```

您可以在這裡找到這個範例程式碼 GitHub。

以檔名 `transcribe-list-medical-jobs.js` 建立一個 Node.js 模組。請務必如先前所示設定 SDK，包括安裝所需的用戶端和套件。使用所需參數建立參數物件，並使用ListMedicalTranscriptionJobsCommand指令列出醫療工作。

> **Note**
>
> ################## KEYWORD#

```
// Import the required AWS SDK clients and commands for Node.js

import { ListMedicalTranscriptionJobsCommand } from "@aws-sdk/client-transcribe";
import { transcribeClient } from "./libs/transcribeClient.js";

// Set the parameters
export const params = {
  JobNameContains: "KEYWORD", // Returns only transcription job names containing this
 string
```

```
};

export const run = async () => {
  try {
    const data = await transcribeClient.send(
      new ListMedicalTranscriptionJobsCommand(params)
    );
    console.log("Success", data.MedicalTranscriptionJobName);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

若要執行範例，請在命令提示字元中輸入下列命令。

```
node transcribe-list-medical-jobs.js
```

您可以在這裡找到此範例程式碼 GitHub。

刪除 Amazon Transcribe 醫療工作

此範例顯示如何使用刪除 Amazon Transcribe 轉錄工作。 AWS SDK for JavaScript如需選用的詳細資訊，請參閱DeleteTranscriptionMedicalJobCommand。

創建一個libs目錄，並使用文件名創建一個 Node.js 模塊transcribeClient.js。將下面的代碼複製並粘貼到其中，以創建 Amazon Transcribe 客戶端對象。以您的##取代「 AWS 地區」。

```
import  { TranscribeClient }  from  "@aws-sdk/client-transcribe";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create Transcribe service object.
const transcribeClient = new TranscribeClient({ region: REGION });
export { transcribeClient };
```

您可以在這裡找到這個範例程式碼 GitHub。

以檔名 transcribe-delete-job.js 建立一個 Node.js 模組。請務必如先前所示設定 SDK，包括安裝所需的用戶端和套件。使用所需參數建立參數物件，並使用DeleteMedicalJobCommand指令刪除醫療工作。

> **Note**
>
> 將 *JOB_NAME* 取代為要刪除的工作名稱。

```
// Import the required AWS SDK clients and commands for Node.js
import { DeleteMedicalTranscriptionJobCommand } from "@aws-sdk/client-transcribe";
import { transcribeClient } from "./libs/transcribeClient.js";

// Set the parameters
export const params = {
  MedicalTranscriptionJobName: "MEDICAL_JOB_NAME", // For example,
 'medical_transciption_demo'
};

export const run = async () => {
  try {
    const data = await transcribeClient.send(
      new DeleteMedicalTranscriptionJobCommand(params)
    );
    console.log("Success - deleted");
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

若要執行範例，請在命令提示字元中輸入下列命令。

```
node transcribe-delete-medical-job.js
```

您可以在這裡找到此範例程式碼 GitHub。

在 Amazon EC2 實例上設置 Node.js

將 Node.js 與 for 搭配使用的常見案SDK例 JavaScript 是在 Amazon 彈性運算雲端 (AmazonEC2) 執行個體上設定和執行 Node.js Web 應用程式。在本教學課程中，您將建立 Linux 執行個體，使用連線至該執行個體SSH，然後安裝 Node.js 以在該執行個體上執行。

必要條件

本教學課程假設您已經啟動了具有可從網際網路存取的公用DNS名稱的 Linux 執行個體，而且您可以使用該執行個體進行連線SSH。如需詳細資訊，請參閱 Amazon EC2 使用者指南中的步驟 1：啟動執行個體。

> **Important**
>
> 在啟動新的 Amazon EC2 實例時，請使用 Amazon Linux 2023 Amazon 機器映像（AMI）。

您還必須先設定安全群組，允許 SSH (連接埠 22)、 HTTP (連接埠 80) 和 HTTPS (連接埠 443) 連線。如需這些先決條件的詳細資訊，請參閱 Amazon EC2使用者指南EC2中的使用 Amazon 設定。

程序

下列程序可協助您在 Amazon Linux 執行個體上安裝 Node.js。您可以使用此伺服器來託管 Node.js Web 應用程式。

在 Linux 執行個體上設定 Node.js

1. 使用時 Connect 到您的 Linux 執行個`ec2-user`體SSH。
2. 通過在命令行中鍵入以下命令來安裝節點版本管理器（`nvm`）。

> **Warning**
>
> AWS 不控制下列程式碼。在您執行前，請務必驗證其真確性及完整性。有關此代碼的更多信息可以在 nvm GitHub 存儲庫中找到。

```
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.7/install.sh | bash
```

我們將使`nvm`用安裝 Node.js，因為`nvm`可以安裝多個版本的 Node.js，並允許您在它們之間切換。

3. nvm通過在命令行中鍵入以下命令來加載。

```
source ~/.bashrc
```

4. 使用 nvm 安裝最新LTS版本的 Node.js，方法是在命令列中輸入下列命令。

```
nvm install --lts
```

安裝 Node.js 也會安裝節點 Package 管理員 (npm)，以便您可以視需要安裝其他模組。

5. 在命令列中輸入以下指令，測試安裝的 Node.js 是否能正常運作。

```
node -e "console.log('Running Node.js ' + process.version)"
```

這會顯示下列訊息，以指出正在執行的 Node.js 版本。

```
Running Node.js VERSION
```

> **Note**
>
> 節點安裝僅適用於目前的 Amazon EC2 工作階段。如果您重新啟動CLI工作階段，則需要再次使用 nvm 來啟用已安裝的節點版本。如果執行個體終止，則需要再次安裝節點。另一種方法是在擁有要保留的組EC2態後，建立 Amazon Machine Image (AMI)，如以下主題所述。

創建一個 Amazon 機器映像（AMI）

在 Amazon EC2 執行個體上安裝 Node.js 之後，您可以從該執行個體建立 Amazon 機器映像 (AMI)。建立AMI使用相同的 Node.js 安裝可以輕鬆佈建多個 Amazon EC2 執行個體。如需AMI從現有執行個體建立的詳細資訊，請參閱 Amazon EC2使用者指南AMI中的建立 Amazon EBS 支援 Linux。

相關資源

如需有關本主題中使用之命令和軟體的詳細資訊，請參閱下列網頁：

- 節點版本管理器（nvm）-請參閱上的 nvm 軟件庫。 GitHub
- 節點 Package 管理員 (npm) — 請參閱 npm 網站。

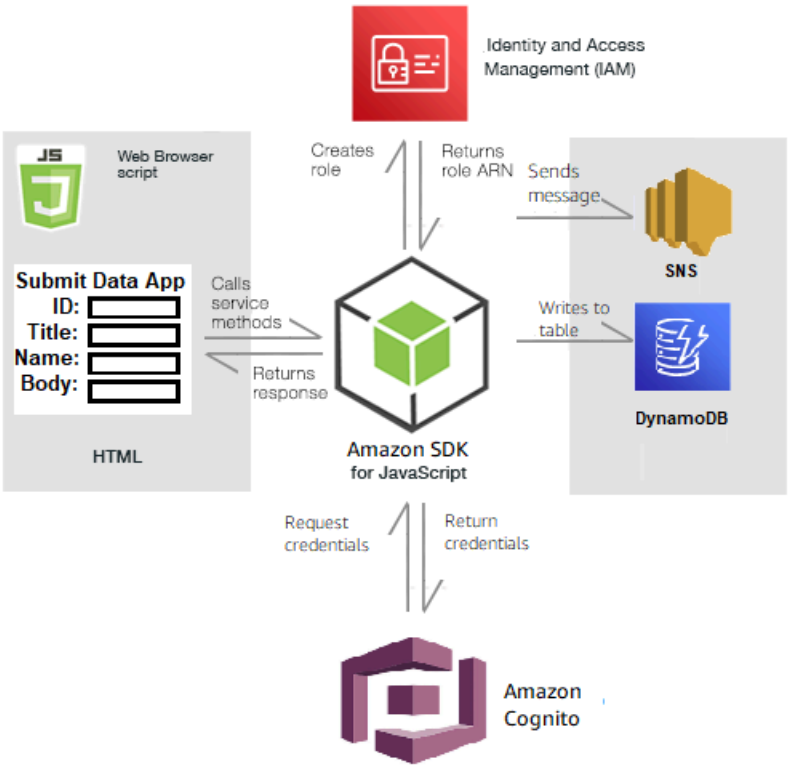
建置應用程式以將資料提交至 DynamoDB

本跨服務 Node.js 教學課程示範如何建立可讓使用者將資料提交至 Amazon DynamoDB 表格的應用程式。此應用程序使用以下服務：

- AWS Identity and Access Management (IAM) 和 Amazon Cognito 的授權和許可。
- 用於建立和更新資料表的 Amazon DynamoDB DynamoDB。
- Amazon Simple Notification Service (Amazon SNS) 可在使用者更新表格時通知應用程式管理員。

該方案

在本教學課程中，HTML 頁面提供了一個以瀏覽器為基礎的應用程式，用於將資料提交至 Amazon DynamoDB 表格。當使用者更新表格時，應用程式會使用 Amazon SNS 通知應用程式管理員。



若要建置應用程式：

1. 先決條件
2. 佈建資源
3. 建立 HTML 格式
4. 建立瀏覽器指令碼

5. 後續步驟

必要條件

完成下列先決條件工作：

- 設置項目環境以運行這些節點 TypeScript 示例，並安裝所需AWS SDK for JavaScript的第三方模塊。按照上的說明進行操作 GitHub。
- 透過使用者登入資料建立共用組態檔。有關提供共用認證檔案的詳細資訊，請參閱 AWSSDK 和工具參考指南中的共用設定和認證檔案。

建立資AWS源

此應用程序需要以下資源：

- AWS Identity and Access Management(IAM) 具有下列許可的未驗證 Amazon Cognito 使用者角色：
  - sns:Publish
  - 動態：PutItem
- 一個 DynamoDB 資料表。

您可以在AWS主控台中手動建立這些資源，但我們建議您使用本教學課程中所AWS CloudFormation述來佈建這些資源。

使用建立AWS資源 AWS CloudFormation

AWS CloudFormation 可讓您以可預期和重複的方式建立及佈建 AWS 基礎設施部署。如需有關 AWS CloudFormation 的詳細資訊，請參閱《使用者指南》AWS CloudFormation。

要使用以下命AWS CloudFormation令創建堆棧AWS CLI：

1. 安裝並設定AWS CLI以下AWS CLI使用者指南中的指示。
2. 在項目文件夾的根目錄`setup.yaml`中創建一個名為的文件，然後將此處的內容複製 GitHub到其中。

> **Note**
>
> AWS CloudFormation範本是使用中AWS CDK提供的來產生的 GitHub。如需關於 AWS CDK 的詳細資訊，請參閱《AWS Cloud Development Kit (AWS CDK) 開發人員指南》。

3. 從命令行運行以下命令，將 *STACK_NAME* 替換為堆棧的唯一名稱，並在您所在##中的 AWS REGION。

> **Important**
>
> 堆疊名稱在AWS區域和AWS帳戶中必須是唯一的。您最多可以指定 128 個字元，且允許使用數字和連字號。

```
aws cloudformation create-stack --stack-name STACK_NAME --template-body file://
setup.yaml --capabilities CAPABILITY_IAM --region REGION
```

若要取得有關指`create-stack`令參數的更多資訊，請參閱《指AWS CLI令參考指南》和《AWS CloudFormation使用指南》。

若要檢視建立的資源，請AWS CloudFormation在AWS管理主控台中開啟，選擇堆疊，然後選取 [資源] 索引標籤。

4. 建立堆疊時，請使用填AWS SDK for JavaScript入 DynamoDB 表格，如中所述。填入表格

填入表格

要填充表格，請先創建一個名為的目錄`libs`，並在其中創建一個名為的文件`dynamoClient.js`，然後將下面的內容粘貼到其中。將##取代為您的AWS區域，並以亞馬遜認可####### *IDENTITY_POOL_ID*。這會建 DynamoDB 用戶端物件。

```
import { CognitoIdentityClient } from "@aws-sdk/client-cognito-identity";
import { fromCognitoIdentityPool } from "@aws-sdk/credential-provider-cognito-
identity";
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

const REGION = "REGION";
const IDENTITY_POOL_ID = "IDENTITY_POOL_ID"; // An Amazon Cognito Identity Pool ID.
```

```
// Create an Amazon DynaomDB service client object.
const dynamoClient = new DynamoDBClient({
  region: REGION,
  credentials: fromCognitoIdentityPool({
    client: new CognitoIdentityClient({ region: REGION }),
    identityPoolId: IDENTITY_POOL_ID,
  }),
});

export { dynamoClient };
```

此程式碼可在此處取得 GitHub。

接下來，在項目文件dynamoAppHelperFiles夾中創建一個文件夾，update-table.js在其中創建一個文件，然後將此處的內容複製 GitHub到其中。

```
// Import required AWS SDK clients and commands for Node.js
import { PutItemCommand } from "@aws-sdk/client-dynamodb";
import { dynamoClient } from "../libs/dynamoClient.js";

// Set the parameters
export const params = {
  TableName: "Items",
  Item: {
    id: { N: "1" },
    title: { S: "aTitle" },
    name: { S: "aName" },
    body: { S: "aBody" },
  },
};

export const run = async () => {
  try {
    const data = await dynamoClient.send(new PutItemCommand(params));
    console.log("success");
    console.log(data);
  } catch (err) {
    console.error(err);
  }
};
run();
```

從命令行運行以下命令。

```
node update-table.js
```

此程式碼可在此處取得 GitHub。

創建應用程序的前端頁面

在這裡，您可以創建應用程序的前端 HTML 瀏覽器頁面。

創建一個DynamoDBApp目錄，創建一個名為的文件index.html，然後從這裡複製代碼 GitHub。該script元素添加了main.js文件，其中包含了示例所需 JavaScript的所有文件。您將在本自學課程稍後建立main.js檔案。中的其餘程式碼index.html會建立瀏覽器頁面，以擷取使用者輸入的資料。

您可以在這裡找到這個範例程式碼 GitHub。

建立瀏覽器指令碼

首先，建立範例所需的服務用戶端物件。創建一個libs目錄，創建一個目錄snsClient.js，並將下面的代碼粘貼到其中。取代每個*##*和*#### _POOL_ID*。

> **Note**
>
> 使用您在建立資AWS源 其中建立的 Amazon Cognito 身分識別集區的識別碼。

```
import { CognitoIdentityClient } from "@aws-sdk/client-cognito-identity";
import { fromCognitoIdentityPool } from "@aws-sdk/credential-provider-cognito-
identity";
import { SNSClient } from "@aws-sdk/client-sns";

const REGION = "REGION";
const IDENTITY_POOL_ID = "IDENTITY_POOL_ID"; // An Amazon Cognito Identity Pool ID.

// Create an Amazon Comprehend service client object.
const snsClient = new SNSClient({
  region: REGION,
  credentials: fromCognitoIdentityPool({
    client: new CognitoIdentityClient({ region: REGION }),
    identityPoolId: IDENTITY_POOL_ID,
  }),
});
```

```
export { snsClient };
```

此程式碼可在此處取得 GitHub。 。

若要建立此範例的瀏覽器指令碼，請在名為的資料夾中建立 Node.js 模組DynamoDBApp，add_data.js並將下面的程式碼貼到其中。此submitData函數會將資料提交至 DynamoDB 表格，並使用 Amazon SNS 將簡訊文字傳送給應用程式管理員。

在submitData函數中，為目標電話號碼、在應用程式界面上輸入的值以及 Amazon S3 儲存貯體的名稱宣告變數。接下來，創建一個參數對象，以將項目添加到表中。如果沒有任何值是空的，將項目submitData添加到表中，並發送消息。請記住，使該功能可供瀏覽器使用window.submitData = submitData。

```
// Import required AWS SDK clients and commands for Node.js
import { PutItemCommand } from "@aws-sdk/client-dynamodb";
import { PublishCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";
import { dynamoClient } from "../libs/dynamoClient.js";

export const submitData = async () => {
  //Set the parameters
  // Capture the values entered in each field in the browser (by id).
  const id = document.getElementById("id").value;
  const title = document.getElementById("title").value;
  const name = document.getElementById("name").value;
  const body = document.getElementById("body").value;
  //Set the table name.
  const tableName = "Items";

  //Set the parameters for the table
  const params = {
    TableName: tableName,
    // Define the attributes and values of the item to be added. Adding ' + "" '
 converts a value to
    // a string.
    Item: {
      id: { N: id + "" },
      title: { S: title + "" },
      name: { S: name + "" },
      body: { S: body + "" },
    },
  };
  // Check that all the fields are completed.
```

```
  if (id != "" && title != "" && name != "" && body != "") {
    try {
      //Upload the item to the table
      await dynamoClient.send(new PutItemCommand(params));
      alert("Data added to table.");
      try {
        // Create the message parameters object.
        const messageParams = {
          Message: "A new item with ID value was added to the DynamoDB",
          PhoneNumber: "PHONE_NUMBER", //PHONE_NUMBER, in the E.164 phone number
 structure.
          // For example, ak standard local formatted number, such as (415) 555-2671,
 is +14155552671 in E.164
          // format, where '1' in the country code.
        };
        // Send the SNS message
        const data = await snsClient.send(new PublishCommand(messageParams));
        console.log(
          "Success, message published. MessageID is " + data.MessageId,
        );
      } catch (err) {
        // Display error message if error is not sent
        console.error(err, err.stack);
      }
    } catch (err) {
      // Display error message if item is no added to table
      console.error(
        "An error occurred. Check the console for further information",
        err,
      );
    }
    // Display alert if all field are not completed.
  } else {
    alert("Enter data in each field.");
  }
};
// Expose the function to the browser
window.submitData = submitData;
```

您可以在這裡找到這個範例程式碼 GitHub。

最後，在命令提示字元中執行下列命令，將此範例中的項目捆綁在名 JavaScript 為的檔案中`main.js`：

```
webpack add_data.js --mode development --target web --devtool false -o main.js
```

> **Note**
>
> 如需有關安裝 webpack 的資訊，請參閱捆綁應用程序與網絡包。

若要執行應用程式，請在瀏覽器`index.html`上開啟。

刪除資源

如本教程開頭所述，請務必終止所有你創建的資源，同時通過本教程，以確保你不收費。您可以透過刪除您在本教學課程建立資AWS源 主題中建立的AWS CloudFormation堆疊來執行此操作，如下所示：

1. 在AWS管理主控台AWS CloudFormation中開啟。
2. 開啟「堆疊」頁面，然後選取堆疊。
3. 選擇刪除。

如需更多AWS跨服務範例，請參閱AWS SDK for JavaScript跨服務範例。

使用 API Gateway 叫用 Lambda

您可以使用 Amazon API Gateway 來叫用 Lambda 函數，這是一項用於大規模建立、發佈、維護、監控和保護 REST、HTTP 和 WebSocket API 的AWS服務。API 開發人員可以建立 API，以存取 AWS 或其他 Web 服務，以及 AWS 雲端中所存放的資料。身為 API Gateway 開發人員，您可以建立 API，以便在自己的用戶端應用程式中使用。如需詳細資訊，請參閱什麼是 Amazon API Gateway。

AWS Lambda是一種運算服務，可讓您在不佈建或管理伺服器的情況下執行程式碼。您可以使用各種程式設計語言建立 Lambda 函數。如需 AWS Lambda 的詳細資訊，請參閱什麼是 AWS Lambda？。

在此範例中，您可以使用 Lambda JavaScript 執行階段 API 建立 Lambda 函數。這個範例會調用不同的 AWS 服務來執行特定使用案例。例如，假設某個組織傳送行動文字訊息給其員工，並在一年週年日期祝賀他們，如本圖所示。

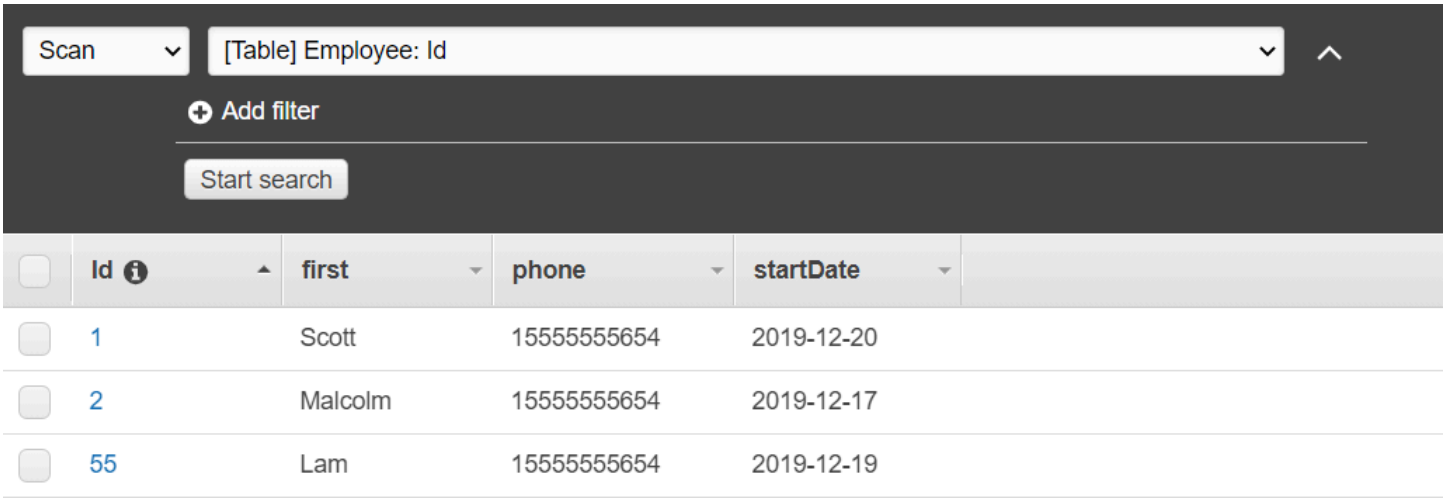
此範例大約需要 20 分鐘才能完成。

此範例說明如何使用 JavaScript 邏輯建立執行此使用案例的解決方案。例如，您將學習如何讀取資料庫，以判斷哪些員工已經達到一年週年紀念日、如何處理資料，以及使用 Lambda 函數傳送文字訊息。然後，您將學習如何使用 API Gateway 使用 Rest 端點來調用此AWS Lambda函數。例如，您可以使用以下 curl 命令叫用 Lambda 函數：

```
curl -XGET "https://xxxxqjko1o3.execute-api.us-east-1.amazonaws.com/cronstage/
employee"
```

本AWS教學課程使用名為員工的 Amazon DynamoDB 表格，其中包含這些欄位。

- id-表的主鍵。
- 名字-員工的名字。
- 電話-員工的電話號碼。
- 開始日期-員工的開始日期。



> **Important**
>
> 完成成本：本文件中包含的AWS服務包含在AWS免費方案中。但是，請務必在完成此範例之後終止所有資源，以確保不會向您收費。

若要建置應用程式：

1. 完成先決條
2. 建立資AWS源
3. 準備瀏覽器腳本
4. 建立並上傳 Lambda 數
5. 部署 Lambda 函數
6. 運行應用程序
7. 刪除資源

必要工作

若要設定和執行此範例，您必須先完成這些任務：

- 設置項目環境以運行這些節點 TypeScript 示例，並安裝所需AWS SDK for JavaScript的第三方模塊。按照上的說明進行操作 GitHub。
- 透過使用者登入資料建立共用組態檔。有關提供共用認證檔案的詳細資訊，請參閱 AWSSDK 和工具參考指南中的共用設定和認證檔案。

建立資AWS源

本教學課程需要下列資源：

- Amazon DynamoDB 表格以名為`Employee`的金鑰命名，以`Id`及上圖中顯示的欄位。請務必輸入正確的資料，包括您要測試此使用案例的有效行動電話。如需詳細資訊，請參閱建立資料表。
- 具有附加權限的 IAM 角色，可執行 Lambda 函數。
- 一個 Amazon S3 存儲桶來託管 Lambda 函數。

您可以手動建立這些資源，但我們建議您使用本教學課程中所AWS CloudFormation述來佈建這些資源。

使用建立AWS資源 AWS CloudFormation

AWS CloudFormation 可讓您以可預期和重複的方式建立及佈建 AWS 基礎設施部署。如需有關 AWS CloudFormation 的詳細資訊，請參閱《 使用者指南》AWS CloudFormation。

要使用以下命AWS CloudFormation令創建堆棧AWS CLI：

1. 安裝並設定AWS CLI以下AWS CLI使用者指南中的指示。
2. 在項目文件夾的根目錄`setup.yaml`中創建一個名為的文件，然後將此處的內容複製 GitHub到其中。

   > **Note**
   >
   > AWS CloudFormation範本是使用中AWS CDK提供的來產生的 GitHub。如需關於 AWS CDK 的詳細資訊，請參閱《AWS Cloud Development Kit (AWS CDK) 開發人員指南》。

3. 從命令列執行下列命令，並以*######### STACK_NAME*。

   > **Important**
   >
   > 堆疊名稱在AWS區域和AWS帳戶中必須是唯一的。您最多可以指定 128 個字元，且允許使用數字和連字號。

   ```
   aws cloudformation create-stack --stack-name STACK_NAME --template-body file://
   setup.yaml --capabilities CAPABILITY_IAM
   ```

   若要取得有關指`create-stack`令參數的更多資訊，請參閱《指AWS CLI令參考指南》和《AWS CloudFormation使用指南》。
4. 接下來，依照程序填入資料表填入表格。

填入表格

要填充表格，請先創建一個名為的目錄`libs`，並在其中創建一個名為的文件`dynamoClient.js`，然後將下面的內容粘貼到其中。

```
const { DynamoDBClient } = require ( "@aws-sdk/client-dynamodb" );
// Set the AWS Region.
const REGION = "REGION"; // e.g. "us-east-1"
 // Create an Amazon Lambda service client object.
const dynamoClient = new DynamoDBClient({region:REGION});
module.exports = { dynamoClient };
```

此程式碼可在此處取得 GitHub。

接下來，在項目文件夾的根目錄`populate-table.js`中創建一個名為的文件，然後將此處的內容複製 GitHub到其中。對於其中一個項目，請將`phone`屬性的值取代為 E.164 格式的有效行動電話號碼，並取代為今天日期的值。`startDate`

從命令行運行以下命令。

```
node populate-table.js
```

```
const { BatchWriteItemCommand } = require ( "aws-sdk/client-dynamodb" );
const {dynamoClient} = require ( "./libs/dynamoClient" );

// Set the parameters.
export const params = {
  RequestItems: {
    Employees: [
      {
        PutRequest: {
          Item: {
            id: { N: "1" },
            firstName: { S: "Bob" },
            phone: { N: "155555555555654" },
            startDate: { S: "2019-12-20" },
          },
        },
      },
      {
        PutRequest: {
          Item: {
            id: { N: "2" },
            firstName: { S: "Xing" },
            phone: { N: "155555555555653" },
```

```
            startDate: { S: "2019-12-17" },
          },
        },
      },
      {
        PutRequest: {
          Item: {
            id: { N: "55" },
            firstName: { S: "Harriette" },
            phone: { N: "155555555555652" },
            startDate: { S: "2019-12-19" },
          },
        },
      },
    ],
  },
};

export const run = async () => {
  try {
    const data = await dbclient.send(new BatchWriteItemCommand(params));
    console.log("Success", data);
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

此程式碼可在此處取得 GitHub。

建立 AWS Lambda 函數

設定軟體開發套件

在libs目錄中，建立名為snsClient.js和的檔案lambdaClient.js，並將下列內容分別貼到這些檔案中。

```
const  { SNSClient } = require ( "@aws-sdk/client-sns" );
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create an Amazon SNS service client object.
const snsClient = new SNSClient({ region: REGION });
```

```
module.exports = { snsClient };
```

將「*##*」取代為「AWS區域」。此程式碼可在此處取得 GitHub。

```
const { LambdaClient } = require ( "@aws-sdk/client-lambda" );
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create an Amazon Lambda service client object.
const lambdaClient = new LambdaClient({ region: REGION });
module.exports = { lambdaClient };
```

將「*##*」取代為「AWS區域」。此程式碼可在此處取得 GitHub。

首先，導入所需的AWS SDK for JavaScript（v3）模塊和命令。然後計算今天的日期並將其分配給參數。第三，建立的參數ScanCommand。將 *TABLE_NAME* 取代為您在此範建立資AWS源 例中所建立之資料表的名稱。

下列程式碼片段說明此步驟。(如需完整範例，請參閱捆綁 Lambda 函數。)

```
"use strict";
const { ScanCommand } = require("@aws-sdk/client-dynamodb");
const { PublishCommand } = require("@aws-sdk/client-sns");
const {snsClient} = require ( "./libs/snsClient" );
const {dynamoClient} = require ( "./libs/dynamoClient" );

// Get today's date.
const today = new Date();
const dd = String(today.getDate()).padStart(2, "0");
const mm = String(today.getMonth() + 1).padStart(2, "0"); //January is 0!
const yyyy = today.getFullYear();
const date = yyyy + "-" + mm + "-" + dd;

// Set the parameters for the ScanCommand method.
const params = {
  // Specify which items in the results are returned.
  FilterExpression: "startDate = :topic",
  // Define the expression attribute value, which are substitutes for the values you
 want to compare.
  ExpressionAttributeValues: {
    ":topic": { S: date },
  },
```

```
  // Set the projection expression, which are the attributes that you want.
  ProjectionExpression: "firstName, phone",
  TableName: "Employees",
};
```

掃描 DynamoDB 料表

首先，建立呼叫的異步/等待函數，sendText以使用 Amazon SNS 發佈文字訊息。PublishCommand然後，新增try區塊模式，針對今天的工作週年紀念日掃描 DynamoDB 表格，然後呼叫sendText函數以傳送文字訊息給這些員工。如果發生錯誤，則調用catch塊。

下列程式碼片段說明此步驟。(如需完整範例，請參閱捆綁 Lambda 函數。)

```
// Helper function to send message using Amazon SNS.
exports.handler = async () => {
  // Helper function to send message using Amazon SNS.
  async function sendText(textParams) {
    try {
      await snsClient.send(new PublishCommand(textParams));
      console.log("Message sent");
    } catch (err) {
      console.log("Error, message not sent ", err);
    }
  }
  try {
    // Scan the table to identify employees with work anniversary today.
    const data = await dynamoClient.send(new ScanCommand(params));
    data.Items.forEach(function (element) {
      const textParams = {
        PhoneNumber: element.phone.N,
        Message:
            "Hi " +
            element.firstName.S +
            "; congratulations on your work anniversary!",
      };
      // Send message using Amazon SNS.
      sendText(textParams);
    });
  } catch (err) {
    console.log("Error, could not scan table ", err);
  }
};
```

捆綁 Lambda 函數

本主題說明如何將此範例中的`mylambdafunction.ts`和所需AWS SDK for JavaScript模組捆綁到名為的隨附檔案中`index.js`。

1. 如果您還沒有，請按照此示例中必要工作的安裝 webpack。

   > **Note**
   >
   > 如需 Webpack 的相關資訊，請參閱捆綁應用程序與網絡包。

2. 在命令列中執行下列命令，將此範例中 JavaScript 的內容捆綁到名為的檔案中`<index.js>`：

   ```
   webpack mylambdafunction.ts --mode development --target node --devtool false --
   output-library-target umd -o index.js
   ```

   > **Important**
   >
   > 請注意輸出已命名`index.js`。這是因為 Lambda 函數必須具有`index.js`處理常式才能運作。

3. 將隨附的輸出檔案壓縮成一個名為的 ZIP 檔案`mylambdafunction.zip`。`index.js`
4. 上傳`mylambdafunction.zip`到您在本教學建立資AWS源 主題中建立的 Amazon S3 儲存貯體。

部署 Lambda 函數。

在項目的根目錄中，創建一個`lambda-function-setup.ts`文件，然後將下面的內容粘貼到其中。

將 *BUCKET_NAME* 取代為您將 Lambda 函數的 ZIP 版本上傳到的 Amazon S3 儲存貯體的名稱。將 *ZIP_FILE_NAME* 取代為您的 Lambda 函數之 ZIP 版本的名稱。將*##*取代為您在本教學建立資AWS源主題中建立的 IAM 角色的 Amazon 資源編號 (ARN)。將 *Lambda ###### Lambda ####*稱。

```
// Load the required Lambda client and commands.
const {
  CreateFunctionCommand
} = require ( "@aws-sdk/client-lambda" );
const { lambdaClient} = require ( "./libs/lambdaClient.js );
```

```
// Set the parameters.
const params = {
  Code: {
    S3Bucket: "BUCKET_NAME", // BUCKET_NAME
    S3Key: "ZIP_FILE_NAME", // ZIP_FILE_NAME
  },
  FunctionName: "LAMBDA_FUNCTION_NAME",
  Handler: "index.handler",
  Role: "IAM_ROLE_ARN", // IAM_ROLE_ARN; e.g., arn:aws:iam::650138640062:role/v3-
lambda-tutorial-lambda-role
  Runtime: "nodejs12.x",
  Description:
    "Scans a DynamoDB table of employee details and using Amazon Simple Notification
 Services (Amazon SNS) to " +
    "send employees an email on each anniversary of their start-date.",
};

const run = async () => {
  try {
    const data = await lambdaClient.send(new CreateFunctionCommand(params));
    console.log("Success", data); // successful response
  } catch (err) {
    console.log("Error", err); // an error occurred
  }
};
run();
```

在命令列輸入下列命令以部署 Lambda 函數。

```
node lambda-function-setup.ts
```

此程式碼範例可在此處取得 GitHub。

設定 API Gateway 以叫用 Lambda 函數

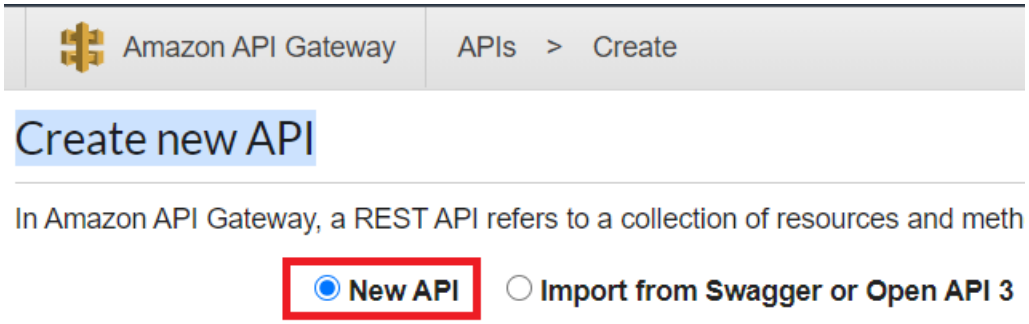
若要建置應用程式：

1. 創建其餘的 API
2. 測試 API Gateway 方法
3. 部署 API Gateway 方法

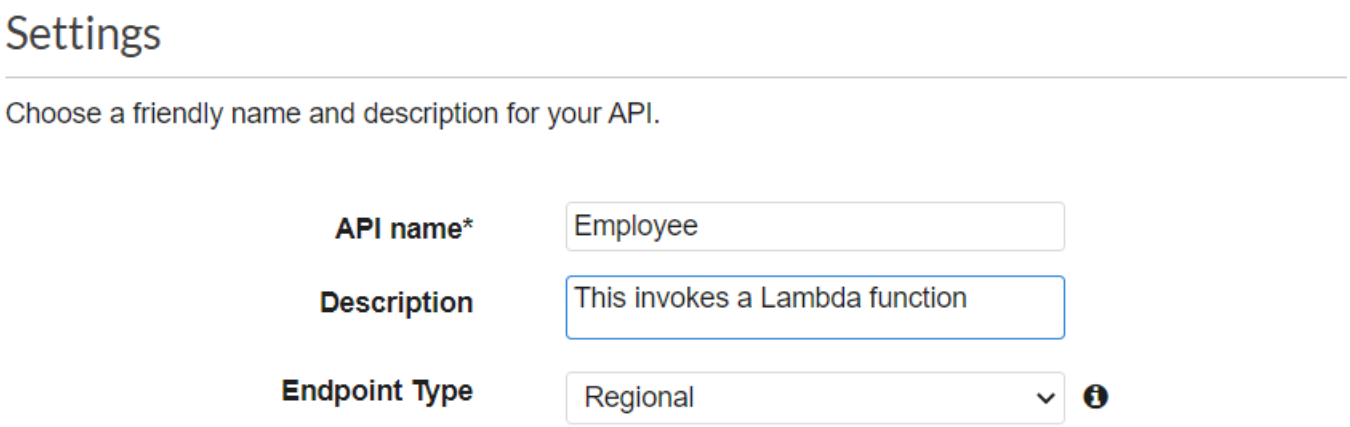
創建其餘的 API

您可以使用 API Gateway 主控台為 Lambda 函數建立休息端點。完成後，您可以使用寧靜的調用調用 Lambda 函數。

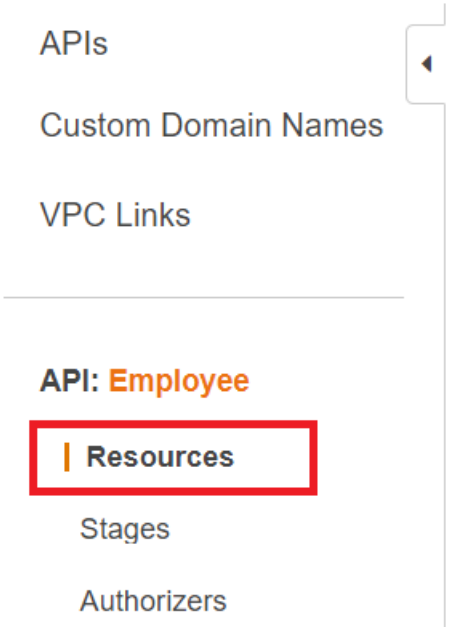
1. 登入 Amazon API Gateway 主控台。
2. 在 [其餘 API] 下，選擇 [建置]。
3. 選取 [新增 API]。



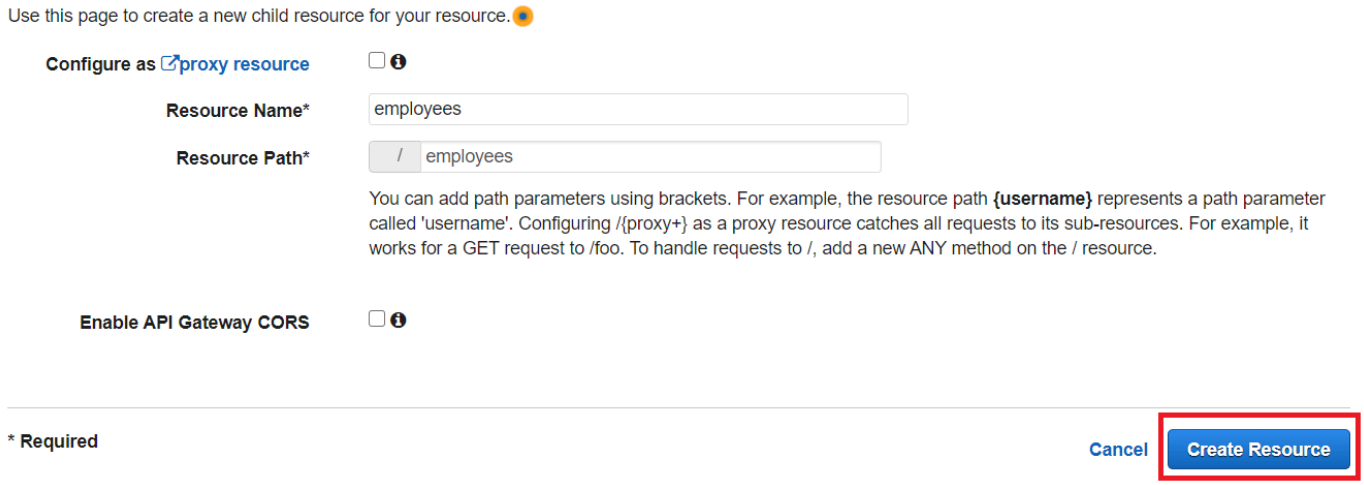
4. 指定員工作為 API 名稱並提供說明。



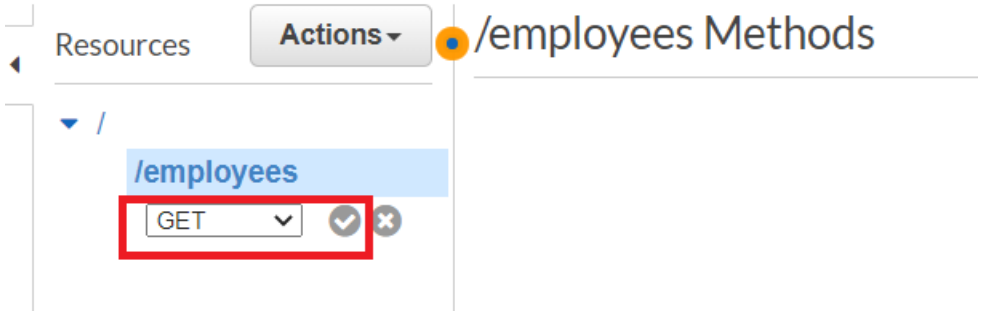
5. 選擇建立 API。
6. 選擇員工部分下的資源。



7. 在名稱欄位中，指定員工。
8. 選擇 Create Resource (建立資源)。
9. 從「動作」 下拉式清單中選擇「建立資源」



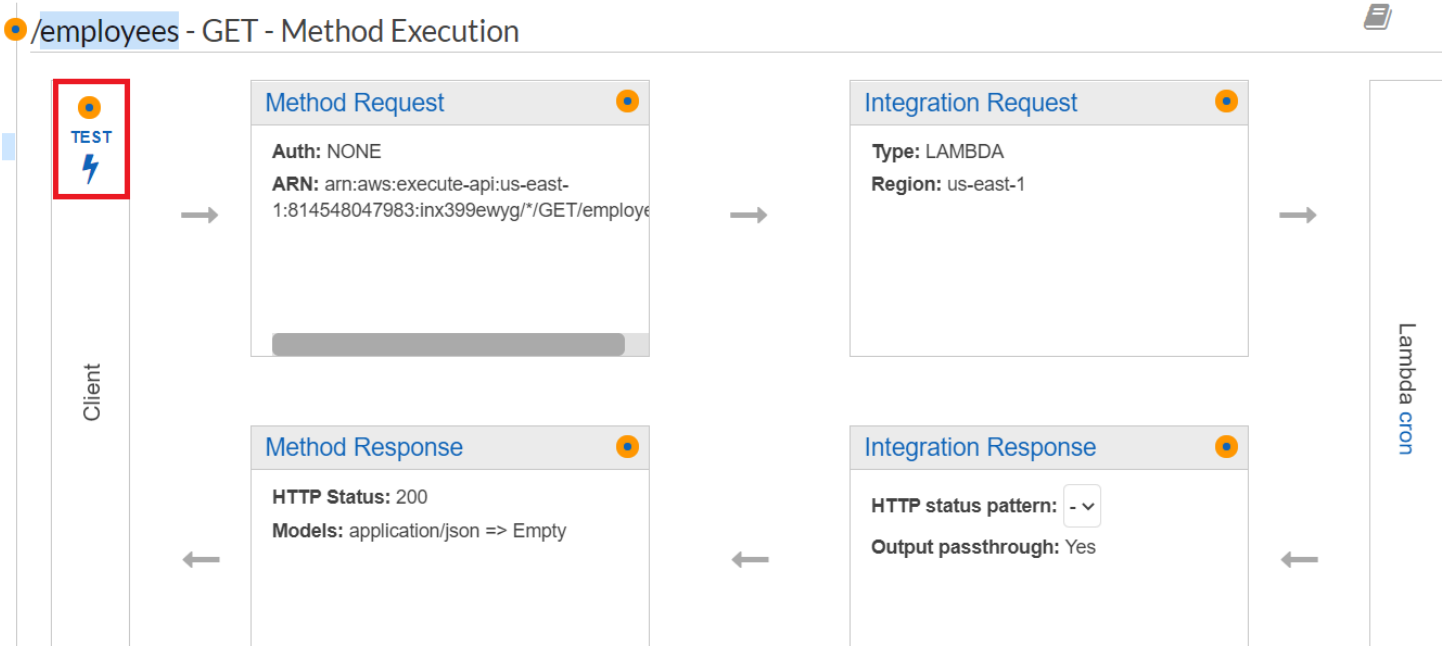
10. 選擇 /employee，從「動作」中選取「建立方法」，然後從/employee 下方的下拉式功能表中選取「GET」。選擇核取記號圖示。



11. 選擇 Lambda 函數，然後輸入 MyLambda 函數作為 Lambda 函數名稱。選擇儲存。

測試 API Gateway 方法

此時在教學課程中，您可以測試叫用 Mylambda 函數 Lambda 函數的 API Gateway 方法。若要測試方法，請選擇「測試」(Test)，如下圖所示。

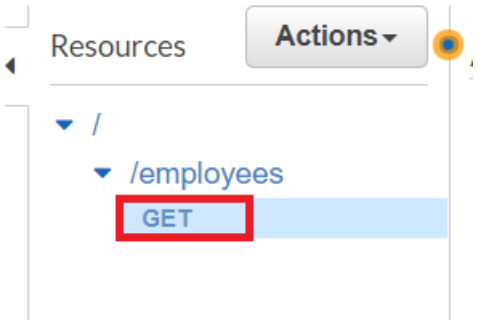


叫用 Lambda 函數之後，您可以檢視記錄檔以查看成功的訊息。

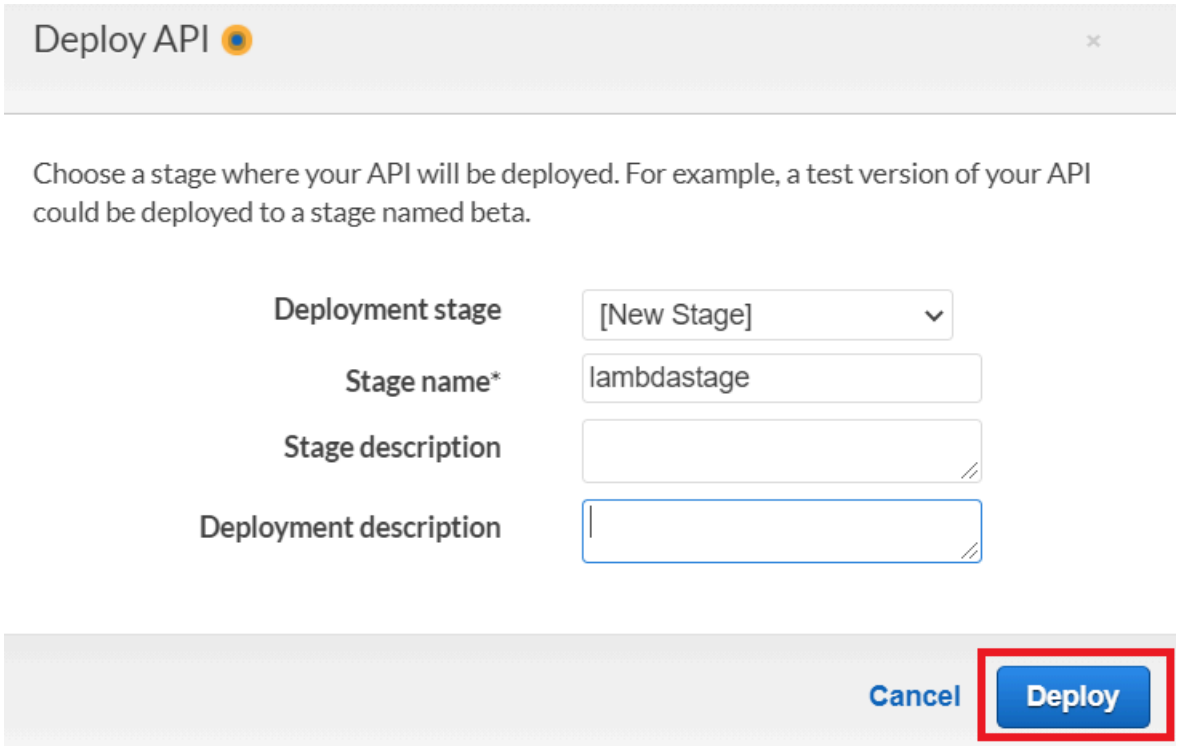
部署 API Gateway 方法

測試成功後，您可以從 Amazon API Gateway 主控台部署該方法。

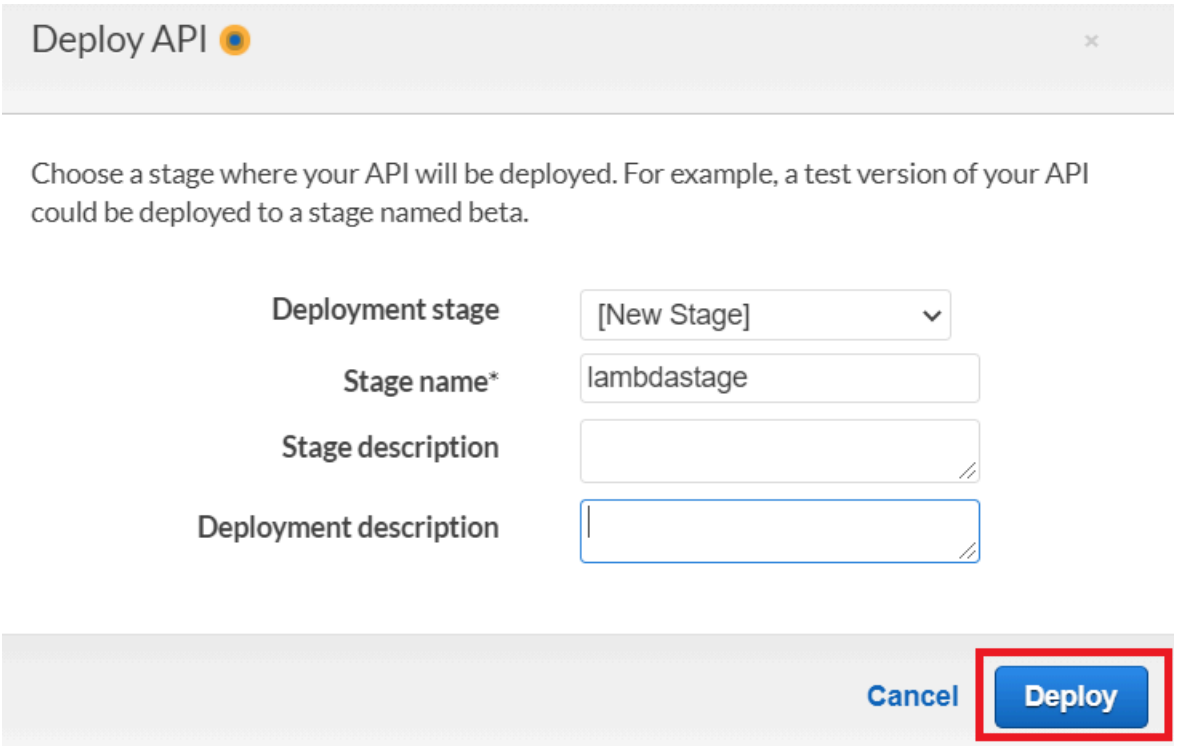
1. 選擇 [取得]。



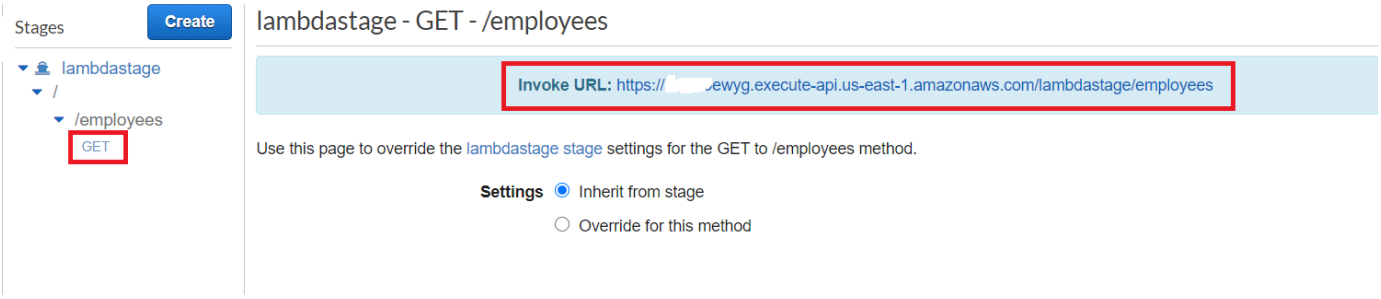
2. 從動作下拉式清單中，選取部署 API。



3. 填寫「部署 API」表單，然後選擇「部署」。



4. 選擇 Save Changes (儲存變更)。
5. 再次選擇 [取得]，請注意 URL 已變更。這是您可以用來叫用 Lambda 函數的叫用網址。



刪除資源

恭喜您！您已經使用 Amazon API Gateway 叫用 Lambda 函數AWS SDK for JavaScript。如本教程開頭所述，請務必終止所有你創建的資源，同時通過本教程，以確保你不收費。您可以透過刪除您在本教學課程建立資AWS源 主題中建立的AWS CloudFormation堆疊來執行此操作，如下所示：

1. 在AWS管理主控台AWS CloudFormation中開啟。
2. 開啟「堆疊」頁面，然後選取堆疊。
3. 選擇 刪除。

建立排程事件以執行AWS Lambda功能

您可以使用 Amazon 事件建立叫用AWS Lambda函數的排程 CloudWatch 事件。您可以將 CloudWatch 事件設定為使用 cron 運算式來排程叫用 Lambda 函數的時間。例如，您可以排程 CloudWatch 事件，以便在每個工作日叫用 Lambda 函數。

AWS Lambda是一種運算服務，可讓您在不佈建或管理伺服器的情況下執行程式碼。您可以使用各種程式設計語言建立 Lambda 函數。如需 AWS Lambda 的詳細資訊，請參閱什麼是 AWS Lambda？。

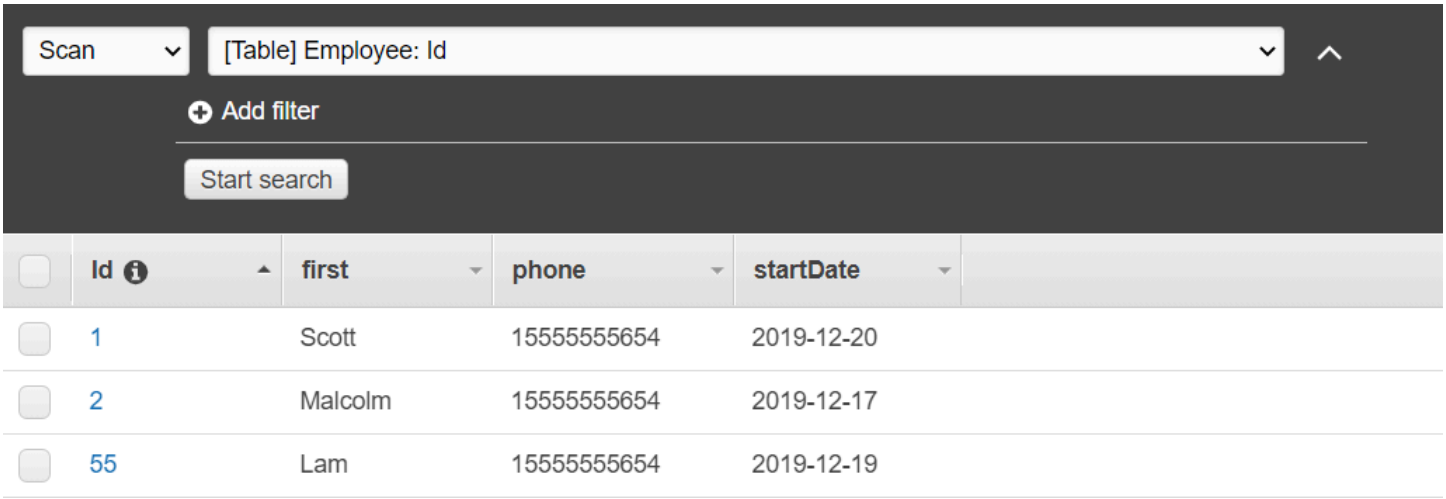
在本教學課程中，您會使用 Lambda JavaScript 執行階段 API 建立 Lambda 函數。這個範例會調用不同的 AWS 服務來執行特定使用案例。例如，假設某個組織傳送行動文字訊息給其員工，並在一年週年日期祝賀他們，如本圖所示。

此教學課程需約 20 分鐘完成。

本教學課程說明如何使用 JavaScript 邏輯來建立執行此使用案例的解決方案。例如，您將學習如何讀取資料庫，以判斷哪些員工已經達到一年週年紀念日、如何處理資料，以及使用 Lambda 函數傳送文字訊息。然後，您將學習如何使用 cron 表達式每個工作日調用 Lambda 函數。

本AWS教學課程使用名為員工的 Amazon DynamoDB 資料表，其中包含這些欄位。

- id-表的主鍵。
- 名字-員工的名字。
- 電話-員工的電話號碼。
- 開始日期-員工的開始日期。



Important

完成成本：本文件中包含的AWS服務包含在AWS免費方案中。但是，請務必在完成此教學課程之後終止所有資源，以確保不會向您收費。

若要建置應用程式：

1. 完成先決條
2. 建立資AWS源
3. 準備瀏覽器腳本
4. 建立並上傳 Lambda 數
5. 部署 Lambda 函數
6. 執行應用程式
7. 刪除資源

必要工作

若要設定和執行此範例，您必須先完成這些任務：

- 設置項目環境以運行這些 Node.js TypeScript 示例，並安裝所需AWS SDK for JavaScript的第三方模塊。按照上的說明進行操作 GitHub。
- 透過使用者登入資料建立共用組態檔。有關提供共用認證檔案的詳細資訊，請參閱 AWSSDK 和工具參考指南中的共用設定和認證檔案。

建立資AWS源

本教學課程需要下列資源。

- 一個名為員工的 Amazon DynamoDB 資料表，其中包含一個名為 ID 的金鑰，以及上圖中顯示的欄位。請務必輸入正確的資料，包括您要測試此使用案例的有效行動電話。如需詳細資訊，請參閱建立資料表。
- 具有附加權限的 IAM 角色，可執行 Lambda 函數。
- 一個 Amazon S3 存儲桶來託管 Lambda 函數。

您可以手動建立這些資源，但我們建議您使用本教學課程中所AWS CloudFormation述來佈建這些資源。

使用建立AWS資源 AWS CloudFormation

AWS CloudFormation 可讓您以可預期和重複的方式建立及佈建 AWS 基礎設施部署。如需 AWS CloudFormation 的相關資訊，請參閱《 使用者指南》AWS CloudFormation。

要使用以下命AWS CloudFormation令創建堆棧AWS CLI：

1. 安裝並設定AWS CLI以下AWS CLI使用者指南中的指示。
2. 在項目文件夾的根目錄`setup.yaml`中創建一個名為的文件，然後將此處的內容複製 GitHub到其中。

> **Note**
>
> AWS CloudFormation範本是使用中AWS CDK提供的來產生的 GitHub。如需關於 AWS CDK 的詳細資訊，請參閱《AWS Cloud Development Kit (AWS CDK) 開發人員指南》。

3. 從命令列執行下列命令，並以*######### STACK_NAME*。

> **Important**
>
> 堆疊名稱在AWS區域和AWS帳戶中必須是唯一的。您最多可以指定 128 個字元，且允許使用數字和連字號。

```
aws cloudformation create-stack --stack-name STACK_NAME --template-body file://
setup.yaml --capabilities CAPABILITY_IAM
```

若要取得有關指`create-stack`令參數的更多資訊，請參閱《指AWS CLI令參考指南》和《AWS CloudFormation使用指南》。

開啟AWS CloudFormation儀表板上的堆疊，然後選擇 [資源] 索引標籤，以檢視主控台中的資源清單。您在本教學課程中需要這些資訊。

4. 建立堆疊時，請使用填AWS SDK for JavaScript入 DynamoDB 表格，如中所述。填入 DynamoDB 料表

填入 DynamoDB 料表

要填充表格，請先創建一個名為的目錄`libs`，並在其中創建一個名為的文件`dynamoClient.js`，然後將下面的內容粘貼到其中。

```
const { DynamoDBClient } = require( "@aws-sdk/client-dynamodb" );
// Set the AWS Region.
const REGION = "REGION"; // e.g. "us-east-1"
// Create an Amazon DynamoDB service client object.
const dynamoClient = new DynamoDBClient({region:REGION});
module.exports = { dynamoClient };
```

此程式碼可在此處取得 GitHub。

接下來，在項目文件夾的根目錄`populate-table.js`中創建一個名為的文件，然後將此處的內容複製 GitHub到其中。對於其中一個項目，請將`phone`屬性的值取代為 E.164 格式的有效行動電話號碼，並取代為今天日期的值。`startDate`

從命令行運行以下命令。

```
node populate-table.js
```

```
const {
BatchWriteItemCommand } = require( "aws-sdk/client-dynamodb" );
const {dynamoClient} = require(  "./libs/dynamoClient" );
// Set the parameters.
```

```
const params = {
  RequestItems: {
    Employees: [
      {
        PutRequest: {
          Item: {
            id: { N: "1" },
            firstName: { S: "Bob" },
            phone: { N: "155555555555654" },
            startDate: { S: "2019-12-20" },
          },
        },
      },
      {
        PutRequest: {
          Item: {
            id: { N: "2" },
            firstName: { S: "Xing" },
            phone: { N: "155555555555653" },
            startDate: { S: "2019-12-17" },
          },
        },
      },
      {
        PutRequest: {
          Item: {
            id: { N: "55" },
            firstName: { S: "Harriette" },
            phone: { N: "155555555555652" },
            startDate: { S: "2019-12-19" },
          },
        },
      },
    ],
  },
};

export const run = async () => {
  try {
    const data = await dbclient.send(new BatchWriteItemCommand(params));
    console.log("Success", data);
  } catch (err) {
    console.log("Error", err);
  }
```

```
};
run();
```

此程式碼可在此處取得 GitHub。

建立 AWS Lambda 函數

設定軟體開發套件

首先匯入必要的 AWS SDK for JavaScript (v3) 模組和命令：以DynamoDBClient及DynamoDBScanCommand，以SNSClient及 Amazon SNS PublishCommand 命令。將「##」取代為「AWS區域」。然後計算今天的日期並將其分配給參數。然後使用您在此範建立資AWS源 例中建立的資料表名稱建立 ScanCommand .Replace *TABLE_NAME* 的參數。

下列程式碼片段說明此步驟。(如需完整範例，請參閱捆綁 Lambda 函數。)

```
"use strict";
// Load the required clients and commands.
const { DynamoDBClient, ScanCommand } = require("@aws-sdk/client-dynamodb");
const { SNSClient, PublishCommand } = require("@aws-sdk/client-sns");

//Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"

// Get today's date.
const today = new Date();
const dd = String(today.getDate()).padStart(2, "0");
const mm = String(today.getMonth() + 1).padStart(2, "0"); //January is 0!
const yyyy = today.getFullYear();
const date = yyyy + "-" + mm + "-" + dd;

// Set the parameters for the ScanCommand method.
const params = {
  // Specify which items in the results are returned.
  FilterExpression: "startDate = :topic",
  // Define the expression attribute value, which are substitutes for the values you
 want to compare.
  ExpressionAttributeValues: {
    ":topic": { S: date },
  },
  // Set the projection expression, which the the attributes that you want.
  ProjectionExpression: "firstName, phone",
  TableName: "TABLE_NAME",
```

```
};
```

掃描 DynamoDB 料表

首先創建一個異步/等待函數調用發布sendText使用 Amazon SNS 的文本消息。PublishCommand然後，新增try區塊模式，針對今天的工作週年紀念日掃描 DynamoDB 表格，然後呼叫sendText函數以傳送文字訊息給這些員工。如果發生錯誤，則會呼叫catch區塊。

下列程式碼片段說明此步驟。(如需完整範例，請參閱捆綁 Lambda 函數。)

```
exports.handler = async (event, context, callback) => {
  // Helper function to send message using Amazon SNS.
  async function sendText(textParams) {
    try {
      const data = await snsclient.send(new PublishCommand(textParams));
      console.log("Message sent");
    } catch (err) {
      console.log("Error, message not sent ", err);
    }
  }
  try {
    // Scan the table to check identify employees with work anniversary today.
    const data = await dbclient.send(new ScanCommand(params));
    data.Items.forEach(function (element, index, array) {
      const textParams = {
        PhoneNumber: element.phone.N,
        Message:
          "Hi " +
          element.firstName.S +
          "; congratulations on your work anniversary!",
      };
      // Send message using Amazon SNS.
      sendText(textParams);
    });
  } catch (err) {
    console.log("Error, could not scan table ", err);
  }
};
```

捆綁 Lambda 函數

本主題說明如何將此範例中的mylambdafunction.js和所需AWS SDK for JavaScript模組捆綁到名為的隨附檔案中index.js。

1. 如果您還沒有，請按照此示例中必要工作的安裝 webpack。

   > **Note**
   >
   > 如需 Webpack 的相關資訊，請參閱捆綁應用程序與網絡包。

2. 在命令列中執行下列命令，將此範例中的 JavaScript 項目捆綁到名為的檔案中<index.js>：

   ```
   webpack mylamdbafunction.js --mode development --target node --devtool false --
   output-library-target umd -o index.js
   ```

   > **Important**
   >
   > 請注意輸出已命名index.js。這是因為 Lambda 函數必須具有index.js處理常式才能運作。

3. 將隨附的輸出檔案壓縮成一個名為的 ZIP 檔案my-lambda-function.zip。index.js

4. 上傳mylambdafunction.zip到您在本教學建立資AWS源 主題中建立的 Amazon S3 儲存貯體。

下面是完整的瀏覽器腳本代碼mylambdafunction.js.

```
"use strict";
// Load the required clients and commands.
const { DynamoDBClient, ScanCommand } = require("@aws-sdk/client-dynamodb");
const { SNSClient, PublishCommand } = require("@aws-sdk/client-sns");

//Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"

// Get today's date.
const today = new Date();
const dd = String(today.getDate()).padStart(2, "0");
const mm = String(today.getMonth() + 1).padStart(2, "0"); //January is 0!
const yyyy = today.getFullYear();
const date = yyyy + "-" + mm + "-" + dd;

// Set the parameters for the ScanCommand method.
const params = {
  // Specify which items in the results are returned.
```

```
  FilterExpression: "startDate = :topic",
  // Define the expression attribute value, which are substitutes for the values you
 want to compare.
  ExpressionAttributeValues: {
    ":topic": { S: date },
  },
  // Set the projection expression, which the the attributes that you want.
  ProjectionExpression: "firstName, phone",
  TableName: "TABLE_NAME",
};

// Create the client service objects.
const dbclient = new DynamoDBClient({ region: REGION });
const snsclient = new SNSClient({ region: REGION });

exports.handler = async (event, context, callback) => {
  // Helper function to send message using Amazon SNS.
  async function sendText(textParams) {
    try {
      const data = await snsclient.send(new PublishCommand(textParams));
      console.log("Message sent");
    } catch (err) {
      console.log("Error, message not sent ", err);
    }
  }
  try {
    // Scan the table to check identify employees with work anniversary today.
    const data = await dbclient.send(new ScanCommand(params));
    data.Items.forEach(function (element, index, array) {
      const textParams = {
        PhoneNumber: element.phone.N,
        Message:
          "Hi " +
          element.firstName.S +
          "; congratulations on your work anniversary!",
      };
      // Send message using Amazon SNS.
      sendText(textParams);
    });
  } catch (err) {
    console.log("Error, could not scan table ", err);
  }
};
```

部署 Lambda 函數。

在項目的根目錄中，創建一個lambda-function-setup.js文件，然後將下面的內容粘貼到其中。

將 *BUCKET_NAME* 取代為您將 Lambda 函數的 ZIP 版本上傳到的 Amazon S3 儲存貯體的名稱。將 *ZIP_FILE_NAME* 取代為您的 Lambda 函數的 ZIP 版本名稱的名稱。將 *IAM_ROLE_ARN* 取代為您在本教學課程主題中建立的 IAM 角色的 Amazon 資源編號 (ARN)。建立資AWS源 將 *Lambda #######* *Lambda ####*稱。

```
// Load the required Lambda client and commands.
const {
   CreateFunctionCommand,
} = require("@aws-sdk/client-lambda");
const {
   lambdaClient
} = require("..libs/lambdaClient.js");

// Instantiate an Lambda client service object.
const lambda = new LambdaClient({ region: REGION });

// Set the parameters.
const params = {
  Code: {
    S3Bucket: "BUCKET_NAME", // BUCKET_NAME
    S3Key: "ZIP_FILE_NAME", // ZIP_FILE_NAME
  },
  FunctionName: "LAMBDA_FUNCTION_NAME",
  Handler: "index.handler",
  Role: "IAM_ROLE_ARN", // IAM_ROLE_ARN; e.g., arn:aws:iam::650138640062:role/v3-
lambda-tutorial-lambda-role
  Runtime: "nodejs12.x",
  Description:
    "Scans a DynamoDB table of employee details and using Amazon Simple Notification
 Services (Amazon SNS) to " +
    "send employees an email the each anniversary of their start-date.",
};

const run = async () => {
  try {
    const data = await lambda.send(new CreateFunctionCommand(params));
    console.log("Success", data); // successful response
  } catch (err) {
    console.log("Error", err); // an error occurred
```

```
  }
};
run();
```

在命令列中輸入以下指令，以部署 Lambda 函數。

```
node lambda-function-setup.js
```

此程式碼範例可在此處取得 GitHub。

設定 CloudWatch 以叫用 Lambda 函數

若要設定 CloudWatch 為叫用 Lambda 函數：

1. 開啟 Lambda 主控台中的 Functions (函數) 頁面。
2. 選擇 Lambda 函數。
3. 在 Designer (設計工具) 下，選擇 Add trigger (新增觸發)。
4. 將觸發類型設定為「CloudWatch 事件/EventBridge」。
5. 針對「規則」，選擇「建立新規則」。
6. 填寫規則名稱和規則說明。
7. 針對規則類型，選取排程運算式。
8. 在「排程運算式」欄位中，輸入 cron 運算式。例如，克朗（0 12？* 星期一至星期五 *）。
9. 選擇新增。

> **Note**
>
> 如需詳細資訊，請參閱搭配 CloudWatch 事件使用 Lambda。

刪除資源

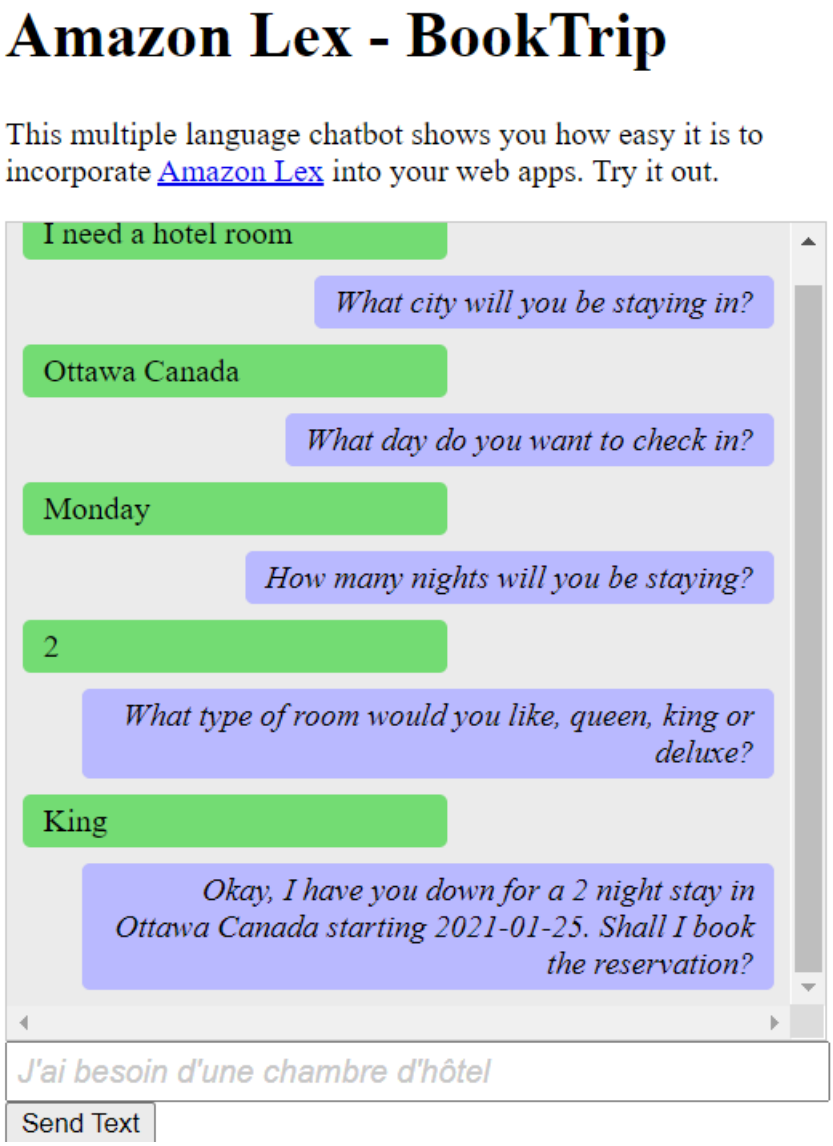
恭喜您！您已透過 Amazon CloudWatch 排定的事件叫用 Lambda 函數使用AWS SDK for JavaScript. 如本教程開頭所述，請務必終止所有你創建的資源，同時通過本教程，以確保你不收費。您可以透過刪除您在本教學課程建立資AWS源 主題中建立的AWS CloudFormation堆疊來執行此操作，如下所示：

1. 開啟 AWS CloudFormation主控台。
2. 在「堆疊」頁面上，選取堆疊。

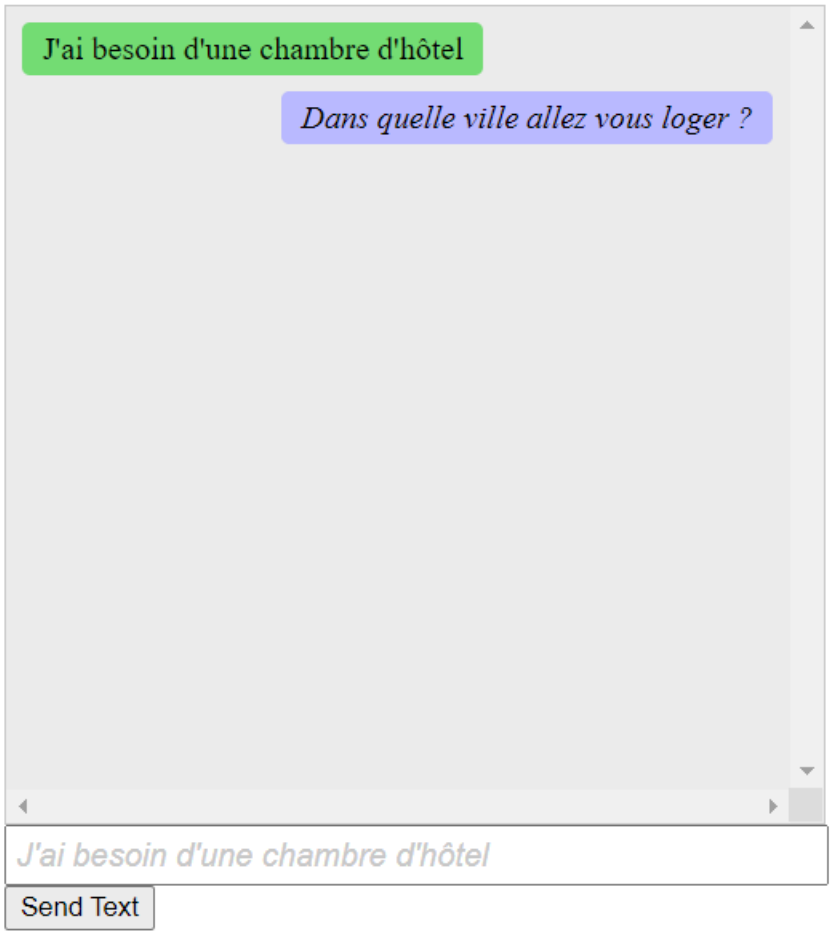
3. 選擇刪除。

建立 Amazon Lex 聊天機器人

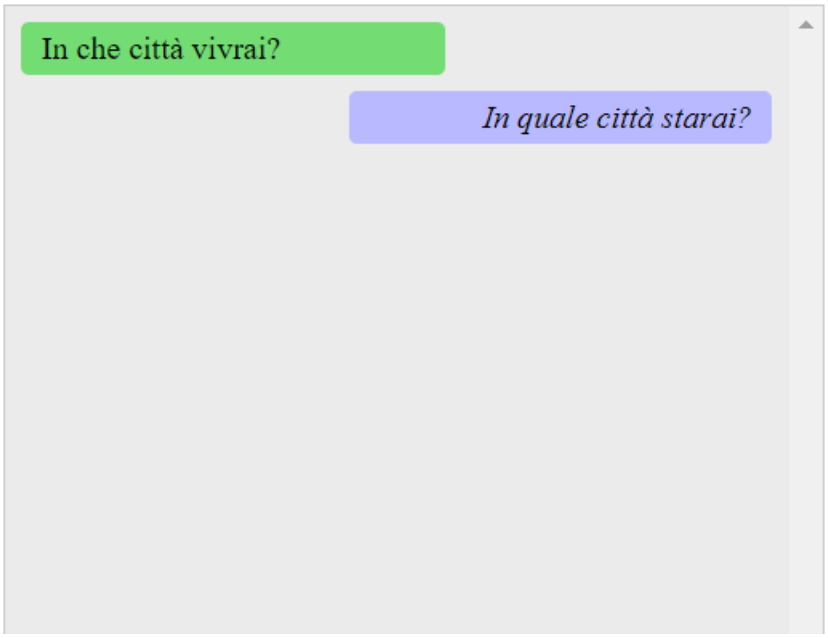
您可以在 Web 應用程式中建立 Amazon Lex 聊天機器人，以吸引您的網站訪客。Amazon Lex 聊天機器人是可與使用者執行線上聊天交談的功能，而無需直接與人員聯絡。例如，下圖顯示了一個 Amazon Lex 聊天機器人，該機器人可以吸引使用者預訂飯店房間。



AWS本教學中建立的 Amazon Lex 聊天機器人能夠處理多種語言。例如，說法文的使用者可以輸入法文文字並取回法文的回覆。



同樣地，使用者也可以使用義大利文與 Amazon Lex 聊天機器人通訊。



本AWS教學課程會引導您建立 Amazon Lex 聊天機器人，並將其整合到 Node.js 網路應用程式中。AWS SDK for JavaScript（v3）用於調用這些AWS服務：

- Amazon Lex
- Amazon Comprehend
- Amazon Translate

完成成本：本文件中包含的AWS服務包含在AWS免費方案中。

附註：請務必在完成本教學課程時終止您建立的所有資源，以確保不會向您收費。

若要建置應用程式：

1. 先決條件
2. 佈建資源
3. 建立 Amazon Lex 聊天機器人
4. 建立 HTML 格式

5. 建立瀏覽器指令碼
6. 後續步驟

必要條件

若要設定和執行此範例，您必須先完成這些任務：

- 設置項目環境以運行這些節點 TypeScript 示例，並安裝所需AWS SDK for JavaScript的第三方模塊。按照上的說明進行操作 GitHub。
- 透過使用者登入資料建立共用組態檔。有關提供共用認證檔案的詳細資訊，請參閱 AWSSDK 和工具參考指南中的共用設定和認證檔案。

> **Important**
>
> 此範例使用電子郵件密碼 6 (ES6)。這需要 Node.js 版本 13.x 或更高版本。要下載並安裝最新版本的 Node.js，請參閱 Node.js 下載。 。
> 但是，如果您更喜歡使用 CommonJS 語法，請參閱JavaScript ES6/共同語法。

建立資AWS源

本教學課程需要下列資源。

- 具有下列權限的未驗證 IAM 角色：
  - Amazon Comprehend
  - Amazon Translate
  - Amazon Lex

您可以手動建立此資源，但我們建議您使用本教學課程中所AWS CloudFormation述來佈建這些資源。

使用建立AWS資源 AWS CloudFormation

AWS CloudFormation 可讓您以可預期和重複的方式建立及佈建 AWS 基礎設施部署。如需 AWS CloudFormation 的相關資訊，請參閱《使用者指南》AWS CloudFormation。

要使用以下命AWS CloudFormation令創建堆棧AWS CLI：

1. 安裝並設定AWS CLI以下AWS CLI使用者指南中的指示。

2. 在項目文件夾的根目錄`setup.yaml`中創建一個名為的文件，然後將此處的內容複製 GitHub到其中。

> **Note**
>
> AWS CloudFormation範本是使用中AWS CDK提供的來產生的 GitHub。如需關於 AWS CDK 的詳細資訊，請參閱《AWS Cloud Development Kit (AWS CDK) 開發人員指南》。

3. 從命令列執行下列命令，並以*`######### STACK_NAME`*。

> **Important**
>
> 堆疊名稱在AWS區域和AWS帳戶中必須是唯一的。您最多可以指定 128 個字元，且允許使用數字和連字號。

```
aws cloudformation create-stack --stack-name STACK_NAME --template-body file://
setup.yaml --capabilities CAPABILITY_IAM
```

若要取得有關指`create-stack`令參數的更多資訊，請參閱《指AWS CLI令參考指南》和《AWS CloudFormation使用指南》。

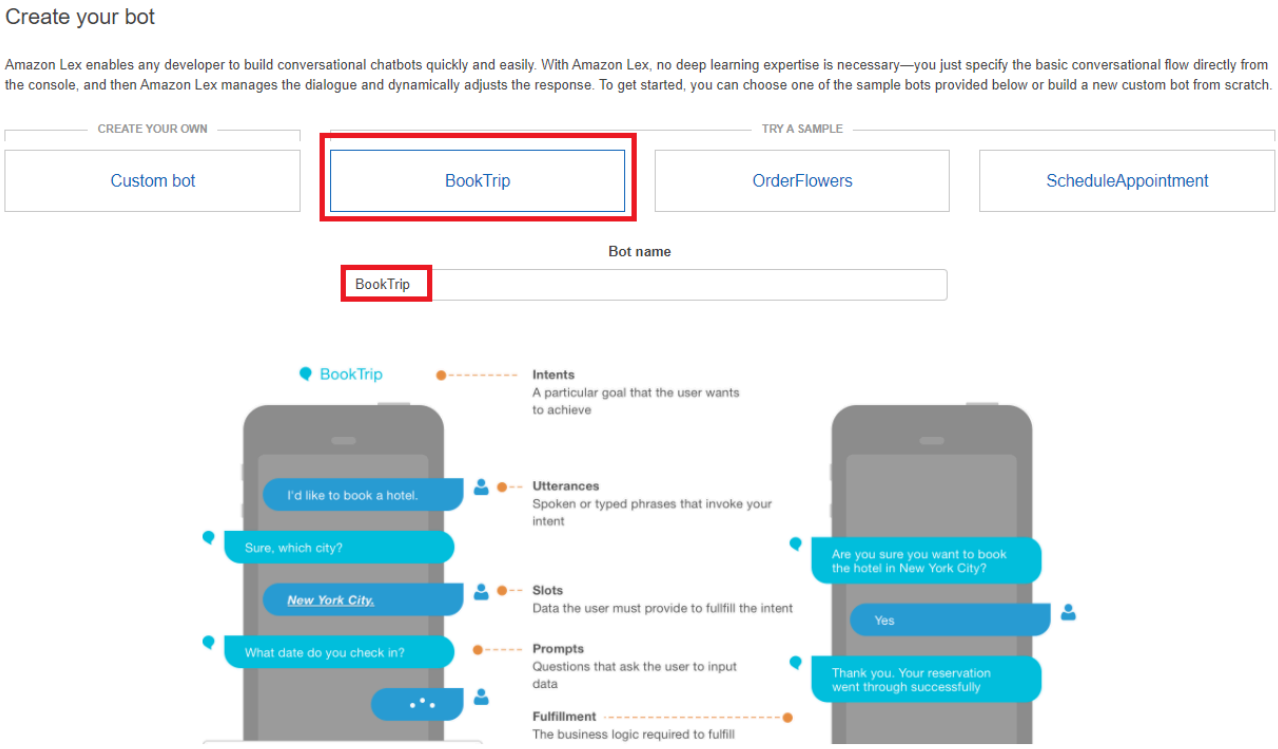
若要檢視建立的資源，請開啟 Amazon Lex 主控台，選擇堆疊，然後選取資源索引標籤。

建立 Amazon Lex 機器人

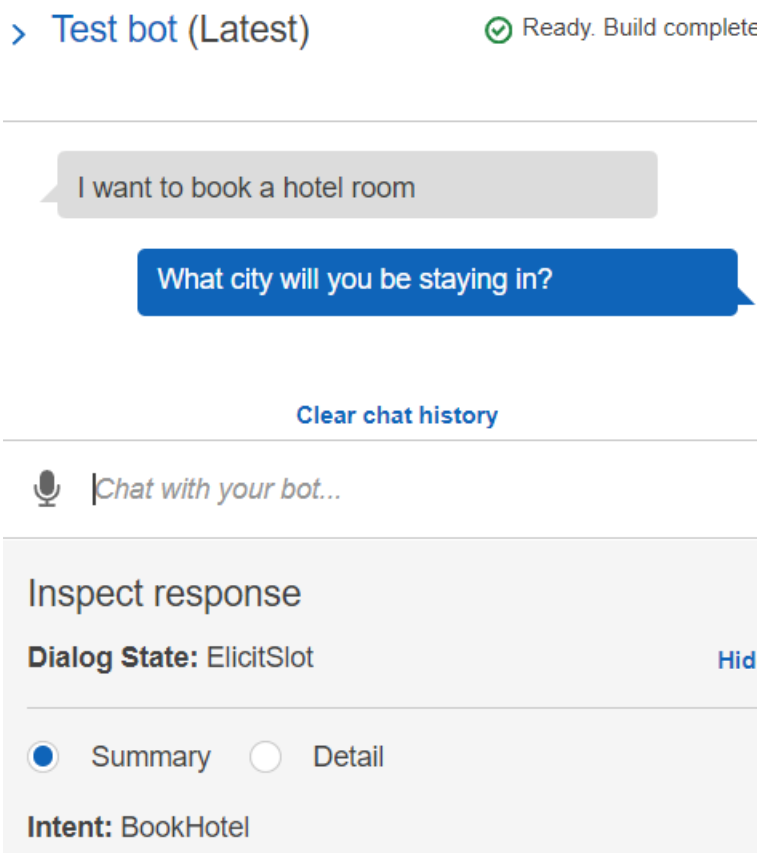
> **Important**
>
> 使用 Amazon Lex 主控台的 V1 建立機器人。這示例不適用於使用 V2 創建的機器人。

第一步是使用 Amazon 網路服務管理主控台建立 Amazon Lex 聊天機器人。在此範例中，使用 Amazon Lex BookTrip範例。如需詳細資訊，請參閱「預訂行程」。

- 登入 Amazon Web Services 管理主控台，然後在亞馬遜網路服務主控台開啟 Amazon Lex 主控台。
- 在 [機器人] 頁面上，選擇 [建立]。
- 選擇BookTrip藍圖 (保留預設機器人名稱 BookTrip)。



- 填寫默認設置，然後選擇創建（控制台顯示BookTrip機器人）。在編輯器索引標籤上，檢閱預先設定的對應方式的詳細資料。
- 在測試視窗中測試機器人。通過鍵入開始測試我想預訂酒店房間。



- 選擇「發佈」並指定別名名稱 (使用時將需要此值AWS SDK for JavaScript)。

> **Note**
>
> 您需要在 JavaScript 程式碼中參考機器人名稱和機器人別名。

建立 HTML 格式

建立名為 `index.html` 的檔案。將下面的代碼複製並粘貼到`index.html`。這個 HTML 引用`main.js`。這是 index.js 的捆綁版本，其中包括所需的AWS SDK for JavaScript模塊。您將在中建立此檔案建立 HTML 格式。 `index.html`還引用`style.css`，它添加了樣式。

```
<!doctype html>
<head>
  <title>Amazon Lex - Sample Application (BookTrip)</title>
  <link type="text/css" rel="stylesheet" href="style.css" />
</head>
```

```
<body>
  <h1 id="title">Amazon Lex - BookTrip</h1>
  <p id="intro">
    This multiple language chatbot shows you how easy it is to incorporate
    <a
      href="https://aws.amazon.com/lex/"
      title="Amazon Lex (product)"
      target="_new"
      >Amazon Lex</a
    >
    into your web apps. Try it out.
  </p>
  <div id="conversation"></div>
  <input
    type="text"
    id="wisdom"
    size="80"
    value=""
    placeholder="J'ai besoin d'une chambre d'hôtel"
  />
  <br />
  <button onclick="createResponse()">Send Text</button>
  <script type="text/javascript" src="./main.js"></script>
</body>
```

此代碼也可以在這裡找到 GitHub。

建立瀏覽器指令碼

建立名為 index.js 的檔案。將下面的代碼複製並粘貼到index.js。匯入所需的AWS SDK for JavaScript模組和指令。為 Amazon Lex，Amazon Comprehend 和 Amazon Translate 創建客戶端。將*##*取代為AWS區域，並以您在中建立的身分集區的識別*#### Identity_POOL_* ID。建立資AWS源 若要擷取此身分集區 ID，請在 Amazon Cognito 主控台中開啟身分集區，選擇編輯身分集區，然後選擇側邊功能表中的範例程式碼。身分集區 ID 在主控台中以紅色文字顯示。

首先，建立一個libs目錄，建立所需的服務用戶端物件，方法是建立三個檔案comprehendClient.jslexClient.js、、和translateClient.js。將下面的適當代碼粘貼到每個文件中，並在每個文件中替換*##*和 *IDENTITY_POOL_* ID。

> **Note**
>
> 使用您在使用建立AWS資源 AWS CloudFormation其中建立的 Amazon Cognito 身分識別集區的識別碼。

```
import { CognitoIdentityClient } from "@aws-sdk/client-cognito-identity";
import { fromCognitoIdentityPool } from "@aws-sdk/credential-provider-cognito-
identity";
import { ComprehendClient } from "@aws-sdk/client-comprehend";

const REGION = "REGION";
const IDENTITY_POOL_ID = "IDENTITY_POOL_ID"; // An Amazon Cognito Identity Pool ID.

// Create an Amazon Comprehend service client object.
const comprehendClient = new ComprehendClient({
  region: REGION,
  credentials: fromCognitoIdentityPool({
    client: new CognitoIdentityClient({ region: REGION }),
    identityPoolId: IDENTITY_POOL_ID,
  }),
});

export { comprehendClient };
```

```
import { CognitoIdentityClient } from "@aws-sdk/client-cognito-identity";
import { fromCognitoIdentityPool } from "@aws-sdk/credential-provider-cognito-
identity";
import { LexRuntimeServiceClient } from "@aws-sdk/client-lex-runtime-service";

const REGION = "REGION";
const IDENTITY_POOL_ID = "IDENTITY_POOL_ID"; // An Amazon Cognito Identity Pool ID.

// Create an Amazon Lex service client object.
const lexClient = new LexRuntimeServiceClient({
  region: REGION,
  credentials: fromCognitoIdentityPool({
    client: new CognitoIdentityClient({ region: REGION }),
    identityPoolId: IDENTITY_POOL_ID,
  }),
});
```

```
export { lexClient };
```

```
import { CognitoIdentityClient } from "@aws-sdk/client-cognito-identity";
import { fromCognitoIdentityPool } from "@aws-sdk/credential-provider-cognito-
identity";
import { TranslateClient } from "@aws-sdk/client-translate";

const REGION = "REGION";
const IDENTITY_POOL_ID = "IDENTITY_POOL_ID"; // An Amazon Cognito Identity Pool ID.

// Create an Amazon Translate service client object.
const translateClient = new TranslateClient({
  region: REGION,
  credentials: fromCognitoIdentityPool({
    client: new CognitoIdentityClient({ region: REGION }),
    identityPoolId: IDENTITY_POOL_ID,
  }),
});

export { translateClient };
```

此程式碼可在此處取得 GitHub。 。

接下來，創建一個index.js文件，並將下面的代碼粘貼到其中。

將 *BOT_ALIAS # BOT_NAME ##*取代為您的 Amazon Lex 機器人的別名和名稱，並以使用者識別碼取代 *USER_* ID。createResponse異步函數執行以下操作：

- 將使用者輸入的文字導入瀏覽器，並使用 Amazon Comprehend 來判斷其語言代碼。
- 採用語言代碼並使用 Amazon Translate 將文本翻譯成英文。
- 取得翻譯後的文字，並使用 Amazon Lex 產生回應。
- 張貼到瀏覽器頁面的響應。

```
import { DetectDominantLanguageCommand } from "@aws-sdk/client-comprehend";
import { TranslateTextCommand } from "@aws-sdk/client-translate";
import { PostTextCommand } from "@aws-sdk/client-lex-runtime-service";
import { lexClient } from "./libs/lexClient.js";
import { translateClient } from "./libs/translateClient.js";
import { comprehendClient } from "./libs/comprehendClient.js";
```

```
var g_text = "";
// Set the focus to the input box.
document.getElementById("wisdom").focus();

function showRequest() {
  var conversationDiv = document.getElementById("conversation");
  var requestPara = document.createElement("P");
  requestPara.className = "userRequest";
  requestPara.appendChild(document.createTextNode(g_text));
  conversationDiv.appendChild(requestPara);
  conversationDiv.scrollTop = conversationDiv.scrollHeight;
}

function showResponse(lexResponse) {
  var conversationDiv = document.getElementById("conversation");
  var responsePara = document.createElement("P");
  responsePara.className = "lexResponse";

  var lexTextResponse = lexResponse;

  responsePara.appendChild(document.createTextNode(lexTextResponse));
  responsePara.appendChild(document.createElement("br"));
  conversationDiv.appendChild(responsePara);
  conversationDiv.scrollTop = conversationDiv.scrollHeight;
}

function handletext(text) {
  g_text = text;
  var xhr = new XMLHttpRequest();
  xhr.addEventListener("load", loadNewItems, false);
  xhr.open("POST", "../text", true); // A Spring MVC controller
  xhr.setRequestHeader("Content-type", "application/x-www-form-urlencoded"); //
necessary
  xhr.send("text=" + text);
}

function loadNewItems() {
  showRequest();

  // Re-enable input.
  var wisdomText = document.getElementById("wisdom");
  wisdomText.value = "";
  wisdomText.locked = false;
}
```

```
// Respond to user's input.
const createResponse = async () => {
  // Confirm there is text to submit.
  var wisdomText = document.getElementById("wisdom");
  if (wisdomText && wisdomText.value && wisdomText.value.trim().length > 0) {
    // Disable input to show it is being sent.
    var wisdom = wisdomText.value.trim();
    wisdomText.value = "...";
    wisdomText.locked = true;
    handletext(wisdom);

    const comprehendParams = {
      Text: wisdom,
    };
    try {
      const data = await comprehendClient.send(
        new DetectDominantLanguageCommand(comprehendParams)
      );
      console.log(
        "Success. The language code is: ",
        data.Languages[0].LanguageCode
      );
      const translateParams = {
        SourceLanguageCode: data.Languages[0].LanguageCode,
        TargetLanguageCode: "en", // For example, "en" for English.
        Text: wisdom,
      };
      try {
        const data = await translateClient.send(
          new TranslateTextCommand(translateParams)
        );
        console.log("Success. Translated text: ", data.TranslatedText);
        const lexParams = {
          botName: "BookTrip",
          botAlias: "mynewalias",
          inputText: data.TranslatedText,
          userId: "chatbot", // For example, 'chatbot-demo'.
        };
        try {
          const data = await lexClient.send(new PostTextCommand(lexParams));
          console.log("Success. Response is: ", data.message);
          var msg = data.message;
          showResponse(msg);
```

```
        } catch (err) {
          console.log("Error responding to message. ", err);
        }
      } catch (err) {
        console.log("Error translating text. ", err);
      }
    } catch (err) {
      console.log("Error identifying language. ", err);
    }
  }
};
// Make the function available to the browser.
window.createResponse = createResponse;
```

此程式碼可在此處取得 GitHub。 。

現在使用 webpack 將`index.js`和AWS SDK for JavaScript模塊捆綁到一個文件中`main.js`。

1. 如果您還沒有，請按照此示例中必要條件的安裝 webpack。

   > **Note**
   >
   > 如需 Webpack 的相關資訊，請參閱捆綁應用程序與網絡包。

2. 在命令列中執行下列命令，將此範例中的 JavaScript 項目捆綁到名為的檔案中`main.js`：

   ```
   webpack index.js --mode development --target web --devtool false -o main.js
   ```

後續步驟

恭喜您！您已建立 Node.js 應用程式，該應用程式使用 Amazon Lex 建立互動式使用者體驗。如本教程開頭所述，請務必終止所有你創建的資源，同時通過本教程，以確保你不收費。您可以透過刪除您在本教學課程建立資AWS源 主題中建立的AWS CloudFormation堆疊來執行此操作，如下所示：

1. 開啟 AWS CloudFormation主控台。
2. 在「堆疊」頁面上，選取堆疊。
3. 選擇刪除。

如需更多AWS跨服務範例，請參閱AWS SDK for JavaScript跨服務範例。

建立範例訊息應用程式

您可以使AWS用和 Amazon Simple Queue Service (Amazon SQS) 建立傳送AWS SDK for JavaScript和擷取訊息的應用程式。訊息會儲存在先進先出 (FIFO) 佇列中，以確保訊息的順序一致。例如，儲存在佇列中的第一個訊息是從佇列讀取的第一封郵件。

> **Note**
>
> 如需 Amazon SQS 的詳細資訊，請參閱什麼是 Amazon 簡單佇列服務？

在本教學課程中，您會建立名為「AWS訊息」的 Node.js 應用程式。

完成成本：本文件中包含的AWS服務包含在AWS免費方案中。

附註：請務必在完成本教學課程時終止您建立的所有資源，以確保不會向您收費。

若要建置應用程式：

1. 先決條件
2. 佈建資源
3. 了解工作流程
4. 建立 HTML 格式
5. 建立瀏覽器指令碼
6. 後續步驟

必要條件

若要設定和執行此範例，您必須先完成這些任務：

- 設置項目環境以運行這些節點 TypeScript 示例，並安裝所需AWS SDK for JavaScript的第三方模塊。按照上的說明進行操作 GitHub。
- 透過使用者登入資料建立共用組態檔。有關提供共用認證檔案的詳細資訊，請參閱 AWSSDK 和工具參考指南中的共用設定和認證檔案。

> **Important**
>
> 此範例使用電子郵件密碼 6 (ES6)。這需要 Node.js 版本 13.x 或更高版本。要下載並安裝最新版本的 Node.js，請參閱 Node.js 下載。 。
> 但是，如果您更喜歡使用 CommonJS 語法，請參閱JavaScript ES6/共同語法。

建立資AWS源

本教學課程需要下列資源。

- 具有 Amazon SQS 許可的未經驗證身分與存取權管理角色。
- 名為訊息的 FIFO Amazon SQS 佇列。FIFO-如需建立佇列的相關資訊，請參閱建立 Amazon SQS 佇列。

您可以手動建立此資源，但建議您使用 AWS CloudFormation (AWS CloudFormation) 佈建這些資源，如本教學課程所述。

> **Note**
>
> 這AWS CloudFormation是一個軟體開發架構，可讓您定義雲端應用程式資源。如需詳細資訊，請參閱《使用者指南》AWS CloudFormation。

使用建立AWS資源 AWS CloudFormation

AWS CloudFormation 可讓您以可預期和重複的方式建立及佈建 AWS 基礎設施部署。如需 AWS CloudFormation 的相關資訊，請參閱《使用者指南》AWS CloudFormation。

要使用以下命AWS CloudFormation令創建堆棧AWS CLI：

1. 安裝並設定AWS CLI以下AWS CLI使用者指南中的指示。
2. 在項目文件夾的根目錄`setup.yaml`中創建一個名為的文件，然後將此處的內容複製 GitHub到其中。

> **Note**
>
> AWS CloudFormation範本是使用中AWS CDK提供的來產生的 GitHub。如需關於 AWS CDK 的詳細資訊，請參閱《AWS Cloud Development Kit (AWS CDK) 開發人員指南》。

3. 從命令列執行下列命令，並以*######### STACK_NAME*。

> **Important**
>
> 堆疊名稱在AWS區域和AWS帳戶中必須是唯一的。您最多可以指定 128 個字元，且允許使用數字和連字號。

```
aws cloudformation create-stack --stack-name STACK_NAME --template-body file://
setup.yaml --capabilities CAPABILITY_IAM
```

若要取得有關指`create-stack`令參數的更多資訊，請參閱《指AWS CLI令參考指南》和《AWS CloudFormation使用指南》。

若要檢視建立的資源，請AWS CloudFormation在AWS管理主控台中開啟，選擇堆疊，然後選取 [資源] 索引標籤。

瞭解AWS訊息應用程式

若要將訊息傳送至 SQS 佇列，請將訊息輸入應用程式，然後選擇傳送。

傳送訊息後，應用程式會顯示訊息。

您可以選擇整個清除，以清除 Amazon SQS 佇列中的訊息。這會導致佇列空白，應用程式中不會顯示任何訊息。

以下說明應用程式如何處理訊息：

- 用戶選擇他們的名稱並輸入他們的消息，並提交啟動函數的消息。`pushMessage`
- `pushMessage`擷取 Amazon SQS 佇列網址，然後傳送訊息文字具有唯一訊息 ID 值 (GUID) 的訊息，並將使用者傳送至 Amazon SQS 佇列。
- `pushMessage`從 Amazon SQS 佇列擷取訊息、擷取每則訊息的使用者和訊息，然後顯示訊息。

- 使用者可以清除訊息，從 Amazon SQS 佇列和使用者介面刪除訊息。

創建 HTML 頁面

現在，您可以建立應用程式圖形化使用者介面 (GUI) 所需的 HTML 檔案。建立名為 `index.html` 的檔案。將下面的代碼複製並粘貼到`index.html`。這個 HTML 引用`main.js`。這是 index.js 的捆綁版本，其中包括所需的AWS SDK for JavaScript模塊。

```
<!doctype html>
<html
  xmlns:th="http://www.thymeleaf.org"
  xmlns:sec="http://www.thymeleaf.org/thymeleaf-extras-springsecurity3"
>
  <head>
    <meta charset="utf-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <link rel="icon" href="./images/favicon.ico" />
    <link
      rel="stylesheet"
      href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
    />
    <link rel="stylesheet" href="./css/styles.css" />
    <script src="https://code.jquery.com/jquery-1.12.4.min.js"></script>
    <script src="https://code.jquery.com/ui/1.11.4/jquery-ui.min.js"></script>
    <script src="./js/main.js"></script>
    <style>
      .messageelement {
        margin: auto;
        border: 2px solid #dedede;
        background-color: #d7d1d0;
        border-radius: 5px;
        max-width: 800px;
        padding: 10px;
        margin: 10px 0;
      }

      .messageelement::after {
        content: "";
        clear: both;
        display: table;
      }
```

```
      .messageelement img {
        float: left;
        max-width: 60px;
        width: 100%;
        margin-right: 20px;
        border-radius: 50%;
      }

      .messageelement img.right {
        float: right;
        margin-left: 20px;
        margin-right: 0;
      }
    </style>
  </head>
  <body>
    <div class="container">
      <h2>AWS Sample Messaging Application</h2>
      <div id="messages"></div>

      <div class="input-group mb-3">
        <div class="input-group-prepend">
          <span class="input-group-text" id="basic-addon1">Sender:</span>
        </div>
        <select name="cars" id="username">
          <option value="Scott">Brian</option>
          <option value="Tricia">Tricia</option>
        </select>
      </div>

      <div class="input-group">
        <div class="input-group-prepend">
          <span class="input-group-text">Message:</span>
        </div>
        <textarea
          class="form-control"
          id="textarea"
          aria-label="With textarea"
        ></textarea>
        <button
          type="button"
          onclick="pushMessage()"
          id="send"
```

```
          class="btn btn-success"
        >
          Send
        </button>
        <button
          type="button"
          onclick="purge()"
          id="refresh"
          class="btn btn-success"
        >
          Purge
        </button>
      </div>
      <!-- All of these child items are hidden and only displayed in a FancyBox
 ------------------------------------------------------>
      <div id="hide" style="display: none">
        <div id="base" class="messageelement">
          <img
            src="../public/images/av2.png"
            alt="Avatar"
            class="right"
            style="width: 100%"
          />
          <p id="text">Excellent! So, what do you want to do today?</p>
          <span class="time-right">11:02</span>
        </div>
      </div>
    </div>
  </body>
</html>
```

此代碼也可以在這裡找到 GitHub。

建立瀏覽器指令碼

在本主題中，您會建立應用程式的瀏覽器指令碼。建立瀏覽器指令碼後，您可以將其捆綁到名為的檔案中，main.js如中所述捆綁 JavaScript。

建立名為 index.js 的檔案。將代碼從這裡複製並粘貼 GitHub到其中。

此代碼將在以下各節中進行說明：

1. 組態

2. 民意科技
3. 推送訊息
4. 清除

組態

首先，建立一個libs目錄，建立名為sqsClient.js的檔案，建立所需的 Amazon SQS 用戶端物件。替換每個## _POOL_ID。

> **Note**
>
> 使用您在建立資AWS源 其中建立的 Amazon Cognito 身分識別集區的識別碼。

```
import { CognitoIdentityClient } from "@aws-sdk/client-cognito-identity";
import { fromCognitoIdentityPool } from "@aws-sdk/credential-providers";
import {SQSClient} from "@aws-sdk/client-sqs";
const REGION = "REGION"; //e.g. "us-east-1"
const IdentityPoolId = "IDENTITY_POOL_ID";
const sqsClient = new SQSClient({
  region: REGION,
  credentials: fromCognitoIdentityPool({
    client: new CognitoIdentityClient({ region: REGION }),
    identityPoolId: IdentityPoolId
  }),
});
```

在中index.js，匯入必要的AWS SDK for JavaScript模組和指令。以您在中### Amazon SQS ###稱取代 SQS 佇列的名稱。建立資AWS源

```
import {
  GetQueueUrlCommand,
  SendMessageCommand,
  ReceiveMessageCommand,
  PurgeQueueCommand,
} from "@aws-sdk/client-sqs";
import { sqsClient } from "./libs/sqsClient.js";
```

```
const QueueName = "SQS_QUEUE_NAME"; // The Amazon SQS queue name, which must end
 in .fifo for this example.
```

民意科技

populateChat函數 onload 會自動擷取 Amazon SQS 佇列的 URL，並擷取佇列中的所有訊息，並顯示這些訊息。

```
$(function () {
  populateChat();
});

const populateChat = async () => {
  try {
    // Set the Amazon SQS Queue parameters.
    const queueParams = {
      QueueName: QueueName,
      Attributes: {
        DelaySeconds: "60",
        MessageRetentionPeriod: "86400",
      },
    };
    // Get the Amazon SQS Queue URL.
    const data = await sqsClient.send(new GetQueueUrlCommand(queueParams));
    console.log("Success. The URL of the SQS Queue is: ", data.QueueUrl);
    // Set the parameters for retrieving the messages in the Amazon SQS Queue.
    var getMessageParams = {
      QueueUrl: data.QueueUrl,
      MaxNumberOfMessages: 10,
      MessageAttributeNames: ["All"],
      VisibilityTimeout: 20,
      WaitTimeSeconds: 20,
    };
    try {
      // Retrieve the messages from the Amazon SQS Queue.
      const data = await sqsClient.send(
        new ReceiveMessageCommand(getMessageParams)
      );
      console.log("Successfully retrieved messages", data.Messages);

      // Loop through messages for user and message body.
      var i;
```

```
      for (i = 0; i < data.Messages.length; i++) {
        const name = data.Messages[i].MessageAttributes.Name.StringValue;
        const body = data.Messages[i].Body;
        // Create the HTML for the message.
        var userText = body + "<br><br><b>" + name;
        var myTextNode = $("#base").clone();
        myTextNode.text(userText);
        var image_url;
        var n = name.localeCompare("Scott");
        if (n == 0) image_url = "./images/av1.png";
        else image_url = "./images/av2.png";
        var images_div =
          '<img src="' +
          image_url +
          '" alt="Avatar" class="right" style=""width:100%;"">';
        myTextNode.html(userText);
        myTextNode.append(images_div);

        // Add the message to the GUI.
        $("#messages").append(myTextNode);
      }
    } catch (err) {
      console.log("Error loading messages: ", err);
    }
  } catch (err) {
    console.log("Error retrieving SQS queue URL: ", err);
  }
};
```

推送訊息

用戶選擇他們的名稱並輸入他們的消息，並提交啟動函數的消息。pushMessage pushMessage擷取 Amazon SQS 佇列網址，然後傳送訊息文字具有唯一訊息 ID 值 (GUID) 的訊息，並將使用者傳送至 Amazon SQS 佇列。然後，它會從 Amazon SQS 佇列擷取所有訊息並加以顯示。

```
const pushMessage = async () => {
  // Get and convert user and message input.
  var user = $("#username").val();
  var message = $("#textarea").val();

  // Create random deduplication ID.
  var dt = new Date().getTime();
  var uuid = "xxxxxxxx-xxxx-4xxx-yxxx-xxxxxxxxxxxx".replace(/[xy]/g, function (
```

```
    c
  ) {
    var r = (dt + Math.random() * 16) % 16 | 0;
    dt = Math.floor(dt / 16);
    return (c == "x" ? r : (r & 0x3) | 0x8).toString(16);
  });

  try {
    // Set the Amazon SQS Queue parameters.
    const queueParams = {
      QueueName: QueueName,
      Attributes: {
        DelaySeconds: "60",
        MessageRetentionPeriod: "86400",
      },
    };
    const data = await sqsClient.send(new GetQueueUrlCommand(queueParams));
    console.log("Success. The URL of the SQS Queue is: ", data.QueueUrl);
    // Set the parameters for the message.
    var messageParams = {
      MessageAttributes: {
        Name: {
          DataType: "String",
          StringValue: user,
        },
      },
      MessageBody: message,
      MessageDeduplicationId: uuid,
      MessageGroupId: "GroupA",
      QueueUrl: data.QueueUrl,
    };
    const result = await sqsClient.send(new SendMessageCommand(messageParams));
    console.log("Success", result.MessageId);

    // Set the parameters for retrieving all messages in the SQS queue.
    var getMessageParams = {
      QueueUrl: data.QueueUrl,
      MaxNumberOfMessages: 10,
      MessageAttributeNames: ["All"],
      VisibilityTimeout: 20,
      WaitTimeSeconds: 20,
    };

    // Retrieve messages from SQS Queue.
```

```
    const final = await sqsClient.send(
      new ReceiveMessageCommand(getMessageParams)
    );
    console.log("Successfully retrieved", final.Messages);
    $("#messages").empty();
    // Loop through messages for user and message body.
    var i;
    for (i = 0; i < final.Messages.length; i++) {
      const name = final.Messages[i].MessageAttributes.Name.StringValue;
      const body = final.Messages[i].Body;
      // Create the HTML for the message.
      var userText = body + "<br><br><b>" + name;
      var myTextNode = $("#base").clone();
      myTextNode.text(userText);
      var image_url;
      var n = name.localeCompare("Scott");
      if (n == 0) image_url = "./images/av1.png";
      else image_url = "./images/av2.png";
      var images_div =
        '<img src="' +
        image_url +
        '" alt="Avatar" class="right" style=""width:100%;"">';
      myTextNode.html(userText);
      myTextNode.append(images_div);
      // Add the HTML to the GUI.
      $("#messages").append(myTextNode);
    }
  } catch (err) {
    console.log("Error", err);
  }
};
// Make the function available to the browser window.
window.pushMessage = pushMessage;
```

清除訊息

purge從 Amazon SQS 佇列和使用者介面刪除訊息。

```
// Delete the message from the Amazon SQS queue.
const purge = async () => {
  try {
    // Set the Amazon SQS Queue parameters.
    const queueParams = {
      QueueName: QueueName,
```

```
      Attributes: {
        DelaySeconds: "60",
        MessageRetentionPeriod: "86400",
      },
    };
    // Get the Amazon SQS Queue URL.
    const data = await sqsClient.send(new GetQueueUrlCommand(queueParams));
    console.log("Success", data.QueueUrl);
    // Delete all the messages in the Amazon SQS Queue.
    const result = await sqsClient.send(
      new PurgeQueueCommand({ QueueUrl: data.QueueUrl })
    );
    // Delete all the messages from the GUI.
    $("#messages").empty();
    console.log("Success. All messages deleted.", data);
  } catch (err) {
    console.log("Error", err);
  }
};

// Make the function available to the browser window.
window.purge = purge;
```

捆綁 JavaScript

這個 comlete 瀏覽器腳本代碼可在這裡找到。 GitHub 。

現在使用 webpack 將`index.js`和AWS SDK for JavaScript模塊捆綁到一個文件中`main.js`。

1. 如果您還沒有，請按照此示例中必要條件的安裝 webpack。

   > **Note**
   >
   > 如需 Webpack 的相關資訊，請參閱捆綁應用程序與網絡包。

2. 在命令列中執行下列命令，將此範例的 JavaScript 內容捆綁到名為的檔案中`<index.js>`：

   ```
   webpack index.js --mode development --target web --devtool false -o main.js
   ```

後續步驟

恭喜您！您已建立並部署使用 Amazon SQS 的AWS簡訊應用程式。正如本教程開頭所述，請務必終止所有您創建的資源，同時通過本教程，以確保你不再收取他們的費用。您可以透過刪除您在本教學課程建立資AWS源 主題中建立的AWS CloudFormation堆疊來執行此操作，如下所示：

1. 在AWS管理主控台AWS CloudFormation中開啟。
2. 開啟「堆疊」頁面，然後選取堆疊。
3. 選擇刪除。

搭 AWS Cloud9 配使用 AWS SDK for JavaScript

您可以使 AWS Cloud9 用在 AWS SDK for JavaScript 瀏覽器中編寫和執行您 JavaScript 的程式碼，以及撰寫、執行和偵錯 Node.js 程式碼，只需使用瀏覽器即可。 AWS Cloud9 包含程式碼編輯器和終端機等工具，以及 Node.js 程式碼的除錯工具。

由於 AWS Cloud9 IDE 是以雲端為基礎，因此您可以使用連接網際網路的機器，在辦公室、家中或任何地方處理專案。有關的一般資訊 AWS Cloud9，請參閱《AWS Cloud9 使用者指南》。

下列步驟說明如何 AWS Cloud9 使用的 SDK 進行設定 JavaScript。

內容

步驟 1：設定要使用的 AWS 帳戶 AWS Cloud9

開始使 AWS Cloud9 用 AWS Cloud9 主控台以具有您 AWS 帳戶存取權限的 (例如 IAM 使用者) 身分登 AWS Cloud9 入主控台。 AWS Identity and Access Management

若要在您的 AWS 帳戶中設定 IAM 實體以存取 AWS Cloud9並登入 AWS Cloud9 主控台，請參閱AWS Cloud9 使用者指南 AWS Cloud9中的 Team 設定。

步驟 2：設定您的 AWS Cloud9 開發環境

登入 AWS Cloud9 主控台後，請使用主控台建立 AWS Cloud9 開發環境。建立環境之後，會 AWS Cloud9 開啟該環境的 IDE。

如需詳細資訊，請參閱《AWS Cloud9 使用指南》 AWS Cloud9中的〈建立環境〉。

> **Note**
>
> 第一次由主控台建立您的環境時，建議您選擇 Create a new instance for environment (EC2) (為環境建立新的執行個體) 選項。此選項指示 AWS Cloud9 建立環境、啟動 Amazon EC2 執行個體，然後將新執行個體連接到新環境。這是開始使用的最快方法 AWS Cloud9。

步驟 3：設定下列項目的 SDK JavaScript

為您的 AWS Cloud9 開發環境開啟 IDE 之後，請遵循下列其中一個或兩個程序，使用 IDE JavaScript 在您的環境中設定 SDK。

若要為 Node.js 設定適用的開 JavaScript 發套件

1. 如果 IDE 中尚未開啟終端機，請開啟終端機。若要執行此操作，請在 IDE 的選單列中選擇 Window, New Terminal (視窗、新增終端機)。
2. 執行下列命令以用npm來安裝 SDK 的用Cloud9戶端 JavaScript。

```
npm install @aws-sdk/client-cloud9
```

如果 IDE 找不到npm，請依照下列順序一次執行下列命令以進行安裝npm。(這些命令會假設您已依照本主題先前所述，選擇 Create a new instance for environment (EC2) (為環境建立新的執行個體 (EC2))。)

> **Warning**
>
> AWS 不控制下列程式碼。在您執行前，請務必驗證其真確性及完整性。有關此代碼的更多信息可以在 nvm（節點版本管理器）GitHub 存儲庫中找到。

```
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.34.0/install.sh | bash #
 Download and install Node Version Manager (nvm).
. ~/.bashrc                                                                    #
 Activate nvm.
nvm install node                                                               #
 Use nvm to install npm (and Node.js at the same time).
```

若要在瀏覽器 JavaScript 中設定 SDK

若要在 HTML 頁面 JavaScript 中使用 SDK，請使用將必要的用戶端模組和所有必要 JavaScript 功能捆綁到單一 JavaScript 檔案中，並將其新增 WebPack 至 HTML 頁面的<head>指令碼標記中。例如：

```
<script src=./main.js></script>
```

> **Note**
>
> 如需 Webpack 的詳細資訊，請參閱 捆綁應用程序與網絡包

步驟 4：下載範例程式碼

使用您在上一個步驟中開啟的終端機，將 SDK 的範例程式碼下載 JavaScript 到開 AWS Cloud9 發環境中。(如果 IDE 尚未開啟終端機，請在 IDE 的選單列中選擇 Window, New Terminal (視窗、新增終端機) 以開啟。)

執行下列命令，下載範例程式碼。此命令將官方 AWS SDK 文檔中使用的所有代碼示例副本下載到您環境的根目錄中。

```
git clone https://github.com/awsdocs/aws-doc-sdk-examples.git
```

若要尋找 SDK 的程式碼範例 JavaScript，請使用 [環境] 視窗開啟 [環境] 視窗*ENVIRONMENT_NAME*\aws-doc-sdk-examples\javascriptv3\example_code/src，其中*####*是您 AWS Cloud9 開發環境的名稱。

若要了解如何使用這些範例和其他程式碼範例，請參閱 SDK 以取得 JavaScript 程式碼範例。

步驟 5：執行和偵錯範例程式碼

若要在 AWS Cloud9 開發環境中執行程式碼，請參閱AWS Cloud9 使用者指南中的執行程式碼。

若要偵錯 Node.js 程式碼，請參閱「AWS Cloud9 使用者指南」中的「偵錯程式碼」。

SDK對於 JavaScript （ v3 ）代碼示例

本主題中的程式碼範例會示範如何搭配使用 AWS SDK for JavaScript (v3) AWS。

基本概念是程式碼範例，會示範如何在服務中執行基本作業。

Actions 是大型程式的程式碼摘錄，必須在內容中執行。雖然動作會顯示如何呼叫個別服務函數，但您可以在其相關案例中查看內容中的動作。

案例是程式碼範例，向您展示如何透過呼叫服務中的多個函式或與其他函式結合來完成特定工作 AWS 服務。

服務

- API使用 for 的閘道範SDK例 JavaScript (v3)
- Aurora 範例使用 SDK for JavaScript (v3)
- Auto Scaling 範SDK例使用 JavaScript (v3)
- Amazon 基岩示例使用 SDK for JavaScript (v3)
- Amazon 基岩運行時示例使用 to JavaScript (SDKv3)
- Amazon 基岩範例的代理程式使SDK用 JavaScript (v3)
- Amazon 基岩運行時示例的代理程序使SDK用 in JavaScript (v3)
- CloudWatch 使用 JavaScript （ v3 ）SDK的示例
- CloudWatch 事件範例使用 SDK for JavaScript (v3)
- CloudWatch 記錄範SDK例使用 JavaScript (v3)
- CodeBuild 使用 JavaScript （ v3 ）SDK的示例
- Amazon Cognito 身份提供商示例使SDK用 JavaScript (v3)
- Amazon Comprehend 使用SDK的例子 JavaScript (v3)
- Amazon DocumentDB 示例使用 SDK for JavaScript (v3)
- 使用SDK的範例 JavaScript (v3)
- Amazon EC2 示例使用 SDK for JavaScript (v3)
- Elastic Load Balancing-第 2 版範SDK例使用 in JavaScript (v3)
- EventBridge 使用 JavaScript （ v3 ）SDK的示例

- AWS Glue 使用 JavaScript （ v3 ）SDK的示例
- HealthImaging 使用 JavaScript （ v3 ）SDK的示例
- IAM使用 JavaScript （ v3 ）SDK的示例
- Kinesis 示例使SDK用 JavaScript (v3)
- 使用SDK的 Lambda 示例 JavaScript (v3)
- Amazon Lex 示SDK例使用 JavaScript （ v3 ）
- Amazon MSK 示例使用 SDK for JavaScript (v3)
- Amazon Personalize 化示例使用 SDK for JavaScript (v3)
- Amazon Personalize 化事件示SDK例使用 JavaScript in (v3)
- Amazon Personalize 化運行時示SDK例使用 JavaScript in (v3)
- Amazon Pinpoint 示例使用 SDK for JavaScript (v3)
- Amazon Polly 示例使SDK用 JavaScript (v3)
- Amazon RDS 示例使用 SDK for JavaScript (v3)
- Amazon RDS 數據服務示SDK例使用 JavaScript (v3)
- Amazon Redshift 示例使用 SDK for JavaScript (v3)
- Amazon Rekognition 示例使用 SDK for JavaScript (v3)
- Amazon S3 示例使SDK用 JavaScript （ v3 ）
- S3 冰川範SDK例使用 JavaScript (v3)
- SageMaker 使用 JavaScript （ v3 ）SDK的示例
- Secrets Manager 範例使用 SDK for JavaScript (v3)
- Amazon SES 示例使用 SDK for JavaScript (v3)
- Amazon SNS 示例使用 SDK for JavaScript (v3)
- Amazon SQS 示例使用 SDK for JavaScript (v3)
- Step Functions 示例使用 SDK for JavaScript (v3)
- AWS STS 使用 JavaScript （ v3 ）SDK的示例
- AWS Support 使用 JavaScript （ v3 ）SDK的示例
- Amazon Textract 範例使用 SDK for JavaScript (v3)
- Amazon Transcribe 示例使用 SDK for JavaScript (v3)
- Amazon Translate 示SDK例使用 JavaScript （ v3 ）

API使用 for 的閘道範SDK例 JavaScript (v3)

下列程式碼範例說明如何透過使用 AWS SDK for JavaScript (v3) 搭配 API Gateway 來執行動作和實作常見案例。

案例是程式碼範例，向您展示如何透過呼叫服務中的多個函式或與其他函式結合來完成特定工作 AWS 服務。

每個範例都包含完整原始程式碼的連結，您可以在其中找到如何在內容中設定和執行程式碼的指示。

主題

案例

建立無伺服器應用程式來管理相片

下列程式碼範例示範如何建立無伺服器應用程式，讓使用者以標籤管理相片。

SDK對於 JavaScript （3）

顯示如何開發照片資產管理應用程式，以便使用 Amazon Rekognition 偵測圖片中的標籤，並將其儲存以供日後擷取。

有關如何設置和運行的完整源代碼和說明，請參閱中的完整示例 GitHub。

如要深入探索此範例的來源，請參閱 AWS 社群上的文章。

此範例中使用的服務

- API閘道
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

使用API閘道來叫用 Lambda 函數

下列程式碼範例顯示如何建立 Amazon API 閘道叫用的 AWS Lambda 函數。

SDK對於 JavaScript （ 3 ）

示範如何使用 Lambda JavaScript 執行階段建立 AWS Lambda 函數API。此範例會呼叫不同的 AWS 服務來執行特定使用案例。此範例示範如何建立 Amazon API 閘道叫用的 Lambda 函數，以掃描 Amazon DynamoDB 表是否有工作週年紀念日，並使用 Amazon 簡單通知服務 (AmazonSNS) 向您的員工傳送文字訊息，並在一年週年紀念日祝賀他們。

有關如何設置和運行的完整源代碼和說明，請參閱中的完整示例GitHub。

此範例也可在 AWS SDK for JavaScript v3 開發人員指南中取得。

此範例中使用的服務

- API閘道
- DynamoDB
- Lambda
- Amazon SNS

Aurora 範例使用 SDK for JavaScript (v3)

下列程式碼範例會示範如何透過使用 AWS SDK for JavaScript (v3) 搭配 Aurora 來執行動作和實作常見案例。

案例是程式碼範例，向您展示如何透過呼叫服務中的多個函式或與其他函式結合來完成特定工作 AWS 服務。

每個範例都包含完整原始程式碼的連結，您可以在其中找到如何在內容中設定和執行程式碼的指示。

主題

- 案例

案例

建立 Aurora 無伺服器工作項目追蹤器

下列程式碼範例示範如何建立可追蹤 Amazon Aurora 無伺服器資料庫中工作項目的 Web 應用程式，並使用 Amazon 簡單電子郵件服務 (AmazonSES) 傳送報告。

SDK對於 JavaScript （ 3 ）

示範如何使用 AWS SDK for JavaScript (v3) 建立 Web 應用程式，以追蹤 Amazon Aurora 資料庫中的工作項目，以及使用 Amazon 簡單電子郵件服務 (AmazonSES) 傳送電子郵件報告的 Web 應用程式。這個範例使用以 React.js 建置的前端與 Express Node.js 後端互動。

- 將 React.js 網路應用程式與 AWS 服務.
- 列出、新增和更新 Aurora 資料表中的項目。
- 使用 Amazon 傳送篩選工作項目的電子郵件報告SES。
- 使用隨附的 AWS CloudFormation 指令碼部署和管理範例資源。

有關如何設置和運行的完整源代碼和說明，請參閱中的完整示例GitHub。

此範例中使用的服務

- Aurora
- Amazon RDS
- Amazon RDS 數據服務
- Amazon SES

Auto Scaling 範SDK例使用 JavaScript (v3)

下列程式碼範例說明如何使用 AWS SDK for JavaScript (v3) 搭配 Auto Scaling 來執行動作及實作常見案例。

Actions 是大型程式的程式碼摘錄，必須在內容中執行。雖然動作會顯示如何呼叫個別服務函數，但您可以在其相關案例中查看內容中的動作。

案例是程式碼範例，向您展示如何透過呼叫服務中的多個函式或與其他函式結合來完成特定工作 AWS 服務。

每個範例都包含完整原始程式碼的連結，您可以在其中找到如何在內容中設定和執行程式碼的指示。

主題

- 動作
- 案例

動作

`AttachLoadBalancerTargetGroups`

下列程式碼範例會示範如何使用`AttachLoadBalancerTargetGroups`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
const client = new AutoScalingClient({});
await client.send(
  new AttachLoadBalancerTargetGroupsCommand({
    AutoScalingGroupName: NAMES.autoScalingGroupName,
    TargetGroupARNs: [state.targetGroupArn],
  }),
);
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考AttachLoadBalancerTargetGroups中的。

案例

建置及管理彈性服務

下列程式碼範例會示範如何建立負載平衡的 Web 服務，以傳回書籍、影片和歌曲建議。此範例顯示服務如何回應失故障，以及如何在發生故障時重組服務以提高復原能力。

- 使用 Amazon EC2 Auto Scaling 群組根據啟動範本建立 Amazon 彈性運算雲端 (AmazonEC2) 執行個體，並將執行個體數量保持在指定範圍內。
- 使用 Elastic Load Balancing 處理和分配HTTP請求。
- 監控 Auto Scaling 群組中執行個體的運作狀態，並且只將請求轉送給運作良好的執行個體。
- 在每個執行個EC2體上執行 Python 網頁伺服器來處理HTTP要求。Web 伺服器會回應建議和運作狀態檢查。

- 使用 Amazon DynamoDB 資料表模擬建議服務。
- 透過更新 AWS Systems Manager 參數來控制 Web 伺服器對要求和健康狀態檢查的回應。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

在命令提示中執行互動式案例。

```
#!/usr/bin/env node
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0


import {
  Scenario,
  parseScenarioArgs,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

/**
 * The workflow steps are split into three stages:
 *   - deploy
 *   - demo
 *   - destroy
 *
 * Each of these stages has a corresponding file prefixed with steps-*.
 */
import { deploySteps } from "./steps-deploy.js";
import { demoSteps } from "./steps-demo.js";
import { destroySteps } from "./steps-destroy.js";

/**
 * The context is passed to every scenario. Scenario steps
 * will modify the context.
 */
const context = {};
```

```
/**
 * Three Scenarios are created for the workflow. A Scenario is an orchestration
 class
 * that simplifies running a series of steps.
 */
export const scenarios = {
  // Deploys all resources necessary for the workflow.
  deploy: new Scenario("Resilient Workflow - Deploy", deploySteps, context),
  // Demonstrates how a fragile web service can be made more resilient.
  demo: new Scenario("Resilient Workflow - Demo", demoSteps, context),
  // Destroys the resources created for the workflow.
  destroy: new Scenario("Resilient Workflow - Destroy", destroySteps, context),
};

// Call function if run directly
import { fileURLToPath } from "url";

if (process.argv[1] === fileURLToPath(import.meta.url)) {
  parseScenarioArgs(scenarios);
}
```

建立步驟以部署所有資源。

```
import { join } from "node:path";
import { readFileSync, writeFileSync } from "node:fs";
import axios from "axios";

import {
  BatchWriteItemCommand,
  CreateTableCommand,
  DynamoDBClient,
  waitUntilTableExists,
} from "@aws-sdk/client-dynamodb";
import {
  EC2Client,
  CreateKeyPairCommand,
  CreateLaunchTemplateCommand,
  DescribeAvailabilityZonesCommand,
  DescribeVpcsCommand,
  DescribeSubnetsCommand,
  DescribeSecurityGroupsCommand,
  AuthorizeSecurityGroupIngressCommand,
```

```
} from "@aws-sdk/client-ec2";
import {
  IAMClient,
  CreatePolicyCommand,
  CreateRoleCommand,
  CreateInstanceProfileCommand,
  AddRoleToInstanceProfileCommand,
  AttachRolePolicyCommand,
  waitUntilInstanceProfileExists,
} from "@aws-sdk/client-iam";
import { SSMClient, GetParameterCommand } from "@aws-sdk/client-ssm";
import {
  CreateAutoScalingGroupCommand,
  AutoScalingClient,
  AttachLoadBalancerTargetGroupsCommand,
} from "@aws-sdk/client-auto-scaling";
import {
  CreateListenerCommand,
  CreateLoadBalancerCommand,
  CreateTargetGroupCommand,
  ElasticLoadBalancingV2Client,
  waitUntilLoadBalancerAvailable,
} from "@aws-sdk/client-elastic-load-balancing-v2";

import {
  ScenarioOutput,
  ScenarioInput,
  ScenarioAction,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import { saveState } from "@aws-doc-sdk-examples/lib/scenario/steps-common.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

import { MESSAGES, NAMES, RESOURCES_PATH, ROOT } from "./constants.js";
import { initParamsSteps } from "./steps-reset-params.js";

/**
 * @type {import('@aws-doc-sdk-examples/lib/scenario.js').Step[]}
 */
export const deploySteps = [
  new ScenarioOutput("introduction", MESSAGES.introduction, { header: true }),
  new ScenarioInput("confirmDeployment", MESSAGES.confirmDeployment, {
    type: "confirm",
  }),
  new ScenarioAction(
```

```
    "handleConfirmDeployment",
    (c) => c.confirmDeployment === false && process.exit(),
  ),
  new ScenarioOutput(
    "creatingTable",
    MESSAGES.creatingTable.replace("${TABLE_NAME}", NAMES.tableName),
  ),
  new ScenarioAction("createTable", async () => {
    const client = new DynamoDBClient({});
    await client.send(
      new CreateTableCommand({
        TableName: NAMES.tableName,
        ProvisionedThroughput: {
          ReadCapacityUnits: 5,
          WriteCapacityUnits: 5,
        },
        AttributeDefinitions: [
          {
            AttributeName: "MediaType",
            AttributeType: "S",
          },
          {
            AttributeName: "ItemId",
            AttributeType: "N",
          },
        ],
        KeySchema: [
          {
            AttributeName: "MediaType",
            KeyType: "HASH",
          },
          {
            AttributeName: "ItemId",
            KeyType: "RANGE",
          },
        ],
      }),
    );
    await waitUntilTableExists({ client }, { TableName: NAMES.tableName });
  }),
  new ScenarioOutput(
    "createdTable",
    MESSAGES.createdTable.replace("${TABLE_NAME}", NAMES.tableName),
  ),
```

```
  new ScenarioOutput(
    "populatingTable",
    MESSAGES.populatingTable.replace("${TABLE_NAME}", NAMES.tableName),
  ),
  new ScenarioAction("populateTable", () => {
    const client = new DynamoDBClient({});
    /**
     * @type {{ default: import("@aws-sdk/client-dynamodb").PutRequest['Item'][] }}
     */
    const recommendations = JSON.parse(
      readFileSync(join(RESOURCES_PATH, "recommendations.json")),
    );

    return client.send(
      new BatchWriteItemCommand({
        RequestItems: {
          [NAMES.tableName]: recommendations.map((item) => ({
            PutRequest: { Item: item },
          })),
        },
      }),
    );
  }),
  new ScenarioOutput(
    "populatedTable",
    MESSAGES.populatedTable.replace("${TABLE_NAME}", NAMES.tableName),
  ),
  new ScenarioOutput(
    "creatingKeyPair",
    MESSAGES.creatingKeyPair.replace("${KEY_PAIR_NAME}", NAMES.keyPairName),
  ),
  new ScenarioAction("createKeyPair", async () => {
    const client = new EC2Client({});
    const { KeyMaterial } = await client.send(
      new CreateKeyPairCommand({
        KeyName: NAMES.keyPairName,
      }),
    );

    writeFileSync(`${NAMES.keyPairName}.pem`, KeyMaterial, { mode: 0o600 });
  }),
  new ScenarioOutput(
    "createdKeyPair",
    MESSAGES.createdKeyPair.replace("${KEY_PAIR_NAME}", NAMES.keyPairName),
```

```
  ),
  new ScenarioOutput(
    "creatingInstancePolicy",
    MESSAGES.creatingInstancePolicy.replace(
      "${INSTANCE_POLICY_NAME}",
      NAMES.instancePolicyName,
    ),
  ),
  new ScenarioAction("createInstancePolicy", async (state) => {
    const client = new IAMClient({});
    const {
      Policy: { Arn },
    } = await client.send(
      new CreatePolicyCommand({
        PolicyName: NAMES.instancePolicyName,
        PolicyDocument: readFileSync(
          join(RESOURCES_PATH, "instance_policy.json"),
        ),
      }),
    );
    state.instancePolicyArn = Arn;
  }),
  new ScenarioOutput("createdInstancePolicy", (state) =>
    MESSAGES.createdInstancePolicy
      .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
      .replace("${INSTANCE_POLICY_ARN}", state.instancePolicyArn),
  ),
  new ScenarioOutput(
    "creatingInstanceRole",
    MESSAGES.creatingInstanceRole.replace(
      "${INSTANCE_ROLE_NAME}",
      NAMES.instanceRoleName,
    ),
  ),
  new ScenarioAction("createInstanceRole", () => {
    const client = new IAMClient({});
    return client.send(
      new CreateRoleCommand({
        RoleName: NAMES.instanceRoleName,
        AssumeRolePolicyDocument: readFileSync(
          join(ROOT, "assume-role-policy.json"),
        ),
      }),
    );
```

```
  }),
  new ScenarioOutput(
    "createdInstanceRole",
    MESSAGES.createdInstanceRole.replace(
      "${INSTANCE_ROLE_NAME}",
      NAMES.instanceRoleName,
    ),
  ),
  new ScenarioOutput(
    "attachingPolicyToRole",
    MESSAGES.attachingPolicyToRole
      .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName)
      .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName),
  ),
  new ScenarioAction("attachPolicyToRole", async (state) => {
    const client = new IAMClient({});
    await client.send(
      new AttachRolePolicyCommand({
        RoleName: NAMES.instanceRoleName,
        PolicyArn: state.instancePolicyArn,
      }),
    );
  }),
  new ScenarioOutput(
    "attachedPolicyToRole",
    MESSAGES.attachedPolicyToRole
      .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
      .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName),
  ),
  new ScenarioOutput(
    "creatingInstanceProfile",
    MESSAGES.creatingInstanceProfile.replace(
      "${INSTANCE_PROFILE_NAME}",
      NAMES.instanceProfileName,
    ),
  ),
  new ScenarioAction("createInstanceProfile", async (state) => {
    const client = new IAMClient({});
    const {
      InstanceProfile: { Arn },
    } = await client.send(
      new CreateInstanceProfileCommand({
        InstanceProfileName: NAMES.instanceProfileName,
      }),
```

```
    );
    state.instanceProfileArn = Arn;

    await waitUntilInstanceProfileExists(
      { client },
      { InstanceProfileName: NAMES.instanceProfileName },
    );
  }),
  new ScenarioOutput("createdInstanceProfile", (state) =>
    MESSAGES.createdInstanceProfile
      .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
      .replace("${INSTANCE_PROFILE_ARN}", state.instanceProfileArn),
  ),
  new ScenarioOutput(
    "addingRoleToInstanceProfile",
    MESSAGES.addingRoleToInstanceProfile
      .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
      .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName),
  ),
  new ScenarioAction("addRoleToInstanceProfile", () => {
    const client = new IAMClient({});
    return client.send(
      new AddRoleToInstanceProfileCommand({
        RoleName: NAMES.instanceRoleName,
        InstanceProfileName: NAMES.instanceProfileName,
      }),
    );
  }),
  new ScenarioOutput(
    "addedRoleToInstanceProfile",
    MESSAGES.addedRoleToInstanceProfile
      .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
      .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName),
  ),
  ...initParamsSteps,
  new ScenarioOutput("creatingLaunchTemplate", MESSAGES.creatingLaunchTemplate),
  new ScenarioAction("createLaunchTemplate", async () => {
    const ssmClient = new SSMClient({});
    const { Parameter } = await ssmClient.send(
      new GetParameterCommand({
        Name: "/aws/service/ami-amazon-linux-latest/amzn2-ami-hvm-x86_64-gp2",
      }),
    );
    const ec2Client = new EC2Client({});
```

```
    await ec2Client.send(
      new CreateLaunchTemplateCommand({
        LaunchTemplateName: NAMES.launchTemplateName,
        LaunchTemplateData: {
          InstanceType: "t3.micro",
          ImageId: Parameter.Value,
          IamInstanceProfile: { Name: NAMES.instanceProfileName },
          UserData: readFileSync(
            join(RESOURCES_PATH, "server_startup_script.sh"),
          ).toString("base64"),
          KeyName: NAMES.keyPairName,
        },
      }),
    );
  }),
  new ScenarioOutput(
    "createdLaunchTemplate",
    MESSAGES.createdLaunchTemplate.replace(
      "${LAUNCH_TEMPLATE_NAME}",
      NAMES.launchTemplateName,
    ),
  ),
  new ScenarioOutput(
    "creatingAutoScalingGroup",
    MESSAGES.creatingAutoScalingGroup.replace(
      "${AUTO_SCALING_GROUP_NAME}",
      NAMES.autoScalingGroupName,
    ),
  ),
  new ScenarioAction("createAutoScalingGroup", async (state) => {
    const ec2Client = new EC2Client({});
    const { AvailabilityZones } = await ec2Client.send(
      new DescribeAvailabilityZonesCommand({}),
    );
    state.availabilityZoneNames = AvailabilityZones.map((az) => az.ZoneName);
    const autoScalingClient = new AutoScalingClient({});
    await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
      autoScalingClient.send(
        new CreateAutoScalingGroupCommand({
          AvailabilityZones: state.availabilityZoneNames,
          AutoScalingGroupName: NAMES.autoScalingGroupName,
          LaunchTemplate: {
            LaunchTemplateName: NAMES.launchTemplateName,
            Version: "$Default",
```

```
        },
        MinSize: 3,
        MaxSize: 3,
      }),
    ),
  );
}),
new ScenarioOutput(
  "createdAutoScalingGroup",
  /**
   * @param {{ availabilityZoneNames: string[] }} state
   */
  (state) =>
    MESSAGES.createdAutoScalingGroup
      .replace("${AUTO_SCALING_GROUP_NAME}", NAMES.autoScalingGroupName)
      .replace(
        "${AVAILABILITY_ZONE_NAMES}",
        state.availabilityZoneNames.join(", "),
      ),
),
new ScenarioInput("confirmContinue", MESSAGES.confirmContinue, {
  type: "confirm",
}),
new ScenarioOutput("loadBalancer", MESSAGES.loadBalancer),
new ScenarioOutput("gettingVpc", MESSAGES.gettingVpc),
new ScenarioAction("getVpc", async (state) => {
  const client = new EC2Client({});
  const { Vpcs } = await client.send(
    new DescribeVpcsCommand({
      Filters: [{ Name: "is-default", Values: ["true"] }],
    }),
  );
  state.defaultVpc = Vpcs[0].VpcId;
}),
new ScenarioOutput("gotVpc", (state) =>
  MESSAGES.gotVpc.replace("${VPC_ID}", state.defaultVpc),
),
new ScenarioOutput("gettingSubnets", MESSAGES.gettingSubnets),
new ScenarioAction("getSubnets", async (state) => {
  const client = new EC2Client({});
  const { Subnets } = await client.send(
    new DescribeSubnetsCommand({
      Filters: [
        { Name: "vpc-id", Values: [state.defaultVpc] },
```

```
          { Name: "availability-zone", Values: state.availabilityZoneNames },
          { Name: "default-for-az", Values: ["true"] },
        ],
      }),
    );
    state.subnets = Subnets.map((subnet) => subnet.SubnetId);
  }),
  new ScenarioOutput(
    "gotSubnets",
    /**
     * @param {{ subnets: string[] }} state
     */
    (state) =>
      MESSAGES.gotSubnets.replace("${SUBNETS}", state.subnets.join(", ")),
  ),
  new ScenarioOutput(
    "creatingLoadBalancerTargetGroup",
    MESSAGES.creatingLoadBalancerTargetGroup.replace(
      "${TARGET_GROUP_NAME}",
      NAMES.loadBalancerTargetGroupName,
    ),
  ),
  new ScenarioAction("createLoadBalancerTargetGroup", async (state) => {
    const client = new ElasticLoadBalancingV2Client({});
    const { TargetGroups } = await client.send(
      new CreateTargetGroupCommand({
        Name: NAMES.loadBalancerTargetGroupName,
        Protocol: "HTTP",
        Port: 80,
        HealthCheckPath: "/healthcheck",
        HealthCheckIntervalSeconds: 10,
        HealthCheckTimeoutSeconds: 5,
        HealthyThresholdCount: 2,
        UnhealthyThresholdCount: 2,
        VpcId: state.defaultVpc,
      }),
    );
    const targetGroup = TargetGroups[0];
    state.targetGroupArn = targetGroup.TargetGroupArn;
    state.targetGroupProtocol = targetGroup.Protocol;
    state.targetGroupPort = targetGroup.Port;
  }),
  new ScenarioOutput(
    "createdLoadBalancerTargetGroup",
```

```
    MESSAGES.createdLoadBalancerTargetGroup.replace(
      "${TARGET_GROUP_NAME}",
      NAMES.loadBalancerTargetGroupName,
    ),
  ),
  new ScenarioOutput(
    "creatingLoadBalancer",
    MESSAGES.creatingLoadBalancer.replace("${LB_NAME}", NAMES.loadBalancerName),
  ),
  new ScenarioAction("createLoadBalancer", async (state) => {
    const client = new ElasticLoadBalancingV2Client({});
    const { LoadBalancers } = await client.send(
      new CreateLoadBalancerCommand({
        Name: NAMES.loadBalancerName,
        Subnets: state.subnets,
      }),
    );
    state.loadBalancerDns = LoadBalancers[0].DNSName;
    state.loadBalancerArn = LoadBalancers[0].LoadBalancerArn;
    await waitUntilLoadBalancerAvailable(
      { client },
      { Names: [NAMES.loadBalancerName] },
    );
  }),
  new ScenarioOutput("createdLoadBalancer", (state) =>
    MESSAGES.createdLoadBalancer
      .replace("${LB_NAME}", NAMES.loadBalancerName)
      .replace("${DNS_NAME}", state.loadBalancerDns),
  ),
  new ScenarioOutput(
    "creatingListener",
    MESSAGES.creatingLoadBalancerListener
      .replace("${LB_NAME}", NAMES.loadBalancerName)
      .replace("${TARGET_GROUP_NAME}", NAMES.loadBalancerTargetGroupName),
  ),
  new ScenarioAction("createListener", async (state) => {
    const client = new ElasticLoadBalancingV2Client({});
    const { Listeners } = await client.send(
      new CreateListenerCommand({
        LoadBalancerArn: state.loadBalancerArn,
        Protocol: state.targetGroupProtocol,
        Port: state.targetGroupPort,
        DefaultActions: [
          { Type: "forward", TargetGroupArn: state.targetGroupArn },
```

```
        ],
      }),
    );
    const listener = Listeners[0];
    state.loadBalancerListenerArn = listener.ListenerArn;
  }),
  new ScenarioOutput("createdListener", (state) =>
    MESSAGES.createdLoadBalancerListener.replace(
      "${LB_LISTENER_ARN}",
      state.loadBalancerListenerArn,
    ),
  ),
  new ScenarioOutput(
    "attachingLoadBalancerTargetGroup",
    MESSAGES.attachingLoadBalancerTargetGroup
      .replace("${TARGET_GROUP_NAME}", NAMES.loadBalancerTargetGroupName)
      .replace("${AUTO_SCALING_GROUP_NAME}", NAMES.autoScalingGroupName),
  ),
  new ScenarioAction("attachLoadBalancerTargetGroup", async (state) => {
    const client = new AutoScalingClient({});
    await client.send(
      new AttachLoadBalancerTargetGroupsCommand({
        AutoScalingGroupName: NAMES.autoScalingGroupName,
        TargetGroupARNs: [state.targetGroupArn],
      }),
    );
  }),
  new ScenarioOutput(
    "attachedLoadBalancerTargetGroup",
    MESSAGES.attachedLoadBalancerTargetGroup,
  ),
  new ScenarioOutput("verifyingInboundPort", MESSAGES.verifyingInboundPort),
  new ScenarioAction(
    "verifyInboundPort",
    /**
     *
     * @param {{ defaultSecurityGroup: import('@aws-sdk/client-ec2').SecurityGroup}}
state
     */
    async (state) => {
      const client = new EC2Client({});
      const { SecurityGroups } = await client.send(
        new DescribeSecurityGroupsCommand({
          Filters: [{ Name: "group-name", Values: ["default"] }],
```

```
      }),
    );
    if (!SecurityGroups) {
      state.verifyInboundPortError = new Error(MESSAGES.noSecurityGroups);
    }
    state.defaultSecurityGroup = SecurityGroups[0];

    /**
     * @type {string}
     */
    const ipResponse = (await axios.get("http://checkip.amazonaws.com")).data;
    state.myIp = ipResponse.trim();
    const myIpRules = state.defaultSecurityGroup.IpPermissions.filter(
      ({ IpRanges }) =>
        IpRanges.some(
          ({ CidrIp }) =>
            CidrIp.startsWith(state.myIp) || CidrIp === "0.0.0.0/0",
        ),
    )
      .filter(({ IpProtocol }) => IpProtocol === "tcp")
      .filter(({ FromPort }) => FromPort === 80);

    state.myIpRules = myIpRules;
  },
),
new ScenarioOutput(
  "verifiedInboundPort",
  /**
   * @param {{ myIpRules: any[] }} state
   */
  (state) => {
    if (state.myIpRules.length > 0) {
      return MESSAGES.foundIpRules.replace(
        "${IP_RULES}",
        JSON.stringify(state.myIpRules, null, 2),
      );
    } else {
      return MESSAGES.noIpRules;
    }
  },
),
new ScenarioInput(
  "shouldAddInboundRule",
  /**
```

```
     * @param {{ myIpRules: any[] }} state
     */
    (state) => {
      if (state.myIpRules.length > 0) {
        return false;
      } else {
        return MESSAGES.noIpRules;
      }
    },
    { type: "confirm" },
  ),
  new ScenarioAction(
    "addInboundRule",
    /**
     * @param {{ defaultSecurityGroup: import('@aws-sdk/client-
ec2').SecurityGroup }} state
     */
    async (state) => {
      if (!state.shouldAddInboundRule) {
        return;
      }

      const client = new EC2Client({});
      await client.send(
        new AuthorizeSecurityGroupIngressCommand({
          GroupId: state.defaultSecurityGroup.GroupId,
          CidrIp: `${state.myIp}/32`,
          FromPort: 80,
          ToPort: 80,
          IpProtocol: "tcp",
        }),
      );
    },
  ),
  new ScenarioOutput("addedInboundRule", (state) => {
    if (state.shouldAddInboundRule) {
      return MESSAGES.addedInboundRule.replace("${IP_ADDRESS}", state.myIp);
    } else {
      return false;
    }
  }),
  new ScenarioOutput("verifyingEndpoint", (state) =>
    MESSAGES.verifyingEndpoint.replace("${DNS_NAME}", state.loadBalancerDns),
  ),
```

```
  new ScenarioAction("verifyEndpoint", async (state) => {
    try {
      const response = await retry({ intervalInMs: 2000, maxRetries: 30 }, () =>
        axios.get(`http://${state.loadBalancerDns}`),
      );
      state.endpointResponse = JSON.stringify(response.data, null, 2);
    } catch (e) {
      state.verifyEndpointError = e;
    }
  }),
  new ScenarioOutput("verifiedEndpoint", (state) => {
    if (state.verifyEndpointError) {
      console.error(state.verifyEndpointError);
    } else {
      return MESSAGES.verifiedEndpoint.replace(
        "${ENDPOINT_RESPONSE}",
        state.endpointResponse,
      );
    }
  }),
  saveState,
];
```

建立步驟以執行示範。

```
import { readFileSync } from "node:fs";
import { join } from "node:path";

import axios from "axios";

import {
  DescribeTargetGroupsCommand,
  DescribeTargetHealthCommand,
  ElasticLoadBalancingV2Client,
} from "@aws-sdk/client-elastic-load-balancing-v2";
import {
  DescribeInstanceInformationCommand,
  PutParameterCommand,
  SSMClient,
  SendCommandCommand,
} from "@aws-sdk/client-ssm";
import {
```

```
  IAMClient,
  CreatePolicyCommand,
  CreateRoleCommand,
  AttachRolePolicyCommand,
  CreateInstanceProfileCommand,
  AddRoleToInstanceProfileCommand,
  waitUntilInstanceProfileExists,
} from "@aws-sdk/client-iam";
import {
  AutoScalingClient,
  DescribeAutoScalingGroupsCommand,
  TerminateInstanceInAutoScalingGroupCommand,
} from "@aws-sdk/client-auto-scaling";
import {
  DescribeIamInstanceProfileAssociationsCommand,
  EC2Client,
  RebootInstancesCommand,
  ReplaceIamInstanceProfileAssociationCommand,
} from "@aws-sdk/client-ec2";

import {
  ScenarioAction,
  ScenarioInput,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/scenario.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

import { MESSAGES, NAMES, RESOURCES_PATH } from "./constants.js";
import { findLoadBalancer } from "./shared.js";

const getRecommendation = new ScenarioAction(
  "getRecommendation",
  async (state) => {
    const loadBalancer = await findLoadBalancer(NAMES.loadBalancerName);
    if (loadBalancer) {
      state.loadBalancerDnsName = loadBalancer.DNSName;
      try {
        state.recommendation = (
          await axios.get(`http://${state.loadBalancerDnsName}`)
        ).data;
      } catch (e) {
        state.recommendation = e instanceof Error ? e.message : e;
      }
    } else {
```

```
      throw new Error(MESSAGES.demoFindLoadBalancerError);
    }
  },
);

const getRecommendationResult = new ScenarioOutput(
  "getRecommendationResult",
  (state) =>
    `Recommendation:\n${JSON.stringify(state.recommendation, null, 2)}`,
  { preformatted: true },
);

const getHealthCheck = new ScenarioAction("getHealthCheck", async (state) => {
  const client = new ElasticLoadBalancingV2Client({});
  const { TargetGroups } = await client.send(
    new DescribeTargetGroupsCommand({
      Names: [NAMES.loadBalancerTargetGroupName],
    }),
  );

  const { TargetHealthDescriptions } = await client.send(
    new DescribeTargetHealthCommand({
      TargetGroupArn: TargetGroups[0].TargetGroupArn,
    }),
  );
  state.targetHealthDescriptions = TargetHealthDescriptions;
});

const getHealthCheckResult = new ScenarioOutput(
  "getHealthCheckResult",
  /**
   * @param {{ targetHealthDescriptions: import('@aws-sdk/client-elastic-load-
balancing-v2').TargetHealthDescription[]}} state
   */
  (state) => {
    const status = state.targetHealthDescriptions
      .map((th) => `${th.Target.Id}: ${th.TargetHealth.State}`)
      .join("\n");
    return `Health check:\n${status}`;
  },
  { preformatted: true },
);

const loadBalancerLoop = new ScenarioAction(
```

```
  "loadBalancerLoop",
  getRecommendation.action,
  {
    whileConfig: {
      whileFn: ({ loadBalancerCheck }) => loadBalancerCheck,
      input: new ScenarioInput(
        "loadBalancerCheck",
        MESSAGES.demoLoadBalancerCheck,
        {
          type: "confirm",
        },
      ),
      output: getRecommendationResult,
    },
  },
);

const healthCheckLoop = new ScenarioAction(
  "healthCheckLoop",
  getHealthCheck.action,
  {
    whileConfig: {
      whileFn: ({ healthCheck }) => healthCheck,
      input: new ScenarioInput("healthCheck", MESSAGES.demoHealthCheck, {
        type: "confirm",
      }),
      output: getHealthCheckResult,
    },
  },
);

const statusSteps = [
  getRecommendation,
  getRecommendationResult,
  getHealthCheck,
  getHealthCheckResult,
];

/**
 * @type {import('@aws-doc-sdk-examples/lib/scenario.js').Step[]}
 */
export const demoSteps = [
  new ScenarioOutput("header", MESSAGES.demoHeader, { header: true }),
  new ScenarioOutput("sanityCheck", MESSAGES.demoSanityCheck),
```

```
...statusSteps,
new ScenarioInput(
  "brokenDependencyConfirmation",
  MESSAGES.demoBrokenDependencyConfirmation,
  { type: "confirm" },
),
new ScenarioAction("brokenDependency", async (state) => {
  if (!state.brokenDependencyConfirmation) {
    process.exit();
  } else {
    const client = new SSMClient({});
    state.badTableName = `fake-table-${Date.now()}`;
    await client.send(
      new PutParameterCommand({
        Name: NAMES.ssmTableNameKey,
        Value: state.badTableName,
        Overwrite: true,
        Type: "String",
      }),
    );
  }
}),
new ScenarioOutput("testBrokenDependency", (state) =>
  MESSAGES.demoTestBrokenDependency.replace(
    "${TABLE_NAME}",
    state.badTableName,
  ),
),
...statusSteps,
new ScenarioInput(
  "staticResponseConfirmation",
  MESSAGES.demoStaticResponseConfirmation,
  { type: "confirm" },
),
new ScenarioAction("staticResponse", async (state) => {
  if (!state.staticResponseConfirmation) {
    process.exit();
  } else {
    const client = new SSMClient({});
    await client.send(
      new PutParameterCommand({
        Name: NAMES.ssmFailureResponseKey,
        Value: "static",
        Overwrite: true,
```

```
          Type: "String",
        }),
      );
    }
  }),
  new ScenarioOutput("testStaticResponse", MESSAGES.demoTestStaticResponse),
  ...statusSteps,
  new ScenarioInput(
    "badCredentialsConfirmation",
    MESSAGES.demoBadCredentialsConfirmation,
    { type: "confirm" },
  ),
  new ScenarioAction("badCredentialsExit", (state) => {
    if (!state.badCredentialsConfirmation) {
      process.exit();
    }
  }),
  new ScenarioAction("fixDynamoDBName", async () => {
    const client = new SSMClient({});
    await client.send(
      new PutParameterCommand({
        Name: NAMES.ssmTableNameKey,
        Value: NAMES.tableName,
        Overwrite: true,
        Type: "String",
      }),
    );
  }),
  new ScenarioAction(
    "badCredentials",
    /**
     * @param {{ targetInstance: import('@aws-sdk/client-auto-scaling').Instance }}
state
     */
    async (state) => {
      await createSsmOnlyInstanceProfile();
      const autoScalingClient = new AutoScalingClient({});
      const { AutoScalingGroups } = await autoScalingClient.send(
        new DescribeAutoScalingGroupsCommand({
          AutoScalingGroupNames: [NAMES.autoScalingGroupName],
        }),
      );
      state.targetInstance = AutoScalingGroups[0].Instances[0];
      const ec2Client = new EC2Client({});
```

```
  const { IamInstanceProfileAssociations } = await ec2Client.send(
    new DescribeIamInstanceProfileAssociationsCommand({
      Filters: [
        { Name: "instance-id", Values: [state.targetInstance.InstanceId] },
      ],
    }),
  );
  state.instanceProfileAssociationId =
    IamInstanceProfileAssociations[0].AssociationId;
  await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
    ec2Client.send(
      new ReplaceIamInstanceProfileAssociationCommand({
        AssociationId: state.instanceProfileAssociationId,
        IamInstanceProfile: { Name: NAMES.ssmOnlyInstanceProfileName },
      }),
    ),
  );

  await ec2Client.send(
    new RebootInstancesCommand({
      InstanceIds: [state.targetInstance.InstanceId],
    }),
  );

  const ssmClient = new SSMClient({});
  await retry({ intervalInMs: 20000, maxRetries: 15 }, async () => {
    const { InstanceInformationList } = await ssmClient.send(
      new DescribeInstanceInformationCommand({}),
    );

    const instance = InstanceInformationList.find(
      (info) => info.InstanceId === state.targetInstance.InstanceId,
    );

    if (!instance) {
      throw new Error("Instance not found.");
    }
  });

  await ssmClient.send(
    new SendCommandCommand({
      InstanceIds: [state.targetInstance.InstanceId],
      DocumentName: "AWS-RunShellScript",
      Parameters: { commands: ["cd / && sudo python3 server.py 80"] },
```

```
        }),
      );
    },
  ),
  new ScenarioOutput(
    "testBadCredentials",
    /**
     * @param {{ targetInstance: import('@aws-sdk/client-ssm').InstanceInformation}}
state
     */
    (state) =>
      MESSAGES.demoTestBadCredentials.replace(
        "${INSTANCE_ID}",
        state.targetInstance.InstanceId,
      ),
  ),
  loadBalancerLoop,
  new ScenarioInput(
    "deepHealthCheckConfirmation",
    MESSAGES.demoDeepHealthCheckConfirmation,
    { type: "confirm" },
  ),
  new ScenarioAction("deepHealthCheckExit", (state) => {
    if (!state.deepHealthCheckConfirmation) {
      process.exit();
    }
  }),
  new ScenarioAction("deepHealthCheck", async () => {
    const client = new SSMClient({});
    await client.send(
      new PutParameterCommand({
        Name: NAMES.ssmHealthCheckKey,
        Value: "deep",
        Overwrite: true,
        Type: "String",
      }),
    );
  }),
  new ScenarioOutput("testDeepHealthCheck", MESSAGES.demoTestDeepHealthCheck),
  healthCheckLoop,
  loadBalancerLoop,
  new ScenarioInput(
    "killInstanceConfirmation",
    /**
```

```
     * @param {{ targetInstance: import('@aws-sdk/client-
ssm').InstanceInformation }} state
     */
    (state) =>
      MESSAGES.demoKillInstanceConfirmation.replace(
        "${INSTANCE_ID}",
        state.targetInstance.InstanceId,
      ),
    { type: "confirm" },
  ),
  new ScenarioAction("killInstanceExit", (state) => {
    if (!state.killInstanceConfirmation) {
      process.exit();
    }
  }),
  new ScenarioAction(
    "killInstance",
    /**
     * @param {{ targetInstance: import('@aws-sdk/client-
ssm').InstanceInformation }} state
     */
    async (state) => {
      const client = new AutoScalingClient({});
      await client.send(
        new TerminateInstanceInAutoScalingGroupCommand({
          InstanceId: state.targetInstance.InstanceId,
          ShouldDecrementDesiredCapacity: false,
        }),
      );
    },
  ),
  new ScenarioOutput("testKillInstance", MESSAGES.demoTestKillInstance),
  healthCheckLoop,
  loadBalancerLoop,
  new ScenarioInput("failOpenConfirmation", MESSAGES.demoFailOpenConfirmation, {
    type: "confirm",
  }),
  new ScenarioAction("failOpenExit", (state) => {
    if (!state.failOpenConfirmation) {
      process.exit();
    }
  }),
  new ScenarioAction("failOpen", () => {
    const client = new SSMClient({});
```

```
    return client.send(
      new PutParameterCommand({
        Name: NAMES.ssmTableNameKey,
        Value: `fake-table-${Date.now()}`,
        Overwrite: true,
        Type: "String",
      }),
    );
  }),
  new ScenarioOutput("testFailOpen", MESSAGES.demoFailOpenTest),
  healthCheckLoop,
  loadBalancerLoop,
  new ScenarioInput(
    "resetTableConfirmation",
    MESSAGES.demoResetTableConfirmation,
    { type: "confirm" },
  ),
  new ScenarioAction("resetTableExit", (state) => {
    if (!state.resetTableConfirmation) {
      process.exit();
    }
  }),
  new ScenarioAction("resetTable", async () => {
    const client = new SSMClient({});
    await client.send(
      new PutParameterCommand({
        Name: NAMES.ssmTableNameKey,
        Value: NAMES.tableName,
        Overwrite: true,
        Type: "String",
      }),
    );
  }),
  new ScenarioOutput("testResetTable", MESSAGES.demoTestResetTable),
  healthCheckLoop,
  loadBalancerLoop,
];

async function createSsmOnlyInstanceProfile() {
  const iamClient = new IAMClient({});
  const { Policy } = await iamClient.send(
    new CreatePolicyCommand({
      PolicyName: NAMES.ssmOnlyPolicyName,
      PolicyDocument: readFileSync(
```

```
      join(RESOURCES_PATH, "ssm_only_policy.json"),
    ),
  }),
);
await iamClient.send(
  new CreateRoleCommand({
    RoleName: NAMES.ssmOnlyRoleName,
    AssumeRolePolicyDocument: JSON.stringify({
      Version: "2012-10-17",
      Statement: [
        {
          Effect: "Allow",
          Principal: { Service: "ec2.amazonaws.com" },
          Action: "sts:AssumeRole",
        },
      ],
    }),
  }),
);
await iamClient.send(
  new AttachRolePolicyCommand({
    RoleName: NAMES.ssmOnlyRoleName,
    PolicyArn: Policy.Arn,
  }),
);
await iamClient.send(
  new AttachRolePolicyCommand({
    RoleName: NAMES.ssmOnlyRoleName,
    PolicyArn: "arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore",
  }),
);
const { InstanceProfile } = await iamClient.send(
  new CreateInstanceProfileCommand({
    InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
  }),
);
await waitUntilInstanceProfileExists(
  { client: iamClient },
  { InstanceProfileName: NAMES.ssmOnlyInstanceProfileName },
);
await iamClient.send(
  new AddRoleToInstanceProfileCommand({
    InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
    RoleName: NAMES.ssmOnlyRoleName,
```

```
    }),
  );

  return InstanceProfile;
}
```

建立步驟以銷毀所有資源。

```
import { unlinkSync } from "node:fs";

import { DynamoDBClient, DeleteTableCommand } from "@aws-sdk/client-dynamodb";
import {
  EC2Client,
  DeleteKeyPairCommand,
  DeleteLaunchTemplateCommand,
  RevokeSecurityGroupIngressCommand,
} from "@aws-sdk/client-ec2";
import {
  IAMClient,
  DeleteInstanceProfileCommand,
  RemoveRoleFromInstanceProfileCommand,
  DeletePolicyCommand,
  DeleteRoleCommand,
  DetachRolePolicyCommand,
  paginateListPolicies,
} from "@aws-sdk/client-iam";
import {
  AutoScalingClient,
  DeleteAutoScalingGroupCommand,
  TerminateInstanceInAutoScalingGroupCommand,
  UpdateAutoScalingGroupCommand,
  paginateDescribeAutoScalingGroups,
} from "@aws-sdk/client-auto-scaling";
import {
  DeleteLoadBalancerCommand,
  DeleteTargetGroupCommand,
  DescribeTargetGroupsCommand,
  ElasticLoadBalancingV2Client,
} from "@aws-sdk/client-elastic-load-balancing-v2";

import {
  ScenarioOutput,
```

```
  ScenarioInput,
  ScenarioAction,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import { loadState } from "@aws-doc-sdk-examples/lib/scenario/steps-common.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

import { MESSAGES, NAMES } from "./constants.js";
import { findLoadBalancer } from "./shared.js";

/**
 * @type {import('@aws-doc-sdk-examples/lib/scenario.js').Step[]}
 */
export const destroySteps = [
  loadState,
  new ScenarioInput("destroy", MESSAGES.destroy, { type: "confirm" }),
  new ScenarioAction(
    "abort",
    (state) => state.destroy === false && process.exit(),
  ),
  new ScenarioAction("deleteTable", async (c) => {
    try {
      const client = new DynamoDBClient({});
      await client.send(new DeleteTableCommand({ TableName: NAMES.tableName }));
    } catch (e) {
      c.deleteTableError = e;
    }
  }),
  new ScenarioOutput("deleteTableResult", (state) => {
    if (state.deleteTableError) {
      console.error(state.deleteTableError);
      return MESSAGES.deleteTableError.replace(
        "${TABLE_NAME}",
        NAMES.tableName,
      );
    } else {
      return MESSAGES.deletedTable.replace("${TABLE_NAME}", NAMES.tableName);
    }
  }),
  new ScenarioAction("deleteKeyPair", async (state) => {
    try {
      const client = new EC2Client({});
      await client.send(
        new DeleteKeyPairCommand({ KeyName: NAMES.keyPairName }),
      );
```

```
      unlinkSync(`${NAMES.keyPairName}.pem`);
    } catch (e) {
      state.deleteKeyPairError = e;
    }
  }),
  new ScenarioOutput("deleteKeyPairResult", (state) => {
    if (state.deleteKeyPairError) {
      console.error(state.deleteKeyPairError);
      return MESSAGES.deleteKeyPairError.replace(
        "${KEY_PAIR_NAME}",
        NAMES.keyPairName,
      );
    } else {
      return MESSAGES.deletedKeyPair.replace(
        "${KEY_PAIR_NAME}",
        NAMES.keyPairName,
      );
    }
  }),
  new ScenarioAction("detachPolicyFromRole", async (state) => {
    try {
      const client = new IAMClient({});
      const policy = await findPolicy(NAMES.instancePolicyName);

      if (!policy) {
        state.detachPolicyFromRoleError = new Error(
          `Policy ${NAMES.instancePolicyName} not found.`,
        );
      } else {
        await client.send(
          new DetachRolePolicyCommand({
            RoleName: NAMES.instanceRoleName,
            PolicyArn: policy.Arn,
          }),
        );
      }
    } catch (e) {
      state.detachPolicyFromRoleError = e;
    }
  }),
  new ScenarioOutput("detachedPolicyFromRole", (state) => {
    if (state.detachPolicyFromRoleError) {
      console.error(state.detachPolicyFromRoleError);
      return MESSAGES.detachPolicyFromRoleError
```

```
      .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
      .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
  } else {
    return MESSAGES.detachedPolicyFromRole
      .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
      .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
  }
}),
new ScenarioAction("deleteInstancePolicy", async (state) => {
  const client = new IAMClient({});
  const policy = await findPolicy(NAMES.instancePolicyName);

  if (!policy) {
    state.deletePolicyError = new Error(
      `Policy ${NAMES.instancePolicyName} not found.`,
    );
  } else {
    return client.send(
      new DeletePolicyCommand({
        PolicyArn: policy.Arn,
      }),
    );
  }
}),
new ScenarioOutput("deletePolicyResult", (state) => {
  if (state.deletePolicyError) {
    console.error(state.deletePolicyError);
    return MESSAGES.deletePolicyError.replace(
      "${INSTANCE_POLICY_NAME}",
      NAMES.instancePolicyName,
    );
  } else {
    return MESSAGES.deletedPolicy.replace(
      "${INSTANCE_POLICY_NAME}",
      NAMES.instancePolicyName,
    );
  }
}),
new ScenarioAction("removeRoleFromInstanceProfile", async (state) => {
  try {
    const client = new IAMClient({});
    await client.send(
      new RemoveRoleFromInstanceProfileCommand({
        RoleName: NAMES.instanceRoleName,
```

```
          InstanceProfileName: NAMES.instanceProfileName,
        }),
      );
    } catch (e) {
      state.removeRoleFromInstanceProfileError = e;
    }
  }),
  new ScenarioOutput("removeRoleFromInstanceProfileResult", (state) => {
    if (state.removeRoleFromInstanceProfile) {
      console.error(state.removeRoleFromInstanceProfileError);
      return MESSAGES.removeRoleFromInstanceProfileError
        .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
        .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
    } else {
      return MESSAGES.removedRoleFromInstanceProfile
        .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
        .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
    }
  }),
  new ScenarioAction("deleteInstanceRole", async (state) => {
    try {
      const client = new IAMClient({});
      await client.send(
        new DeleteRoleCommand({
          RoleName: NAMES.instanceRoleName,
        }),
      );
    } catch (e) {
      state.deleteInstanceRoleError = e;
    }
  }),
  new ScenarioOutput("deleteInstanceRoleResult", (state) => {
    if (state.deleteInstanceRoleError) {
      console.error(state.deleteInstanceRoleError);
      return MESSAGES.deleteInstanceRoleError.replace(
        "${INSTANCE_ROLE_NAME}",
        NAMES.instanceRoleName,
      );
    } else {
      return MESSAGES.deletedInstanceRole.replace(
        "${INSTANCE_ROLE_NAME}",
        NAMES.instanceRoleName,
      );
    }
```

```
  }),
  new ScenarioAction("deleteInstanceProfile", async (state) => {
    try {
      const client = new IAMClient({});
      await client.send(
        new DeleteInstanceProfileCommand({
          InstanceProfileName: NAMES.instanceProfileName,
        }),
      );
    } catch (e) {
      state.deleteInstanceProfileError = e;
    }
  }),
  new ScenarioOutput("deleteInstanceProfileResult", (state) => {
    if (state.deleteInstanceProfileError) {
      console.error(state.deleteInstanceProfileError);
      return MESSAGES.deleteInstanceProfileError.replace(
        "${INSTANCE_PROFILE_NAME}",
        NAMES.instanceProfileName,
      );
    } else {
      return MESSAGES.deletedInstanceProfile.replace(
        "${INSTANCE_PROFILE_NAME}",
        NAMES.instanceProfileName,
      );
    }
  }),
  new ScenarioAction("deleteAutoScalingGroup", async (state) => {
    try {
      await terminateGroupInstances(NAMES.autoScalingGroupName);
      await retry({ intervalInMs: 60000, maxRetries: 60 }, async () => {
        await deleteAutoScalingGroup(NAMES.autoScalingGroupName);
      });
    } catch (e) {
      state.deleteAutoScalingGroupError = e;
    }
  }),
  new ScenarioOutput("deleteAutoScalingGroupResult", (state) => {
    if (state.deleteAutoScalingGroupError) {
      console.error(state.deleteAutoScalingGroupError);
      return MESSAGES.deleteAutoScalingGroupError.replace(
        "${AUTO_SCALING_GROUP_NAME}",
        NAMES.autoScalingGroupName,
      );
```

```
    } else {
      return MESSAGES.deletedAutoScalingGroup.replace(
        "${AUTO_SCALING_GROUP_NAME}",
        NAMES.autoScalingGroupName,
      );
    }
  }),
  new ScenarioAction("deleteLaunchTemplate", async (state) => {
    const client = new EC2Client({});
    try {
      await client.send(
        new DeleteLaunchTemplateCommand({
          LaunchTemplateName: NAMES.launchTemplateName,
        }),
      );
    } catch (e) {
      state.deleteLaunchTemplateError = e;
    }
  }),
  new ScenarioOutput("deleteLaunchTemplateResult", (state) => {
    if (state.deleteLaunchTemplateError) {
      console.error(state.deleteLaunchTemplateError);
      return MESSAGES.deleteLaunchTemplateError.replace(
        "${LAUNCH_TEMPLATE_NAME}",
        NAMES.launchTemplateName,
      );
    } else {
      return MESSAGES.deletedLaunchTemplate.replace(
        "${LAUNCH_TEMPLATE_NAME}",
        NAMES.launchTemplateName,
      );
    }
  }),
  new ScenarioAction("deleteLoadBalancer", async (state) => {
    try {
      const client = new ElasticLoadBalancingV2Client({});
      const loadBalancer = await findLoadBalancer(NAMES.loadBalancerName);
      await client.send(
        new DeleteLoadBalancerCommand({
          LoadBalancerArn: loadBalancer.LoadBalancerArn,
        }),
      );
      await retry({ intervalInMs: 1000, maxRetries: 60 }, async () => {
        const lb = await findLoadBalancer(NAMES.loadBalancerName);
```

```
      if (lb) {
        throw new Error("Load balancer still exists.");
      }
    });
  } catch (e) {
    state.deleteLoadBalancerError = e;
  }
}),
new ScenarioOutput("deleteLoadBalancerResult", (state) => {
  if (state.deleteLoadBalancerError) {
    console.error(state.deleteLoadBalancerError);
    return MESSAGES.deleteLoadBalancerError.replace(
      "${LB_NAME}",
      NAMES.loadBalancerName,
    );
  } else {
    return MESSAGES.deletedLoadBalancer.replace(
      "${LB_NAME}",
      NAMES.loadBalancerName,
    );
  }
}),
new ScenarioAction("deleteLoadBalancerTargetGroup", async (state) => {
  const client = new ElasticLoadBalancingV2Client({});
  try {
    const { TargetGroups } = await client.send(
      new DescribeTargetGroupsCommand({
        Names: [NAMES.loadBalancerTargetGroupName],
      }),
    );

    await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
      client.send(
        new DeleteTargetGroupCommand({
          TargetGroupArn: TargetGroups[0].TargetGroupArn,
        }),
      ),
    );
  } catch (e) {
    state.deleteLoadBalancerTargetGroupError = e;
  }
}),
new ScenarioOutput("deleteLoadBalancerTargetGroupResult", (state) => {
  if (state.deleteLoadBalancerTargetGroupError) {
```

```
      console.error(state.deleteLoadBalancerTargetGroupError);
      return MESSAGES.deleteLoadBalancerTargetGroupError.replace(
        "${TARGET_GROUP_NAME}",
        NAMES.loadBalancerTargetGroupName,
      );
    } else {
      return MESSAGES.deletedLoadBalancerTargetGroup.replace(
        "${TARGET_GROUP_NAME}",
        NAMES.loadBalancerTargetGroupName,
      );
    }
  }),
  new ScenarioAction("detachSsmOnlyRoleFromProfile", async (state) => {
    try {
      const client = new IAMClient({});
      await client.send(
        new RemoveRoleFromInstanceProfileCommand({
          InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
          RoleName: NAMES.ssmOnlyRoleName,
        }),
      );
    } catch (e) {
      state.detachSsmOnlyRoleFromProfileError = e;
    }
  }),
  new ScenarioOutput("detachSsmOnlyRoleFromProfileResult", (state) => {
    if (state.detachSsmOnlyRoleFromProfileError) {
      console.error(state.detachSsmOnlyRoleFromProfileError);
      return MESSAGES.detachSsmOnlyRoleFromProfileError
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
        .replace("${PROFILE_NAME}", NAMES.ssmOnlyInstanceProfileName);
    } else {
      return MESSAGES.detachedSsmOnlyRoleFromProfile
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
        .replace("${PROFILE_NAME}", NAMES.ssmOnlyInstanceProfileName);
    }
  }),
  new ScenarioAction("detachSsmOnlyCustomRolePolicy", async (state) => {
    try {
      const iamClient = new IAMClient({});
      const ssmOnlyPolicy = await findPolicy(NAMES.ssmOnlyPolicyName);
      await iamClient.send(
        new DetachRolePolicyCommand({
          RoleName: NAMES.ssmOnlyRoleName,
```

```
          PolicyArn: ssmOnlyPolicy.Arn,
        }),
      );
    } catch (e) {
      state.detachSsmOnlyCustomRolePolicyError = e;
    }
  }),
  new ScenarioOutput("detachSsmOnlyCustomRolePolicyResult", (state) => {
    if (state.detachSsmOnlyCustomRolePolicyError) {
      console.error(state.detachSsmOnlyCustomRolePolicyError);
      return MESSAGES.detachSsmOnlyCustomRolePolicyError
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
        .replace("${POLICY_NAME}", NAMES.ssmOnlyPolicyName);
    } else {
      return MESSAGES.detachedSsmOnlyCustomRolePolicy
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
        .replace("${POLICY_NAME}", NAMES.ssmOnlyPolicyName);
    }
  }),
  new ScenarioAction("detachSsmOnlyAWSRolePolicy", async (state) => {
    try {
      const iamClient = new IAMClient({});
      await iamClient.send(
        new DetachRolePolicyCommand({
          RoleName: NAMES.ssmOnlyRoleName,
          PolicyArn: "arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore",
        }),
      );
    } catch (e) {
      state.detachSsmOnlyAWSRolePolicyError = e;
    }
  }),
  new ScenarioOutput("detachSsmOnlyAWSRolePolicyResult", (state) => {
    if (state.detachSsmOnlyAWSRolePolicyError) {
      console.error(state.detachSsmOnlyAWSRolePolicyError);
      return MESSAGES.detachSsmOnlyAWSRolePolicyError
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
        .replace("${POLICY_NAME}", "AmazonSSMManagedInstanceCore");
    } else {
      return MESSAGES.detachedSsmOnlyAWSRolePolicy
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
        .replace("${POLICY_NAME}", "AmazonSSMManagedInstanceCore");
    }
  }),
```

```
  new ScenarioAction("deleteSsmOnlyInstanceProfile", async (state) => {
    try {
      const iamClient = new IAMClient({});
      await iamClient.send(
        new DeleteInstanceProfileCommand({
          InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
        }),
      );
    } catch (e) {
      state.deleteSsmOnlyInstanceProfileError = e;
    }
  }),
  new ScenarioOutput("deleteSsmOnlyInstanceProfileResult", (state) => {
    if (state.deleteSsmOnlyInstanceProfileError) {
      console.error(state.deleteSsmOnlyInstanceProfileError);
      return MESSAGES.deleteSsmOnlyInstanceProfileError.replace(
        "${INSTANCE_PROFILE_NAME}",
        NAMES.ssmOnlyInstanceProfileName,
      );
    } else {
      return MESSAGES.deletedSsmOnlyInstanceProfile.replace(
        "${INSTANCE_PROFILE_NAME}",
        NAMES.ssmOnlyInstanceProfileName,
      );
    }
  }),
  new ScenarioAction("deleteSsmOnlyPolicy", async (state) => {
    try {
      const iamClient = new IAMClient({});
      const ssmOnlyPolicy = await findPolicy(NAMES.ssmOnlyPolicyName);
      await iamClient.send(
        new DeletePolicyCommand({
          PolicyArn: ssmOnlyPolicy.Arn,
        }),
      );
    } catch (e) {
      state.deleteSsmOnlyPolicyError = e;
    }
  }),
  new ScenarioOutput("deleteSsmOnlyPolicyResult", (state) => {
    if (state.deleteSsmOnlyPolicyError) {
      console.error(state.deleteSsmOnlyPolicyError);
      return MESSAGES.deleteSsmOnlyPolicyError.replace(
        "${POLICY_NAME}",
```

```
        NAMES.ssmOnlyPolicyName,
      );
    } else {
      return MESSAGES.deletedSsmOnlyPolicy.replace(
        "${POLICY_NAME}",
        NAMES.ssmOnlyPolicyName,
      );
    }
  }),
  new ScenarioAction("deleteSsmOnlyRole", async (state) => {
    try {
      const iamClient = new IAMClient({});
      await iamClient.send(
        new DeleteRoleCommand({
          RoleName: NAMES.ssmOnlyRoleName,
        }),
      );
    } catch (e) {
      state.deleteSsmOnlyRoleError = e;
    }
  }),
  new ScenarioOutput("deleteSsmOnlyRoleResult", (state) => {
    if (state.deleteSsmOnlyRoleError) {
      console.error(state.deleteSsmOnlyRoleError);
      return MESSAGES.deleteSsmOnlyRoleError.replace(
        "${ROLE_NAME}",
        NAMES.ssmOnlyRoleName,
      );
    } else {
      return MESSAGES.deletedSsmOnlyRole.replace(
        "${ROLE_NAME}",
        NAMES.ssmOnlyRoleName,
      );
    }
  }),
  new ScenarioAction(
    "revokeSecurityGroupIngress",
    async (
      /** @type {{ myIp: string, defaultSecurityGroup: { GroupId: string } }} */
state,
    ) => {
      const ec2Client = new EC2Client({});

      try {
```

```
        await ec2Client.send(
          new RevokeSecurityGroupIngressCommand({
            GroupId: state.defaultSecurityGroup.GroupId,
            CidrIp: `${state.myIp}/32`,
            FromPort: 80,
            ToPort: 80,
            IpProtocol: "tcp",
          }),
        );
      } catch (e) {
        state.revokeSecurityGroupIngressError = e;
      }
    },
  ),
  new ScenarioOutput("revokeSecurityGroupIngressResult", (state) => {
    if (state.revokeSecurityGroupIngressError) {
      console.error(state.revokeSecurityGroupIngressError);
      return MESSAGES.revokeSecurityGroupIngressError.replace(
        "${IP}",
        state.myIp,
      );
    } else {
      return MESSAGES.revokedSecurityGroupIngress.replace("${IP}", state.myIp);
    }
  }),
];

/**
 * @param {string} policyName
 */
async function findPolicy(policyName) {
  const client = new IAMClient({});
  const paginatedPolicies = paginateListPolicies({ client }, {});
  for await (const page of paginatedPolicies) {
    const policy = page.Policies.find((p) => p.PolicyName === policyName);
    if (policy) {
      return policy;
    }
  }
}

/**
 * @param {string} groupName
 */
```

```
async function deleteAutoScalingGroup(groupName) {
  const client = new AutoScalingClient({});
  try {
    await client.send(
      new DeleteAutoScalingGroupCommand({
        AutoScalingGroupName: groupName,
      }),
    );
  } catch (err) {
    if (!(err instanceof Error)) {
      throw err;
    } else {
      console.log(err.name);
      throw err;
    }
  }
}

/**
 * @param {string} groupName
 */
async function terminateGroupInstances(groupName) {
  const autoScalingClient = new AutoScalingClient({});
  const group = await findAutoScalingGroup(groupName);
  await autoScalingClient.send(
    new UpdateAutoScalingGroupCommand({
      AutoScalingGroupName: group.AutoScalingGroupName,
      MinSize: 0,
    }),
  );
  for (const i of group.Instances) {
    await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
      autoScalingClient.send(
        new TerminateInstanceInAutoScalingGroupCommand({
          InstanceId: i.InstanceId,
          ShouldDecrementDesiredCapacity: true,
        }),
      ),
    );
  }
}

async function findAutoScalingGroup(groupName) {
  const client = new AutoScalingClient({});
```

```
  const paginatedGroups = paginateDescribeAutoScalingGroups({ client }, {});
  for await (const page of paginatedGroups) {
    const group = page.AutoScalingGroups.find(
      (g) => g.AutoScalingGroupName === groupName,
    );
    if (group) {
      return group;
    }
  }
  throw new Error(`Auto scaling group ${groupName} not found.`);
}
```

- 如需詳API細資訊，請參閱「AWS SDK for JavaScript API參考」中的下列主題。
  - AttachLoadBalancerTargetGroups
  - CreateAutoScalingGroup
  - CreateInstanceProfile
  - CreateLaunchTemplate
  - CreateListener
  - CreateLoadBalancer
  - CreateTargetGroup
  - DeleteAutoScalingGroup
  - DeleteInstanceProfile
  - DeleteLaunchTemplate
  - DeleteLoadBalancer
  - DeleteTargetGroup
  - DescribeAutoScalingGroups
  - DescribeAvailabilityZones
  - DescribeIamInstanceProfileAssociations
  - DescribeInstances
  - DescribeLoadBalancers
  - DescribeSubnets
  - DescribeTargetGroups
  - DescribeTargetHealth

- DescribeVpcs
- RebootInstances
- ReplaceIamInstanceProfileAssociation
- TerminateInstanceInAutoScalingGroup
- UpdateAutoScalingGroup

Amazon 基岩示例使用 SDK for JavaScript (v3)

下列程式碼範例說明如何透過使用 AWS SDK for JavaScript (v3) 搭配 Amazon 基岩來執行動作和實作常見案例。

Actions 是大型程式的程式碼摘錄，必須在內容中執行。雖然動作會顯示如何呼叫個別服務函數，但您可以在其相關案例中查看內容中的動作。

每個範例都包含完整原始程式碼的連結，您可以在其中找到如何在內容中設定和執行程式碼的指示。

開始使用

你好 Amazon 基岩

下列程式碼範例說明如何開始使用 Amazon 基岩。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import { fileURLToPath } from "url";

import {
  BedrockClient,
  ListFoundationModelsCommand,
} from "@aws-sdk/client-bedrock";

const REGION = "us-east-1";
const client = new BedrockClient({ region: REGION });
```

```
export const main = async () => {
  const command = new ListFoundationModelsCommand({});

  const response = await client.send(command);
  const models = response.modelSummaries;

  console.log("Listing the available Bedrock foundation models:");

  for (let model of models) {
    console.log("=".repeat(42));
    console.log(` Model: ${model.modelId}`);
    console.log("-".repeat(42));
    console.log(` Name: ${model.modelName}`);
    console.log(` Provider: ${model.providerName}`);
    console.log(` Model ARN: ${model.modelArn}`);
    console.log(` Input modalities: ${model.inputModalities}`);
    console.log(` Output modalities: ${model.outputModalities}`);
    console.log(` Supported customizations: ${model.customizationsSupported}`);
    console.log(` Supported inference types: ${model.inferenceTypesSupported}`);
    console.log(` Lifecycle status: ${model.modelLifecycle.status}`);
    console.log("=".repeat(42) + "\n");
  }

  const active = models.filter(
    (m) => m.modelLifecycle.status === "ACTIVE",
  ).length;
  const legacy = models.filter(
    (m) => m.modelLifecycle.status === "LEGACY",
  ).length;

  console.log(
    `There are ${active} active and ${legacy} legacy foundation models in
 ${REGION}.`,
  );

  return response;
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  await main();
}
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考ListFoundationModels中的。

主題

- 動作

動作

`GetFoundationModel`

下列程式碼範例會示範如何使用`GetFoundationModel`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

取得有關基礎模型的詳細資訊。

```
import { fileURLToPath } from "url";

import {
  BedrockClient,
  GetFoundationModelCommand,
} from "@aws-sdk/client-bedrock";

/**
 * Get details about an Amazon Bedrock foundation model.
 *
 * @return {FoundationModelDetails} - The list of available bedrock foundation
 models.
 */
export const getFoundationModel = async () => {
  const client = new BedrockClient();
```

```
  const command = new GetFoundationModelCommand({
    modelIdentifier: "amazon.titan-embed-text-v1",
  });

  const response = await client.send(command);

  return response.modelDetails;
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  const model = await getFoundationModel();
  console.log(model);
}
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考GetFoundationModel中的。

`ListFoundationModels`

下列程式碼範例會示範如何使用`ListFoundationModels`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

列出可用的基礎模型。

```
import { fileURLToPath } from "url";

import {
  BedrockClient,
  ListFoundationModelsCommand,
} from "@aws-sdk/client-bedrock";

/**
```

```
 * List the available Amazon Bedrock foundation models.
 *
 * @return {FoundationModelSummary[]} - The list of available bedrock foundation
 models.
 */
export const listFoundationModels = async () => {
  const client = new BedrockClient();

  const input = {
    // byProvider: 'STRING_VALUE',
    // byCustomizationType: 'FINE_TUNING' || 'CONTINUED_PRE_TRAINING',
    // byOutputModality: 'TEXT' || 'IMAGE' || 'EMBEDDING',
    // byInferenceType: 'ON_DEMAND' || 'PROVISIONED',
  };

  const command = new ListFoundationModelsCommand(input);

  const response = await client.send(command);

  return response.modelSummaries;
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  const models = await listFoundationModels();
  console.log(models);
}
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考ListFoundationModels中的。

Amazon 基岩運行時示例使用 to JavaScript (SDKv3)

下列程式碼範例說明如何透過使用 AWS SDK for JavaScript (v3) 搭配 Amazon 基岩執行階段來執行動作和實作常見案例。

案例是程式碼範例，向您展示如何透過呼叫服務中的多個函式或與其他函式結合來完成特定工作 AWS 服務。

每個範例都包含完整原始程式碼的連結，您可以在其中找到如何在內容中設定和執行程式碼的指示。

開始使用

你好 Amazon 基岩

下列程式碼範例說明如何開始使用 Amazon 基岩。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
/**
 * @typedef {Object} Content
 * @property {string} text
 *
 * @typedef {Object} Usage
 * @property {number} input_tokens
 * @property {number} output_tokens
 *
 * @typedef {Object} ResponseBody
 * @property {Content[]} content
 * @property {Usage} usage
 */

import { fileURLToPath } from "url";
import {
  BedrockRuntimeClient,
  InvokeModelCommand,
} from "@aws-sdk/client-bedrock-runtime";

const AWS_REGION = "us-east-1";

const MODEL_ID = "anthropic.claude-3-haiku-20240307-v1:0";
const PROMPT = "Hi. In a short paragraph, explain what you can do.";

const hello = async () => {
  console.log("=".repeat(35));
  console.log("Welcome to the Amazon Bedrock demo!");
  console.log("=".repeat(35));

  console.log("Model: Anthropic Claude 3 Haiku");
```

```
  console.log(`Prompt: ${PROMPT}\n`);
  console.log("Invoking model...\n");

  // Create a new Bedrock Runtime client instance.
  const client = new BedrockRuntimeClient({ region: AWS_REGION });

  // Prepare the payload for the model.
  const payload = {
    anthropic_version: "bedrock-2023-05-31",
    max_tokens: 1000,
    messages: [{ role: "user", content: [{ type: "text", text: PROMPT }] }],
  };

  // Invoke Claude with the payload and wait for the response.
  const apiResponse = await client.send(
    new InvokeModelCommand({
      contentType: "application/json",
      body: JSON.stringify(payload),
      modelId: MODEL_ID,
    }),
  );

  // Decode and return the response(s)
  const decodedResponseBody = new TextDecoder().decode(apiResponse.body);
  /** @type {ResponseBody} */
  const responseBody = JSON.parse(decodedResponseBody);
  const responses = responseBody.content;

  if (responses.length === 1) {
    console.log(`Response: ${responses[0].text}`);
  } else {
    console.log("Haiku returned multiple responses:");
    console.log(responses);
  }

  console.log(`\nNumber of input tokens:   ${responseBody.usage.input_tokens}`);
  console.log(`Number of output tokens: ${responseBody.usage.output_tokens}`);
};

if (process.argv[1] === fileURLToPath(import.meta.url)) {
  await hello();
}
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考InvokeModel中的。

主題

- 案例
- AI21侏羅西克實驗室 -2
- Amazon 泰坦文本
- Anthropic Claude
- Cohere Command
- 美洲駝
- 米斯特拉尔 AI

案例

在 Amazon 基岩上調用多個基礎模型

下列程式碼範例示範如何準備並傳送提示至 Amazon 基岩上的各種大語言模型 (LLMs)

SDK對於 JavaScript （ 3 ）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import { fileURLToPath } from "url";
import {
  Scenario,
  ScenarioAction,
  ScenarioInput,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import { FoundationModels } from "../config/foundation_models.js";

/**
 * @typedef {Object} ModelConfig
 * @property {Function} module
```

```
 * @property {Function} invoker
 * @property {string} modelId
 * @property {string} modelName
 */

const greeting = new ScenarioOutput(
  "greeting",
  "Welcome to the Amazon Bedrock Runtime client demo!",
  { header: true },
);

const selectModel = new ScenarioInput("model", "First, select a model:", {
  type: "select",
  choices: Object.values(FoundationModels).map((model) => ({
    name: model.modelName,
    value: model,
  })),
});

const enterPrompt = new ScenarioInput("prompt", "Now, enter your prompt:", {
  type: "input",
});

const printDetails = new ScenarioOutput(
  "print details",
  /**
   * @param {{ model: ModelConfig, prompt: string }} c
   */
  (c) => console.log(`Invoking ${c.model.modelName} with '${c.prompt}'...`),
  { slow: false },
);

const invokeModel = new ScenarioAction(
  "invoke model",
  /**
   * @param {{ model: ModelConfig, prompt: string, response: string }} c
   */
  async (c) => {
    const modelModule = await c.model.module();
    const invoker = c.model.invoker(modelModule);
    c.response = await invoker(c.prompt, c.model.modelId);
  },
);
```

```
const printResponse = new ScenarioOutput(
  "print response",
  /**
   * @param {{ response: string }} c
   */
  (c) => c.response,
  { slow: false },
);

const scenario = new Scenario("Amazon Bedrock Runtime Demo", [
  greeting,
  selectModel,
  enterPrompt,
  printDetails,
  invokeModel,
  printResponse,
]);

if (process.argv[1] === fileURLToPath(import.meta.url)) {
  scenario.run();
}
```

- 如需詳API細資訊，請參閱「AWS SDK for JavaScript API參考」中的下列主題。
  - InvokeModel
  - InvokeModelWithResponseStream

AI21侏羅西克實驗室 -2

匡威

下面的代碼示例演示了如何發送文本消息到AI21實驗室 Jurassic-2，使用基岩的匡威。API

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

發送短信到AI21實驗室 Jurassic-2，使用基岩的匡威。API

```
// Use the Conversation API to send a text message to AI21 Labs Jurassic-2.

import {
  BedrockRuntimeClient,
  ConverseCommand,
} from "@aws-sdk/client-bedrock-runtime";

// Create a Bedrock Runtime client in the AWS Region you want to use.
const client = new BedrockRuntimeClient({ region: "us-east-1" });

// Set the model ID, e.g., Jurassic-2 Mid.
const modelId = "ai21.j2-mid-v1";

// Start a conversation with the user message.
const userMessage =
  "Describe the purpose of a 'hello world' program in one line.";
const conversation = [
  {
    role: "user",
    content: [{ text: userMessage }],
  },
];

// Create a command with the model ID, the message, and a basic configuration.
const command = new ConverseCommand({
  modelId,
  messages: conversation,
  inferenceConfig: { maxTokens: 512, temperature: 0.5, topP: 0.9 },
});

try {
  // Send the command to the model and wait for the response
  const response = await client.send(command);

  // Extract and print the response text.
  const responseText = response.output.message.content[0].text;
  console.log(responseText);
} catch (err) {
  console.log(`ERROR: Can't invoke '${modelId}'. Reason: ${err}`);
  process.exit(1);
}
```

- 有API關詳細信息，請參閱AWS SDK for JavaScript API參考文獻中的匡威。

InvokeModel

下面的代碼示例演示了如何發送文本消息到AI21實驗室 Jurassic-2，使用調用模型。API

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

使用調用模型API發送文本消息。

```
import { fileURLToPath } from "url";

import { FoundationModels } from "../../config/foundation_models.js";
import {
  BedrockRuntimeClient,
  InvokeModelCommand,
} from "@aws-sdk/client-bedrock-runtime";

/**
 * @typedef {Object} Data
 * @property {string} text
 *
 * @typedef {Object} Completion
 * @property {Data} data
 *
 * @typedef {Object} ResponseBody
 * @property {Completion[]} completions
 */

/**
 * Invokes an AI21 Labs Jurassic-2 model.
 *
 * @param {string} prompt - The input text prompt for the model to complete.
```

```
 * @param {string} [modelId] - The ID of the model to use. Defaults to "ai21.j2-mid-
v1".
 */
export const invokeModel = async (prompt, modelId = "ai21.j2-mid-v1") => {
  // Create a new Bedrock Runtime client instance.
  const client = new BedrockRuntimeClient({ region: "us-east-1" });

  // Prepare the payload for the model.
  const payload = {
    prompt,
    maxTokens: 500,
    temperature: 0.5,
  };

  // Invoke the model with the payload and wait for the response.
  const command = new InvokeModelCommand({
    contentType: "application/json",
    body: JSON.stringify(payload),
    modelId,
  });
  const apiResponse = await client.send(command);

  // Decode and return the response(s).
  const decodedResponseBody = new TextDecoder().decode(apiResponse.body);
  /** @type {ResponseBody} */
  const responseBody = JSON.parse(decodedResponseBody);
  return responseBody.completions[0].data.text;
};

// Invoke the function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  const prompt =
    'Complete the following in one sentence: "Once upon a time..."';
  const modelId = FoundationModels.JURASSIC2_MID.modelId;
  console.log(`Prompt: ${prompt}`);
  console.log(`Model ID: ${modelId}`);

  try {
    console.log("-".repeat(53));
    const response = await invokeModel(prompt, modelId);
    console.log(response);
  } catch (err) {
    console.log(err);
  }
```

```
}
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考InvokeModel中的。

Amazon 泰坦文本

匡威

下面的代碼示例演示了如何使用基岩的匡威發送文本消息到 Amazon Titan 文本。API

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

發送短信 Amazon 泰坦文本，使用基岩的匡威。API

```
// Use the Conversation API to send a text message to Amazon Titan Text.

import {
  BedrockRuntimeClient,
  ConverseCommand,
} from "@aws-sdk/client-bedrock-runtime";

// Create a Bedrock Runtime client in the AWS Region you want to use.
const client = new BedrockRuntimeClient({ region: "us-east-1" });

// Set the model ID, e.g., Titan Text Premier.
const modelId = "amazon.titan-text-premier-v1:0";

// Start a conversation with the user message.
const userMessage =
  "Describe the purpose of a 'hello world' program in one line.";
const conversation = [
  {
    role: "user",
    content: [{ text: userMessage }],
  },
```

```
];

// Create a command with the model ID, the message, and a basic configuration.
const command = new ConverseCommand({
  modelId,
  messages: conversation,
  inferenceConfig: { maxTokens: 512, temperature: 0.5, topP: 0.9 },
});

try {
  // Send the command to the model and wait for the response
  const response = await client.send(command);

  // Extract and print the response text.
  const responseText = response.output.message.content[0].text;
  console.log(responseText);
} catch (err) {
  console.log(`ERROR: Can't invoke '${modelId}'. Reason: ${err}`);
  process.exit(1);
}
```

- 有API關詳細信息，請參閱AWS SDK for JavaScript API參考文獻中的匡威。

ConverseStream

下列程式碼範例示範如何使用基岩的交談傳送文字訊息至 Amazon Titan Titan Text，API並即時處理回應串流。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

使用基岩的交談功能傳送文字訊息至 Amazon Titan Titan 文字，API並即時處理回應串流。

```
// Use the Conversation API to send a text message to Amazon Titan Text.
```

```
import {
  BedrockRuntimeClient,
  ConverseStreamCommand,
} from "@aws-sdk/client-bedrock-runtime";

// Create a Bedrock Runtime client in the AWS Region you want to use.
const client = new BedrockRuntimeClient({ region: "us-east-1" });

// Set the model ID, e.g., Titan Text Premier.
const modelId = "amazon.titan-text-premier-v1:0";

// Start a conversation with the user message.
const userMessage =
  "Describe the purpose of a 'hello world' program in one line.";
const conversation = [
  {
    role: "user",
    content: [{ text: userMessage }],
  },
];

// Create a command with the model ID, the message, and a basic configuration.
const command = new ConverseStreamCommand({
  modelId,
  messages: conversation,
  inferenceConfig: { maxTokens: 512, temperature: 0.5, topP: 0.9 },
});

try {
  // Send the command to the model and wait for the response
  const response = await client.send(command);

  // Extract and print the streamed response text in real-time.
  for await (const item of response.stream) {
    if (item.contentBlockDelta) {
      process.stdout.write(item.contentBlockDelta.delta?.text);
    }
  }
} catch (err) {
  console.log(`ERROR: Can't invoke '${modelId}'. Reason: ${err}`);
  process.exit(1);
}
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考ConverseStream中的。

InvokeModel

下列程式碼範例顯示如何使用叫用模型，將文字訊息傳送至 Amazon Titan 文字API。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

使用調用模型API發送文本消息。

```
import { fileURLToPath } from "url";

import { FoundationModels } from "../../config/foundation_models.js";
import {
  BedrockRuntimeClient,
  InvokeModelCommand,
} from "@aws-sdk/client-bedrock-runtime";

/**
 * @typedef {Object} ResponseBody
 * @property {Object[]} results
 */

/**
 * Invokes an Amazon Titan Text generation model.
 *
 * @param {string} prompt - The input text prompt for the model to complete.
 * @param {string} [modelId] - The ID of the model to use. Defaults to
 "amazon.titan-text-express-v1".
 */
export const invokeModel = async (
  prompt,
  modelId = "amazon.titan-text-express-v1",
```

```
) => {
  // Create a new Bedrock Runtime client instance.
  const client = new BedrockRuntimeClient({ region: "us-east-1" });

  // Prepare the payload for the model.
  const payload = {
    inputText: prompt,
    textGenerationConfig: {
      maxTokenCount: 4096,
      stopSequences: [],
      temperature: 0,
      topP: 1,
    },
  };

  // Invoke the model with the payload and wait for the response.
  const command = new InvokeModelCommand({
    contentType: "application/json",
    body: JSON.stringify(payload),
    modelId,
  });
  const apiResponse = await client.send(command);

  // Decode and return the response.
  const decodedResponseBody = new TextDecoder().decode(apiResponse.body);
  /** @type {ResponseBody} */
  const responseBody = JSON.parse(decodedResponseBody);
  return responseBody.results[0].outputText;
};

// Invoke the function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  const prompt =
    'Complete the following in one sentence: "Once upon a time..."';
  const modelId = FoundationModels.TITAN_TEXT_G1_EXPRESS.modelId;
  console.log(`Prompt: ${prompt}`);
  console.log(`Model ID: ${modelId}`);

  try {
    console.log("-".repeat(53));
    const response = await invokeModel(prompt, modelId);
    console.log(response);
  } catch (err) {
    console.log(err);
```

```
  }
}
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考InvokeModel中的。

Anthropic Claude

匡威

下面的代碼示例演示了如何使用基岩的匡威發送文本消息人性克勞德。API

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

發送短信給人類克勞德，使用基岩的匡威。API

```
// Use the Conversation API to send a text message to Anthropic Claude.

import {
  BedrockRuntimeClient,
  ConverseCommand,
} from "@aws-sdk/client-bedrock-runtime";

// Create a Bedrock Runtime client in the AWS Region you want to use.
const client = new BedrockRuntimeClient({ region: "us-east-1" });

// Set the model ID, e.g., Claude 3 Haiku.
const modelId = "anthropic.claude-3-haiku-20240307-v1:0";

// Start a conversation with the user message.
const userMessage =
  "Describe the purpose of a 'hello world' program in one line.";
const conversation = [
  {
    role: "user",
    content: [{ text: userMessage }],
```

```
  },
];

// Create a command with the model ID, the message, and a basic configuration.
const command = new ConverseCommand({
  modelId,
  messages: conversation,
  inferenceConfig: { maxTokens: 512, temperature: 0.5, topP: 0.9 },
});

try {
  // Send the command to the model and wait for the response
  const response = await client.send(command);

  // Extract and print the response text.
  const responseText = response.output.message.content[0].text;
  console.log(responseText);
} catch (err) {
  console.log(`ERROR: Can't invoke '${modelId}'. Reason: ${err}`);
  process.exit(1);
}
```

- 有API關詳細信息，請參閱AWS SDK for JavaScript API參考文獻中的匡威。

ConverseStream

下面的代碼示例演示了如何使用基岩的匡威發送文本消息給 Eurpic 克勞德，API並實時處理響應流。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

發送文本消息給人為克勞德，使用基岩的匡威API和處理實時響應流。

```
// Use the Conversation API to send a text message to Anthropic Claude.
```

```
import {
  BedrockRuntimeClient,
  ConverseStreamCommand,
} from "@aws-sdk/client-bedrock-runtime";

// Create a Bedrock Runtime client in the AWS Region you want to use.
const client = new BedrockRuntimeClient({ region: "us-east-1" });

// Set the model ID, e.g., Claude 3 Haiku.
const modelId = "anthropic.claude-3-haiku-20240307-v1:0";

// Start a conversation with the user message.
const userMessage =
  "Describe the purpose of a 'hello world' program in one line.";
const conversation = [
  {
    role: "user",
    content: [{ text: userMessage }],
  },
];

// Create a command with the model ID, the message, and a basic configuration.
const command = new ConverseStreamCommand({
  modelId,
  messages: conversation,
  inferenceConfig: { maxTokens: 512, temperature: 0.5, topP: 0.9 },
});

try {
  // Send the command to the model and wait for the response
  const response = await client.send(command);

  // Extract and print the streamed response text in real-time.
  for await (const item of response.stream) {
    if (item.contentBlockDelta) {
      process.stdout.write(item.contentBlockDelta.delta?.text);
    }
  }
} catch (err) {
  console.log(`ERROR: Can't invoke '${modelId}'. Reason: ${err}`);
  process.exit(1);
}
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考ConverseStream中的。

InvokeModel

下列程式碼範例示範如何使用叫用模型，將文字訊息傳送至人性克勞德。API

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

使用調用模型API發送文本消息。

```
import { fileURLToPath } from "url";

import { FoundationModels } from "../../config/foundation_models.js";
import {
  BedrockRuntimeClient,
  InvokeModelCommand,
  InvokeModelWithResponseStreamCommand,
} from "@aws-sdk/client-bedrock-runtime";

/**
 * @typedef {Object} ResponseContent
 * @property {string} text
 *
 * @typedef {Object} MessagesResponseBody
 * @property {ResponseContent[]} content
 *
 * @typedef {Object} Delta
 * @property {string} text
 *
 * @typedef {Object} Message
 * @property {string} role
 *
 * @typedef {Object} Chunk
```

```
 * @property {string} type
 * @property {Delta} delta
 * @property {Message} message
 */

/**
 * Invokes Anthropic Claude 3 using the Messages API.
 *
 * To learn more about the Anthropic Messages API, go to:
 * https://docs.aws.amazon.com/bedrock/latest/userguide/model-parameters-anthropic-
claude-messages.html
 *
 * @param {string} prompt - The input text prompt for the model to complete.
 * @param {string} [modelId] - The ID of the model to use. Defaults to
 "anthropic.claude-3-haiku-20240307-v1:0".
 */
export const invokeModel = async (
  prompt,
  modelId = "anthropic.claude-3-haiku-20240307-v1:0",
) => {
  // Create a new Bedrock Runtime client instance.
  const client = new BedrockRuntimeClient({ region: "us-east-1" });

  // Prepare the payload for the model.
  const payload = {
    anthropic_version: "bedrock-2023-05-31",
    max_tokens: 1000,
    messages: [
      {
        role: "user",
        content: [{ type: "text", text: prompt }],
      },
    ],
  };

  // Invoke Claude with the payload and wait for the response.
  const command = new InvokeModelCommand({
    contentType: "application/json",
    body: JSON.stringify(payload),
    modelId,
  });
  const apiResponse = await client.send(command);

  // Decode and return the response(s)
```

```
  const decodedResponseBody = new TextDecoder().decode(apiResponse.body);
  /** @type {MessagesResponseBody} */
  const responseBody = JSON.parse(decodedResponseBody);
  return responseBody.content[0].text;
};

/**
 * Invokes Anthropic Claude 3 and processes the response stream.
 *
 * To learn more about the Anthropic Messages API, go to:
 * https://docs.aws.amazon.com/bedrock/latest/userguide/model-parameters-anthropic-
claude-messages.html
 *
 * @param {string} prompt - The input text prompt for the model to complete.
 * @param {string} [modelId] - The ID of the model to use. Defaults to
 "anthropic.claude-3-haiku-20240307-v1:0".
 */
export const invokeModelWithResponseStream = async (
  prompt,
  modelId = "anthropic.claude-3-haiku-20240307-v1:0",
) => {
  // Create a new Bedrock Runtime client instance.
  const client = new BedrockRuntimeClient({ region: "us-east-1" });

  // Prepare the payload for the model.
  const payload = {
    anthropic_version: "bedrock-2023-05-31",
    max_tokens: 1000,
    messages: [
      {
        role: "user",
        content: [{ type: "text", text: prompt }],
      },
    ],
  };

  // Invoke Claude with the payload and wait for the API to respond.
  const command = new InvokeModelWithResponseStreamCommand({
    contentType: "application/json",
    body: JSON.stringify(payload),
    modelId,
  });
  const apiResponse = await client.send(command);
```

```
  let completeMessage = "";

  // Decode and process the response stream
  for await (const item of apiResponse.body) {
    /** @type Chunk */
    const chunk = JSON.parse(new TextDecoder().decode(item.chunk.bytes));
    const chunk_type = chunk.type;

    if (chunk_type === "content_block_delta") {
      const text = chunk.delta.text;
      completeMessage = completeMessage + text;
      process.stdout.write(text);
    }
  }

  // Return the final response
  return completeMessage;
};

// Invoke the function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  const prompt = 'Write a paragraph starting with: "Once upon a time..."';
  const modelId = FoundationModels.CLAUDE_3_HAIKU.modelId;
  console.log(`Prompt: ${prompt}`);
  console.log(`Model ID: ${modelId}`);

  try {
    console.log("-".repeat(53));
    const response = await invokeModel(prompt, modelId);
    console.log("\n" + "-".repeat(53));
    console.log("Final structured response:");
    console.log(response);
  } catch (err) {
    console.log(`\n${err}`);
  }
}
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考InvokeModel中的。

InvokeModelWithResponseStream

下列程式碼範例示範如何使用 Invoke 模型，將文字訊息傳送至 Resortic Claude 模型API，以及列印回應資料流。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

使用調用模型API發送文本消息並實時處理響應流。

```
import { fileURLToPath } from "url";

import { FoundationModels } from "../../config/foundation_models.js";
import {
  BedrockRuntimeClient,
  InvokeModelCommand,
  InvokeModelWithResponseStreamCommand,
} from "@aws-sdk/client-bedrock-runtime";

/**
 * @typedef {Object} ResponseContent
 * @property {string} text
 *
 * @typedef {Object} MessagesResponseBody
 * @property {ResponseContent[]} content
 *
 * @typedef {Object} Delta
 * @property {string} text
 *
 * @typedef {Object} Message
 * @property {string} role
 *
 * @typedef {Object} Chunk
 * @property {string} type
 * @property {Delta} delta
 * @property {Message} message
 */
```

```
/**
 * Invokes Anthropic Claude 3 using the Messages API.
 *
 * To learn more about the Anthropic Messages API, go to:
 * https://docs.aws.amazon.com/bedrock/latest/userguide/model-parameters-anthropic-
claude-messages.html
 *
 * @param {string} prompt - The input text prompt for the model to complete.
 * @param {string} [modelId] - The ID of the model to use. Defaults to
 "anthropic.claude-3-haiku-20240307-v1:0".
 */
export const invokeModel = async (
  prompt,
  modelId = "anthropic.claude-3-haiku-20240307-v1:0",
) => {
  // Create a new Bedrock Runtime client instance.
  const client = new BedrockRuntimeClient({ region: "us-east-1" });

  // Prepare the payload for the model.
  const payload = {
    anthropic_version: "bedrock-2023-05-31",
    max_tokens: 1000,
    messages: [
      {
        role: "user",
        content: [{ type: "text", text: prompt }],
      },
    ],
  };

  // Invoke Claude with the payload and wait for the response.
  const command = new InvokeModelCommand({
    contentType: "application/json",
    body: JSON.stringify(payload),
    modelId,
  });
  const apiResponse = await client.send(command);

  // Decode and return the response(s)
  const decodedResponseBody = new TextDecoder().decode(apiResponse.body);
  /** @type {MessagesResponseBody} */
  const responseBody = JSON.parse(decodedResponseBody);
  return responseBody.content[0].text;
```

```
};

/**
 * Invokes Anthropic Claude 3 and processes the response stream.
 *
 * To learn more about the Anthropic Messages API, go to:
 * https://docs.aws.amazon.com/bedrock/latest/userguide/model-parameters-anthropic-
claude-messages.html
 *
 * @param {string} prompt - The input text prompt for the model to complete.
 * @param {string} [modelId] - The ID of the model to use. Defaults to
 "anthropic.claude-3-haiku-20240307-v1:0".
 */
export const invokeModelWithResponseStream = async (
  prompt,
  modelId = "anthropic.claude-3-haiku-20240307-v1:0",
) => {
  // Create a new Bedrock Runtime client instance.
  const client = new BedrockRuntimeClient({ region: "us-east-1" });

  // Prepare the payload for the model.
  const payload = {
    anthropic_version: "bedrock-2023-05-31",
    max_tokens: 1000,
    messages: [
      {
        role: "user",
        content: [{ type: "text", text: prompt }],
      },
    ],
  };

  // Invoke Claude with the payload and wait for the API to respond.
  const command = new InvokeModelWithResponseStreamCommand({
    contentType: "application/json",
    body: JSON.stringify(payload),
    modelId,
  });
  const apiResponse = await client.send(command);

  let completeMessage = "";

  // Decode and process the response stream
  for await (const item of apiResponse.body) {
```

```
    /** @type Chunk */
    const chunk = JSON.parse(new TextDecoder().decode(item.chunk.bytes));
    const chunk_type = chunk.type;

    if (chunk_type === "content_block_delta") {
      const text = chunk.delta.text;
      completeMessage = completeMessage + text;
      process.stdout.write(text);
    }
  }

  // Return the final response
  return completeMessage;
};

// Invoke the function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  const prompt = 'Write a paragraph starting with: "Once upon a time..."';
  const modelId = FoundationModels.CLAUDE_3_HAIKU.modelId;
  console.log(`Prompt: ${prompt}`);
  console.log(`Model ID: ${modelId}`);

  try {
    console.log("-".repeat(53));
    const response = await invokeModel(prompt, modelId);
    console.log("\n" + "-".repeat(53));
    console.log("Final structured response:");
    console.log(response);
  } catch (err) {
    console.log(`\n${err}`);
  }
}
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考InvokeModelWithResponseStream中的。

Cohere Command

匡威

下面的代碼示例演示了如何發送文本消息 Cohere 命令，使用基岩的匡威。API

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

發送短信給 Cohere 命令，使用基岩的匡威。API

```
// Use the Conversation API to send a text message to Cohere Command.

import {
  BedrockRuntimeClient,
  ConverseCommand,
} from "@aws-sdk/client-bedrock-runtime";

// Create a Bedrock Runtime client in the AWS Region you want to use.
const client = new BedrockRuntimeClient({ region: "us-east-1" });

// Set the model ID, e.g., Command R.
const modelId = "cohere.command-r-v1:0";

// Start a conversation with the user message.
const userMessage =
  "Describe the purpose of a 'hello world' program in one line.";
const conversation = [
  {
    role: "user",
    content: [{ text: userMessage }],
  },
];

// Create a command with the model ID, the message, and a basic configuration.
const command = new ConverseCommand({
  modelId,
  messages: conversation,
  inferenceConfig: { maxTokens: 512, temperature: 0.5, topP: 0.9 },
});

try {
  // Send the command to the model and wait for the response
  const response = await client.send(command);
```

```
  // Extract and print the response text.
  const responseText = response.output.message.content[0].text;
  console.log(responseText);
} catch (err) {
  console.log(`ERROR: Can't invoke '${modelId}'. Reason: ${err}`);
  process.exit(1);
}
```

- 有API關詳細信息，請參閱AWS SDK for JavaScript API參考文獻中的匡威。

ConverseStream

下面的代碼示例演示了如何使用基岩的匡威發送文本消息到 Cohere 命令，API並實時處理響應流。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

使用基岩的反向傳送文字訊息至 Cohere 命令，API並即時處理回應串流。

```
// Use the Conversation API to send a text message to Cohere Command.

import {
  BedrockRuntimeClient,
  ConverseStreamCommand,
} from "@aws-sdk/client-bedrock-runtime";

// Create a Bedrock Runtime client in the AWS Region you want to use.
const client = new BedrockRuntimeClient({ region: "us-east-1" });

// Set the model ID, e.g., Command R.
const modelId = "cohere.command-r-v1:0";

// Start a conversation with the user message.
```

```
const userMessage =
  "Describe the purpose of a 'hello world' program in one line.";
const conversation = [
  {
    role: "user",
    content: [{ text: userMessage }],
  },
];

// Create a command with the model ID, the message, and a basic configuration.
const command = new ConverseStreamCommand({
  modelId,
  messages: conversation,
  inferenceConfig: { maxTokens: 512, temperature: 0.5, topP: 0.9 },
});

try {
  // Send the command to the model and wait for the response
  const response = await client.send(command);

  // Extract and print the streamed response text in real-time.
  for await (const item of response.stream) {
    if (item.contentBlockDelta) {
      process.stdout.write(item.contentBlockDelta.delta?.text);
    }
  }
} catch (err) {
  console.log(`ERROR: Can't invoke '${modelId}'. Reason: ${err}`);
  process.exit(1);
}
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考ConverseStream中的。

美洲駝

匡威

下面的代碼示例演示了如何使用基岩的匡威發送文本消息 Meta Lama。API

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

發送短信給元駱駝，使用基岩的匡威。API

```
// Use the Conversation API to send a text message to Meta Llama.

import {
  BedrockRuntimeClient,
  ConverseCommand,
} from "@aws-sdk/client-bedrock-runtime";

// Create a Bedrock Runtime client in the AWS Region you want to use.
const client = new BedrockRuntimeClient({ region: "us-east-1" });

// Set the model ID, e.g., Llama 3 8b Instruct.
const modelId = "meta.llama3-8b-instruct-v1:0";

// Start a conversation with the user message.
const userMessage =
  "Describe the purpose of a 'hello world' program in one line.";
const conversation = [
  {
    role: "user",
    content: [{ text: userMessage }],
  },
];

// Create a command with the model ID, the message, and a basic configuration.
const command = new ConverseCommand({
  modelId,
  messages: conversation,
  inferenceConfig: { maxTokens: 512, temperature: 0.5, topP: 0.9 },
});

try {
  // Send the command to the model and wait for the response
  const response = await client.send(command);
```

```
  // Extract and print the response text.
  const responseText = response.output.message.content[0].text;
  console.log(responseText);
} catch (err) {
  console.log(`ERROR: Can't invoke '${modelId}'. Reason: ${err}`);
  process.exit(1);
}
```

- 有API關詳細信息，請參閱AWS SDK for JavaScript API參考文獻中的匡威。

ConverseStream

下面的代碼示例演示了如何使用基岩的匡威發送文本消息到 Meta Lama API 並實時處理響應流。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

使用基岩的匡威發送短信給 Meta Lama，API並實時處理響應流。

```
// Use the Conversation API to send a text message to Meta Llama.

import {
  BedrockRuntimeClient,
  ConverseStreamCommand,
} from "@aws-sdk/client-bedrock-runtime";

// Create a Bedrock Runtime client in the AWS Region you want to use.
const client = new BedrockRuntimeClient({ region: "us-east-1" });

// Set the model ID, e.g., Llama 3 8b Instruct.
const modelId = "meta.llama3-8b-instruct-v1:0";

// Start a conversation with the user message.
```

```
const userMessage =
  "Describe the purpose of a 'hello world' program in one line.";
const conversation = [
  {
    role: "user",
    content: [{ text: userMessage }],
  },
];

// Create a command with the model ID, the message, and a basic configuration.
const command = new ConverseStreamCommand({
  modelId,
  messages: conversation,
  inferenceConfig: { maxTokens: 512, temperature: 0.5, topP: 0.9 },
});

try {
  // Send the command to the model and wait for the response
  const response = await client.send(command);

  // Extract and print the streamed response text in real-time.
  for await (const item of response.stream) {
    if (item.contentBlockDelta) {
      process.stdout.write(item.contentBlockDelta.delta?.text);
    }
  }
} catch (err) {
  console.log(`ERROR: Can't invoke '${modelId}'. Reason: ${err}`);
  process.exit(1);
}
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考ConverseStream中的。

InvokeModel: 美洲駝 2

下面的代碼示例演示了如何發送文本消息元駱駝 2，使用調用模型API。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

使用調用模型API發送文本消息。

```
// Send a prompt to Meta Llama 2 and print the response.

import {
  BedrockRuntimeClient,
  InvokeModelCommand,
} from "@aws-sdk/client-bedrock-runtime";

// Create a Bedrock Runtime client in the AWS Region of your choice.
const client = new BedrockRuntimeClient({ region: "us-west-2" });

// Set the model ID, e.g., Llama 2 Chat 13B.
const modelId = "meta.llama2-13b-chat-v1";

// Define the user message to send.
const userMessage =
  "Describe the purpose of a 'hello world' program in one sentence.";

// Embed the message in Llama 2's prompt format.
const prompt = `<s>[INST] ${userMessage} [/INST]`;

// Format the request payload using the model's native structure.
const request = {
  prompt,
  // Optional inference parameters:
  max_gen_len: 512,
  temperature: 0.5,
  top_p: 0.9,
};

// Encode and send the request.
const response = await client.send(
  new InvokeModelCommand({
    contentType: "application/json",
```

```
    body: JSON.stringify(request),
    modelId,
  }),
);

// Decode the native response body.
/** @type {{ generation: string }} */
const nativeResponse = JSON.parse(new TextDecoder().decode(response.body));

// Extract and print the generated text.
const responseText = nativeResponse.generation;
console.log(responseText);

// Learn more about the Llama 2 prompt format at:
// https://llama.meta.com/docs/model-cards-and-prompt-formats/meta-llama-2
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考InvokeModel中的。

InvokeModel: 美洲駝 3

下面的代碼示例演示了如何發送文本消息元駱駝 3，使用調用模型API。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

使用調用模型API發送文本消息。

```
// Send a prompt to Meta Llama 3 and print the response.

import {
  BedrockRuntimeClient,
  InvokeModelCommand,
} from "@aws-sdk/client-bedrock-runtime";

// Create a Bedrock Runtime client in the AWS Region of your choice.
```

```
const client = new BedrockRuntimeClient({ region: "us-west-2" });

// Set the model ID, e.g., Llama 3 8B Instruct.
const modelId = "meta.llama3-8b-instruct-v1:0";

// Define the user message to send.
const userMessage =
  "Describe the purpose of a 'hello world' program in one sentence.";

// Embed the message in Llama 3's prompt format.
const prompt = `
<|begin_of_text|>
<|start_header_id|>user<|end_header_id|>
${userMessage}
<|eot_id|>
<|start_header_id|>assistant<|end_header_id|>
`;

// Format the request payload using the model's native structure.
const request = {
  prompt,
  // Optional inference parameters:
  max_gen_len: 512,
  temperature: 0.5,
  top_p: 0.9,
};

// Encode and send the request.
const response = await client.send(
  new InvokeModelCommand({
    contentType: "application/json",
    body: JSON.stringify(request),
    modelId,
  }),
);

// Decode the native response body.
/** @type {{ generation: string }} */
const nativeResponse = JSON.parse(new TextDecoder().decode(response.body));

// Extract and print the generated text.
const responseText = nativeResponse.generation;
console.log(responseText);
```

```
// Learn more about the Llama 3 prompt format at:
// https://llama.meta.com/docs/model-cards-and-prompt-formats/meta-llama-3/#special-tokens-used-with-meta-llama-3
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考InvokeModel中的。

InvokeModelWithResponseStream: 美洲駝 2

下面的代碼示例演示了如何發送文本消息到 Meta Lama 2，使用調用模型API，並打印響應流。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

使用調用模型API發送文本消息並實時處理響應流。

```
// Send a prompt to Meta Llama 2 and print the response stream in real-time.

import {
  BedrockRuntimeClient,
  InvokeModelWithResponseStreamCommand,
} from "@aws-sdk/client-bedrock-runtime";

// Create a Bedrock Runtime client in the AWS Region of your choice.
const client = new BedrockRuntimeClient({ region: "us-west-2" });

// Set the model ID, e.g., Llama 2 Chat 13B.
const modelId = "meta.llama2-13b-chat-v1";

// Define the user message to send.
const userMessage =
  "Describe the purpose of a 'hello world' program in one sentence.";

// Embed the message in Llama 2's prompt format.
const prompt = `<s>[INST] ${userMessage} [/INST]`;
```

```
// Format the request payload using the model's native structure.
const request = {
  prompt,
  // Optional inference parameters:
  max_gen_len: 512,
  temperature: 0.5,
  top_p: 0.9,
};

// Encode and send the request.
const responseStream = await client.send(
  new InvokeModelWithResponseStreamCommand({
    contentType: "application/json",
    body: JSON.stringify(request),
    modelId,
  }),
);

// Extract and print the response stream in real-time.
for await (const event of responseStream.body) {
  /** @type {{ generation: string }} */
  const chunk = JSON.parse(new TextDecoder().decode(event.chunk.bytes));
  if (chunk.generation) {
    process.stdout.write(chunk.generation);
  }
}

// Learn more about the Llama 3 prompt format at:
// https://llama.meta.com/docs/model-cards-and-prompt-formats/meta-llama-3/#special-
tokens-used-with-meta-llama-3
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考InvokeModelWithResponseStream中的。

InvokeModelWithResponseStream: 美洲駝 3

下面的代碼示例演示了如何發送文本消息到 Meta Lama 3，使用調用模型API，並打印響應流。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

使用調用模型API發送文本消息並實時處理響應流。

```
// Send a prompt to Meta Llama 3 and print the response stream in real-time.

import {
  BedrockRuntimeClient,
  InvokeModelWithResponseStreamCommand,
} from "@aws-sdk/client-bedrock-runtime";

// Create a Bedrock Runtime client in the AWS Region of your choice.
const client = new BedrockRuntimeClient({ region: "us-west-2" });

// Set the model ID, e.g., Llama 3 8B Instruct.
const modelId = "meta.llama3-8b-instruct-v1:0";

// Define the user message to send.
const userMessage =
  "Describe the purpose of a 'hello world' program in one sentence.";

// Embed the message in Llama 3's prompt format.
const prompt = `
<|begin_of_text|>
<|start_header_id|>user<|end_header_id|>
${userMessage}
<|eot_id|>
<|start_header_id|>assistant<|end_header_id|>
`;

// Format the request payload using the model's native structure.
const request = {
  prompt,
  // Optional inference parameters:
  max_gen_len: 512,
  temperature: 0.5,
  top_p: 0.9,
```

```
};

// Encode and send the request.
const responseStream = await client.send(
  new InvokeModelWithResponseStreamCommand({
    contentType: "application/json",
    body: JSON.stringify(request),
    modelId,
  }),
);

// Extract and print the response stream in real-time.
for await (const event of responseStream.body) {
  /** @type {{ generation: string }} */
  const chunk = JSON.parse(new TextDecoder().decode(event.chunk.bytes));
  if (chunk.generation) {
    process.stdout.write(chunk.generation);
  }
}

// Learn more about the Llama 3 prompt format at:
// https://llama.meta.com/docs/model-cards-and-prompt-formats/meta-llama-3/#special-
tokens-used-with-meta-llama-3
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考InvokeModelWithResponseStream中的。

米斯特拉尔 AI

匡威

下面的代碼示例演示了如何發送短信到米斯特拉爾，使用基岩的匡威。API

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

發送短信到米斯特拉爾，使用基岩的匡威。API

```
// Use the Conversation API to send a text message to Mistral.

import {
  BedrockRuntimeClient,
  ConverseCommand,
} from "@aws-sdk/client-bedrock-runtime";

// Create a Bedrock Runtime client in the AWS Region you want to use.
const client = new BedrockRuntimeClient({ region: "us-east-1" });

// Set the model ID, e.g., Mistral Large.
const modelId = "mistral.mistral-large-2402-v1:0";

// Start a conversation with the user message.
const userMessage =
  "Describe the purpose of a 'hello world' program in one line.";
const conversation = [
  {
    role: "user",
    content: [{ text: userMessage }],
  },
];

// Create a command with the model ID, the message, and a basic configuration.
const command = new ConverseCommand({
  modelId,
  messages: conversation,
  inferenceConfig: { maxTokens: 512, temperature: 0.5, topP: 0.9 },
});

try {
  // Send the command to the model and wait for the response
  const response = await client.send(command);

  // Extract and print the response text.
  const responseText = response.output.message.content[0].text;
  console.log(responseText);
} catch (err) {
  console.log(`ERROR: Can't invoke '${modelId}'. Reason: ${err}`);
  process.exit(1);
}
```

- 有API關詳細信息，請參閱AWS SDK for JavaScript API參考文獻中的匡威。

ConverseStream

下面的代碼示例演示了如何發送文本消息到米斯特拉爾，使用基岩的匡威API和處理實時響應流。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

發送短信到米斯特拉爾，使用基岩的匡威API和處理實時響應流。

```
// Use the Conversation API to send a text message to Mistral.

import {
  BedrockRuntimeClient,
  ConverseStreamCommand,
} from "@aws-sdk/client-bedrock-runtime";

// Create a Bedrock Runtime client in the AWS Region you want to use.
const client = new BedrockRuntimeClient({ region: "us-east-1" });

// Set the model ID, e.g., Mistral Large.
const modelId = "mistral.mistral-large-2402-v1:0";

// Start a conversation with the user message.
const userMessage =
  "Describe the purpose of a 'hello world' program in one line.";
const conversation = [
  {
    role: "user",
    content: [{ text: userMessage }],
  },
];

// Create a command with the model ID, the message, and a basic configuration.
```

```
const command = new ConverseStreamCommand({
  modelId,
  messages: conversation,
  inferenceConfig: { maxTokens: 512, temperature: 0.5, topP: 0.9 },
});

try {
  // Send the command to the model and wait for the response
  const response = await client.send(command);

  // Extract and print the streamed response text in real-time.
  for await (const item of response.stream) {
    if (item.contentBlockDelta) {
      process.stdout.write(item.contentBlockDelta.delta?.text);
    }
  }
} catch (err) {
  console.log(`ERROR: Can't invoke '${modelId}'. Reason: ${err}`);
  process.exit(1);
}
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考ConverseStream中的。

InvokeModel

下面的代碼示例演示了如何發送文本消息到米斯特拉爾模型，使用調用模型API。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

使用調用模型API發送文本消息。

```
import { fileURLToPath } from "url";
```

```
import { FoundationModels } from "../../config/foundation_models.js";
import {
  BedrockRuntimeClient,
  InvokeModelCommand,
} from "@aws-sdk/client-bedrock-runtime";

/**
 * @typedef {Object} Output
 * @property {string} text
 *
 * @typedef {Object} ResponseBody
 * @property {Output[]} outputs
 */

/**
 * Invokes a Mistral 7B Instruct model.
 *
 * @param {string} prompt - The input text prompt for the model to complete.
 * @param {string} [modelId] - The ID of the model to use. Defaults to
 "mistral.mistral-7b-instruct-v0:2".
 */
export const invokeModel = async (
  prompt,
  modelId = "mistral.mistral-7b-instruct-v0:2",
) => {
  // Create a new Bedrock Runtime client instance.
  const client = new BedrockRuntimeClient({ region: "us-east-1" });

  // Mistral instruct models provide optimal results when embedding
  // the prompt into the following template:
  const instruction = `<s>[INST] ${prompt} [/INST]`;

  // Prepare the payload.
  const payload = {
    prompt: instruction,
    max_tokens: 500,
    temperature: 0.5,
  };

  // Invoke the model with the payload and wait for the response.
  const command = new InvokeModelCommand({
    contentType: "application/json",
    body: JSON.stringify(payload),
    modelId,
```

```
  });
  const apiResponse = await client.send(command);

  // Decode and return the response.
  const decodedResponseBody = new TextDecoder().decode(apiResponse.body);
  /** @type {ResponseBody} */
  const responseBody = JSON.parse(decodedResponseBody);
  return responseBody.outputs[0].text;
};

// Invoke the function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  const prompt =
    'Complete the following in one sentence: "Once upon a time..."';
  const modelId = FoundationModels.MISTRAL_7B.modelId;
  console.log(`Prompt: ${prompt}`);
  console.log(`Model ID: ${modelId}`);

  try {
    console.log("-".repeat(53));
    const response = await invokeModel(prompt, modelId);
    console.log(response);
  } catch (err) {
    console.log(err);
  }
}
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考InvokeModel中的。

Amazon 基岩範例的代理程式使SDK用 JavaScript (v3)

下列程式碼範例說明如何透過使用 AWS SDK for JavaScript (v3) 搭配 Amazon 基岩代理程式來執行動作和實作常見案例。

Actions 是大型程式的程式碼摘錄，必須在內容中執行。雖然動作會顯示如何呼叫個別服務函數，但您可以在其相關案例中查看內容中的動作。

每個範例都包含完整原始程式碼的連結，您可以在其中找到如何在內容中設定和執行程式碼的指示。

開始使用

Amazon 基岩你好代理商

下列程式碼範例顯示如何開始使用 Amazon 基岩代理程式。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import { fileURLToPath } from "url";

import {
  BedrockAgentClient,
  GetAgentCommand,
  paginateListAgents,
} from "@aws-sdk/client-bedrock-agent";

/**
 * @typedef {Object} AgentSummary
 */

/**
 * A simple scenario to demonstrate basic setup and interaction with the Bedrock
 Agents Client.
 *
 * This function first initializes the Amazon Bedrock Agents client for a specific
 region.
 * It then retrieves a list of existing agents using the streamlined paginator
 approach.
 * For each agent found, it retrieves detailed information using a command object.
 *
 * Demonstrates:
 * - Use of the Bedrock Agents client to initialize and communicate with the AWS
 service.
 * - Listing resources in a paginated response pattern.
 * - Accessing an individual resource using a command object.
 *
 * @returns {Promise<void>} A promise that resolves when the function has completed
 execution.
```

```
 */
export const main = async () => {
  const region = "us-east-1";

  console.log("=".repeat(68));

  console.log(`Initializing Amazon Bedrock Agents client for ${region}...`);
  const client = new BedrockAgentClient({ region });

  console.log(`Retrieving the list of existing agents...`);
  const paginatorConfig = { client };
  const pages = paginateListAgents(paginatorConfig, {});

  /** @type {AgentSummary[]} */
  const agentSummaries = [];
  for await (const page of pages) {
    agentSummaries.push(...page.agentSummaries);
  }

  console.log(`Found ${agentSummaries.length} agents in ${region}.`);

  if (agentSummaries.length > 0) {
    for (const agentSummary of agentSummaries) {
      const agentId = agentSummary.agentId;
      console.log("=".repeat(68));
      console.log(`Retrieving agent with ID: ${agentId}:`);
      console.log("-".repeat(68));

      const command = new GetAgentCommand({ agentId });
      const response = await client.send(command);
      const agent = response.agent;

      console.log(` Name: ${agent.agentName}`);
      console.log(` Status: ${agent.agentStatus}`);
      console.log(` ARN: ${agent.agentArn}`);
      console.log(` Foundation model: ${agent.foundationModel}`);
    }
  }
  console.log("=".repeat(68));
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  await main();
```

```
}
```

- 如需詳API細資訊，請參閱「AWS SDK for JavaScript API參考」中的下列主題。
  - GetAgent
  - ListAgents

主題

- 動作

動作

`CreateAgent`

下列程式碼範例會示範如何使用`CreateAgent`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

建立 代理程式。

```
import { fileURLToPath } from "url";
import { checkForPlaceholders } from "../lib/utils.js";

import {
  BedrockAgentClient,
  CreateAgentCommand,
} from "@aws-sdk/client-bedrock-agent";

/**
 * Creates an Amazon Bedrock Agent.
 *
 * @param {string} agentName - A name for the agent that you create.
```

```
 * @param {string} foundationModel - The foundation model to be used by the agent
 you create.
 * @param {string} agentResourceRoleArn - The ARN of the IAM role with permissions
 required by the agent.
 * @param {string} [region='us-east-1'] - The AWS region in use.
 * @returns {Promise<import("@aws-sdk/client-bedrock-agent").Agent>} An object
 containing details of the created agent.
 */
export const createAgent = async (
  agentName,
  foundationModel,
  agentResourceRoleArn,
  region = "us-east-1",
) => {
  const client = new BedrockAgentClient({ region });

  const command = new CreateAgentCommand({
    agentName,
    foundationModel,
    agentResourceRoleArn,
  });
  const response = await client.send(command);

  return response.agent;
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  // Replace the placeholders for agentName and accountId, and roleName with a
 unique name for the new agent,
  // the id of your AWS account, and the name of an existing execution role that the
 agent can use inside your account.
  // For foundationModel, specify the desired model. Ensure to remove the brackets
 '[]' before adding your data.

  // A string (max 100 chars) that can include letters, numbers, dashes '-', and
 underscores '_'.
  const agentName = "[your-bedrock-agent-name]";

  // Your AWS account id.
  const accountId = "[123456789012]";

  // The name of the agent's execution role. It must be prefixed by
 `AmazonBedrockExecutionRoleForAgents_`.
```

```
  const roleName = "[AmazonBedrockExecutionRoleForAgents_your-role-name]";

  // The ARN for the agent's execution role.
  // Follow the ARN format: 'arn:aws:iam::account-id:role/role-name'
  const roleArn = `arn:aws:iam::${accountId}:role/${roleName}`;

  // Specify the model for the agent. Change if a different model is preferred.
  const foundationModel = "anthropic.claude-v2";

  // Check for unresolved placeholders in agentName and roleArn.
  checkForPlaceholders([agentName, roleArn]);

  console.log(`Creating a new agent...`);

  const agent = await createAgent(agentName, foundationModel, roleArn);
  console.log(agent);
}
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考CreateAgent中的。

`DeleteAgent`

下列程式碼範例會示範如何使用`DeleteAgent`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

刪除代理程式。

```
import { fileURLToPath } from "url";
import { checkForPlaceholders } from "../lib/utils.js";

import {
  BedrockAgentClient,
```

```
  DeleteAgentCommand,
} from "@aws-sdk/client-bedrock-agent";

/**
 * Deletes an Amazon Bedrock Agent.
 *
 * @param {string} agentId - The unique identifier of the agent to delete.
 * @param {string} [region='us-east-1'] - The AWS region in use.
 * @returns {Promise<import("@aws-sdk/client-bedrock-
agent").DeleteAgentCommandOutput>} An object containing the agent id, the status,
 and some additional metadata.
 */
export const deleteAgent = (agentId, region = "us-east-1") => {
  const client = new BedrockAgentClient({ region });
  const command = new DeleteAgentCommand({ agentId });
  return client.send(command);
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  // Replace the placeholders for agentId with an existing agent's id.
  // Ensure to remove the brackets (`[]`) before adding your data.

  // The agentId must be an alphanumeric string with exactly 10 characters.
  const agentId = "[ABC123DE45]";

  // Check for unresolved placeholders in agentId.
  checkForPlaceholders([agentId]);

  console.log(`Deleting agent with ID ${agentId}...`);

  const response = await deleteAgent(agentId);
  console.log(response);
}
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考DeleteAgent中的。

GetAgent

下列程式碼範例會示範如何使用GetAgent。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

找個特工

```
import { fileURLToPath } from "url";
import { checkForPlaceholders } from "../lib/utils.js";

import {
  BedrockAgentClient,
  GetAgentCommand,
} from "@aws-sdk/client-bedrock-agent";

/**
 * Retrieves the details of an Amazon Bedrock Agent.
 *
 * @param {string} agentId - The unique identifier of the agent.
 * @param {string} [region='us-east-1'] - The AWS region in use.
 * @returns {Promise<import("@aws-sdk/client-bedrock-agent").Agent>} An object
 containing the agent details.
 */
export const getAgent = async (agentId, region = "us-east-1") => {
  const client = new BedrockAgentClient({ region });

  const command = new GetAgentCommand({ agentId });
  const response = await client.send(command);
  return response.agent;
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  // Replace the placeholders for agentId with an existing agent's id.
  // Ensure to remove the brackets '[]' before adding your data.

  // The agentId must be an alphanumeric string with exactly 10 characters.
  const agentId = "[ABC123DE45]";
```

```
  // Check for unresolved placeholders in agentId.
  checkForPlaceholders([agentId]);

  console.log(`Retrieving agent with ID ${agentId}...`);

  const agent = await getAgent(agentId);
  console.log(agent);
}
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考GetAgent中的。

`ListAgentActionGroups`

下列程式碼範例會示範如何使用`ListAgentActionGroups`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

列出代理程式的動作群組。

```
import { fileURLToPath } from "url";
import { checkForPlaceholders } from "../lib/utils.js";

import {
  BedrockAgentClient,
  ListAgentActionGroupsCommand,
  paginateListAgentActionGroups,
} from "@aws-sdk/client-bedrock-agent";

/**
 * Retrieves a list of Action Groups of an agent utilizing the paginator function.
 *
 * This function leverages a paginator, which abstracts the complexity of
 pagination, providing
 * a straightforward way to handle paginated results inside a `for await...of` loop.
```

```
 *
 * @param {string} agentId - The unique identifier of the agent.
 * @param {string} agentVersion - The version of the agent.
 * @param {string} [region='us-east-1'] - The AWS region in use.
 * @returns {Promise<ActionGroupSummary[]>} An array of action group summaries.
 */
export const listAgentActionGroupsWithPaginator = async (
  agentId,
  agentVersion,
  region = "us-east-1",
) => {
  const client = new BedrockAgentClient({ region });

  // Create a paginator configuration
  const paginatorConfig = {
    client,
    pageSize: 10, // optional, added for demonstration purposes
  };

  const params = { agentId, agentVersion };

  const pages = paginateListAgentActionGroups(paginatorConfig, params);

  // Paginate until there are no more results
  const actionGroupSummaries = [];
  for await (const page of pages) {
    actionGroupSummaries.push(...page.actionGroupSummaries);
  }

  return actionGroupSummaries;
};

/**
 * Retrieves a list of Action Groups of an agent utilizing the
 ListAgentActionGroupsCommand.
 *
 * This function demonstrates the manual approach, sending a command to the client
 and processing the response.
 * Pagination must manually be managed. For a simplified approach that abstracts
 away pagination logic, see
 * the `listAgentActionGroupsWithPaginator()` example below.
 *
 * @param {string} agentId - The unique identifier of the agent.
 * @param {string} agentVersion - The version of the agent.
```

```
 * @param {string} [region='us-east-1'] - The AWS region in use.
 * @returns {Promise<ActionGroupSummary[]>} An array of action group summaries.
 */
export const listAgentActionGroupsWithCommandObject = async (
  agentId,
  agentVersion,
  region = "us-east-1",
) => {
  const client = new BedrockAgentClient({ region });

  let nextToken;
  const actionGroupSummaries = [];
  do {
    const command = new ListAgentActionGroupsCommand({
      agentId,
      agentVersion,
      nextToken,
      maxResults: 10, // optional, added for demonstration purposes
    });

    /** @type {{actionGroupSummaries: ActionGroupSummary[], nextToken?: string}} */
    const response = await client.send(command);

    for (const actionGroup of response.actionGroupSummaries || []) {
      actionGroupSummaries.push(actionGroup);
    }

    nextToken = response.nextToken;
  } while (nextToken);

  return actionGroupSummaries;
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  // Replace the placeholders for agentId and agentVersion with an existing agent's
 id and version.
  // Ensure to remove the brackets '[]' before adding your data.

  // The agentId must be an alphanumeric string with exactly 10 characters.
  const agentId = "[ABC123DE45]";

  // A string either containing `DRAFT` or a number with 1-5 digits (e.g., '123' or
 'DRAFT').
```

```
  const agentVersion = "[DRAFT]";

  // Check for unresolved placeholders in agentId and agentVersion.
  checkForPlaceholders([agentId, agentVersion]);

  console.log("=".repeat(68));
  console.log(
    "Listing agent action groups using ListAgentActionGroupsCommand:",
  );

  for (const actionGroup of await listAgentActionGroupsWithCommandObject(
    agentId,
    agentVersion,
  )) {
    console.log(actionGroup);
  }

  console.log("=".repeat(68));
  console.log(
    "Listing agent action groups using the paginateListAgents function:",
  );
  for (const actionGroup of await listAgentActionGroupsWithPaginator(
    agentId,
    agentVersion,
  )) {
    console.log(actionGroup);
  }
}
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考ListAgentActionGroups中的。

`ListAgents`

下列程式碼範例會示範如何使用`ListAgents`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

列出屬於某個帳號的代理程式。

```
import { fileURLToPath } from "url";

import {
  BedrockAgentClient,
  ListAgentsCommand,
  paginateListAgents,
} from "@aws-sdk/client-bedrock-agent";

/**
 * Retrieves a list of available Amazon Bedrock agents utilizing the paginator
 function.
 *
 * This function leverages a paginator, which abstracts the complexity of
 pagination, providing
 * a straightforward way to handle paginated results inside a `for await...of` loop.
 *
 * @param {string} [region='us-east-1'] - The AWS region in use.
 * @returns {Promise<AgentSummary[]>} An array of agent summaries.
 */
export const listAgentsWithPaginator = async (region = "us-east-1") => {
  const client = new BedrockAgentClient({ region });

  const paginatorConfig = {
    client,
    pageSize: 10, // optional, added for demonstration purposes
  };

  const pages = paginateListAgents(paginatorConfig, {});

  // Paginate until there are no more results
  const agentSummaries = [];
  for await (const page of pages) {
    agentSummaries.push(...page.agentSummaries);
  }

  return agentSummaries;
};

/**
 * Retrieves a list of available Amazon Bedrock agents utilizing the
 ListAgentsCommand.
```

```
 *
 * This function demonstrates the manual approach, sending a command to the client
 and processing the response.
 * Pagination must manually be managed. For a simplified approach that abstracts
 away pagination logic, see
 * the `listAgentsWithPaginator()` example below.
 *
 * @param {string} [region='us-east-1'] - The AWS region in use.
 * @returns {Promise<AgentSummary[]>} An array of agent summaries.
 */
export const listAgentsWithCommandObject = async (region = "us-east-1") => {
  const client = new BedrockAgentClient({ region });

  let nextToken;
  const agentSummaries = [];
  do {
    const command = new ListAgentsCommand({
      nextToken,
      maxResults: 10, // optional, added for demonstration purposes
    });

    /** @type {{agentSummaries: AgentSummary[], nextToken?: string}} */
    const paginatedResponse = await client.send(command);

    agentSummaries.push(...(paginatedResponse.agentSummaries || []));

    nextToken = paginatedResponse.nextToken;
  } while (nextToken);

  return agentSummaries;
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  console.log("=".repeat(68));
  console.log("Listing agents using ListAgentsCommand:");
  for (const agent of await listAgentsWithCommandObject()) {
    console.log(agent);
  }

  console.log("=".repeat(68));
  console.log("Listing agents using the paginateListAgents function:");
  for (const agent of await listAgentsWithPaginator()) {
    console.log(agent);
```

```
  }
}
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考ListAgents中的。

Amazon 基岩運行時示例的代理程序使SDK用 in JavaScript (v3)

下列程式碼範例說明如何透過使用 AWS SDK for JavaScript (v3) 搭配 Amazon 基岩執行階段的代理程式來執行動作和實作常見案例。

Actions 是大型程式的程式碼摘錄，必須在內容中執行。雖然動作會顯示如何呼叫個別服務函數，但您可以在其相關案例中查看內容中的動作。

每個範例都包含完整原始程式碼的連結，您可以在其中找到如何在內容中設定和執行程式碼的指示。

主題

- 動作

動作

`InvokeAgent`

下列程式碼範例會示範如何使用`InvokeAgent`。

SDK對於 JavaScript （ 3 ）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import {
  BedrockAgentRuntimeClient,
  InvokeAgentCommand,
} from "@aws-sdk/client-bedrock-agent-runtime";

/**
```

```
 * @typedef {Object} ResponseBody
 * @property {string} completion
 */

/**
 * Invokes a Bedrock agent to run an inference using the input
 * provided in the request body.
 *
 * @param {string} prompt - The prompt that you want the Agent to complete.
 * @param {string} sessionId - An arbitrary identifier for the session.
 */
export const invokeBedrockAgent = async (prompt, sessionId) => {
  const client = new BedrockAgentRuntimeClient({ region: "us-east-1" });
  // const client = new BedrockAgentRuntimeClient({
  //   region: "us-east-1",
  //   credentials: {
  //     accessKeyId: "accessKeyId", // permission to invoke agent
  //     secretAccessKey: "accessKeySecret",
  //   },
  // });

  const agentId = "AJBHXXILZN";
  const agentAliasId = "AVKP1ITZAA";

  const command = new InvokeAgentCommand({
    agentId,
    agentAliasId,
    sessionId,
    inputText: prompt,
  });

  try {
    let completion = "";
    const response = await client.send(command);

    if (response.completion === undefined) {
      throw new Error("Completion is undefined");
    }

    for await (let chunkEvent of response.completion) {
      const chunk = chunkEvent.chunk;
      console.log(chunk);
      const decodedResponse = new TextDecoder("utf-8").decode(chunk.bytes);
      completion += decodedResponse;
```

```
    }

    return { sessionId: sessionId, completion };
  } catch (err) {
    console.error(err);
  }
};

// Call function if run directly
import { fileURLToPath } from "url";
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  const result = await invokeBedrockAgent("I need help.", "123");
  console.log(result);
}
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考InvokeAgent中的。

CloudWatch 使用 JavaScript （ v3 ）SDK的示例

下列程式碼範例會示範如何使用 AWS SDK for JavaScript (v3) 搭配使用來執行動作及實作常見案例 CloudWatch。

Actions 是大型程式的程式碼摘錄，必須在內容中執行。雖然動作會顯示如何呼叫個別服務函數，但您可以在其相關案例中查看內容中的動作。

每個範例都包含完整原始程式碼的連結，您可以在其中找到如何在內容中設定和執行程式碼的指示。

主題

- 動作

動作

`DeleteAlarms`

下列程式碼範例會示範如何使用`DeleteAlarms`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

匯入SDK和用戶端模組並呼叫API.

```
import { DeleteAlarmsCommand } from "@aws-sdk/client-cloudwatch";
import { client } from "../libs/client.js";

const run = async () => {
  const command = new DeleteAlarmsCommand({
    AlarmNames: [process.env.CLOUDWATCH_ALARM_NAME], // Set the value of
 CLOUDWATCH_ALARM_NAME to the name of an existing alarm.
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

在單獨的模組中建立用戶端並將其匯出。

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考DeleteAlarms中的。

SDK對於 JavaScript （第 2 個）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

匯入SDK和用戶端模組並呼叫API.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create CloudWatch service object
var cw = new AWS.CloudWatch({ apiVersion: "2010-08-01" });

var params = {
  AlarmNames: ["Web_Server_CPU_Utilization"],
};

cw.deleteAlarms(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考DeleteAlarms中的。

`DescribeAlarmsForMetric`

下列程式碼範例會示範如何使用`DescribeAlarmsForMetric`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

匯入SDK和用戶端模組並呼叫API.

```
import { DescribeAlarmsCommand } from "@aws-sdk/client-cloudwatch";
import { client } from "../libs/client.js";

const run = async () => {
  const command = new DescribeAlarmsCommand({
    AlarmNames: [process.env.CLOUDWATCH_ALARM_NAME], // Set the value of
 CLOUDWATCH_ALARM_NAME to the name of an existing alarm.
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

在單獨的模組中建立用戶端並將其匯出。

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考DescribeAlarmsForMetric中的。

SDK對於 JavaScript （第 2 個）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create CloudWatch service object
var cw = new AWS.CloudWatch({ apiVersion: "2010-08-01" });

cw.describeAlarms({ StateValue: "INSUFFICIENT_DATA" }, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    // List the names of all current alarms in the console
    data.MetricAlarms.forEach(function (item, index, array) {
      console.log(item.AlarmName);
    });
  }
});
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考DescribeAlarmsForMetric中的。

`DisableAlarmActions`

下列程式碼範例會示範如何使用`DisableAlarmActions`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

匯入SDK和用戶端模組並呼叫API.

```
import { DisableAlarmActionsCommand } from "@aws-sdk/client-cloudwatch";
import { client } from "../libs/client.js";

const run = async () => {
  const command = new DisableAlarmActionsCommand({
    AlarmNames: process.env.CLOUDWATCH_ALARM_NAME, // Set the value of
 CLOUDWATCH_ALARM_NAME to the name of an existing alarm.
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

在單獨的模組中建立用戶端並將其匯出。

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考DisableAlarmActions中的。

SDK對於 JavaScript （ 第 2 個 ）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

匯入SDK和用戶端模組並呼叫API.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create CloudWatch service object
var cw = new AWS.CloudWatch({ apiVersion: "2010-08-01" });

cw.disableAlarmActions(
  { AlarmNames: ["Web_Server_CPU_Utilization"] },
  function (err, data) {
    if (err) {
      console.log("Error", err);
    } else {
      console.log("Success", data);
    }
  }
);
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考DisableAlarmActions中的。

`EnableAlarmActions`

下列程式碼範例會示範如何使用`EnableAlarmActions`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

匯入SDK和用戶端模組並呼叫API.

```
import { EnableAlarmActionsCommand } from "@aws-sdk/client-cloudwatch";
import { client } from "../libs/client.js";

const run = async () => {
  const command = new EnableAlarmActionsCommand({
    AlarmNames: [process.env.CLOUDWATCH_ALARM_NAME], // Set the value of
 CLOUDWATCH_ALARM_NAME to the name of an existing alarm.
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

在單獨的模組中建立用戶端並將其匯出。

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考EnableAlarmActions中的。

SDK對於 JavaScript （ 第 2 個 ）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

匯入SDK和用戶端模組並呼叫API.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create CloudWatch service object
var cw = new AWS.CloudWatch({ apiVersion: "2010-08-01" });

var params = {
  AlarmName: "Web_Server_CPU_Utilization",
  ComparisonOperator: "GreaterThanThreshold",
  EvaluationPeriods: 1,
  MetricName: "CPUUtilization",
  Namespace: "AWS/EC2",
  Period: 60,
  Statistic: "Average",
  Threshold: 70.0,
  ActionsEnabled: true,
  AlarmActions: ["ACTION_ARN"],
  AlarmDescription: "Alarm when server CPU exceeds 70%",
  Dimensions: [
    {
      Name: "InstanceId",
      Value: "INSTANCE_ID",
    },
  ],
  Unit: "Percent",
};

cw.putMetricAlarm(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
```

```
    console.log("Alarm action added", data);
    var paramsEnableAlarmAction = {
      AlarmNames: [params.AlarmName],
    };
    cw.enableAlarmActions(paramsEnableAlarmAction, function (err, data) {
      if (err) {
        console.log("Error", err);
      } else {
        console.log("Alarm action enabled", data);
      }
    });
  }
});
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考EnableAlarmActions中的。

`ListMetrics`

下列程式碼範例會示範如何使用`ListMetrics`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

匯入SDK和用戶端模組並呼叫API.

```
import { ListMetricsCommand } from "@aws-sdk/client-cloudwatch";
import { client } from "../libs/client.js";

export const main = () => {
  // Use the AWS console to see available namespaces and metric names. Custom
 metrics can also be created.
  // https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/
viewing_metrics_with_cloudwatch.html
  const command = new ListMetricsCommand({
```

```
    Dimensions: [
      {
        Name: "LogGroupName",
      },
    ],
    MetricName: "IncomingLogEvents",
    Namespace: "AWS/Logs",
  });

  return client.send(command);
};
```

在單獨的模組中建立用戶端並將其匯出。

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考ListMetrics中的。

SDK對於 JavaScript（第 2 個）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create CloudWatch service object
var cw = new AWS.CloudWatch({ apiVersion: "2010-08-01" });

var params = {
  Dimensions: [
    {
```

```
      Name: "LogGroupName" /* required */,
    },
  ],
  MetricName: "IncomingLogEvents",
  Namespace: "AWS/Logs",
};

cw.listMetrics(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Metrics", JSON.stringify(data.Metrics));
  }
});
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考ListMetrics中的。

`PutMetricAlarm`

下列程式碼範例會示範如何使用`PutMetricAlarm`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

匯入SDK和用戶端模組並呼叫API.

```
import { PutMetricAlarmCommand } from "@aws-sdk/client-cloudwatch";
import { client } from "../libs/client.js";

const run = async () => {
  // This alarm triggers when CPUUtilization exceeds 70% for one minute.
  const command = new PutMetricAlarmCommand({
    AlarmName: process.env.CLOUDWATCH_ALARM_NAME, // Set the value of
 CLOUDWATCH_ALARM_NAME to the name of an existing alarm.
```

```
    ComparisonOperator: "GreaterThanThreshold",
    EvaluationPeriods: 1,
    MetricName: "CPUUtilization",
    Namespace: "AWS/EC2",
    Period: 60,
    Statistic: "Average",
    Threshold: 70.0,
    ActionsEnabled: false,
    AlarmDescription: "Alarm when server CPU exceeds 70%",
    Dimensions: [
      {
        Name: "InstanceId",
        Value: process.env.EC2_INSTANCE_ID, // Set the value of EC_INSTANCE_ID to
 the Id of an existing Amazon EC2 instance.
      },
    ],
    Unit: "Percent",
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

在單獨的模組中建立用戶端並將其匯出。

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考PutMetricAlarm中的。

SDK對於 JavaScript （ 第 2 個 ）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create CloudWatch service object
var cw = new AWS.CloudWatch({ apiVersion: "2010-08-01" });

var params = {
  AlarmName: "Web_Server_CPU_Utilization",
  ComparisonOperator: "GreaterThanThreshold",
  EvaluationPeriods: 1,
  MetricName: "CPUUtilization",
  Namespace: "AWS/EC2",
  Period: 60,
  Statistic: "Average",
  Threshold: 70.0,
  ActionsEnabled: false,
  AlarmDescription: "Alarm when server CPU exceeds 70%",
  Dimensions: [
    {
      Name: "InstanceId",
      Value: "INSTANCE_ID",
    },
  ],
  Unit: "Percent",
};

cw.putMetricAlarm(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
```

```
});
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考PutMetricAlarm中的。

`PutMetricData`

下列程式碼範例會示範如何使用PutMetricData。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

匯入SDK和用戶端模組並呼叫API.

```
import { PutMetricDataCommand } from "@aws-sdk/client-cloudwatch";
import { client } from "../libs/client.js";

const run = async () => {
  // See https://docs.aws.amazon.com/AmazonCloudWatch/latest/APIReference/
API_PutMetricData.html#API_PutMetricData_RequestParameters
  // and https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/
publishingMetrics.html
  // for more information about the parameters in this command.
  const command = new PutMetricDataCommand({
    MetricData: [
      {
        MetricName: "PAGES_VISITED",
        Dimensions: [
          {
            Name: "UNIQUE_PAGES",
            Value: "URLS",
          },
        ],
        Unit: "None",
        Value: 1.0,
```

```
      },
    ],
    Namespace: "SITE/TRAFFIC",
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

在單獨的模組中建立用戶端並將其匯出。

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考PutMetricData中的。

SDK對於 JavaScript （ 第 2 個 ）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create CloudWatch service object
var cw = new AWS.CloudWatch({ apiVersion: "2010-08-01" });
```

```
// Create parameters JSON for putMetricData
var params = {
  MetricData: [
    {
      MetricName: "PAGES_VISITED",
      Dimensions: [
        {
          Name: "UNIQUE_PAGES",
          Value: "URLS",
        },
      ],
      Unit: "None",
      Value: 1.0,
    },
  ],
  Namespace: "SITE/TRAFFIC",
};

cw.putMetricData(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", JSON.stringify(data));
  }
});
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考PutMetricData中的。

CloudWatch 事件範例使用 SDK for JavaScript (v3)

下列程式碼範例會示範如何透過使用 AWS SDK for JavaScript (v3) 搭配 E CloudWatch vents 來執行動作及實作常見案例。

Actions 是大型程式的程式碼摘錄，必須在內容中執行。雖然動作會顯示如何呼叫個別服務函數，但您可以在其相關案例中查看內容中的動作。

每個範例都包含完整原始程式碼的連結，您可以在其中找到如何在內容中設定和執行程式碼的指示。

主題

- 動作

動作

`PutEvents`

下列程式碼範例會示範如何使用PutEvents。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

匯入SDK和用戶端模組並呼叫API.

```
import { PutEventsCommand } from "@aws-sdk/client-cloudwatch-events";
import { client } from "../libs/client.js";

const run = async () => {
  const command = new PutEventsCommand({
    // The list of events to send to Amazon CloudWatch Events.
    Entries: [
      {
        // The name of the application or service that is sending the event.
        Source: "my.app",

        // The name of the event that is being sent.
        DetailType: "My Custom Event",

        // The data that is sent with the event.
        Detail: JSON.stringify({ timeOfEvent: new Date().toISOString() }),
      },
    ],
  });

  try {
    return await client.send(command);
  } catch (err) {
```

```
    console.error(err);
  }
};

export default run();
```

在單獨的模組中建立用戶端並將其匯出。

```
import { CloudWatchEventsClient } from "@aws-sdk/client-cloudwatch-events";

export const client = new CloudWatchEventsClient({});
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考PutEvents中的。

SDK對於 JavaScript（第 2 個）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create CloudWatchEvents service object
var cwevents = new AWS.CloudWatchEvents({ apiVersion: "2015-10-07" });

var params = {
  Entries: [
    {
      Detail: '{ "key1": "value1", "key2": "value2" }',
      DetailType: "appRequestSubmitted",
      Resources: ["RESOURCE_ARN"],
      Source: "com.company.app",
    },
```

```
  ],
};

cwevents.putEvents(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.Entries);
  }
});
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考PutEvents中的。

`PutRule`

下列程式碼範例會示範如何使用PutRule。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

匯入SDK和用戶端模組並呼叫API.

```
import { PutRuleCommand } from "@aws-sdk/client-cloudwatch-events";
import { client } from "../libs/client.js";

const run = async () => {
  // Request parameters for PutRule.
  // https://docs.aws.amazon.com/eventbridge/latest/APIReference/
API_PutRule.html#API_PutRule_RequestParameters
  const command = new PutRuleCommand({
    Name: process.env.CLOUDWATCH_EVENTS_RULE,

    // The event pattern for the rule.
    //  Example: {"source": ["my.app"]}
```

```
    EventPattern: process.env.CLOUDWATCH_EVENTS_RULE_PATTERN,

    // The state of the rule. Valid values: ENABLED, DISABLED
    State: "ENABLED",
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

在單獨的模組中建立用戶端並將其匯出。

```
import { CloudWatchEventsClient } from "@aws-sdk/client-cloudwatch-events";

export const client = new CloudWatchEventsClient({});
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考PutRule中的。

SDK對於 JavaScript （ 第 2 個 ）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create CloudWatchEvents service object
var cwevents = new AWS.CloudWatchEvents({ apiVersion: "2015-10-07" });
```

```
var params = {
  Name: "DEMO_EVENT",
  RoleArn: "IAM_ROLE_ARN",
  ScheduleExpression: "rate(5 minutes)",
  State: "ENABLED",
};

cwevents.putRule(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.RuleArn);
  }
});
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考PutRule中的。

`PutTargets`

下列程式碼範例會示範如何使用`PutTargets`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

匯入SDK和用戶端模組並呼叫API.

```
import { PutTargetsCommand } from "@aws-sdk/client-cloudwatch-events";
import { client } from "../libs/client.js";

const run = async () => {
  const command = new PutTargetsCommand({
    // The name of the Amazon CloudWatch Events rule.
    Rule: process.env.CLOUDWATCH_EVENTS_RULE,
```

```
    // The targets to add to the rule.
    Targets: [
      {
        Arn: process.env.CLOUDWATCH_EVENTS_TARGET_ARN,
        // The ID of the target. Choose a unique ID for each target.
        Id: process.env.CLOUDWATCH_EVENTS_TARGET_ID,
      },
    ],
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

在單獨的模組中建立用戶端並將其匯出。

```
import { CloudWatchEventsClient } from "@aws-sdk/client-cloudwatch-events";

export const client = new CloudWatchEventsClient({});
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考PutTargets中的。

SDK對於 JavaScript（第 2 個）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });
```

```
// Create CloudWatchEvents service object
var cwevents = new AWS.CloudWatchEvents({ apiVersion: "2015-10-07" });

var params = {
  Rule: "DEMO_EVENT",
  Targets: [
    {
      Arn: "LAMBDA_FUNCTION_ARN",
      Id: "myCloudWatchEventsTarget",
    },
  ],
};

cwevents.putTargets(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考PutTargets中的。

CloudWatch 記錄範SDK例使用 JavaScript (v3)

下列程式碼範例說明如何透過使用 AWS SDK for JavaScript (v3) 搭配 CloudWatch Logs 來執行動作和實作常見案例。

Actions 是大型程式的程式碼摘錄，必須在內容中執行。雖然動作會顯示如何呼叫個別服務函數，但您可以在其相關案例中查看內容中的動作。

案例是程式碼範例，向您展示如何透過呼叫服務中的多個函式或與其他函式結合來完成特定工作 AWS 服務。

每個範例都包含完整原始程式碼的連結，您可以在其中找到如何在內容中設定和執行程式碼的指示。

主題

- 動作

- 案例

動作

`CreateLogGroup`

下列程式碼範例會示範如何使用CreateLogGroup。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import { CreateLogGroupCommand } from "@aws-sdk/client-cloudwatch-logs";
import { client } from "../libs/client.js";

const run = async () => {
  const command = new CreateLogGroupCommand({
    // The name of the log group.
    logGroupName: process.env.CLOUDWATCH_LOGS_LOG_GROUP,
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考CreateLogGroup中的。

`DeleteLogGroup`

下列程式碼範例會示範如何使用DeleteLogGroup。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import { DeleteLogGroupCommand } from "@aws-sdk/client-cloudwatch-logs";
import { client } from "../libs/client.js";

const run = async () => {
  const command = new DeleteLogGroupCommand({
    // The name of the log group.
    logGroupName: process.env.CLOUDWATCH_LOGS_LOG_GROUP,
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考DeleteLogGroup中的。

`DeleteSubscriptionFilter`

下列程式碼範例會示範如何使用`DeleteSubscriptionFilter`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import { DeleteSubscriptionFilterCommand } from "@aws-sdk/client-cloudwatch-logs";
import { client } from "../libs/client.js";

const run = async () => {
  const command = new DeleteSubscriptionFilterCommand({
    // The name of the filter.
    filterName: process.env.CLOUDWATCH_LOGS_FILTER_NAME,
    // The name of the log group.
    logGroupName: process.env.CLOUDWATCH_LOGS_LOG_GROUP,
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考DeleteSubscriptionFilter中的。

SDK對於 JavaScript （第 2 個）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the CloudWatchLogs service object
var cwl = new AWS.CloudWatchLogs({ apiVersion: "2014-03-28" });

var params = {
  filterName: "FILTER",
  logGroupName: "LOG_GROUP",
```

```
};

cwl.deleteSubscriptionFilter(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考DeleteSubscriptionFilter中的。

`DescribeLogGroups`

下列程式碼範例會示範如何使用`DescribeLogGroups`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import {
  paginateDescribeLogGroups,
  CloudWatchLogsClient,
} from "@aws-sdk/client-cloudwatch-logs";

const client = new CloudWatchLogsClient({});

export const main = async () => {
  const paginatedLogGroups = paginateDescribeLogGroups({ client }, {});
  const logGroups = [];

  for await (const page of paginatedLogGroups) {
    if (page.logGroups && page.logGroups.every((lg) => !!lg)) {
      logGroups.push(...page.logGroups);
    }
```

```
  }

  console.log(logGroups);
  return logGroups;
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考DescribeLogGroups中的。

`DescribeSubscriptionFilters`

下列程式碼範例會示範如何使用`DescribeSubscriptionFilters`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import { DescribeSubscriptionFiltersCommand } from "@aws-sdk/client-cloudwatch-
logs";
import { client } from "../libs/client.js";

const run = async () => {
  // This will return a list of all subscription filters in your account
  // matching the log group name.
  const command = new DescribeSubscriptionFiltersCommand({
    logGroupName: process.env.CLOUDWATCH_LOGS_LOG_GROUP,
    limit: 1,
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考DescribeSubscriptionFilters中的。

SDK對於 JavaScript （第 2 個）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the CloudWatchLogs service object
var cwl = new AWS.CloudWatchLogs({ apiVersion: "2014-03-28" });

var params = {
  logGroupName: "GROUP_NAME",
  limit: 5,
};

cwl.describeSubscriptionFilters(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.subscriptionFilters);
  }
});
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考DescribeSubscriptionFilters中的。

`GetQueryResults`

下列程式碼範例會示範如何使用`GetQueryResults`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
/**
 * Simple wrapper for the GetQueryResultsCommand.
 * @param {string} queryId
 */
_getQueryResults(queryId) {
  return this.client.send(new GetQueryResultsCommand({ queryId }));
}
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考GetQueryResults中的。

`PutSubscriptionFilter`

下列程式碼範例會示範如何使用`PutSubscriptionFilter`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import { PutSubscriptionFilterCommand } from "@aws-sdk/client-cloudwatch-logs";
import { client } from "../libs/client.js";

const run = async () => {
  const command = new PutSubscriptionFilterCommand({
    // An ARN of a same-account Kinesis stream, Kinesis Firehose
    // delivery stream, or Lambda function.
    // https://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/
SubscriptionFilters.html
```

```
    destinationArn: process.env.CLOUDWATCH_LOGS_DESTINATION_ARN,

    // A name for the filter.
    filterName: process.env.CLOUDWATCH_LOGS_FILTER_NAME,

    // A filter pattern for subscribing to a filtered stream of log events.
    // https://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/
FilterAndPatternSyntax.html
    filterPattern: process.env.CLOUDWATCH_LOGS_FILTER_PATTERN,

    // The name of the log group. Messages in this group matching the filter pattern
    // will be sent to the destination ARN.
    logGroupName: process.env.CLOUDWATCH_LOGS_LOG_GROUP,
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考PutSubscriptionFilter中的。

SDK對於 JavaScript （ 第 2 個 ）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the CloudWatchLogs service object
var cwl = new AWS.CloudWatchLogs({ apiVersion: "2014-03-28" });
```

```
var params = {
  destinationArn: "LAMBDA_FUNCTION_ARN",
  filterName: "FILTER_NAME",
  filterPattern: "ERROR",
  logGroupName: "LOG_GROUP",
};

cwl.putSubscriptionFilter(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考PutSubscriptionFilter中的。

`StartLiveTail`

下列程式碼範例會示範如何使用`StartLiveTail`。

SDK對於 JavaScript（3）

包括必需的檔案。

```
import { CloudWatchLogsClient, StartLiveTailCommand } from "@aws-sdk/client-
cloudwatch-logs";
```

處理來自即時尾巴工作階段的事件。

```
async function handleResponseAsync(response) {
    try {
      for await (const event of response.responseStream) {
        if (event.sessionStart !== undefined) {
          console.log(event.sessionStart);
        } else if (event.sessionUpdate !== undefined) {
          for (const logEvent of event.sessionUpdate.sessionResults) {
            const timestamp = logEvent.timestamp;
```

```
            const date = new Date(timestamp);
            console.log("[" + date + "] " + logEvent.message);
          }
        } else {
            console.error("Unknown event type");
        }
      }
    } catch (err) {
        // On-stream exceptions are captured here
        console.error(err)
    }
}
```

啟動「即時尾巴」工作階段。

```
    const client = new CloudWatchLogsClient();

    const command = new StartLiveTailCommand({
        logGroupIdentifiers: logGroupIdentifiers,
        logStreamNames: logStreamNames,
        logEventFilterPattern: filterPattern
    });
    try{
        const response = await client.send(command);
        handleResponseAsync(response);
    } catch (err){
        // Pre-stream exceptions are captured here
        console.log(err);
    }
```

經過一段時間後，停止「即時尾端」工作階段。

```
    /* Set a timeout to close the client. This will stop the Live Tail session. */
    setTimeout(function() {
        console.log("Client timeout");
        client.destroy();
      }, 10000);
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考StartLiveTail中的。

`StartQuery`

下列程式碼範例會示範如何使用`StartQuery`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
/**
 * Wrapper for the StartQueryCommand. Uses a static query string
 * for consistency.
 * @param {[Date, Date]} dateRange
 * @param {number} maxLogs
 * @returns {Promise<{ queryId: string }>}
 */
async _startQuery([startDate, endDate], maxLogs = 10000) {
  try {
    return await this.client.send(
      new StartQueryCommand({
        logGroupNames: this.logGroupNames,
        queryString: "fields @timestamp, @message | sort @timestamp asc",
        startTime: startDate.valueOf(),
        endTime: endDate.valueOf(),
        limit: maxLogs,
      }),
    );
  } catch (err) {
    /** @type {string} */
    const message = err.message;
    if (message.startsWith("Query's end date and time")) {
      // This error indicates that the query's start or end date occur
      // before the log group was created.
      throw new DateOutOfBoundsError(message);
    }

    throw err;
  }
}
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考StartQuery中的。

案例

執行大型查詢

下列程式碼範例顯示如何使用記 CloudWatch 錄來查詢 10,000 筆以上的記錄。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

這是入口點。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { CloudWatchLogsClient } from "@aws-sdk/client-cloudwatch-logs";
import { CloudWatchQuery } from "./cloud-watch-query.js";

console.log("Starting a recursive query...");

if (!process.env.QUERY_START_DATE || !process.env.QUERY_END_DATE) {
  throw new Error(
    "QUERY_START_DATE and QUERY_END_DATE environment variables are required.",
  );
}

const cloudWatchQuery = new CloudWatchQuery(new CloudWatchLogsClient({}), {
  logGroupNames: ["/workflows/cloudwatch-logs/large-query"],
  dateRange: [
    new Date(parseInt(process.env.QUERY_START_DATE)),
    new Date(parseInt(process.env.QUERY_END_DATE)),
  ],
});
```

```
await cloudWatchQuery.run();

console.log(
  `Queries finished in ${cloudWatchQuery.secondsElapsed} seconds.\nTotal logs found:
 ${cloudWatchQuery.results.length}`,
);
```

這是在必要時將查詢拆分為多個步驟的類。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import {
  StartQueryCommand,
  GetQueryResultsCommand,
} from "@aws-sdk/client-cloudwatch-logs";
import { splitDateRange } from "@aws-doc-sdk-examples/lib/utils/util-date.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

class DateOutOfBoundsError extends Error {}

export class CloudWatchQuery {
  /**
   * Run a query for all CloudWatch Logs within a certain date range.
   * CloudWatch logs return a max of 10,000 results. This class
   * performs a binary search across all of the logs in the provided
   * date range if a query returns the maximum number of results.
   *
   * @param {import('@aws-sdk/client-cloudwatch-logs').CloudWatchLogsClient} client
   * @param {{ logGroupNames: string[], dateRange: [Date, Date], queryConfig:
 { limit: number } }} config
   */
  constructor(client, { logGroupNames, dateRange, queryConfig }) {
    this.client = client;
    /**
     * All log groups are queried.
     */
    this.logGroupNames = logGroupNames;

    /**
     * The inclusive date range that is queried.
     */
    this.dateRange = dateRange;
```

```
  /**
   * CloudWatch Logs never returns more than 10,000 logs.
   */
  this.limit = queryConfig?.limit ?? 10000;

  /**
   * @type {import("@aws-sdk/client-cloudwatch-logs").ResultField[][]}
   */
  this.results = [];
}

/**
 * Run the query.
 */
async run() {
  this.secondsElapsed = 0;
  const start = new Date();
  this.results = await this._largeQuery(this.dateRange);
  const end = new Date();
  this.secondsElapsed = (end - start) / 1000;
  return this.results;
}

/**
 * Recursively query for logs.
 * @param {[Date, Date]} dateRange
 * @returns {Promise<import("@aws-sdk/client-cloudwatch-logs").ResultField[][]>}
 */
async _largeQuery(dateRange) {
  const logs = await this._query(dateRange, this.limit);

  console.log(
    `Query date range: ${dateRange
      .map((d) => d.toISOString())
      .join(" to ")}. Found ${logs.length} logs.`,
  );

  if (logs.length < this.limit) {
    return logs;
  }

  const lastLogDate = this._getLastLogDate(logs);
  const offsetLastLogDate = new Date(lastLogDate);
```

```
    offsetLastLogDate.setMilliseconds(lastLogDate.getMilliseconds() + 1);
    const subDateRange = [offsetLastLogDate, dateRange[1]];
    const [r1, r2] = splitDateRange(subDateRange);
    const results = await Promise.all([
      this._largeQuery(r1),
      this._largeQuery(r2),
    ]);
    return [logs, ...results].flat();
  }

  /**
   * Find the most recent log in a list of logs.
   * @param {import("@aws-sdk/client-cloudwatch-logs").ResultField[][]} logs
   */
  _getLastLogDate(logs) {
    const timestamps = logs
      .map(
        (log) =>
          log.find((fieldMeta) => fieldMeta.field === "@timestamp")?.value,
      )
      .filter((t) => !!t)
      .map((t) => `${t}Z`)
      .sort();

    if (!timestamps.length) {
      throw new Error("No timestamp found in logs.");
    }

    return new Date(timestamps[timestamps.length - 1]);
  }

  /**
   * Simple wrapper for the GetQueryResultsCommand.
   * @param {string} queryId
   */
  _getQueryResults(queryId) {
    return this.client.send(new GetQueryResultsCommand({ queryId }));
  }

  /**
   * Starts a query and waits for it to complete.
   * @param {[Date, Date]} dateRange
   * @param {number} maxLogs
   */
```

```
  async _query(dateRange, maxLogs) {
    try {
      const { queryId } = await this._startQuery(dateRange, maxLogs);
      const { results } = await this._waitUntilQueryDone(queryId);
      return results ?? [];
    } catch (err) {
      /**
       * This error is thrown when StartQuery returns an error indicating
       * that the query's start or end date occur before the log group was
       * created.
       */
      if (err instanceof DateOutOfBoundsError) {
        return [];
      } else {
        throw err;
      }
    }
  }

  /**
   * Wrapper for the StartQueryCommand. Uses a static query string
   * for consistency.
   * @param {[Date, Date]} dateRange
   * @param {number} maxLogs
   * @returns {Promise<{ queryId: string }>}
   */
  async _startQuery([startDate, endDate], maxLogs = 10000) {
    try {
      return await this.client.send(
        new StartQueryCommand({
          logGroupNames: this.logGroupNames,
          queryString: "fields @timestamp, @message | sort @timestamp asc",
          startTime: startDate.valueOf(),
          endTime: endDate.valueOf(),
          limit: maxLogs,
        }),
      );
    } catch (err) {
      /** @type {string} */
      const message = err.message;
      if (message.startsWith("Query's end date and time")) {
        // This error indicates that the query's start or end date occur
        // before the log group was created.
        throw new DateOutOfBoundsError(message);
```

```
      }

      throw err;
    }
  }

  /**
   * Call GetQueryResultsCommand until the query is done.
   * @param {string} queryId
   */
  _waitUntilQueryDone(queryId) {
    const getResults = async () => {
      const results = await this._getQueryResults(queryId);
      const queryDone = [
        "Complete",
        "Failed",
        "Cancelled",
        "Timeout",
        "Unknown",
      ].includes(results.status);

      return { queryDone, results };
    };

    return retry(
      { intervalInMs: 1000, maxRetries: 60, quiet: true },
      async () => {
        const { queryDone, results } = await getResults();
        if (!queryDone) {
          throw new Error("Query not done.");
        }

        return results;
      },
    );
  }
}
```

- 如需詳API細資訊，請參閱「AWS SDK for JavaScript API參考」中的下列主題。
  - GetQueryResults
  - StartQuery

CodeBuild 使用 JavaScript （ v3 ）SDK的示例

下列程式碼範例會示範如何使用 AWS SDK for JavaScript (v3) 搭配使用來執行動作及實作常見案例 CodeBuild。

Actions 是大型程式的程式碼摘錄，必須在內容中執行。雖然動作會顯示如何呼叫個別服務函數，但您可以在其相關案例中查看內容中的動作。

每個範例都包含完整原始程式碼的連結，您可以在其中找到如何在內容中設定和執行程式碼的指示。

主題

動作

`CreateProject`

下列程式碼範例會示範如何使用`CreateProject`。

SDK對於 JavaScript （ 3 ）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

建立專案。

```
import {
  ArtifactsType,
  CodeBuildClient,
  ComputeType,
  CreateProjectCommand,
  EnvironmentType,
  SourceType,
} from "@aws-sdk/client-codebuild";

// Create the AWS CodeBuild project.
export const createProject = async (
  projectName = "MyCodeBuilder",
```

```
  roleArn = "arn:aws:iam::xxxxxxxxxxxx:role/CodeBuildAdmin",
  buildOutputBucket = "xxxx",
  githubUrl = "https://...",
) => {
  const codeBuildClient = new CodeBuildClient({});

  const response = await codeBuildClient.send(
    new CreateProjectCommand({
      artifacts: {
        // The destination of the build artifacts.
        type: ArtifactsType.S3,
        location: buildOutputBucket,
      },
      // Information about the build environment. The combination of "computeType"
 and "type" determines the
      // requirements for the environment such as CPU, memory, and disk space.
      environment: {
        // Build environment compute types.
        // https://docs.aws.amazon.com/codebuild/latest/userguide/build-env-ref-
compute-types.html
        computeType: ComputeType.BUILD_GENERAL1_SMALL,
        // Docker image identifier.
        // See https://docs.aws.amazon.com/codebuild/latest/userguide/build-env-ref-
available.html
        image: "aws/codebuild/standard:7.0",
        // Build environment type.
        type: EnvironmentType.LINUX_CONTAINER,
      },
      name: projectName,
      // A role ARN with permission to create a CodeBuild project, write to the
 artifact location, and write CloudWatch logs.
      serviceRole: roleArn,
      source: {
        // The type of repository that contains the source code to be built.
        type: SourceType.GITHUB,
        // The location of the repository that contains the source code to be built.
        location: githubUrl,
      },
    }),
  );
  console.log(response);
  //   {
  //     '$metadata': {
  //       httpStatusCode: 200,
```

```
  //       requestId: 'b428b244-777b-49a6-a48d-5dffedced8e7',
  //       extendedRequestId: undefined,
  //       cfId: undefined,
  //       attempts: 1,
  //       totalRetryDelay: 0
  //     },
  //     project: {
  //       arn: 'arn:aws:codebuild:us-east-1:xxxxxxxxxxxx:project/MyCodeBuilder',
  //       artifacts: {
  //         encryptionDisabled: false,
  //         location: 'xxxxxx-xxxxxxx-xxxxxx',
  //         name: 'MyCodeBuilder',
  //         namespaceType: 'NONE',
  //         packaging: 'NONE',
  //         type: 'S3'
  //       },
  //       badge: { badgeEnabled: false },
  //       cache: { type: 'NO_CACHE' },
  //       created: 2023-08-18T14:46:48.979Z,
  //       encryptionKey: 'arn:aws:kms:us-east-1:xxxxxxxxxxxx:alias/aws/s3',
  //       environment: {
  //         computeType: 'BUILD_GENERAL1_SMALL',
  //         environmentVariables: [],
  //         image: 'aws/codebuild/standard:7.0',
  //         imagePullCredentialsType: 'CODEBUILD',
  //         privilegedMode: false,
  //         type: 'LINUX_CONTAINER'
  //       },
  //       lastModified: 2023-08-18T14:46:48.979Z,
  //       name: 'MyCodeBuilder',
  //       projectVisibility: 'PRIVATE',
  //       queuedTimeoutInMinutes: 480,
  //       serviceRole: 'arn:aws:iam::xxxxxxxxxxxx:role/CodeBuildAdmin',
  //       source: {
  //         insecureSsl: false,
  //         location: 'https://...',
  //         reportBuildStatus: false,
  //         type: 'GITHUB'
  //       },
  //       timeoutInMinutes: 60
  //     }
  //   }
  return response;
};
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考CreateProject中的。

Amazon Cognito 身份提供商示例使SDK用 JavaScript (v3)

下列程式碼範例說明如何透過使用 AWS SDK for JavaScript (v3) 搭配 Amazon Cognito 身分識別提供者來執行動作和實作常見案例。

Actions 是大型程式的程式碼摘錄，必須在內容中執行。雖然動作會顯示如何呼叫個別服務函數，但您可以在其相關案例中查看內容中的動作。

案例是程式碼範例，向您展示如何透過呼叫服務中的多個函式或與其他函式結合來完成特定工作 AWS 服務。

每個範例都包含完整原始程式碼的連結，您可以在其中找到如何在內容中設定和執行程式碼的指示。

開始使用

Hello Amazon Cognito

下列程式碼範例顯示如何開始使用 Amazon Cognito。

SDK對於 JavaScript （ 3 ）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import {
  paginateListUserPools,
  CognitoIdentityProviderClient,
} from "@aws-sdk/client-cognito-identity-provider";

const client = new CognitoIdentityProviderClient({});

export const helloCognito = async () => {
```

```
  const paginator = paginateListUserPools({ client }, {});

  const userPoolNames = [];

  for await (const page of paginator) {
    const names = page.UserPools.map((pool) => pool.Name);
    userPoolNames.push(...names);
  }

  console.log("User pool names: ");
  console.log(userPoolNames.join("\n"));
  return userPoolNames;
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考ListUserPools中的。

主題

動作

`AdminGetUser`

下列程式碼範例會示範如何使用AdminGetUser。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
const adminGetUser = ({ userPoolId, username }) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new AdminGetUserCommand({
    UserPoolId: userPoolId,
```

```
    Username: username,
  });

  return client.send(command);
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考AdminGetUser中的。

`AdminInitiateAuth`

下列程式碼範例會示範如何使用`AdminInitiateAuth`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
const adminInitiateAuth = ({ clientId, userPoolId, username, password }) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new AdminInitiateAuthCommand({
    ClientId: clientId,
    UserPoolId: userPoolId,
    AuthFlow: AuthFlowType.ADMIN_USER_PASSWORD_AUTH,
    AuthParameters: { USERNAME: username, PASSWORD: password },
  });

  return client.send(command);
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考AdminInitiateAuth中的。

`AdminRespondToAuthChallenge`

下列程式碼範例會示範如何使用`AdminRespondToAuthChallenge`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
const adminRespondToAuthChallenge = ({
  userPoolId,
  clientId,
  username,
  totp,
  session,
}) => {
  const client = new CognitoIdentityProviderClient({});
  const command = new AdminRespondToAuthChallengeCommand({
    ChallengeName: ChallengeNameType.SOFTWARE_TOKEN_MFA,
    ChallengeResponses: {
      SOFTWARE_TOKEN_MFA_CODE: totp,
      USERNAME: username,
    },
    ClientId: clientId,
    UserPoolId: userPoolId,
    Session: session,
  });

  return client.send(command);
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考AdminRespondToAuthChallenge中的。

`AssociateSoftwareToken`

下列程式碼範例會示範如何使用`AssociateSoftwareToken`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
const associateSoftwareToken = (session) => {
  const client = new CognitoIdentityProviderClient({});
  const command = new AssociateSoftwareTokenCommand({
    Session: session,
  });

  return client.send(command);
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考AssociateSoftwareToken中的。

`ConfirmDevice`

下列程式碼範例會示範如何使用`ConfirmDevice`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
const confirmDevice = ({ deviceKey, accessToken, passwordVerifier, salt }) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new ConfirmDeviceCommand({
    DeviceKey: deviceKey,
    AccessToken: accessToken,
    DeviceSecretVerifierConfig: {
      PasswordVerifier: passwordVerifier,
```

```
      Salt: salt,
    },
  });

  return client.send(command);
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考ConfirmDevice中的。

`ConfirmSignUp`

下列程式碼範例會示範如何使用`ConfirmSignUp`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
const confirmSignUp = ({ clientId, username, code }) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new ConfirmSignUpCommand({
    ClientId: clientId,
    Username: username,
    ConfirmationCode: code,
  });

  return client.send(command);
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考ConfirmSignUp中的。

`InitiateAuth`

下列程式碼範例會示範如何使用`InitiateAuth`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
const initiateAuth = ({ username, password, clientId }) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new InitiateAuthCommand({
    AuthFlow: AuthFlowType.USER_PASSWORD_AUTH,
    AuthParameters: {
      USERNAME: username,
      PASSWORD: password,
    },
    ClientId: clientId,
  });

  return client.send(command);
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考InitiateAuth中的。

`ListUsers`

下列程式碼範例會示範如何使用`ListUsers`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
const listUsers = ({ userPoolId }) => {
  const client = new CognitoIdentityProviderClient({});
```

```
  const command = new ListUsersCommand({
    UserPoolId: userPoolId,
  });

  return client.send(command);
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考ListUsers中的。

`ResendConfirmationCode`

下列程式碼範例會示範如何使用`ResendConfirmationCode`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
const resendConfirmationCode = ({ clientId, username }) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new ResendConfirmationCodeCommand({
    ClientId: clientId,
    Username: username,
  });

  return client.send(command);
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考ResendConfirmationCode中的。

`RespondToAuthChallenge`

下列程式碼範例會示範如何使用`RespondToAuthChallenge`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
const respondToAuthChallenge = ({
  clientId,
  username,
  session,
  userPoolId,
  code,
}) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new RespondToAuthChallengeCommand({
    ChallengeName: ChallengeNameType.SOFTWARE_TOKEN_MFA,
    ChallengeResponses: {
      SOFTWARE_TOKEN_MFA_CODE: code,
      USERNAME: username,
    },
    ClientId: clientId,
    UserPoolId: userPoolId,
    Session: session,
  });

  return client.send(command);
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考RespondToAuthChallenge中的。

`SignUp`

下列程式碼範例會示範如何使用SignUp。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
const signUp = ({ clientId, username, password, email }) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new SignUpCommand({
    ClientId: clientId,
    Username: username,
    Password: password,
    UserAttributes: [{ Name: "email", Value: email }],
  });

  return client.send(command);
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考SignUp中的。

`VerifySoftwareToken`

下列程式碼範例會示範如何使用`VerifySoftwareToken`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
const verifySoftwareToken = (totp) => {
  const client = new CognitoIdentityProviderClient({});

  // The 'Session' is provided in the response to 'AssociateSoftwareToken'.
```

```
  const session = process.env.SESSION;

  if (!session) {
    throw new Error(
      "Missing a valid Session. Did you run 'admin-initiate-auth'?",
    );
  }

  const command = new VerifySoftwareTokenCommand({
    Session: session,
    UserCode: totp,
  });

  return client.send(command);
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考VerifySoftwareToken中的。

案例

使用需要的使用者集區註冊使用者 MFA

以下程式碼範例顯示做法：

- 使用使用者名稱、密碼和電子郵件地址註冊並確認使用者。
- 通過將MFA應用程序與用戶關聯來設置多因素身份驗證。
- 使用密碼和MFA代碼登入。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

為獲得最佳體驗，請複製 GitHub 儲存庫並執行此範例。下列程式碼代表完整範例應用程式的範例。

```
import { log } from "@aws-doc-sdk-examples/lib/utils/util-log.js";
import { signUp } from "../../../actions/sign-up.js";
import { FILE_USER_POOLS } from "./constants.js";
import { getSecondValuesFromEntries } from "@aws-doc-sdk-examples/lib/utils/util-
csv.js";

const validateClient = (clientId) => {
  if (!clientId) {
    throw new Error(
      `App client id is missing. Did you run 'create-user-pool'?`,
    );
  }
};

const validateUser = (username, password, email) => {
  if (!(username && password && email)) {
    throw new Error(
      `Username, password, and email must be provided as arguments to the 'sign-up'
 command.`,
    );
  }
};

const signUpHandler = async (commands) => {
  const [_, username, password, email] = commands;

  try {
    validateUser(username, password, email);
    /**
     * @type {string[]}
     */
    const values = getSecondValuesFromEntries(FILE_USER_POOLS);
    const clientId = values[0];
    validateClient(clientId);
    log(`Signing up.`);
    await signUp({ clientId, username, password, email });
    log(`Signed up. A confirmation email has been sent to: ${email}.`);
    log(`Run 'confirm-sign-up ${username} <code>' to confirm your account.`);
  } catch (err) {
    log(err);
  }
};
```

```
export { signUpHandler };

const signUp = ({ clientId, username, password, email }) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new SignUpCommand({
    ClientId: clientId,
    Username: username,
    Password: password,
    UserAttributes: [{ Name: "email", Value: email }],
  });

  return client.send(command);
};

import { log } from "@aws-doc-sdk-examples/lib/utils/util-log.js";
import { confirmSignUp } from "../../../actions/confirm-sign-up.js";
import { FILE_USER_POOLS } from "./constants.js";
import { getSecondValuesFromEntries } from "@aws-doc-sdk-examples/lib/utils/util-
csv.js";

const validateClient = (clientId) => {
  if (!clientId) {
    throw new Error(
      `App client id is missing. Did you run 'create-user-pool'?`,
    );
  }
};

const validateUser = (username) => {
  if (!username) {
    throw new Error(
      `Username name is missing. It must be provided as an argument to the 'confirm-
sign-up' command.`,
    );
  }
};

const validateCode = (code) => {
  if (!code) {
    throw new Error(
      `Verification code is missing. It must be provided as an argument to the
 'confirm-sign-up' command.`,
    );
```

```
  }
};

const confirmSignUpHandler = async (commands) => {
  const [_, username, code] = commands;

  try {
    validateUser(username);
    validateCode(code);
    /**
     * @type {string[]}
     */
    const values = getSecondValuesFromEntries(FILE_USER_POOLS);
    const clientId = values[0];
    validateClient(clientId);
    log(`Confirming user.`);
    await confirmSignUp({ clientId, username, code });
    log(
      `User confirmed. Run 'admin-initiate-auth ${username} <password>' to sign
 in.`,
    );
  } catch (err) {
    log(err);
  }
};

export { confirmSignUpHandler };

const confirmSignUp = ({ clientId, username, code }) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new ConfirmSignUpCommand({
    ClientId: clientId,
    Username: username,
    ConfirmationCode: code,
  });

  return client.send(command);
};

import qrcode from "qrcode-terminal";
import { log } from "@aws-doc-sdk-examples/lib/utils/util-log.js";
import { adminInitiateAuth } from "../../../actions/admin-initiate-auth.js";
```

```
import { associateSoftwareToken } from "../../../actions/associate-software-
token.js";
import { FILE_USER_POOLS } from "./constants.js";
import { getFirstEntry } from "@aws-doc-sdk-examples/lib/utils/util-csv.js";

const handleMfaSetup = async (session, username) => {
  const { SecretCode, Session } = await associateSoftwareToken(session);

  // Store the Session for use with 'VerifySoftwareToken'.
  process.env.SESSION = Session;

  console.log(
    "Scan this code in your preferred authenticator app, then run 'verify-software-
token' to finish the setup.",
  );
  qrcode.generate(
    `otpauth://totp/${username}?secret=${SecretCode}`,
    { small: true },
    console.log,
  );
};

const handleSoftwareTokenMfa = (session) => {
  // Store the Session for use with 'AdminRespondToAuthChallenge'.
  process.env.SESSION = session;
};

const validateClient = (id) => {
  if (!id) {
    throw new Error(
      `User pool client id is missing. Did you run 'create-user-pool'?`,
    );
  }
};

const validateId = (id) => {
  if (!id) {
    throw new Error(`User pool id is missing. Did you run 'create-user-pool'?`);
  }
};

const validateUser = (username, password) => {
  if (!(username && password)) {
    throw new Error(
```

```
      `Username and password must be provided as arguments to the 'admin-initiate-
auth' command.`,
    );
  }
};

const adminInitiateAuthHandler = async (commands) => {
  const [_, username, password] = commands;

  try {
    validateUser(username, password);

    const [userPoolId, clientId] = getFirstEntry(FILE_USER_POOLS);
    validateId(userPoolId);
    validateClient(clientId);

    log("Signing in.");
    const { ChallengeName, Session } = await adminInitiateAuth({
      clientId,
      userPoolId,
      username,
      password,
    });

    if (ChallengeName === "MFA_SETUP") {
      log("MFA setup is required.");
      return handleMfaSetup(Session, username);
    }

    if (ChallengeName === "SOFTWARE_TOKEN_MFA") {
      handleSoftwareTokenMfa(Session);
      log(`Run 'admin-respond-to-auth-challenge ${username} <totp>'`);
    }
  } catch (err) {
    log(err);
  }
};

export { adminInitiateAuthHandler };

const adminInitiateAuth = ({ clientId, userPoolId, username, password }) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new AdminInitiateAuthCommand({
```

```
    ClientId: clientId,
    UserPoolId: userPoolId,
    AuthFlow: AuthFlowType.ADMIN_USER_PASSWORD_AUTH,
    AuthParameters: { USERNAME: username, PASSWORD: password },
  });

  return client.send(command);
};

import { log } from "@aws-doc-sdk-examples/lib/utils/util-log.js";
import { adminRespondToAuthChallenge } from "../../../actions/admin-respond-to-auth-
challenge.js";
import { getFirstEntry } from "@aws-doc-sdk-examples/lib/utils/util-csv.js";
import { FILE_USER_POOLS } from "./constants.js";

const verifyUsername = (username) => {
  if (!username) {
    throw new Error(
      `Username is missing. It must be provided as an argument to the 'admin-
respond-to-auth-challenge' command.`,
    );
  }
};

const verifyTotp = (totp) => {
  if (!totp) {
    throw new Error(
      `Time-based one-time password (TOTP) is missing. It must be provided as an
 argument to the 'admin-respond-to-auth-challenge' command.`,
    );
  }
};

const storeAccessToken = (token) => {
  process.env.AccessToken = token;
};

const adminRespondToAuthChallengeHandler = async (commands) => {
  const [_, username, totp] = commands;

  try {
    verifyUsername(username);
    verifyTotp(totp);
```

```
    const [userPoolId, clientId] = getFirstEntry(FILE_USER_POOLS);
    const session = process.env.SESSION;

    const { AuthenticationResult } = await adminRespondToAuthChallenge({
      clientId,
      userPoolId,
      username,
      totp,
      session,
    });

    storeAccessToken(AuthenticationResult.AccessToken);

    log("Successfully authenticated.");
  } catch (err) {
    log(err);
  }
};

export { adminRespondToAuthChallengeHandler };

const respondToAuthChallenge = ({
  clientId,
  username,
  session,
  userPoolId,
  code,
}) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new RespondToAuthChallengeCommand({
    ChallengeName: ChallengeNameType.SOFTWARE_TOKEN_MFA,
    ChallengeResponses: {
      SOFTWARE_TOKEN_MFA_CODE: code,
      USERNAME: username,
    },
    ClientId: clientId,
    UserPoolId: userPoolId,
    Session: session,
  });

  return client.send(command);
};
```

```
import { log } from "@aws-doc-sdk-examples/lib/utils/util-log.js";
import { verifySoftwareToken } from "../../../actions/verify-software-token.js";

const validateTotp = (totp) => {
  if (!totp) {
    throw new Error(
      `Time-based one-time password (TOTP) must be provided to the 'validate-
software-token' command.`,
    );
  }
};
const verifySoftwareTokenHandler = async (commands) => {
  const [_, totp] = commands;

  try {
    validateTotp(totp);

    log("Verifying TOTP.");
    await verifySoftwareToken(totp);
    log("TOTP Verified. Run 'admin-initiate-auth' again to sign-in.");
  } catch (err) {
    console.log(err);
  }
};

export { verifySoftwareTokenHandler };

const verifySoftwareToken = (totp) => {
  const client = new CognitoIdentityProviderClient({});

  // The 'Session' is provided in the response to 'AssociateSoftwareToken'.
  const session = process.env.SESSION;

  if (!session) {
    throw new Error(
      "Missing a valid Session. Did you run 'admin-initiate-auth'?",
    );
  }

  const command = new VerifySoftwareTokenCommand({
    Session: session,
    UserCode: totp,
  });
```

```
  return client.send(command);
};
```

- 如需詳API細資訊，請參閱「AWS SDK for JavaScript API參考」中的下列主題。
  - AdminGetUser
  - AdminInitiateAuth
  - AdminRespondToAuthChallenge
  - AssociateSoftwareToken
  - ConfirmDevice
  - ConfirmSignUp
  - InitiateAuth
  - ListUsers
  - ResendConfirmationCode
  - RespondToAuthChallenge
  - SignUp
  - VerifySoftwareToken

Amazon Comprehend 使用SDK的例子 JavaScript (v3)

下列程式碼範例說明如何透過使用 AWS SDK for JavaScript (v3) 搭配 Amazon Comprehend 來執行動作和實作常見案例。

案例是程式碼範例，向您展示如何透過呼叫服務中的多個函式或與其他函式結合來完成特定工作 AWS 服務。

每個範例都包含完整原始程式碼的連結，您可以在其中找到如何在內容中設定和執行程式碼的指示。

主題

- 案例

案例

建置 Amazon Transcribe 串流應用程式

下面的程式碼範例說明如何建置可即時記錄、轉錄和翻譯直播音訊並透過電子郵件傳送結果的應用程式。

SDK對於 JavaScript（3）

示範如何使用 Amazon Transcribe 建置可即時記錄、轉錄和轉譯即時音訊的應用程式，並使用 Amazon 簡易電子郵件服務 (Amazon) 將結果以電子郵件傳送給結果。SES

有關如何設置和運行的完整源代碼和說明，請參閱中的完整示例GitHub。

此範例中使用的服務

- Amazon Comprehend
- Amazon SES
- Amazon Transcribe
- Amazon Translate

建立 Amazon Lex 聊天機器人

以下代碼示例演示瞭如何創建聊天機器人以吸引您的網站訪問者。

SDK對於 JavaScript（3）

示範如何使用 Amazon Lex 在 Web 應API用程式中建立 Chatbot，以吸引您的網站訪客。

如需有關如何設定和執行的完整原始程式碼和指示，請參閱開發人 AWS SDK for JavaScript 員指南中的建立 Amazon Lex 聊天機器人的完整範例。

此範例中使用的服務

- Amazon Comprehend
- Amazon Lex
- Amazon Translate

建立應用程式以分析客戶意見回饋

下列程式碼範例會示範如何建立可分析客戶評論卡、從其原始語言進行翻譯、判斷對方情緒，以及透過翻譯後的文字產生音訊檔案的應用程式。

SDK對於 JavaScript（3）

此範例應用程式會分析和存儲客戶的意見回饋卡。具體來說，它滿足了紐約市一家虛構飯店的需求。飯店以實體評論卡的形式收到賓客以各種語言撰寫的意見回饋。這些意見回饋透過 Web 用戶端上傳至應用程式。評論卡的影像上傳後，系統會執行下列步驟：

- 文字內容是使用 Amazon Textract 從影像中擷取。
- Amazon Comprehend 會決定擷取文字及其用語的情感。
- 擷取的文字內容會使用 Amazon Translate 翻譯成英文。
- Amazon Polly 會使用擷取的文字內容合成音訊檔案。

完整的應用程式可透過 AWS CDK 部署。如需原始程式碼和部署指示，請參閱中的專案 GitHub。下列摘錄顯示如何在 AWS SDK for JavaScript Lambda 函數內部使用。

```
import {
  ComprehendClient,
  DetectDominantLanguageCommand,
  DetectSentimentCommand,
} from "@aws-sdk/client-comprehend";

/**
 * Determine the language and sentiment of the extracted text.
 *
 * @param {{ source_text: string}} extractTextOutput
 */
export const handler = async (extractTextOutput) => {
  const comprehendClient = new ComprehendClient({});

  const detectDominantLanguageCommand = new DetectDominantLanguageCommand({
    Text: extractTextOutput.source_text,
  });

  // The source language is required for sentiment analysis and
  // translation in the next step.
  const { Languages } = await comprehendClient.send(
    detectDominantLanguageCommand,
  );
```

```
  const languageCode = Languages[0].LanguageCode;

  const detectSentimentCommand = new DetectSentimentCommand({
    Text: extractTextOutput.source_text,
    LanguageCode: languageCode,
  });

  const { Sentiment } = await comprehendClient.send(detectSentimentCommand);

  return {
    sentiment: Sentiment,
    language_code: languageCode,
  };
};
```

```
import {
  DetectDocumentTextCommand,
  TextractClient,
} from "@aws-sdk/client-textract";

/**
 * Fetch the S3 object from the event and analyze it using Amazon Textract.
 *
 * @param {import("@types/aws-lambda").EventBridgeEvent<"Object Created">}
 eventBridgeS3Event
 */
export const handler = async (eventBridgeS3Event) => {
  const textractClient = new TextractClient();

  const detectDocumentTextCommand = new DetectDocumentTextCommand({
    Document: {
      S3Object: {
        Bucket: eventBridgeS3Event.bucket,
        Name: eventBridgeS3Event.object,
      },
    },
  });

  // Textract returns a list of blocks. A block can be a line, a page, word, etc.
  // Each block also contains geometry of the detected text.
  // For more information on the Block type, see https://docs.aws.amazon.com/
textract/latest/dg/API_Block.html.
```

```
  const { Blocks } = await textractClient.send(detectDocumentTextCommand);

  // For the purpose of this example, we are only interested in words.
  const extractedWords = Blocks.filter((b) => b.BlockType === "WORD").map(
    (b) => b.Text,
  );

  return extractedWords.join(" ");
};
```

```
import { PollyClient, SynthesizeSpeechCommand } from "@aws-sdk/client-polly";
import { S3Client } from "@aws-sdk/client-s3";
import { Upload } from "@aws-sdk/lib-storage";

/**
 * Synthesize an audio file from text.
 *
 * @param {{ bucket: string, translated_text: string, object: string}}
 sourceDestinationConfig
 */
export const handler = async (sourceDestinationConfig) => {
  const pollyClient = new PollyClient({});

  const synthesizeSpeechCommand = new SynthesizeSpeechCommand({
    Engine: "neural",
    Text: sourceDestinationConfig.translated_text,
    VoiceId: "Ruth",
    OutputFormat: "mp3",
  });

  const { AudioStream } = await pollyClient.send(synthesizeSpeechCommand);

  const audioKey = `${sourceDestinationConfig.object}.mp3`;

  // Store the audio file in S3.
  const s3Client = new S3Client();
  const upload = new Upload({
    client: s3Client,
    params: {
      Bucket: sourceDestinationConfig.bucket,
      Key: audioKey,
      Body: AudioStream,
      ContentType: "audio/mp3",
```

```
    },
  });

  await upload.done();
  return audioKey;
};
```

```
import {
  TranslateClient,
  TranslateTextCommand,
} from "@aws-sdk/client-translate";

/**
 * Translate the extracted text to English.
 *
 * @param {{ extracted_text: string, source_language_code: string}}
 textAndSourceLanguage
 */
export const handler = async (textAndSourceLanguage) => {
  const translateClient = new TranslateClient({});

  const translateCommand = new TranslateTextCommand({
    SourceLanguageCode: textAndSourceLanguage.source_language_code,
    TargetLanguageCode: "en",
    Text: textAndSourceLanguage.extracted_text,
  });

  const { TranslatedText } = await translateClient.send(translateCommand);

  return { translated_text: TranslatedText };
};
```

此範例中使用的服務

- Amazon Comprehend
- Lambda
- Amazon Polly
- Amazon Textract
- Amazon Translate

Amazon DocumentDB 示例使用 SDK for JavaScript (v3)

下列程式碼範例說明如何透過使用 AWS SDK for JavaScript (v3) 搭配 Amazon DocumentDB 來執行動作和實作常見案例。

每個範例都包含完整原始程式碼的連結，您可以在其中找到如何在內容中設定和執行程式碼的指示。

主題

無伺服器範例

從 Amazon DocumentDB 觸發器調用 Lambda 函數

下列程式碼範例示範如何實作 Lambda 函數，此函數會接收由 DocumentDB 變更串流接收記錄所觸發的事件。該函數檢索 DocumentDB 有效載荷和記錄的內容。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在無伺服器範例儲存庫中設定和執行。

使用 Amazon DocumentDB 事件與 Lambda 使用. JavaScript

```
console.log('Loading function');
exports.handler = async (event, context) => {
    event.events.forEach(record => {
        logDocumentDBEvent(record);
    });
    return 'OK';
};

const logDocumentDBEvent = (record) => {
    console.log('Operation type: ' + record.event.operationType);
    console.log('db: ' + record.event.ns.db);
    console.log('collection: ' + record.event.ns.coll);
    console.log('Full document:', JSON.stringify(record.event.fullDocument, null,
 2));
```

```
};
```

使用使用 Lambda 使用 Amazon DocumentDB 事件 TypeScript

```
import { DocumentDBEventRecord, DocumentDBEventSubscriptionContext } from 'aws-
lambda';

console.log('Loading function');

export const handler = async (
  event: DocumentDBEventSubscriptionContext,
  context: any
): Promise<string> => {
  event.events.forEach((record: DocumentDBEventRecord) => {
    logDocumentDBEvent(record);
  });
  return 'OK';
};

const logDocumentDBEvent = (record: DocumentDBEventRecord): void => {
  console.log('Operation type: ' + record.event.operationType);
  console.log('db: ' + record.event.ns.db);
  console.log('collection: ' + record.event.ns.coll);
  console.log('Full document:', JSON.stringify(record.event.fullDocument, null, 2));
};
```

使用SDK的範例 JavaScript (v3)

下列程式碼範例說明如何透過使用 AWS SDK for JavaScript (v3) 搭配 DynamoDB 來執行動作和實作常見案例。

基本概念是程式碼範例，會示範如何在服務中執行基本作業。

Actions 是大型程式的程式碼摘錄，必須在內容中執行。雖然動作會顯示如何呼叫個別服務函數，但您可以在其相關案例中查看內容中的動作。

案例是程式碼範例，向您展示如何透過呼叫服務中的多個函式或與其他函式結合來完成特定工作 AWS 服務。

每個範例都包含完整原始程式碼的連結，您可以在其中找到如何在內容中設定和執行程式碼的指示。

開始使用

Hello DynamoDB

下列程式碼範例示範如何開始使用 DynamoDB。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

如需有關在中使用 DynamoDB 的詳細資訊 AWS SDK for JavaScript，請參閱使用. JavaScript

```
import { ListTablesCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new ListTablesCommand({});

  const response = await client.send(command);
  console.log(response.TableNames.join("\n"));
  return response;
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考ListTables中的。

主題

- 基本概念
- 動作
- 案例
- 無伺服器範例

基本概念

學習基礎知識

以下程式碼範例顯示做法：

- 建立可存放電影資料的資料表。
- 放入、取得和更新資料表中的單個電影。
- 從範例JSON檔案將影片資料寫入資料表。
- 查詢特定年份發表的電影。
- 掃描某個年份範圍內發表的電影。
- 從資料表刪除電影，然後刪除資料表。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import { readFileSync } from "fs";
import {
  BillingMode,
  CreateTableCommand,
  DeleteTableCommand,
  DynamoDBClient,
  waitUntilTableExists,
} from "@aws-sdk/client-dynamodb";

/**
 * This module is a convenience library. It abstracts Amazon DynamoDB's data type
 * descriptors (such as S, N, B, and BOOL) by marshalling JavaScript objects into
 * AttributeValue shapes.
 */
import {
  BatchWriteCommand,
  DeleteCommand,
  DynamoDBDocumentClient,
```

```
  GetCommand,
  PutCommand,
  UpdateCommand,
  paginateQuery,
  paginateScan,
} from "@aws-sdk/lib-dynamodb";

// These modules are local to our GitHub repository. We recommend cloning
// the project from GitHub if you want to run this example.
// For more information, see https://github.com/awsdocs/aws-doc-sdk-examples.
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { dirnameFromMetaUrl } from "@aws-doc-sdk-examples/lib/utils/util-fs.js";
import { chunkArray } from "@aws-doc-sdk-examples/lib/utils/util-array.js";

const dirname = dirnameFromMetaUrl(import.meta.url);
const tableName = getUniqueName("Movies");
const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

const log = (msg) => console.log(`[SCENARIO] ${msg}`);

export const main = async () => {
  /**
   * Create a table.
   */

  const createTableCommand = new CreateTableCommand({
    TableName: tableName,
    // This example performs a large write to the database.
    // Set the billing mode to PAY_PER_REQUEST to
    // avoid throttling the large write.
    BillingMode: BillingMode.PAY_PER_REQUEST,
    // Define the attributes that are necessary for the key schema.
    AttributeDefinitions: [
      {
        AttributeName: "year",
        // 'N' is a data type descriptor that represents a number type.
        // For a list of all data type descriptors, see the following link.
        // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Programming.LowLevelAPI.html#Programming.LowLevelAPI.DataTypeDescriptors
        AttributeType: "N",
      },
      { AttributeName: "title", AttributeType: "S" },
    ],
```

```
    // The KeySchema defines the primary key. The primary key can be
    // a partition key, or a combination of a partition key and a sort key.
    // Key schema design is important. For more info, see
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/best-
practices.html
    KeySchema: [
      // The way your data is accessed determines how you structure your keys.
      // The movies table will be queried for movies by year. It makes sense
      // to make year our partition (HASH) key.
      { AttributeName: "year", KeyType: "HASH" },
      { AttributeName: "title", KeyType: "RANGE" },
    ],
  });

  log("Creating a table.");
  const createTableResponse = await client.send(createTableCommand);
  log(`Table created: ${JSON.stringify(createTableResponse.TableDescription)}`);

  // This polls with DescribeTableCommand until the requested table is 'ACTIVE'.
  // You can't write to a table before it's active.
  log("Waiting for the table to be active.");
  await waitUntilTableExists({ client }, { TableName: tableName });
  log("Table active.");

  /**
   * Add a movie to the table.
   */

  log("Adding a single movie to the table.");
  // PutCommand is the first example usage of 'lib-dynamodb'.
  const putCommand = new PutCommand({
    TableName: tableName,
    Item: {
      // In 'client-dynamodb', the AttributeValue would be required (`year: { N:
 1981 }`)
      // 'lib-dynamodb' simplifies the usage ( `year: 1981` )
      year: 1981,
      // The preceding KeySchema defines 'title' as our sort (RANGE) key, so 'title'
      // is required.
      title: "The Evil Dead",
      // Every other attribute is optional.
      info: {
        genres: ["Horror"],
      },
```

```
    },
  });
  await docClient.send(putCommand);
  log("The movie was added.");

  /**
   * Get a movie from the table.
   */

  log("Getting a single movie from the table.");
  const getCommand = new GetCommand({
    TableName: tableName,
    // Requires the complete primary key. For the movies table, the primary key
    // is only the id (partition key).
    Key: {
      year: 1981,
      title: "The Evil Dead",
    },
    // Set this to make sure that recent writes are reflected.
    // For more information, see https://docs.aws.amazon.com/amazondynamodb/latest/
developerguide/HowItWorks.ReadConsistency.html.
    ConsistentRead: true,
  });
  const getResponse = await docClient.send(getCommand);
  log(`Got the movie: ${JSON.stringify(getResponse.Item)}`);

  /**
   * Update a movie in the table.
   */

  log("Updating a single movie in the table.");
  const updateCommand = new UpdateCommand({
    TableName: tableName,
    Key: { year: 1981, title: "The Evil Dead" },
    // This update expression appends "Comedy" to the list of genres.
    // For more information on update expressions, see
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Expressions.UpdateExpressions.html
    UpdateExpression: "set #i.#g = list_append(#i.#g, :vals)",
    ExpressionAttributeNames: { "#i": "info", "#g": "genres" },
    ExpressionAttributeValues: {
      ":vals": ["Comedy"],
    },
    ReturnValues: "ALL_NEW",
```

```
});
const updateResponse = await docClient.send(updateCommand);
log(`Movie updated: ${JSON.stringify(updateResponse.Attributes)}`);

/**
 * Delete a movie from the table.
 */

log("Deleting a single movie from the table.");
const deleteCommand = new DeleteCommand({
  TableName: tableName,
  Key: { year: 1981, title: "The Evil Dead" },
});
await client.send(deleteCommand);
log("Movie deleted.");

/**
 * Upload a batch of movies.
 */

log("Adding movies from local JSON file.");
const file = readFileSync(
  `${dirname}../../../../resources/sample_files/movies.json`,
);
const movies = JSON.parse(file.toString());
// chunkArray is a local convenience function. It takes an array and returns
// a generator function. The generator function yields every N items.
const movieChunks = chunkArray(movies, 25);
// For every chunk of 25 movies, make one BatchWrite request.
for (const chunk of movieChunks) {
  const putRequests = chunk.map((movie) => ({
    PutRequest: {
      Item: movie,
    },
  }));

  const command = new BatchWriteCommand({
    RequestItems: {
      [tableName]: putRequests,
    },
  });

  await docClient.send(command);
}
```

```
  log("Movies added.");

  /**
   * Query for movies by year.
   */

  log("Querying for all movies from 1981.");
  const paginatedQuery = paginateQuery(
    { client: docClient },
    {
      TableName: tableName,
      //For more information about query expressions, see
      // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Query.html#Query.KeyConditionExpressions
      KeyConditionExpression: "#y = :y",
      // 'year' is a reserved word in DynamoDB. Indicate that it's an attribute
      // name by using an expression attribute name.
      ExpressionAttributeNames: { "#y": "year" },
      ExpressionAttributeValues: { ":y": 1981 },
      ConsistentRead: true,
    },
  );
  /**
   * @type { Record<string, any>[] };
   */
  const movies1981 = [];
  for await (const page of paginatedQuery) {
    movies1981.push(...page.Items);
  }
  log(`Movies: ${movies1981.map((m) => m.title).join(", ")}`);

  /**
   * Scan the table for movies between 1980 and 1990.
   */

  log(`Scan for movies released between 1980 and 1990`);
  // A 'Scan' operation always reads every item in the table. If your design
 requires
  // the use of 'Scan', consider indexing your table or changing your design.
  // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/bp-query-
scan.html
  const paginatedScan = paginateScan(
    { client: docClient },
    {
```

```
      TableName: tableName,
      // Scan uses a filter expression instead of a key condition expression. Scan
 will
      // read the entire table and then apply the filter.
      FilterExpression: "#y between :y1 and :y2",
      ExpressionAttributeNames: { "#y": "year" },
      ExpressionAttributeValues: { ":y1": 1980, ":y2": 1990 },
      ConsistentRead: true,
    },
  );
  /**
   * @type { Record<string, any>[] };
   */
  const movies1980to1990 = [];
  for await (const page of paginatedScan) {
    movies1980to1990.push(...page.Items);
  }
  log(
    `Movies: ${movies1980to1990
      .map((m) => `${m.title} (${m.year})`)
      .join(", ")}`,
  );

  /**
   * Delete the table.
   */

  const deleteTableCommand = new DeleteTableCommand({ TableName: tableName });
  log(`Deleting table ${tableName}.`);
  await client.send(deleteTableCommand);
  log("Table deleted.");
};
```

- 如需詳API細資訊，請參閱「AWS SDK for JavaScript API參考」中的下列主題。
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem

- PutItem
- 查詢
- 掃描
- UpdateItem

動作

`BatchExecuteStatement`

下列程式碼範例會示範如何使用BatchExecuteStatement。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

使用 PartiQL 建立一批項目。

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

import {
  DynamoDBDocumentClient,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const breakfastFoods = ["Eggs", "Bacon", "Sausage"];
  const command = new BatchExecuteStatementCommand({
    Statements: breakfastFoods.map((food) => ({
      Statement: `INSERT INTO BreakfastFoods value {'Name':?}`,
      Parameters: [food],
    })),
  });
```

```
  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

使用 PartiQL 取得一批項目。

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

import {
  DynamoDBDocumentClient,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new BatchExecuteStatementCommand({
    Statements: [
      {
        Statement: "SELECT * FROM PepperMeasurements WHERE Unit=?",
        Parameters: ["Teaspoons"],
        ConsistentRead: true,
      },
      {
        Statement: "SELECT * FROM PepperMeasurements WHERE Unit=?",
        Parameters: ["Grams"],
        ConsistentRead: true,
      },
    ],
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

使用 PartiQL 更新一批項目。

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
```

```
import {
  DynamoDBDocumentClient,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const eggUpdates = [
    ["duck", "fried"],
    ["chicken", "omelette"],
  ];
  const command = new BatchExecuteStatementCommand({
    Statements: eggUpdates.map((change) => ({
      Statement: "UPDATE Eggs SET Style=? where Variety=?",
      Parameters: [change[1], change[0]],
    })),
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

使用 PartiQL 刪除一批項目。

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

import {
  DynamoDBDocumentClient,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new BatchExecuteStatementCommand({
    Statements: [
      {
```

```
        Statement: "DELETE FROM Flavors where Name=?",
        Parameters: ["Grape"],
      },
      {
        Statement: "DELETE FROM Flavors where Name=?",
        Parameters: ["Strawberry"],
      },
    ],
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考BatchExecuteStatement中的。

`BatchGetItem`

下列程式碼範例會示範如何使用`BatchGetItem`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

此範例使用文件用戶端來簡化 DynamoDB 中處理項目的作業。API詳情請參閱BatchGet。

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { BatchGetCommand, DynamoDBDocumentClient } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new BatchGetCommand({
    // Each key in this object is the name of a table. This example refers
    // to a Books table.
```

```
    RequestItems: {
      Books: {
        // Each entry in Keys is an object that specifies a primary key.
        Keys: [
          {
            Title: "How to AWS",
          },
          {
            Title: "DynamoDB for DBAs",
          },
        ],
        // Only return the "Title" and "PageCount" attributes.
        ProjectionExpression: "Title, PageCount",
      },
    },
  });

  const response = await docClient.send(command);
  console.log(response.Responses["Books"]);
  return response;
};
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考BatchGetItem中的。

SDK對於 JavaScript （ 第 2 個 ）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });
```

```
var params = {
  RequestItems: {
    TABLE_NAME: {
      Keys: [
        { KEY_NAME: { N: "KEY_VALUE_1" } },
        { KEY_NAME: { N: "KEY_VALUE_2" } },
        { KEY_NAME: { N: "KEY_VALUE_3" } },
      ],
      ProjectionExpression: "KEY_NAME, ATTRIBUTE",
    },
  },
};

ddb.batchGetItem(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    data.Responses.TABLE_NAME.forEach(function (element, index, array) {
      console.log(element);
    });
  }
});
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考BatchGetItem中的。

`BatchWriteItem`

下列程式碼範例會示範如何使用`BatchWriteItem`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

此範例使用文件用戶端來簡化 DynamoDB 中處理項目的作業。API詳情請參閱BatchWrite。

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
```

```
import {
  BatchWriteCommand,
  DynamoDBDocumentClient,
} from "@aws-sdk/lib-dynamodb";
import { readFileSync } from "fs";

// These modules are local to our GitHub repository. We recommend cloning
// the project from GitHub if you want to run this example.
// For more information, see https://github.com/awsdocs/aws-doc-sdk-examples.
import { dirnameFromMetaUrl } from "@aws-doc-sdk-examples/lib/utils/util-fs.js";
import { chunkArray } from "@aws-doc-sdk-examples/lib/utils/util-array.js";

const dirname = dirnameFromMetaUrl(import.meta.url);

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const file = readFileSync(
    `${dirname}../../../../../resources/sample_files/movies.json`,
  );

  const movies = JSON.parse(file.toString());

  // chunkArray is a local convenience function. It takes an array and returns
  // a generator function. The generator function yields every N items.
  const movieChunks = chunkArray(movies, 25);

  // For every chunk of 25 movies, make one BatchWrite request.
  for (const chunk of movieChunks) {
    const putRequests = chunk.map((movie) => ({
      PutRequest: {
        Item: movie,
      },
    }));

    const command = new BatchWriteCommand({
      RequestItems: {
        // An existing table is required. A composite key of 'title' and 'year' is
 recommended
        // to account for duplicate titles.
        ["BatchWriteMoviesTable"]: putRequests,
      },
    });
```

```
    await docClient.send(command);
  }
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考BatchWriteItem中的。

SDK對於 JavaScript（第 2 個）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  RequestItems: {
    TABLE_NAME: [
      {
        PutRequest: {
          Item: {
            KEY: { N: "KEY_VALUE" },
            ATTRIBUTE_1: { S: "ATTRIBUTE_1_VALUE" },
            ATTRIBUTE_2: { N: "ATTRIBUTE_2_VALUE" },
          },
        },
      },
      {
        PutRequest: {
          Item: {
            KEY: { N: "KEY_VALUE" },
            ATTRIBUTE_1: { S: "ATTRIBUTE_1_VALUE" },
            ATTRIBUTE_2: { N: "ATTRIBUTE_2_VALUE" },
          },
```

```
        },
      },
    ],
  },
};

ddb.batchWriteItem(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考BatchWriteItem中的。

`CreateTable`

下列程式碼範例會示範如何使用`CreateTable`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import { CreateTableCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new CreateTableCommand({
    TableName: "EspressoDrinks",
    // For more information about data types,
    // see https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
HowItWorks.NamingRulesDataTypes.html#HowItWorks.DataTypes and
```

```
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Programming.LowLevelAPI.html#Programming.LowLevelAPI.DataTypeDescriptors
    AttributeDefinitions: [
      {
        AttributeName: "DrinkName",
        AttributeType: "S",
      },
    ],
    KeySchema: [
      {
        AttributeName: "DrinkName",
        KeyType: "HASH",
      },
    ],
    ProvisionedThroughput: {
      ReadCapacityUnits: 1,
      WriteCapacityUnits: 1,
    },
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考CreateTable中的。

SDK對於 JavaScript（第 2 個）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });
```

```
// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  AttributeDefinitions: [
    {
      AttributeName: "CUSTOMER_ID",
      AttributeType: "N",
    },
    {
      AttributeName: "CUSTOMER_NAME",
      AttributeType: "S",
    },
  ],
  KeySchema: [
    {
      AttributeName: "CUSTOMER_ID",
      KeyType: "HASH",
    },
    {
      AttributeName: "CUSTOMER_NAME",
      KeyType: "RANGE",
    },
  ],
  ProvisionedThroughput: {
    ReadCapacityUnits: 1,
    WriteCapacityUnits: 1,
  },
  TableName: "CUSTOMER_LIST",
  StreamSpecification: {
    StreamEnabled: false,
  },
};

// Call DynamoDB to create the table
ddb.createTable(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Table Created", data);
  }
});
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考CreateTable中的。

`DeleteItem`

下列程式碼範例會示範如何使用`DeleteItem`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

此範例使用文件用戶端來簡化 DynamoDB 中處理項目的作業。API詳情請參閱DeleteCommand。

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { DynamoDBDocumentClient, DeleteCommand } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new DeleteCommand({
    TableName: "Sodas",
    Key: {
      Flavor: "Cola",
    },
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考DeleteItem中的。

SDK對於 JavaScript （第 2 個）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

從資料表刪除項目。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  TableName: "TABLE",
  Key: {
    KEY_NAME: { N: "VALUE" },
  },
};

// Call DynamoDB to delete the item from the table
ddb.deleteItem(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

使用 DynamoDB 文件用戶端從資料表刪除項目。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });
```

```
// Create DynamoDB document client
var docClient = new AWS.DynamoDB.DocumentClient({ apiVersion: "2012-08-10" });

var params = {
  Key: {
    HASH_KEY: VALUE,
  },
  TableName: "TABLE",
};

docClient.delete(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考DeleteItem中的。

`DeleteTable`

下列程式碼範例會示範如何使用DeleteTable。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import { DeleteTableCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new DeleteTableCommand({
    TableName: "DecafCoffees",
```

```
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考DeleteTable中的。

SDK對於 JavaScript（第 2 個）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  TableName: process.argv[2],
};

// Call DynamoDB to delete the specified table
ddb.deleteTable(params, function (err, data) {
  if (err && err.code === "ResourceNotFoundException") {
    console.log("Error: Table not found");
  } else if (err && err.code === "ResourceInUseException") {
    console.log("Error: Table in use");
  } else {
    console.log("Success", data);
  }
});
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考DeleteTable中的。

`DescribeTable`

下列程式碼範例會示範如何使用`DescribeTable`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import { DescribeTableCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new DescribeTableCommand({
    TableName: "Pastries",
  });

  const response = await client.send(command);
  console.log(`TABLE NAME: ${response.Table.TableName}`);
  console.log(`TABLE ITEM COUNT: ${response.Table.ItemCount}`);
  return response;
};
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考DescribeTable中的。

SDK對於 JavaScript（第 2 個）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  TableName: process.argv[2],
};

// Call DynamoDB to retrieve the selected table descriptions
ddb.describeTable(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.Table.KeySchema);
  }
});
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考DescribeTable中的。

`DescribeTimeToLive`

下列程式碼範例會示範如何使用`DescribeTimeToLive`。

SDK對於 JavaScript（3）

```
import { DynamoDBClient, DescribeTimeToLiveCommand } from "@aws-sdk/client-
dynamodb";

const describeTableTTL = async (tableName, region) => {
    const client = new DynamoDBClient({
        region: region,
        endpoint: `https://dynamodb.${region}.amazonaws.com`
    });

    try {
```

```
        const ttlDescription = await client.send(new
 DescribeTimeToLiveCommand({ TableName: tableName }));

        if (ttlDescription.TimeToLiveDescription.TimeToLiveStatus === 'ENABLED') {
            console.log("TTL is enabled for table %s.", tableName);
        } else {
            console.log("TTL is not enabled for table %s.", tableName);
        }

        return ttlDescription;
    } catch (e) {
        console.error(`Error describing table: ${e}`);
        throw e;
    }
}

// enter table name and change region if desired.
describeTableTTL('your-table-name', 'us-east-1');
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考DescribeTimeToLive中的。

`ExecuteStatement`

下列程式碼範例會示範如何使用`ExecuteStatement`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

使用 PartiQL 建立項目。

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

import {
  ExecuteStatementCommand,
  DynamoDBDocumentClient,
} from "@aws-sdk/lib-dynamodb";
```

```
const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new ExecuteStatementCommand({
    Statement: `INSERT INTO Flowers value {'Name':?}`,
    Parameters: ["Rose"],
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

使用 PartiQL 取得項目。

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

import {
  ExecuteStatementCommand,
  DynamoDBDocumentClient,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new ExecuteStatementCommand({
    Statement: "SELECT * FROM CloudTypes WHERE IsStorm=?",
    Parameters: [false],
    ConsistentRead: true,
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

使用 PartiQL 更新項目。

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
```

```
import {
  ExecuteStatementCommand,
  DynamoDBDocumentClient,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new ExecuteStatementCommand({
    Statement: "UPDATE EyeColors SET IsRecessive=? where Color=?",
    Parameters: [true, "blue"],
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

使用 PartiQL 刪除項目。

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

import {
  ExecuteStatementCommand,
  DynamoDBDocumentClient,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new ExecuteStatementCommand({
    Statement: "DELETE FROM PaintColors where Name=?",
    Parameters: ["Purple"],
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考ExecuteStatement中的。

`GetItem`

下列程式碼範例會示範如何使用`GetItem`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

此範例使用文件用戶端來簡化 DynamoDB 中處理項目的作業。API詳情請參閱GetCommand。

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { DynamoDBDocumentClient, GetCommand } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new GetCommand({
    TableName: "AngryAnimals",
    Key: {
      CommonName: "Shoebill",
    },
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考GetItem中的。

SDK對於 JavaScript （第 2 個）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

從資料表取得項目。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  TableName: "TABLE",
  Key: {
    KEY_NAME: { N: "001" },
  },
  ProjectionExpression: "ATTRIBUTE_NAME",
};

// Call DynamoDB to read the item from the table
ddb.getItem(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.Item);
  }
});
```

使用 DynamoDB 文件用戶端從資料表取得項目。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });
```

```
// Create DynamoDB document client
var docClient = new AWS.DynamoDB.DocumentClient({ apiVersion: "2012-08-10" });

var params = {
  TableName: "EPISODES_TABLE",
  Key: { KEY_NAME: VALUE },
};

docClient.get(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.Item);
  }
});
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考GetItem中的。

`ListTables`

下列程式碼範例會示範如何使用`ListTables`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import { ListTablesCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new ListTablesCommand({});

  const response = await client.send(command);
```

```
  console.log(response);
  return response;
};
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考ListTables中的。

SDK對於 JavaScript（第 2 個）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

// Call DynamoDB to retrieve the list of tables
ddb.listTables({ Limit: 10 }, function (err, data) {
  if (err) {
    console.log("Error", err.code);
  } else {
    console.log("Table names are ", data.TableNames);
  }
});
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考ListTables中的。

`PutItem`

下列程式碼範例會示範如何使用`PutItem`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

此範例使用文件用戶端來簡化 DynamoDB 中處理項目的作業。API詳情請參閱PutCommand。

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { PutCommand, DynamoDBDocumentClient } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new PutCommand({
    TableName: "HappyAnimals",
    Item: {
      CommonName: "Shiba Inu",
    },
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考PutItem中的。

SDK對於 JavaScript（第 2 個）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

將項目放入資料表。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  TableName: "CUSTOMER_LIST",
  Item: {
    CUSTOMER_ID: { N: "001" },
    CUSTOMER_NAME: { S: "Richard Roe" },
  },
};

// Call DynamoDB to add the item to the table
ddb.putItem(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

使用 DynamoDB 文件用戶端將項目放入資料表。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create DynamoDB document client
var docClient = new AWS.DynamoDB.DocumentClient({ apiVersion: "2012-08-10" });

var params = {
  TableName: "TABLE",
  Item: {
    HASHKEY: VALUE,
    ATTRIBUTE_1: "STRING_VALUE",
    ATTRIBUTE_2: VALUE_2,
  },
```

```
};

docClient.put(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考PutItem中的。

Query

下列程式碼範例會示範如何使用Query。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

此範例使用文件用戶端來簡化 DynamoDB 中處理項目的作業。API詳情請參閱QueryCommand。

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { QueryCommand, DynamoDBDocumentClient } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new QueryCommand({
    TableName: "CoffeeCrop",
    KeyConditionExpression:
      "OriginCountry = :originCountry AND RoastDate > :roastDate",
    ExpressionAttributeValues: {
      ":originCountry": "Ethiopia",
      ":roastDate": "2023-05-01",
```

```
    },
    ConsistentRead: true,
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考中的查詢。

SDK對於 JavaScript（第 2 個）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create DynamoDB document client
var docClient = new AWS.DynamoDB.DocumentClient({ apiVersion: "2012-08-10" });

var params = {
  ExpressionAttributeValues: {
    ":s": 2,
    ":e": 9,
    ":topic": "PHRASE",
  },
  KeyConditionExpression: "Season = :s and Episode > :e",
  FilterExpression: "contains (Subtitle, :topic)",
  TableName: "EPISODES_TABLE",
};

docClient.query(params, function (err, data) {
  if (err) {
```

```
    console.log("Error", err);
  } else {
    console.log("Success", data.Items);
  }
});
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考中的查詢。

Scan

下列程式碼範例會示範如何使用Scan。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

此範例使用文件用戶端來簡化 DynamoDB 中處理項目的作業。API詳情請參閱ScanCommand。

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { DynamoDBDocumentClient, ScanCommand } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new ScanCommand({
    ProjectionExpression: "#Name, Color, AvgLifeSpan",
    ExpressionAttributeNames: { "#Name": "Name" },
    TableName: "Birds",
  });

  const response = await docClient.send(command);
  for (const bird of response.Items) {
    console.log(`${bird.Name} - (${bird.Color}, ${bird.AvgLifeSpan})`);
  }
  return response;
```

```
};
```

- 如需詳API細資訊，請參閱在AWS SDK for JavaScript API參考中掃描。

SDK對於 JavaScript（第 2 個）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
// Load the AWS SDK for Node.js.
var AWS = require("aws-sdk");
// Set the AWS Region.
AWS.config.update({ region: "REGION" });

// Create DynamoDB service object.
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

const params = {
  // Specify which items in the results are returned.
  FilterExpression: "Subtitle = :topic AND Season = :s AND Episode = :e",
  // Define the expression attribute value, which are substitutes for the values you
 want to compare.
  ExpressionAttributeValues: {
    ":topic": { S: "SubTitle2" },
    ":s": { N: 1 },
    ":e": { N: 2 },
  },
  // Set the projection expression, which are the attributes that you want.
  ProjectionExpression: "Season, Episode, Title, Subtitle",
  TableName: "EPISODES_TABLE",
};

ddb.scan(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
    data.Items.forEach(function (element, index, array) {
      console.log(
```

```
        "printing",
        element.Title.S + " (" + element.Subtitle.S + ")"
      );
    });
  }
});
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱在AWS SDK for JavaScript API參考中掃描。

`UpdateItem`

下列程式碼範例會示範如何使用`UpdateItem`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

此範例使用文件用戶端來簡化 DynamoDB 中處理項目的作業。API詳情請參閱UpdateCommand。

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { DynamoDBDocumentClient, UpdateCommand } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new UpdateCommand({
    TableName: "Dogs",
    Key: {
      Breed: "Labrador",
    },
    UpdateExpression: "set Color = :color",
    ExpressionAttributeValues: {
      ":color": "black",
    },
```

```
    ReturnValues: "ALL_NEW",
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考UpdateItem中的。

`UpdateTimeToLive`

下列程式碼範例會示範如何使用UpdateTimeToLive。

SDK對於 JavaScript（3）

在現有TTL的 DynamoDB 資料表上啟用。

```
import { DynamoDBClient, UpdateTimeToLiveCommand } from "@aws-sdk/client-dynamodb";

const enableTTL = async (tableName, ttlAttribute) => {

    const client = new DynamoDBClient({});

    const params = {
        TableName: tableName,
        TimeToLiveSpecification: {
            Enabled: true,
            AttributeName: ttlAttribute
        }
    };

    try {
        const response = await client.send(new UpdateTimeToLiveCommand(params));
        if (response.$metadata.httpStatusCode === 200) {
            console.log(`TTL enabled successfully for table ${tableName}, using
 attribute name ${ttlAttribute}.`);
        } else {
            console.log(`Failed to enable TTL for table ${tableName}, response
 object: ${response}`);
        }
        return response;
```

```
    } catch (e) {
        console.error(`Error enabling TTL: ${e}`);
        throw e;
    }
};

// call with your own values
enableTTL('ExampleTable', 'exampleTtlAttribute');
```

在現有的 DynamoDB 資料TTL表上停用。

```
import { DynamoDBClient, UpdateTimeToLiveCommand } from "@aws-sdk/client-dynamodb";

const disableTTL = async (tableName, ttlAttribute) => {

    const client = new DynamoDBClient({});

    const params = {
        TableName: tableName,
        TimeToLiveSpecification: {
            Enabled: false,
            AttributeName: ttlAttribute
        }
    };

    try {
        const response = await client.send(new UpdateTimeToLiveCommand(params));
        if (response.$metadata.httpStatusCode === 200) {
            console.log(`TTL disabled successfully for table ${tableName}, using
 attribute name ${ttlAttribute}.`);
        } else {
            console.log(`Failed to disable TTL for table ${tableName}, response
 object: ${response}`);
        }
        return response;
    } catch (e) {
        console.error(`Error disabling TTL: ${e}`);
        throw e;
    }
};

// call with your own values
disableTTL('ExampleTable', 'exampleTtlAttribute');
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考UpdateTimeToLive中的。

案例

建置應用程式以將資料提交至 DynamoDB 資料表

下列程式碼範例顯示如何建立將資料提交至 Amazon DynamoDB 表的應用程式，並在使用者更新表格時通知您。

SDK對於 JavaScript（3）

此範例說明如何建立可讓使用者將資料提交至 Amazon DynamoDB 表格的應用程式，以及如何使用 Amazon 簡單通知服務 (AmazonSNS) 傳送文字訊息給管理員。

有關如何設置和運行的完整源代碼和說明，請參閱中的完整示例GitHub。

此範例也可在 AWS SDK for JavaScript v3 開發人員指南中取得。

此範例中使用的服務

- DynamoDB
- Amazon SNS

有條件地更新項目 TTL

下列程式碼範例會示範如何有條件地更新項目。TTL

SDK對於 JavaScript（3）

針TTL對資料表中現有 DynamoDB 項目進行更新，且具有條件。

```
import { DynamoDBClient, UpdateItemCommand } from "@aws-sdk/client-dynamodb";
import { marshall, unmarshall } from "@aws-sdk/util-dynamodb";

const updateDynamoDBItem = async (tableName, region, partitionKey, sortKey,
 newAttribute) => {
    const client = new DynamoDBClient({
        region: region,
        endpoint: `https://dynamodb.${region}.amazonaws.com`
    });
```

```
    const currentTime = Math.floor(Date.now() / 1000);

    const params = {
        TableName: tableName,
        Key: marshall({
            artist: partitionKey,
            album: sortKey
        }),
        UpdateExpression: "SET newAttribute = :newAttribute",
        ConditionExpression: "expireAt > :expiration",
        ExpressionAttributeValues: marshall({
            ':newAttribute': newAttribute,
            ':expiration': currentTime
        }),
        ReturnValues: "ALL_NEW"
    };

    try {
        const response = await client.send(new UpdateItemCommand(params));
        const responseData = unmarshall(response.Attributes);
        console.log("Item updated successfully: ", responseData);
        return responseData;
    } catch (error) {
        if (error.name === "ConditionalCheckFailedException") {
            console.log("Condition check failed: Item's 'expireAt' is expired.");
        } else {
            console.error("Error updating item: ", error);
        }
        throw error;
    }
};

// Enter your values here
updateDynamoDBItem('your-table-name', "us-east-1",'your-partition-key-value', 'your-
sort-key-value', 'your-new-attribute-value');
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考UpdateItem中的。

建立無伺服器應用程式來管理相片

下列程式碼範例示範如何建立無伺服器應用程式，讓使用者以標籤管理相片。

SDK對於 JavaScript（3）

顯示如何開發照片資產管理應用程式，以便使用 Amazon Rekognition 偵測圖片中的標籤，並將其儲存以供日後擷取。

有關如何設置和運行的完整源代碼和說明，請參閱中的完整示例 GitHub。

如要深入探索此範例的來源，請參閱 AWS 社群上的文章。

此範例中使用的服務

- API閘道
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

建立 Web 應用程式以追蹤 DynamoDB 資料

下列程式碼範例示範如何建立可追蹤 Amazon DynamoDB 表中工作項目的 Web 應用程式，並使用 Amazon 簡單電子郵件服務 (AmazonSES) 傳送報告。

SDK對於 JavaScript（3）

示範如何使用 Amazon DynamoDB API 建立可追蹤 DynamoDB 工作資料的動態 Web 應用程式。

有關如何設置和運行的完整源代碼和說明，請參閱中的完整示例GitHub。

此範例中使用的服務

- DynamoDB
- Amazon SES

創建一個物件 TTL

下列程式碼範例會示範如何使用建立項目TTL。

SDK對於 JavaScript（3）

```
import { DynamoDBClient, PutItemCommand } from "@aws-sdk/client-dynamodb";
```

```
function createDynamoDBItem(table_name, region, partition_key, sort_key) {
    const client = new DynamoDBClient({
        region: region,
        endpoint: `https://dynamodb.${region}.amazonaws.com`
    });

    // Get the current time in epoch second format
    const current_time = Math.floor(new Date().getTime() / 1000);

    // Calculate the expireAt time (90 days from now) in epoch second format
    const expire_at = Math.floor((new Date().getTime() + 90 * 24 * 60 * 60 * 1000) /
 1000);

    // Create DynamoDB item
    const item = {
        'partitionKey': {'S': partition_key},
        'sortKey': {'S': sort_key},
        'createdAt': {'N': current_time.toString()},
        'expireAt': {'N': expire_at.toString()}
    };

    const putItemCommand = new PutItemCommand({
        TableName: table_name,
        Item: item,
        ProvisionedThroughput: {
            ReadCapacityUnits: 1,
            WriteCapacityUnits: 1,
        },
    });

    client.send(putItemCommand, function(err, data) {
        if (err) {
            console.log("Exception encountered when creating item %s, here's what
 happened: ", data, ex);
            throw err;
        } else {
            console.log("Item created successfully: %s.", data);
            return data;
        }
    });
}

// use your own values
```

```
createDynamoDBItem('your-table-name', 'us-east-1', 'your-partition-key-value',
 'your-sort-key-value');
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考PutItem中的。

PPE在影像中偵測

下列程式碼範例示範如何建立使用 Amazon Rekognition 偵測影像中個人防護設備 (PPE) 的應用程式。

SDK對於 JavaScript （ 3 ）

示範如何搭配使用 Amazon Rekognition AWS SDK for JavaScript 來建立應用程式，以偵測位於亞馬遜簡單儲存服務 (Amazon S3PPE) 儲存貯體中的映像中的個人防護設備 ()。該應用程式會將結果儲存到 Amazon DynamoDB 表，並使用 Amazon 簡單電子郵件服務 (AmazonSES) 向管理員傳送包含結果的電子郵件通知。

了解如何：

- 使用 Amazon Cognito 建立未經身分驗證的使用者。
- 分析圖像以PPE使用 Amazon Rekognition。
- 驗證 Amazon 的電子郵件地址SES。
- 以結果更新 DynamoDB 資料表。
- 使用 Amazon 發送電子郵件通知SES。

有關如何設置和運行的完整源代碼和說明，請參閱中的完整示例GitHub。

此範例中使用的服務

- DynamoDB
- Amazon Rekognition
- Amazon S3
- Amazon SES

從瀏覽器調用 Lambda 函數

下列程式碼範例會示範如何從瀏覽器叫用 AWS Lambda 函數。

SDK對於 JavaScript（第 2 個）

您可以建立以瀏覽器為基礎的應用程式，使用 AWS Lambda 函數更新具有使用者選擇的 Amazon DynamoDB 表格。

有關如何設置和運行的完整源代碼和說明，請參閱中的完整示例GitHub。

此範例中使用的服務

- DynamoDB
- Lambda

SDK對於 JavaScript（3）

您可以建立以瀏覽器為基礎的應用程式，使用 AWS Lambda 函數更新具有使用者選擇的 Amazon DynamoDB 表格。此應用程序使用 AWS SDK for JavaScript v3。

有關如何設置和運行的完整源代碼和說明，請參閱中的完整示例GitHub。

此範例中使用的服務

- DynamoDB
- Lambda

使用多批 PartiQL 陳述式查詢資料表

以下程式碼範例顯示做法：

- 通過運行多個SELECT語句獲取一批項目。
- 通過運行多個INSERT語句添加一批項目。
- 透過執行多個UPDATE陳述式來更新一批項目。
- 執行多個DELETE陳述式以刪除批次項目。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

執行批次 PartiQL 陳述式。

```
import {
  BillingMode,
  CreateTableCommand,
  DeleteTableCommand,
  DescribeTableCommand,
  DynamoDBClient,
  waitUntilTableExists,
} from "@aws-sdk/client-dynamodb";
import {
  DynamoDBDocumentClient,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";
import { ScenarioInput } from "@aws-doc-sdk-examples/lib/scenario";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

const log = (msg) => console.log(`[SCENARIO] ${msg}`);
const tableName = "Cities";

export const main = async (confirmAll = false) => {
  /**
   * Delete table if it exists.
   */
  try {
    await client.send(new DescribeTableCommand({ TableName: tableName }));
    // If no error was thrown, the table exists.
    const input = new ScenarioInput(
      "deleteTable",
      `A table named ${tableName} already exists. If you choose not to delete
this table, the scenario cannot continue. Delete it?`,
      { confirmAll },
    );
    const deleteTable = await input.handle({});
    if (deleteTable) {
      await client.send(new DeleteTableCommand({ tableName }));
    } else {
      console.warn(
        "Scenario could not run. Either delete ${tableName} or provide a unique
 table name.",
      );
      return;
```

```
    }
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "ResourceNotFoundException"
    ) {
      // Do nothing. This means the table is not there.
    } else {
      throw caught;
    }
  }

  /**
   * Create a table.
   */

  log("Creating a table.");
  const createTableCommand = new CreateTableCommand({
    TableName: tableName,
    // This example performs a large write to the database.
    // Set the billing mode to PAY_PER_REQUEST to
    // avoid throttling the large write.
    BillingMode: BillingMode.PAY_PER_REQUEST,
    // Define the attributes that are necessary for the key schema.
    AttributeDefinitions: [
      {
        AttributeName: "name",
        // 'S' is a data type descriptor that represents a number type.
        // For a list of all data type descriptors, see the following link.
        // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Programming.LowLevelAPI.html#Programming.LowLevelAPI.DataTypeDescriptors
        AttributeType: "S",
      },
    ],
    // The KeySchema defines the primary key. The primary key can be
    // a partition key, or a combination of a partition key and a sort key.
    // Key schema design is important. For more info, see
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/best-
practices.html
    KeySchema: [{ AttributeName: "name", KeyType: "HASH" }],
  });
  await client.send(createTableCommand);
  log(`Table created: ${tableName}.`);
```

```
  /**
   * Wait until the table is active.
   */

  // This polls with DescribeTableCommand until the requested table is 'ACTIVE'.
  // You can't write to a table before it's active.
  log("Waiting for the table to be active.");
  await waitUntilTableExists({ client }, { TableName: tableName });
  log("Table active.");

  /**
   * Insert items.
   */

  log("Inserting cities into the table.");
  const addItemsStatementCommand = new BatchExecuteStatementCommand({
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.insert.html
    Statements: [
      {
        Statement: `INSERT INTO ${tableName} value {'name':?, 'population':?}`,
        Parameters: ["Alachua", 10712],
      },
      {
        Statement: `INSERT INTO ${tableName} value {'name':?, 'population':?}`,
        Parameters: ["High Springs", 6415],
      },
    ],
  });
  await docClient.send(addItemsStatementCommand);
  log(`Cities inserted.`);

  /**
   * Select items.
   */

  log("Selecting cities from the table.");
  const selectItemsStatementCommand = new BatchExecuteStatementCommand({
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.select.html
    Statements: [
      {
        Statement: `SELECT * FROM ${tableName} WHERE name=?`,
        Parameters: ["Alachua"],
```

```
      },
      {
        Statement: `SELECT * FROM ${tableName} WHERE name=?`,
        Parameters: ["High Springs"],
      },
    ],
  });
  const selectItemResponse = await docClient.send(selectItemsStatementCommand);
  log(
    `Got cities: ${selectItemResponse.Responses.map(
      (r) => `${r.Item.name} (${r.Item.population})`,
    ).join(", ")}`,
  );

  /**
   * Update items.
   */

  log("Modifying the populations.");
  const updateItemStatementCommand = new BatchExecuteStatementCommand({
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.update.html
    Statements: [
      {
        Statement: `UPDATE ${tableName} SET population=? WHERE name=?`,
        Parameters: [10, "Alachua"],
      },
      {
        Statement: `UPDATE ${tableName} SET population=? WHERE name=?`,
        Parameters: [5, "High Springs"],
      },
    ],
  });
  await docClient.send(updateItemStatementCommand);
  log(`Updated cities.`);

  /**
   * Delete the items.
   */

  log("Deleting the cities.");
  const deleteItemStatementCommand = new BatchExecuteStatementCommand({
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.delete.html
```

```
    Statements: [
      {
        Statement: `DELETE FROM ${tableName} WHERE name=?`,
        Parameters: ["Alachua"],
      },
      {
        Statement: `DELETE FROM ${tableName} WHERE name=?`,
        Parameters: ["High Springs"],
      },
    ],
  });
  await docClient.send(deleteItemStatementCommand);
  log("Cities deleted.");

  /**
   * Delete the table.
   */

  log("Deleting the table.");
  const deleteTableCommand = new DeleteTableCommand({ TableName: tableName });
  await client.send(deleteTableCommand);
  log("Table deleted.");
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考BatchExecuteStatement中的。

使用 PartiQL 查詢資料表

以下程式碼範例顯示做法：

- 透過執行SELECT陳述式取得項目。
- 執行INSERT陳述式以新增項目。
- 執行UPDATE陳述式以更新項目。
- 執行DELETE陳述式以刪除項目。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

執行單一 PartiQL 陳述式。

```
import {
  BillingMode,
  CreateTableCommand,
  DeleteTableCommand,
  DescribeTableCommand,
  DynamoDBClient,
  waitUntilTableExists,
} from "@aws-sdk/client-dynamodb";
import {
  DynamoDBDocumentClient,
  ExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";
import { ScenarioInput } from "@aws-doc-sdk-examples/lib/scenario";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

const log = (msg) => console.log(`[SCENARIO] ${msg}`);
const tableName = "SingleOriginCoffees";

export const main = async (confirmAll = false) => {
  /**
   * Delete table if it exists.
   */
  try {
    await client.send(new DescribeTableCommand({ TableName: tableName }));
    // If no error was thrown, the table exists.
    const input = new ScenarioInput(
      "deleteTable",
      `A table named ${tableName} already exists. If you choose not to delete
this table, the scenario cannot continue. Delete it?`,
      { confirmAll },
    );
```

```
    const deleteTable = await input.handle({});
    if (deleteTable) {
      await client.send(new DeleteTableCommand({ tableName }));
    } else {
      console.warn(
        "Scenario could not run. Either delete ${tableName} or provide a unique
 table name.",
      );
      return;
    }
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "ResourceNotFoundException"
    ) {
      // Do nothing. This means the table is not there.
    } else {
      throw caught;
    }
  }

  /**
   * Create a table.
   */

  log("Creating a table.");
  const createTableCommand = new CreateTableCommand({
    TableName: tableName,
    // This example performs a large write to the database.
    // Set the billing mode to PAY_PER_REQUEST to
    // avoid throttling the large write.
    BillingMode: BillingMode.PAY_PER_REQUEST,
    // Define the attributes that are necessary for the key schema.
    AttributeDefinitions: [
      {
        AttributeName: "varietal",
        // 'S' is a data type descriptor that represents a number type.
        // For a list of all data type descriptors, see the following link.
        // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Programming.LowLevelAPI.html#Programming.LowLevelAPI.DataTypeDescriptors
        AttributeType: "S",
      },
    ],
    // The KeySchema defines the primary key. The primary key can be
```

```
    // a partition key, or a combination of a partition key and a sort key.
    // Key schema design is important. For more info, see
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/best-
practices.html
    KeySchema: [{ AttributeName: "varietal", KeyType: "HASH" }],
  });
  await client.send(createTableCommand);
  log(`Table created: ${tableName}.`);

  /**
   * Wait until the table is active.
   */

  // This polls with DescribeTableCommand until the requested table is 'ACTIVE'.
  // You can't write to a table before it's active.
  log("Waiting for the table to be active.");
  await waitUntilTableExists({ client }, { TableName: tableName });
  log("Table active.");

  /**
   * Insert an item.
   */

  log("Inserting a coffee into the table.");
  const addItemStatementCommand = new ExecuteStatementCommand({
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.insert.html
    Statement: `INSERT INTO ${tableName} value {'varietal':?, 'profile':?}`,
    Parameters: ["arabica", ["chocolate", "floral"]],
  });
  await client.send(addItemStatementCommand);
  log(`Coffee inserted.`);

  /**
   * Select an item.
   */

  log("Selecting the coffee from the table.");
  const selectItemStatementCommand = new ExecuteStatementCommand({
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.select.html
    Statement: `SELECT * FROM ${tableName} WHERE varietal=?`,
    Parameters: ["arabica"],
  });
```

```
  const selectItemResponse = await docClient.send(selectItemStatementCommand);
  log(`Got coffee: ${JSON.stringify(selectItemResponse.Items[0])}`);

  /**
   * Update the item.
   */

  log("Add a flavor profile to the coffee.");
  const updateItemStatementCommand = new ExecuteStatementCommand({
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.update.html
    Statement: `UPDATE ${tableName} SET profile=list_append(profile, ?) WHERE
 varietal=?`,
    Parameters: [["fruity"], "arabica"],
  });
  await client.send(updateItemStatementCommand);
  log(`Updated coffee`);

  /**
   * Delete the item.
   */

  log("Deleting the coffee.");
  const deleteItemStatementCommand = new ExecuteStatementCommand({
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.delete.html
    Statement: `DELETE FROM ${tableName} WHERE varietal=?`,
    Parameters: ["arabica"],
  });
  await docClient.send(deleteItemStatementCommand);
  log("Coffee deleted.");

  /**
   * Delete the table.
   */

  log("Deleting the table.");
  const deleteTableCommand = new DeleteTableCommand({ TableName: tableName });
  await client.send(deleteTableCommand);
  log("Table deleted.");
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考ExecuteStatement中的。

查詢TTL項目

下列程式碼範例會示範如何查詢TTL項目。

SDK對於 JavaScript（3）

```
import { DynamoDBClient, QueryCommand } from "@aws-sdk/client-dynamodb";
import { marshall, unmarshall } from "@aws-sdk/util-dynamodb";

async function queryDynamoDBItems(tableName, region, primaryKey) {
    const client = new DynamoDBClient({
        region: region,
        endpoint: `https://dynamodb.${region}.amazonaws.com`
    });

    const currentTime = Math.floor(Date.now() / 1000);

    const params = {
        TableName: tableName,
        KeyConditionExpression: "#pk = :pk",
        FilterExpression: "#ea > :ea",
        ExpressionAttributeNames: {
            "#pk": "primaryKey",
            "#ea": "expireAt"
        },
        ExpressionAttributeValues: marshall({
            ":pk": primaryKey,
            ":ea": currentTime
        })
    };

    try {
        const { Items } = await client.send(new QueryCommand(params));
        Items.forEach(item => {
            console.log(unmarshall(item))
        });
        return Items;
    } catch (err) {
        console.error(`Error querying items: ${err}`);
        throw err;
    }
}

//enter your own values here
```

```
queryDynamoDBItems('your-table-name', 'your-partition-key-value');
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考中的查詢。

更新項目 TTL

下列程式碼範例顯示如何更新項目TTL。

SDK對於 JavaScript（3）

```
import { DynamoDBClient, UpdateItemCommand } from "@aws-sdk/client-dynamodb";
import { marshall, unmarshall } from "@aws-sdk/util-dynamodb";

async function updateDynamoDBItem(tableName, region, partitionKey, sortKey) {
    const client = new DynamoDBClient({
        region: region,
        endpoint: `https://dynamodb.${region}.amazonaws.com`
    });

    const currentTime = Math.floor(Date.now() / 1000);
    const expireAt = Math.floor((Date.now() + 90 * 24 * 60 * 60 * 1000) / 1000);

    const params = {
        TableName: tableName,
        Key: marshall({
            partitionKey: partitionKey,
            sortKey: sortKey
        }),
        UpdateExpression: "SET updatedAt = :c, expireAt = :e",
        ExpressionAttributeValues: marshall({
            ":c": currentTime,
            ":e": expireAt
        }),
    };

    try {
        const data = await client.send(new UpdateItemCommand(params));
        const responseData = unmarshall(data.Attributes);
        console.log("Item updated successfully: %s", responseData);
        return responseData;
    } catch (err) {
        console.error("Error updating item:", err);
```

```
        throw err;
    }
}

//enter your values here
updateDynamoDBItem('your-table-name', 'us-east-1', 'your-partition-key-value',
 'your-sort-key-value');
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考UpdateItem中的。

無伺服器範例

從 DynamoDB 觸發程序叫用 Lambda 函數

下列程式碼範例示範如何實作 Lambda 函數，該函數會接收透過從 DynamoDB 串流接收記錄而觸發的事件。此函數會擷取 DynamoDB 承載並記錄記錄內容。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在無伺服器範例儲存庫中設定和執行。

使用 Lambda 使用. JavaScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
    console.log(JSON.stringify(event, null, 2));
    event.Records.forEach(record => {
        logDynamoDBRecord(record);
    });
};

const logDynamoDBRecord = (record) => {
    console.log(record.eventID);
    console.log(record.eventName);
    console.log(`DynamoDB Record: ${JSON.stringify(record.dynamodb)}`);
};
```

使用 Lambda 使用. TypeScript

```
export const handler = async (event, context) => {
    console.log(JSON.stringify(event, null, 2));
    event.Records.forEach(record => {
        logDynamoDBRecord(record);
    });
}
const logDynamoDBRecord = (record) => {
    console.log(record.eventID);
    console.log(record.eventName);
    console.log(`DynamoDB Record: ${JSON.stringify(record.dynamodb)}`);
};
```

使用 DynamoDB 觸發程序報告 Lambda 函數的批次項目失敗

下列程式碼範例示範如何針對接收來自 DynamoDB 串流之事件的 Lambda 函數實作部分批次回應。此函數會在回應中報告批次項目失敗，指示 Lambda 稍後重試這些訊息。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在無伺服器範例儲存庫中設定和執行。

使用 Lambda 使用報告批次項目失敗 JavaScript

```
export const handler = async (event) => {
  const records = event.Records;
  let curRecordSequenceNumber = "";

  for (const record of records) {
    try {
      // Process your record
      curRecordSequenceNumber = record.dynamodb.SequenceNumber;
    } catch (e) {
      // Return failed record's sequence number
      return { batchItemFailures: [{ itemIdentifier: curRecordSequenceNumber }] };
```

```
    }
  }

  return { batchItemFailures: [] };
};
```

使用 Lambda 使用報告批次項目失敗 TypeScript

```
import {
  DynamoDBBatchResponse,
  DynamoDBBatchItemFailure,
  DynamoDBStreamEvent,
} from "aws-lambda";

export const handler = async (
  event: DynamoDBStreamEvent
): Promise<DynamoDBBatchResponse> => {
  const batchItemFailures: DynamoDBBatchItemFailure[] = [];
  let curRecordSequenceNumber;

  for (const record of event.Records) {
    curRecordSequenceNumber = record.dynamodb?.SequenceNumber;

    if (curRecordSequenceNumber) {
      batchItemFailures.push({
        itemIdentifier: curRecordSequenceNumber,
      });
    }
  }

  return { batchItemFailures: batchItemFailures };
};
```

Amazon EC2 示例使用 SDK for JavaScript (v3)

下列程式碼範例說明如何透過使用 AWS SDK for JavaScript (v3) 搭配 Amazon 來執行動作和實作常見案例EC2。

基本概念是程式碼範例，會示範如何在服務中執行基本作業。

Actions 是大型程式的程式碼摘錄，必須在內容中執行。雖然動作會顯示如何呼叫個別服務函數，但您可以在其相關案例中查看內容中的動作。

案例是程式碼範例，向您展示如何透過呼叫服務中的多個函式或與其他函式結合來完成特定工作 AWS 服務。

每個範例都包含完整原始程式碼的連結，您可以在其中找到如何在內容中設定和執行程式碼的指示。

開始使用

你好 Amazon EC2

下列程式碼範例說明如何開始使用 Amazon EC2。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import { DescribeSecurityGroupsCommand, EC2Client } from "@aws-sdk/client-ec2";

// Call DescribeSecurityGroups and display the result.
export const main = async () => {
  const client = new EC2Client();
  try {
    const { SecurityGroups } = await client.send(
      new DescribeSecurityGroupsCommand({}),
    );

    const securityGroupList = SecurityGroups.slice(0, 9)
      .map((sg) => ` • ${sg.GroupId}: ${sg.GroupName}`)
      .join("\n");

    console.log(
      "Hello, Amazon EC2! Let's list up to 10 of your security groups:",
    );
    console.log(securityGroupList);
  } catch (err) {
```

```
    console.error(err);
  }
};

// Call function if run directly.
import { fileURLToPath } from "url";
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  main();
}
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考DescribeSecurityGroups中的。

主題

- 基本概念
- 動作
- 案例

基本概念

學習基礎知識

以下程式碼範例顯示做法：

- 建立金鑰對和安全群組。
- 選取 Amazon 機器映像 (AMI) 和相容的執行個體類型，然後建立執行個體。
- 停止並重新啟動執行個體。
- 將彈性 IP 地址與您的執行個體建立關聯.
- 使用 Connect 至執行個體SSH，然後清理資源。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

此檔案包含搭配使用的常見動作清單EC2。這些步驟是使用 Scenario 框架構構建的，該框架可簡化運行交互式示例。有關完整上下文，請訪問存 GitHub 儲庫。

```
import { tmpdir } from "node:os";
import { writeFile, mkdtemp, rm } from "node:fs/promises";
import { join } from "node:path";
import { get } from "node:http";

import {
  AllocateAddressCommand,
  AssociateAddressCommand,
  AuthorizeSecurityGroupIngressCommand,
  CreateKeyPairCommand,
  CreateSecurityGroupCommand,
  DeleteKeyPairCommand,
  DeleteSecurityGroupCommand,
  DisassociateAddressCommand,
  paginateDescribeImages,
  paginateDescribeInstances,
  paginateDescribeInstanceTypes,
  ReleaseAddressCommand,
  RunInstancesCommand,
  StartInstancesCommand,
  StopInstancesCommand,
  TerminateInstancesCommand,
  waitUntilInstanceStatusOk,
  waitUntilInstanceStopped,
  waitUntilInstanceTerminated,
} from "@aws-sdk/client-ec2";

import {
  ScenarioAction,
  ScenarioInput,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

import { paginateGetParametersByPath, SSMClient } from "@aws-sdk/client-ssm";

/**
 * @typedef {{
 *   ec2Client: import('@aws-sdk/client-ec2').EC2Client,
 *   errors: Error[],
 *   keyPairId?: string,
```

```
 *   tmpDirectory?: string,
 *   securityGroupId?: string,
 *   ipAddress?: string,
 *   images?: import('@aws-sdk/client-ec2').Image[],
 *   image?: import('@aws-sdk/client-ec2').Image,
 *   instanceTypes?: import('@aws-sdk/client-ec2').InstanceTypeInfo[],
 *   instanceId?: string,
 *   instanceIpAddress?: string,
 *   allocationId?: string,
 *   allocatedIpAddress?: string,
 *   associationId?: string,
 * }} State
 */

/**
 * A skip function provided to the `skipWhen` of a Step when you want
 * to ignore that step if any errors have occurred.
 * @param {State} state
 */
const skipWhenErrors = (state) => state.errors.length > 0;

const MAX_WAITER_TIME_IN_SECONDS = 60 * 8;

export const confirm = new ScenarioInput("confirmContinue", "Continue?", {
  type: "confirm",
  skipWhen: skipWhenErrors,
});

export const exitOnNoConfirm = new ScenarioAction(
  `exitOnConfirmContinueFalse`,
  (/** @type { { earlyExit: boolean } & Record<string, any>} */ state) => {
    if (!state[confirm.name]) {
      state.earlyExit = true;
    }
  },
  {
    skipWhen: skipWhenErrors,
  },
);

export const greeting = new ScenarioOutput(
  "greeting",
  `Welcome to the Amazon EC2 basic usage scenario.
Before you launch an instances, you'll need to provide a few things:
```

```
 • A key pair - This is for SSH access to your EC2 instance. You only need to
 provide the name.
 • A security group - This is used for configuring access to your instance. Again,
 only the name is needed.
 • An IP address - Your public IP address will be fetched.
 • An Amazon Machine Image (AMI)
 • A compatible instance type`,
  { header: true, preformatted: true, skipWhen: skipWhenErrors },
);

export const provideKeyPairName = new ScenarioInput(
  "keyPairName",
  "Provide a name for a new key pair.",
  { type: "input", default: "ec2-example-key-pair", skipWhen: skipWhenErrors },
);

export const createKeyPair = new ScenarioAction(
  "createKeyPair",
  async (/** @type {State} */ state) => {
    try {
      // Create a key pair in Amazon EC2.
      const { KeyMaterial, KeyPairId } = await state.ec2Client.send(
        // A unique name for the key pair. Up to 255 ASCII characters.
        new CreateKeyPairCommand({ KeyName: state[provideKeyPairName.name] }),
      );

      state.keyPairId = KeyPairId;

      // Save the private key in a temporary location.
      state.tmpDirectory = await mkdtemp(join(tmpdir(), "ec2-scenario-tmp"));
      await writeFile(
        `${state.tmpDirectory}/${state[provideKeyPairName.name]}.pem`,
        KeyMaterial,
        {
          mode: 0o400,
        },
      );
    } catch (caught) {
      if (
        caught instanceof Error &&
        caught.name === "InvalidKeyPair.Duplicate"
      ) {
        caught.message = `${caught.message}. Try another key name.`;
      }
```

```
      state.errors.push(caught);
    }
  },
  { skipWhen: skipWhenErrors },
);

export const logKeyPair = new ScenarioOutput(
  "logKeyPair",
  (/** @type {State} */ state) =>
    `Created the key pair ${state[provideKeyPairName.name]}.`,
  { skipWhen: skipWhenErrors },
);

export const confirmDeleteKeyPair = new ScenarioInput(
  "confirmDeleteKeyPair",
  "Do you want to delete the key pair?",
  {
    type: "confirm",
    // Don't do anything when a key pair was never created.
    skipWhen: (/** @type {State} */ state) => !state.keyPairId,
  },
);

export const maybeDeleteKeyPair = new ScenarioAction(
  "deleteKeyPair",
  async (/** @type {State} */ state) => {
    try {
      // Delete a key pair by name from EC2
      await state.ec2Client.send(
        new DeleteKeyPairCommand({ KeyName: state[provideKeyPairName.name] }),
      );
    } catch (caught) {
      if (
        caught instanceof Error &&
        // Occurs when a required parameter (e.g. KeyName) is undefined.
        caught.name === "MissingParameter"
      ) {
        caught.message = `${caught.message}. Did you provide the required value?`;
      }
      state.errors.push(caught);
    }
  },
  {
```

```
    // Don't do anything when there's no key pair to delete or the user chooses
    // to keep it.
    skipWhen: (/** @type {State} */ state) =>
      !state.keyPairId || !state[confirmDeleteKeyPair.name],
  },
);

export const provideSecurityGroupName = new ScenarioInput(
  "securityGroupName",
  "Provide a name for a new security group.",
  { type: "input", default: "ec2-scenario-sg", skipWhen: skipWhenErrors },
);

export const createSecurityGroup = new ScenarioAction(
  "createSecurityGroup",
  async (/** @type {State} */ state) => {
    try {
      // Create a new security group that will be used to configure ingress/egress
 for
      // an EC2 instance.
      const { GroupId } = await state.ec2Client.send(
        new CreateSecurityGroupCommand({
          GroupName: state[provideSecurityGroupName.name],
          Description: "A security group for the Amazon EC2 example.",
        }),
      );
      state.securityGroupId = GroupId;
    } catch (caught) {
      if (caught instanceof Error && caught.name === "InvalidGroup.Duplicate") {
        caught.message = `${caught.message}. Please provide a different name for
 your security group.`;
      }

      state.errors.push(caught);
    }
  },
  { skipWhen: skipWhenErrors },
);

export const logSecurityGroup = new ScenarioOutput(
  "logSecurityGroup",
  (/** @type {State} */ state) =>
    `Created the security group ${state.securityGroupId}.`,
  { skipWhen: skipWhenErrors },
```

```
);

export const confirmDeleteSecurityGroup = new ScenarioInput(
  "confirmDeleteSecurityGroup",
  "Do you want to delete the security group?",
  {
    type: "confirm",
    // Don't do anything when a security group was never created.
    skipWhen: (/** @type {State} */ state) => !state.securityGroupId,
  },
);

export const maybeDeleteSecurityGroup = new ScenarioAction(
  "deleteSecurityGroup",
  async (/** @type {State} */ state) => {
    try {
      // Delete the security group if the 'skipWhen' condition below is not met.
      await state.ec2Client.send(
        new DeleteSecurityGroupCommand({
          GroupId: state.securityGroupId,
        }),
      );
    } catch (caught) {
      if (
        caught instanceof Error &&
        caught.name === "InvalidGroupId.Malformed"
      ) {
        caught.message = `${caught.message}. Please provide a valid GroupId.`;
      }
      state.errors.push(caught);
    }
  },
  {
    // Don't do anything when there's no security group to delete
    // or the user chooses to keep it.
    skipWhen: (/** @type {State} */ state) =>
      !state.securityGroupId || !state[confirmDeleteSecurityGroup.name],
  },
);

export const authorizeSecurityGroupIngress = new ScenarioAction(
  "authorizeSecurity",
  async (/** @type {State} */ state) => {
    try {
```

```
      // Get the public IP address of the machine running this example.
      const ipAddress = await new Promise((res, rej) => {
        get("http://checkip.amazonaws.com", (response) => {
          let data = "";
          response.on("data", (chunk) => (data += chunk));
          response.on("end", () => res(data.trim()));
        }).on("error", (err) => {
          rej(err);
        });
      });
      state[`ipAddress`] = ipAddress;
      // Allow ingress from the IP address above to the security group.
      // This will allow you to SSH into the EC2 instance.
      const command = new AuthorizeSecurityGroupIngressCommand({
        GroupId: state.securityGroupId,
        IpPermissions: [
          {
            IpProtocol: "tcp",
            FromPort: 22,
            ToPort: 22,
            IpRanges: [{ CidrIp: `${ipAddress}/32` }],
          },
        ],
      });

      await state.ec2Client.send(command);
    } catch (caught) {
      if (
        caught instanceof Error &&
        caught.name === "InvalidGroupId.Malformed"
      ) {
        caught.message = `${caught.message}. Please provide a valid GroupId.`;
      }

      state.errors.push(caught);
    }
  },
  { skipWhen: skipWhenErrors },
);

export const logSecurityGroupIngress = new ScenarioOutput(
  "logSecurityGroupIngress",
  (/** @type {State} */ state) =>
    `Allowed SSH access from your public IP: ${state.ipAddress}.`,
```

```
  { skipWhen: skipWhenErrors },
);

export const getImages = new ScenarioAction(
  "images",
  async (/** @type {State} */ state) => {
    const AMIs = [];
    // Some AWS services publish information about common artifacts as AWS Systems
 Manager (SSM)
    // public parameters. For example, the Amazon Elastic Compute Cloud (Amazon EC2)
    // service publishes information about Amazon Machine Images (AMIs) as public
 parameters.

    // Create the paginator for getting images. Actions that return multiple pages
 of
    // results have paginators to simplify those calls.
    const getParametersByPathPaginator = paginateGetParametersByPath(
      {
        // Not storing this client in state since it's only used once.
        client: new SSMClient({}),
      },
      {
        // The path to the public list of the latest amazon-linux instances.
        Path: "/aws/service/ami-amazon-linux-latest",
      },
    );

    try {
      for await (const page of getParametersByPathPaginator) {
        page.Parameters.forEach((param) => {
          // Filter by Amazon Linux 2
          if (param.Name.includes("amzn2")) {
            AMIs.push(param.Value);
          }
        });
      }
    } catch (caught) {
      if (caught instanceof Error && caught.name === "InvalidFilterValue") {
        caught.message = `${caught.message} Please provide a valid filter value for
 paginateGetParametersByPath.`;
      }
      state.errors.push(caught);
      return;
    }
```

```
    const imageDetails = [];
    const describeImagesPaginator = paginateDescribeImages(
      { client: state.ec2Client },
      // The images found from the call to SSM.
      { ImageIds: AMIs },
    );

    try {
      // Get more details for the images found above.
      for await (const page of describeImagesPaginator) {
        imageDetails.push(...(page.Images || []));
      }

      // Store the image details for later use.
      state["images"] = imageDetails;
    } catch (caught) {
      if (caught instanceof Error && caught.name === "InvalidAMIID.NotFound") {
        caught.message = `${caught.message}. Please provide a valid image id.`;
      }

      state.errors.push(caught);
    }
  },
  { skipWhen: skipWhenErrors },
);

export const provideImage = new ScenarioInput(
  "image",
  "Select one of the following images.",
  {
    type: "select",
    choices: (/** @type { State } */ state) =>
      state.images.map((image) => ({
        name: `${image.ImageId} - ${image.Description}`,
        value: image,
      })),
    default: (/** @type { State } */ state) => state.images[0],
    skipWhen: skipWhenErrors,
  },
);

export const getCompatibleInstanceTypes = new ScenarioAction(
  "getCompatibleInstanceTypes",
```

```
  async (/** @type {State} */ state) => {
    // Get more details about instance types that match the architecture of
    // the provided image.
    const paginator = paginateDescribeInstanceTypes(
      { client: state.ec2Client, pageSize: 25 },
      {
        Filters: [
          {
            Name: "processor-info.supported-architecture",
            // The value selected from provideImage()
            Values: [state.image.Architecture],
          },
          // Filter for smaller, less expensive, types.
          { Name: "instance-type", Values: ["*.micro", "*.small"] },
        ],
      },
    );

    const instanceTypes = [];

    try {
      for await (const page of paginator) {
        if (page.InstanceTypes.length) {
          instanceTypes.push(...(page.InstanceTypes || []));
        }
      }

      if (!instanceTypes.length) {
        state.errors.push(
          "No instance types matched the instance type filters.",
        );
      }
    } catch (caught) {
      if (caught instanceof Error && caught.name === "InvalidParameterValue") {
        caught.message = `${caught.message}. Please check the provided values and
try again.`;
      }

      state.errors.push(caught);
    }

    state["instanceTypes"] = instanceTypes;
  },
  { skipWhen: skipWhenErrors },
```

```
);

export const provideInstanceType = new ScenarioInput(
  "instanceType",
  "Select an instance type.",
  {
    choices: (/** @type {State} */ state) =>
      state.instanceTypes.map((instanceType) => ({
        name: `${instanceType.InstanceType} - Memory:
${instanceType.MemoryInfo.SizeInMiB}`,
        value: instanceType.InstanceType,
      })),
    type: "select",
    default: (/** @type {State} */ state) =>
      state.instanceTypes[0].InstanceType,
    skipWhen: skipWhenErrors,
  },
);

export const runInstance = new ScenarioAction(
  "runInstance",
  async (/** @type { State } */ state) => {
    const { Instances } = await state.ec2Client.send(
      new RunInstancesCommand({
        KeyName: state[provideKeyPairName.name],
        SecurityGroupIds: [state.securityGroupId],
        ImageId: state.image.ImageId,
        InstanceType: state[provideInstanceType.name],
        // Availability Zones have capacity limitations that may impact your ability
 to launch instances.
        // The `RunInstances` operation will only succeed if it can allocate at
 least the `MinCount` of instances.
        // However, EC2 will attempt to launch up to the `MaxCount` of instances,
 even if the full request cannot be satisfied.
        // If you need a specific number of instances, use `MinCount` and `MaxCount`
 set to the same value.
        // If you want to launch up to a certain number of instances, use `MaxCount`
 and let EC2 provision as many as possible.
        // If you require a minimum number of instances, but do not want to exceed a
 maximum, use both `MinCount` and `MaxCount`.
        MinCount: 1,
        MaxCount: 1,
      }),
    );
```

```
    state.instanceId = Instances[0].InstanceId;

    try {
      // Poll `DescribeInstanceStatus` until status is "ok".
      await waitUntilInstanceStatusOk(
        {
          client: state.ec2Client,
          maxWaitTime: MAX_WAITER_TIME_IN_SECONDS,
        },
        { InstanceIds: [Instances[0].InstanceId] },
      );
    } catch (caught) {
      if (caught instanceof Error && caught.name === "TimeoutError") {
        caught.message = `${caught.message}. Try increasing the maxWaitTime in the
 waiter.`;
      }

      state.errors.push(caught);
    }
  },
  { skipWhen: skipWhenErrors },
);

export const logRunInstance = new ScenarioOutput(
  "logRunInstance",
  "The next step is to run your EC2 instance for the first time. This can take a few
 minutes.",
  { header: true, skipWhen: skipWhenErrors },
);

export const describeInstance = new ScenarioAction(
  "describeInstance",
  async (/** @type { State } */ state) => {
    /** @type { import("@aws-sdk/client-ec2").Instance[] } */
    const instances = [];

    try {
      const paginator = paginateDescribeInstances(
        {
          client: state.ec2Client,
        },
        {
          // Only get our created instance.
```

```
          InstanceIds: [state.instanceId],
        },
      );

      for await (const page of paginator) {
        for (const reservation of page.Reservations) {
          instances.push(...reservation.Instances);
        }
      }
      if (instances.length !== 1) {
        throw new Error(`Instance ${state.instanceId} not found.`);
      }

      // The only info we need is the IP address for SSH purposes.
      state.instanceIpAddress = instances[0].PublicIpAddress;
    } catch (caught) {
      if (caught instanceof Error && caught.name === "InvalidParameterValue") {
        caught.message = `${caught.message}. Please check provided values and try
 again.`;
      }

      state.errors.push(caught);
    }
  },
  { skipWhen: skipWhenErrors },
);

export const logSSHConnectionInfo = new ScenarioOutput(
  "logSSHConnectionInfo",
  (/** @type { State } */ state) =>
    `You can now SSH into your instance using the following command:
ssh -i ${state.tmpDirectory}/${state[provideKeyPairName.name]}.pem ec2-user@
${state.instanceIpAddress}`,
  { preformatted: true, skipWhen: skipWhenErrors },
);

export const logStopInstance = new ScenarioOutput(
  "logStopInstance",
  "Stopping your EC2 instance.",
  { skipWhen: skipWhenErrors },
);

export const stopInstance = new ScenarioAction(
  "stopInstance",
```

```
  async (/** @type { State } */ state) => {
    try {
      await state.ec2Client.send(
        new StopInstancesCommand({
          InstanceIds: [state.instanceId],
        }),
      );

      await waitUntilInstanceStopped(
        {
          client: state.ec2Client,
          maxWaitTime: MAX_WAITER_TIME_IN_SECONDS,
        },
        { InstanceIds: [state.instanceId] },
      );
    } catch (caught) {
      if (caught instanceof Error && caught.name === "TimeoutError") {
        caught.message = `${caught.message}. Try increasing the maxWaitTime in the
 waiter.`;
      }

      state.errors.push(caught);
    }
  },
  // Don't try to stop an instance that doesn't exist.
  { skipWhen: (/** @type { State } */ state) => !state.instanceId },
);

export const logIpAddressBehavior = new ScenarioOutput(
  "logIpAddressBehavior",
  [
    "When you run an instance, by default it's assigned an IP address.",
    "That IP address is not static. It will change every time the instance is
 restarted.",
    "The next step is to stop and restart your instance to demonstrate this
 behavior.",
  ].join(" "),
  { header: true, skipWhen: skipWhenErrors },
);

export const logStartInstance = new ScenarioOutput(
  "logStartInstance",
  (/** @type { State } */ state) => `Starting instance ${state.instanceId}`,
  { skipWhen: skipWhenErrors },
```

```
);

export const startInstance = new ScenarioAction(
  "startInstance",
  async (/** @type { State } */ state) => {
    try {
      await state.ec2Client.send(
        new StartInstancesCommand({
          InstanceIds: [state.instanceId],
        }),
      );

      await waitUntilInstanceStatusOk(
        {
          client: state.ec2Client,
          maxWaitTime: MAX_WAITER_TIME_IN_SECONDS,
        },
        { InstanceIds: [state.instanceId] },
      );
    } catch (caught) {
      if (caught instanceof Error && caught.name === "TimeoutError") {
        caught.message = `${caught.message}. Try increasing the maxWaitTime in the
 waiter.`;
      }

      state.errors.push(caught);
    }
  },
  { skipWhen: skipWhenErrors },
);

export const logIpAllocation = new ScenarioOutput(
  "logIpAllocation",
  [
    "It is possible to have a static IP address.",
    "To demonstrate this, an IP will be allocated and associated to your EC2
 instance.",
  ].join(" "),
  { header: true, skipWhen: skipWhenErrors },
);

export const allocateIp = new ScenarioAction(
  "allocateIp",
  async (/** @type { State } */ state) => {
```

```
    try {
      // An Elastic IP address is allocated to your AWS account, and is yours until
 you release it.
      const { AllocationId, PublicIp } = await state.ec2Client.send(
        new AllocateAddressCommand({}),
      );
      state.allocationId = AllocationId;
      state.allocatedIpAddress = PublicIp;
    } catch (caught) {
      if (caught instanceof Error && caught.name === "MissingParameter") {
        caught.message = `${caught.message}. Did you provide these values?`;
      }
      state.errors.push(caught);
    }
  },
  { skipWhen: skipWhenErrors },
);

export const associateIp = new ScenarioAction(
  "associateIp",
  async (/** @type { State } */ state) => {
    try {
      // Associate an allocated IP address to an EC2 instance. An IP address can be
 allocated
      // with the AllocateAddress action.
      const { AssociationId } = await state.ec2Client.send(
        new AssociateAddressCommand({
          AllocationId: state.allocationId,
          InstanceId: state.instanceId,
        }),
      );
      state.associationId = AssociationId;
      // Update the IP address that is being tracked to match
      // the one just associated.
      state.instanceIpAddress = state.allocatedIpAddress;
    } catch (caught) {
      if (
        caught instanceof Error &&
        caught.name === "InvalidAllocationID.NotFound"
      ) {
        caught.message = `${caught.message}. Did you provide the ID of a valid
 Elastic IP address AllocationId?`;
      }
      state.errors.push(caught);
```

```
    }
  },
  { skipWhen: skipWhenErrors },
);

export const logStaticIpProof = new ScenarioOutput(
  "logStaticIpProof",
  "The IP address should remain the same even after stopping and starting the
 instance.",
  { header: true, skipWhen: skipWhenErrors },
);

export const logCleanUp = new ScenarioOutput(
  "logCleanUp",
  "That's it! You can choose to clean up the resources now, or clean them up on your
 own later.",
  { header: true, skipWhen: skipWhenErrors },
);

export const confirmDisassociateAddress = new ScenarioInput(
  "confirmDisassociateAddress",
  "Do you want to disassociate and release the static IP address created earlier?",
  {
    type: "confirm",
    skipWhen: (/** @type { State } */ state) => !state.associationId,
  },
);

export const maybeDisassociateAddress = new ScenarioAction(
  "maybeDisassociateAddress",
  async (/** @type { State } */ state) => {
    try {
      await state.ec2Client.send(
        new DisassociateAddressCommand({
          AssociationId: state.associationId,
        }),
      );
    } catch (caught) {
      if (
        caught instanceof Error &&
        caught.name === "InvalidAssociationID.NotFound"
      ) {
        caught.message = `${caught.message}. Please provide a valid association
 ID.`;
```

```
      }
      state.errors.push(caught);
    }
  },
  {
    skipWhen: (/** @type { State } */ state) =>
      !state[confirmDisassociateAddress.name] || !state.associationId,
  },
);

export const maybeReleaseAddress = new ScenarioAction(
  "maybeReleaseAddress",
  async (/** @type { State } */ state) => {
    try {
      await state.ec2Client.send(
        new ReleaseAddressCommand({
          AllocationId: state.allocationId,
        }),
      );
    } catch (caught) {
      if (
        caught instanceof Error &&
        caught.name === "InvalidAllocationID.NotFound"
      ) {
        caught.message = `${caught.message}. Please provide a valid AllocationID.`;
      }
      state.errors.push(caught);
    }
  },
  {
    skipWhen: (/** @type { State } */ state) =>
      !state[confirmDisassociateAddress.name] || !state.allocationId,
  },
);

export const confirmTerminateInstance = new ScenarioInput(
  "confirmTerminateInstance",
  "Do you want to terminate the instance?",
  // Don't do anything when an instance was never run.
  {
    skipWhen: (/** @type { State } */ state) => !state.instanceId,
    type: "confirm",
  },
);
```

```
export const maybeTerminateInstance = new ScenarioAction(
  "terminateInstance",
  async (/** @type { State } */ state) => {
    try {
      await state.ec2Client.send(
        new TerminateInstancesCommand({
          InstanceIds: [state.instanceId],
        }),
      );
      await waitUntilInstanceTerminated(
        { client: state.ec2Client },
        { InstanceIds: [state.instanceId] },
      );
    } catch (caught) {
      if (caught instanceof Error && caught.name === "TimeoutError") {
        caught.message = `${caught.message}. Try increasing the maxWaitTime in the
 waiter.`;
      }

      state.errors.push(caught);
    }
  },
  {
    // Don't do anything when there's no instance to terminate or the
    // use chooses not to terminate.
    skipWhen: (/** @type { State } */ state) =>
      !state.instanceId || !state[confirmTerminateInstance.name],
  },
);

export const deleteTemporaryDirectory = new ScenarioAction(
  "deleteTemporaryDirectory",
  async (/** @type { State } */ state) => {
    try {
      await rm(state.tmpDirectory, { recursive: true });
    } catch (caught) {
      state.errors.push(caught);
    }
  },
);

export const logErrors = new ScenarioOutput(
  "logErrors",
```

```
  (/** @type {State}*/ state) => {
    const errorList = state.errors
      .map((err) => `• ${err.name}: ${err.message}`)
      .join("\n");
    return `Scenario errors found:\n${errorList}`;
  },
  {
    preformatted: true,
    header: true,
    // Don't log errors when there aren't any!
    skipWhen: (/** @type {State} */ state) => state.errors.length === 0,
  },
);
```

- 如需詳API細資訊，請參閱「AWS SDK for JavaScript API參考」中的下列主題。
  - AllocateAddress
  - AssociateAddress
  - AuthorizeSecurityGroupIngress
  - CreateKeyPair
  - CreateSecurityGroup
  - DeleteKeyPair
  - DeleteSecurityGroup
  - DescribeImages
  - DescribeInstanceTypes
  - DescribeInstances
  - DescribeKeyPairs
  - DescribeSecurityGroups
  - DisassociateAddress
  - ReleaseAddress
  - RunInstances
  - StartInstances
  - StopInstances
  - TerminateInstances
  - UnmonitorInstances

動作

`AllocateAddress`

下列程式碼範例會示範如何使用`AllocateAddress`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import { AllocateAddressCommand, EC2Client } from "@aws-sdk/client-ec2";

/**
 * Allocates an Elastic IP address to your AWS account.
 */
export const main = async () => {
  const client = new EC2Client({});
  const command = new AllocateAddressCommand({});

  try {
    const { AllocationId, PublicIp } = await client.send(command);
    console.log("A new IP address has been allocated to your account:");
    console.log(`ID: ${AllocationId} Public IP: ${PublicIp}`);
    console.log(
      "You can view your IP addresses in the AWS Management Console for Amazon EC2.
 Look under Network & Security > Elastic IPs",
    );
  } catch (caught) {
    if (caught instanceof Error && caught.name === "MissingParameter") {
      console.warn(`${caught.message}. Did you provide these values?`);
    } else {
      throw caught;
    }
  }
};
import { fileURLToPath } from "url";
// Call function if run directly.
```

```
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  main();
}
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考AllocateAddress中的。

`AssociateAddress`

下列程式碼範例會示範如何使用`AssociateAddress`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import { AssociateAddressCommand, EC2Client } from "@aws-sdk/client-ec2";

/**
 * Associates an Elastic IP address, or carrier IP address (for instances that are
 in subnets in Wavelength Zones)
 * with an instance or a network interface.
 * @param {{ instanceId: string, allocationId: string }} options
 */
export const main = async ({ instanceId, allocationId }) => {
  const client = new EC2Client({});
  const command = new AssociateAddressCommand({
    // You need to allocate an Elastic IP address before associating it with an
 instance.
    // You can do that with the AllocateAddressCommand.
    AllocationId: allocationId,
    // You need to create an EC2 instance before an IP address can be associated
 with it.
    // You can do that with the RunInstancesCommand.
    InstanceId: instanceId,
  });

  try {
    const { AssociationId } = await client.send(command);
```

```
    console.log(
      `Address with allocation ID ${allocationId} is now associated with instance
 ${instanceId}.`,
      `The association ID is ${AssociationId}.`,
    );
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "InvalidAllocationID.NotFound"
    ) {
      console.warn(
        `${caught.message}. Did you provide the ID of a valid Elastic IP address
 AllocationId?`,
      );
    } else {
      throw caught;
    }
  }
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考AssociateAddress中的。

`AuthorizeSecurityGroupIngress`

下列程式碼範例會示範如何使用`AuthorizeSecurityGroupIngress`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import {
  AuthorizeSecurityGroupIngressCommand,
  EC2Client,
} from "@aws-sdk/client-ec2";

/**
```

```
 * Adds the specified inbound (ingress) rules to a security group.
 * @param {{ groupId: string, ipAddress: string }} options
 */
export const main = async ({ groupId, ipAddress }) => {
  const client = new EC2Client({});
  const command = new AuthorizeSecurityGroupIngressCommand({
    // Use a group ID from the AWS console or
    // the DescribeSecurityGroupsCommand.
    GroupId: groupId,
    IpPermissions: [
      {
        IpProtocol: "tcp",
        FromPort: 22,
        ToPort: 22,
        // The IP address to authorize.
        // For more information on this notation, see
        // https://en.wikipedia.org/wiki/Classless_Inter-
Domain_Routing#CIDR_notation
        IpRanges: [{ CidrIp: `${ipAddress}/32` }],
      },
    ],
  });

  try {
    const { SecurityGroupRules } = await client.send(command);
    console.log(JSON.stringify(SecurityGroupRules, null, 2));
  } catch (caught) {
    if (caught instanceof Error && caught.name === "InvalidGroupId.Malformed") {
      console.warn(`${caught.message}. Please provide a valid GroupId.`);
    } else {
      throw caught;
    }
  }
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考AuthorizeSecurityGroupIngress中的。

CreateKeyPair

下列程式碼範例會示範如何使用CreateKeyPair。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import { CreateKeyPairCommand, EC2Client } from "@aws-sdk/client-ec2";

/**
 * Creates an ED25519 or 2048-bit RSA key pair with the specified name and in the
 specified PEM or PPK format.
 * Amazon EC2 stores the public key and displays the private key for you to save to
 a file.
 * @param {{ keyName: string }} options
 */
export const main = async ({ keyName }) => {
  const client = new EC2Client({});
  const command = new CreateKeyPairCommand({
    KeyName: keyName,
  });

  try {
    const { KeyMaterial, KeyName } = await client.send(command);
    console.log(KeyName);
    console.log(KeyMaterial);
  } catch (caught) {
    if (caught instanceof Error && caught.name === "InvalidKeyPair.Duplicate") {
      console.warn(`${caught.message}. Try another key name.`);
    } else {
      throw caught;
    }
  }
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考CreateKeyPair中的。

`CreateLaunchTemplate`

下列程式碼範例會示範如何使用`CreateLaunchTemplate`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
const ssmClient = new SSMClient({});
const { Parameter } = await ssmClient.send(
  new GetParameterCommand({
    Name: "/aws/service/ami-amazon-linux-latest/amzn2-ami-hvm-x86_64-gp2",
  }),
);
const ec2Client = new EC2Client({});
await ec2Client.send(
  new CreateLaunchTemplateCommand({
    LaunchTemplateName: NAMES.launchTemplateName,
    LaunchTemplateData: {
      InstanceType: "t3.micro",
      ImageId: Parameter.Value,
      IamInstanceProfile: { Name: NAMES.instanceProfileName },
      UserData: readFileSync(
        join(RESOURCES_PATH, "server_startup_script.sh"),
      ).toString("base64"),
      KeyName: NAMES.keyPairName,
    },
  }),
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考CreateLaunchTemplate中的。

`CreateSecurityGroup`

下列程式碼範例會示範如何使用`CreateSecurityGroup`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import { CreateSecurityGroupCommand, EC2Client } from "@aws-sdk/client-ec2";

/**
 * Creates a security group.
 * @param {{ groupName: string, description: string }} options
 */
export const main = async ({ groupName, description }) => {
  const client = new EC2Client({});
  const command = new CreateSecurityGroupCommand({
    // Up to 255 characters in length. Cannot start with sg-.
    GroupName: groupName,
    // Up to 255 characters in length.
    Description: description,
  });

  try {
    const { GroupId } = await client.send(command);
    console.log(GroupId);
  } catch (caught) {
    if (caught instanceof Error && caught.name === "InvalidParameterValue") {
      console.warn(`${caught.message}.`);
    } else {
      throw caught;
    }
  }
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考CreateSecurityGroup中的。

`DeleteKeyPair`

下列程式碼範例會示範如何使用`DeleteKeyPair`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import { DeleteKeyPairCommand, EC2Client } from "@aws-sdk/client-ec2";

/**
 * Deletes the specified key pair, by removing the public key from Amazon EC2.
 * @param {{ keyName: string }} options
 */
export const main = async ({ keyName }) => {
  const client = new EC2Client({});
  const command = new DeleteKeyPairCommand({
    KeyName: keyName,
  });

  try {
    await client.send(command);
    console.log("Successfully deleted key pair.");
  } catch (caught) {
    if (caught instanceof Error && caught.name === "MissingParameter") {
      console.warn(`${caught.message}. Did you provide the required value?`);
    } else {
      throw caught;
    }
  }
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考DeleteKeyPair中的。

`DeleteLaunchTemplate`

下列程式碼範例會示範如何使用`DeleteLaunchTemplate`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    await client.send(
      new DeleteLaunchTemplateCommand({
        LaunchTemplateName: NAMES.launchTemplateName,
      }),
    );
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考DeleteLaunchTemplate中的。

`DeleteSecurityGroup`

下列程式碼範例會示範如何使用`DeleteSecurityGroup`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import { DeleteSecurityGroupCommand, EC2Client } from "@aws-sdk/client-ec2";

/**
 * Deletes a security group.
 * @param {{ groupId: string }} options
 */
export const main = async ({ groupId }) => {
  const client = new EC2Client({});
  const command = new DeleteSecurityGroupCommand({
    GroupId: groupId,
```

```
  });

  try {
    await client.send(command);
    console.log("Security group deleted successfully.");
  } catch (caught) {
    if (caught instanceof Error && caught.name === "InvalidGroupId.Malformed") {
      console.warn(`${caught.message}. Please provide a valid GroupId.`);
    } else {
      throw caught;
    }
  }
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考DeleteSecurityGroup中的。

`DescribeAddresses`

下列程式碼範例會示範如何使用`DescribeAddresses`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import { DescribeAddressesCommand, EC2Client } from "@aws-sdk/client-ec2";

/**
 * Describes the specified Elastic IP addresses or all of your Elastic IP addresses.
 * @param {{ allocationId: string }} options
 */
export const main = async ({ allocationId }) => {
  const client = new EC2Client({});
  const command = new DescribeAddressesCommand({
    // You can omit this property to show all addresses.
    AllocationIds: [allocationId],
  });
```

```
  try {
    const { Addresses } = await client.send(command);
    const addressList = Addresses.map((address) => ` • ${address.PublicIp}`);
    console.log("Elastic IP addresses:");
    console.log(addressList.join("\n"));
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "InvalidAllocationID.NotFound"
    ) {
      console.warn(`${caught.message}. Please provide a valid AllocationId.`);
    } else {
      throw caught;
    }
  }
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考DescribeAddresses中的。

`DescribeIamInstanceProfileAssociations`

下列程式碼範例會示範如何使用`DescribeIamInstanceProfileAssociations`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    const ec2Client = new EC2Client({});
    const { IamInstanceProfileAssociations } = await ec2Client.send(
      new DescribeIamInstanceProfileAssociationsCommand({
        Filters: [
          { Name: "instance-id", Values: [state.targetInstance.InstanceId] },
        ],
      }),
    );
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考DescribeIamInstanceProfileAssociations中的。

`DescribeImages`

下列程式碼範例會示範如何使用`DescribeImages`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import { EC2Client, paginateDescribeImages } from "@aws-sdk/client-ec2";

/**
 * Describes the specified images (AMIs, AKIs, and ARIs) available to you or all of
 the images available to you.
 * @param {{ architecture: string, pageSize: number }} options
 */
export const main = async ({ architecture, pageSize }) => {
  pageSize = parseInt(pageSize);
  const client = new EC2Client({});

  // The paginate function is a wrapper around the base command.
  const paginator = paginateDescribeImages(
    // Without limiting the page size, this call can take a long time. pageSize is
 just sugar for
    // the MaxResults property in the base command.
    { client, pageSize },
    {
      // There are almost 70,000 images available. Be specific with your filtering
      // to increase efficiency.
      // See https://docs.aws.amazon.com/AWSJavaScriptSDK/v3/latest/clients/client-
ec2/interfaces/describeimagescommandinput.html#filters
      Filters: [{ Name: "architecture", Values: [architecture] }],
    },
```

```
  );

  /**
   * @type {import('@aws-sdk/client-ec2').Image[]}
   */
  const images = [];
  let recordsScanned = 0;

  try {
    for await (const page of paginator) {
      recordsScanned += pageSize;
      if (page.Images.length) {
        images.push(...page.Images);
        break;
      } else {
        console.log(
          `No matching image found yet. Searched ${recordsScanned} records.`,
        );
      }
    }

    if (images.length) {
      console.log(
        `Found ${images.length} images:\n\n${images.map((image) =>
 image.Name).join("\n")}\n`,
      );
    } else {
      console.log(
        `No matching images found. Searched ${recordsScanned} records.\n`,
      );
    }

    return images;
  } catch (caught) {
    if (caught instanceof Error && caught.name === "InvalidParameterValue") {
      console.warn(`${caught.message}`);
      return [];
    }
    throw caught;
  }
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考DescribeImages中的。

`DescribeInstanceTypes`

下列程式碼範例會示範如何使用`DescribeInstanceTypes`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import { EC2Client, paginateDescribeInstanceTypes } from "@aws-sdk/client-ec2";

/**
 * Describes the specified instance types. By default, all instance types for the
 * current Region are described. Alternatively, you can filter the results.
 * @param {{ pageSize: string, supportedArch: string[], freeTier: boolean }} options
 */
export const main = async ({ pageSize, supportedArch, freeTier }) => {
  pageSize = parseInt(pageSize);
  const client = new EC2Client({});

  // The paginate function is a wrapper around the underlying command.
  const paginator = paginateDescribeInstanceTypes(
    // Without limiting the page size, this call can take a long time. pageSize is
 just sugar for
    // the MaxResults property in the underlying command.
    { client, pageSize },
    {
      Filters: [
        {
          Name: "processor-info.supported-architecture",
          Values: supportedArch,
        },
        { Name: "free-tier-eligible", Values: [freeTier ? "true" : "false"] },
      ],
    },
  );

  try {
    /**
     * @type {import('@aws-sdk/client-ec2').InstanceTypeInfo[]}
```

```
     */
    const instanceTypes = [];

    for await (const page of paginator) {
      if (page.InstanceTypes.length) {
        instanceTypes.push(...page.InstanceTypes);

        // When we have at least 1 result, we can stop.
        if (instanceTypes.length >= 1) {
          break;
        }
      }
    }
    console.log(
      `Memory size in MiB for matching instance types:\n\n${instanceTypes.map((it)
 => `${it.InstanceType}: ${it.MemoryInfo.SizeInMiB} MiB`).join("\n")}`,
    );
  } catch (caught) {
    if (caught instanceof Error && caught.name === "InvalidParameterValue") {
      console.warn(`${caught.message}`);
      return [];
    }
    throw caught;
  }
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考DescribeInstanceTypes中的。

`DescribeInstances`

下列程式碼範例會示範如何使用`DescribeInstances`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import { EC2Client, paginateDescribeInstances } from "@aws-sdk/client-ec2";
```

```
/**
 * List all of your EC2 instances running with the provided architecture that
 * were launched in the past month.
 * @param {{ pageSize: string, architectures: string[] }} options
 */
export const main = async ({ pageSize, architectures }) => {
  pageSize = parseInt(pageSize);
  const client = new EC2Client({});
  const d = new Date();
  const year = d.getFullYear();
  const month = `0${d.getMonth() + 1}`.slice(-2);
  const launchTimePattern = `${year}-${month}-*`;

  const paginator = paginateDescribeInstances(
    {
      client,
      pageSize,
    },
    {
      Filters: [
        { Name: "architecture", Values: architectures },
        { Name: "instance-state-name", Values: ["running"] },
        {
          Name: "launch-time",
          Values: [launchTimePattern],
        },
      ],
    },
  );

  try {
    /**
     * @type {import('@aws-sdk/client-ec2').Instance[]}
     */
    const instanceList = [];
    for await (const page of paginator) {
      const { Reservations } = page;
      Reservations.forEach((r) => instanceList.push(...r.Instances));
    }
    console.log(
      `Running instances launched this month:\n\n${instanceList.map((instance) =>
 instance.InstanceId).join("\n")}`,
    );
```

```
  } catch (caught) {
    if (caught instanceof Error && caught.name === "InvalidParameterValue") {
      console.warn(`${caught.message}.`);
    } else {
      throw caught;
    }
  }
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考DescribeInstances中的。

`DescribeKeyPairs`

下列程式碼範例會示範如何使用`DescribeKeyPairs`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import { DescribeKeyPairsCommand, EC2Client } from "@aws-sdk/client-ec2";

/**
 * List all key pairs in the current AWS account.
 * @param {{ dryRun: boolean }}
 */
export const main = async ({ dryRun }) => {
  const client = new EC2Client({});
  const command = new DescribeKeyPairsCommand({ DryRun: dryRun });

  try {
    const { KeyPairs } = await client.send(command);
    const keyPairList = KeyPairs.map(
      (kp) => ` • ${kp.KeyPairId}: ${kp.KeyName}`,
    ).join("\n");
    console.log("The following key pairs were found in your account:");
    console.log(keyPairList);
```

```
  } catch (caught) {
    if (caught instanceof Error && caught.name === "DryRunOperation") {
      console.log(`${caught.message}`);
    } else {
      throw caught;
    }
  }
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考DescribeKeyPairs中的。

`DescribeRegions`

下列程式碼範例會示範如何使用`DescribeRegions`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import { DescribeRegionsCommand, EC2Client } from "@aws-sdk/client-ec2";

/**
 * List all available AWS regions.
 * @param {{ regionNames: string[], includeOptInRegions: boolean }} options
 */
export const main = async ({ regionNames, includeOptInRegions }) => {
  const client = new EC2Client({});
  const command = new DescribeRegionsCommand({
    // By default this command will not show regions that require you to opt-in.
    // When AllRegions is true, even the regions that require opt-in will be
 returned.
    AllRegions: includeOptInRegions,
    // You can omit the Filters property if you want to get all regions.
    Filters: regionNames?.length
      ? [
          {
```

```
            Name: "region-name",
            // You can specify multiple values for a filter.
            // You can also use '*' as a wildcard. This will return all
            // of the regions that start with `us-east-`.
            Values: regionNames,
          },
        ]
      : undefined,
  });

  try {
    const { Regions } = await client.send(command);
    const regionsList = Regions.map((reg) => ` • ${reg.RegionName}`);
    console.log("Found regions:");
    console.log(regionsList.join("\n"));
  } catch (caught) {
    if (caught instanceof Error && caught.name === "DryRunOperation") {
      console.log(`${caught.message}`);
    } else {
      throw caught;
    }
  }
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考DescribeRegions中的。

`DescribeSecurityGroups`

下列程式碼範例會示範如何使用`DescribeSecurityGroups`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import { DescribeSecurityGroupsCommand, EC2Client } from "@aws-sdk/client-ec2";
```

```
/**
 * Describes the specified security groups or all of your security groups.
 * @param {{ groupIds: string[] }} options
 */
export const main = async ({ groupIds = [] }) => {
  const client = new EC2Client({});
  const command = new DescribeSecurityGroupsCommand({
    GroupIds: groupIds,
  });

  try {
    const { SecurityGroups } = await client.send(command);
    const sgList = SecurityGroups.map(
      (sg) => `• ${sg.GroupName} (${sg.GroupId}): ${sg.Description}`,
    ).join("\n");
    if (sgList.length) {
      console.log(`Security groups:\n${sgList}`);
    } else {
      console.log("No security groups found.");
    }
  } catch (caught) {
    if (caught instanceof Error && caught.name === "InvalidGroupId.Malformed") {
      console.warn(`${caught.message}. Please provide a valid GroupId.`);
    } else if (
      caught instanceof Error &&
      caught.name === "InvalidGroup.NotFound"
    ) {
      console.warn(caught.message);
    } else {
      throw caught;
    }
  }
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考DescribeSecurityGroups中的。

`DescribeSubnets`

下列程式碼範例會示範如何使用DescribeSubnets。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
const client = new EC2Client({});
const { Subnets } = await client.send(
  new DescribeSubnetsCommand({
    Filters: [
      { Name: "vpc-id", Values: [state.defaultVpc] },
      { Name: "availability-zone", Values: state.availabilityZoneNames },
      { Name: "default-for-az", Values: ["true"] },
    ],
  }),
);
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考DescribeSubnets中的。

`DescribeVpcs`

下列程式碼範例會示範如何使用`DescribeVpcs`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
const client = new EC2Client({});
const { Vpcs } = await client.send(
  new DescribeVpcsCommand({
    Filters: [{ Name: "is-default", Values: ["true"] }],
  }),
);
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考DescribeVpcs中的。

`DisassociateAddress`

下列程式碼範例會示範如何使用DisassociateAddress。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import { DisassociateAddressCommand, EC2Client } from "@aws-sdk/client-ec2";

/**
 * Disassociate an Elastic IP address from an instance.
 * @param {{ associationId: string }} options
 */
export const main = async ({ associationId }) => {
  const client = new EC2Client({});
  const command = new DisassociateAddressCommand({
    // You can also use PublicIp, but that is for EC2 classic which is being
 retired.
    AssociationId: associationId,
  });

  try {
    await client.send(command);
    console.log("Successfully disassociated address");
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "InvalidAssociationID.NotFound"
    ) {
      console.warn(`${caught.message}.`);
    } else {
      throw caught;
    }
```

```
  }
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考DisassociateAddress中的。

`MonitorInstances`

下列程式碼範例會示範如何使用MonitorInstances。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import { EC2Client, MonitorInstancesCommand } from "@aws-sdk/client-ec2";

/**
 * Turn on detailed monitoring for the selected instance.
 * By default, metrics are sent to Amazon CloudWatch every 5 minutes.
 * For a cost you can enable detailed monitoring which sends metrics every minute.
 * @param {{ instanceIds: string[] }} options
 */
export const main = async ({ instanceIds }) => {
  const client = new EC2Client({});
  const command = new MonitorInstancesCommand({
    InstanceIds: instanceIds,
  });

  try {
    const { InstanceMonitorings } = await client.send(command);
    const instancesBeingMonitored = InstanceMonitorings.map(
      (im) =>
        `  • Detailed monitoring state for ${im.InstanceId} is
 ${im.Monitoring.State}.`,
    );
    console.log("Monitoring status:");
    console.log(instancesBeingMonitored.join("\n"));
  } catch (caught) {
```

```
    if (caught instanceof Error && caught.name === "InvalidParameterValue") {
      console.warn(`${caught.message}`);
    } else {
      throw caught;
    }
  }
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考MonitorInstances中的。

`RebootInstances`

下列程式碼範例會示範如何使用RebootInstances。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import { EC2Client, RebootInstancesCommand } from "@aws-sdk/client-ec2";

/**
 * Requests a reboot of the specified instances. This operation is asynchronous;
 * it only queues a request to reboot the specified instances.
 * @param {{ instanceIds: string[] }} options
 */
export const main = async ({ instanceIds }) => {
  const client = new EC2Client({});
  const command = new RebootInstancesCommand({
    InstanceIds: instanceIds,
  });

  try {
    await client.send(command);
    console.log("Instance rebooted successfully.");
  } catch (caught) {
    if (
      caught instanceof Error &&
```

```
      caught.name === "InvalidInstanceID.NotFound"
    ) {
      console.warn(
        `${caught.message}. Please provide the InstanceId of a valid instance to
 reboot.`,
      );
    } else {
      throw caught;
    }
  }
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考RebootInstances中的。

`ReleaseAddress`

下列程式碼範例會示範如何使用`ReleaseAddress`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import { ReleaseAddressCommand, EC2Client } from "@aws-sdk/client-ec2";

/**
 * Release an Elastic IP address.
 * @param {{ allocationId: string }} options
 */
export const main = async ({ allocationId }) => {
  const client = new EC2Client({});
  const command = new ReleaseAddressCommand({
    // You can also use PublicIp, but that is for EC2 classic which is being
 retired.
    AllocationId: allocationId,
  });

  try {
```

```
    await client.send(command);
    console.log("Successfully released address.");
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "InvalidAllocationID.NotFound"
    ) {
      console.warn(`${caught.message}. Please provide a valid AllocationID.`);
    } else {
      throw caught;
    }
  }
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考ReleaseAddress中的。

`ReplaceIamInstanceProfileAssociation`

下列程式碼範例會示範如何使用`ReplaceIamInstanceProfileAssociation`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
      ec2Client.send(
        new ReplaceIamInstanceProfileAssociationCommand({
          AssociationId: state.instanceProfileAssociationId,
          IamInstanceProfile: { Name: NAMES.ssmOnlyInstanceProfileName },
        }),
      ),
    );
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考ReplaceIamInstanceProfileAssociation中的。

`RunInstances`

下列程式碼範例會示範如何使用RunInstances。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import { EC2Client, RunInstancesCommand } from "@aws-sdk/client-ec2";

/**
 * Create new EC2 instances.
 * @param {{
 *  keyName: string,
 *  securityGroupIds: string[],
 *  imageId: string,
 *  instanceType: import('@aws-sdk/client-ec2')._InstanceType,
 *  minCount?: number,
 *  maxCount?: number }} options
 */
export const main = async ({
  keyName,
  securityGroupIds,
  imageId,
  instanceType,
  minCount = "1",
  maxCount = "1",
}) => {
  const client = new EC2Client({});
  minCount = parseInt(minCount);
  maxCount = parseInt(maxCount);
  const command = new RunInstancesCommand({
    // Your key pair name.
    KeyName: keyName,
    // Your security group.
    SecurityGroupIds: securityGroupIds,
    // An Amazon Machine Image (AMI). There are multiple ways to search for AMIs.
 For more information, see:
    // https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/finding-an-ami.html
```

```
    ImageId: imageId,
    // An instance type describing the resources provided to your instance. There
 are multiple
    // ways to search for instance types. For more information see:
    // https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/instance-discovery.html
    InstanceType: instanceType,
    // Availability Zones have capacity limitations that may impact your ability to
 launch instances.
    // The `RunInstances` operation will only succeed if it can allocate at least
 the `MinCount` of instances.
    // However, EC2 will attempt to launch up to the `MaxCount` of instances, even
 if the full request cannot be satisfied.
    // If you need a specific number of instances, use `MinCount` and `MaxCount` set
 to the same value.
    // If you want to launch up to a certain number of instances, use `MaxCount` and
 let EC2 provision as many as possible.
    // If you require a minimum number of instances, but do not want to exceed a
 maximum, use both `MinCount` and `MaxCount`.
    MinCount: minCount,
    MaxCount: maxCount,
  });

  try {
    const { Instances } = await client.send(command);
    const instanceList = Instances.map(
      (instance) => `• ${instance.InstanceId}`,
    ).join("\n");
    console.log(`Launched instances:\n${instanceList}`);
  } catch (caught) {
    if (caught instanceof Error && caught.name === "ResourceCountExceeded") {
      console.warn(`${caught.message}`);
    } else {
      throw caught;
    }
  }
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考RunInstances中的。

`StartInstances`

下列程式碼範例會示範如何使用`StartInstances`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import { EC2Client, StartInstancesCommand } from "@aws-sdk/client-ec2";
import { fileURLToPath } from "url";
import { parseArgs } from "util";

/**
 * Starts an Amazon EBS-backed instance that you've previously stopped.
 * @param {{ instanceIds }} options
 */
export const main = async ({ instanceIds }) => {
  const client = new EC2Client({});
  const command = new StartInstancesCommand({
    InstanceIds: instanceIds,
  });

  try {
    const { StartingInstances } = await client.send(command);
    const instanceIdList = StartingInstances.map(
      (instance) => ` • ${instance.InstanceId}`,
    );
    console.log("Starting instances:");
    console.log(instanceIdList.join("\n"));
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "InvalidInstanceID.NotFound"
    ) {
      console.warn(`${caught.message}`);
    } else {
      throw caught;
    }
  }
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考StartInstances中的。

`StopInstances`

下列程式碼範例會示範如何使用StopInstances。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import { EC2Client, StopInstancesCommand } from "@aws-sdk/client-ec2";
import { fileURLToPath } from "url";
import { parseArgs } from "util";

/**
 * Stop one or more EC2 instances.
 * @param {{ instanceIds: string[] }} options
 */
export const main = async ({ instanceIds }) => {
  const client = new EC2Client({});
  const command = new StopInstancesCommand({
    InstanceIds: instanceIds,
  });

  try {
    const { StoppingInstances } = await client.send(command);
    const instanceIdList = StoppingInstances.map(
      (instance) => ` • ${instance.InstanceId}`,
    );
    console.log("Stopping instances:");
    console.log(instanceIdList.join("\n"));
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "InvalidInstanceID.NotFound"
    ) {
      console.warn(`${caught.message}`);
    } else {
```

```
      throw caught;
    }
  }
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考StopInstances中的。

`TerminateInstances`

下列程式碼範例會示範如何使用TerminateInstances。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import { EC2Client, TerminateInstancesCommand } from "@aws-sdk/client-ec2";
import { fileURLToPath } from "url";
import { parseArgs } from "util";

/**
 * Terminate one or more EC2 instances.
 * @param {{ instanceIds: string[] }} options
 */
export const main = async ({ instanceIds }) => {
  const client = new EC2Client({});
  const command = new TerminateInstancesCommand({
    InstanceIds: instanceIds,
  });

  try {
    const { TerminatingInstances } = await client.send(command);
    const instanceList = TerminatingInstances.map(
      (instance) => ` • ${instance.InstanceId}`,
    );
    console.log("Terminating instances:");
    console.log(instanceList.join("\n"));
  } catch (caught) {
```

```
    if (
      caught instanceof Error &&
      caught.name === "InvalidInstanceID.NotFound"
    ) {
      console.warn(`${caught.message}`);
    } else {
      throw caught;
    }
  }
  ``;
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考TerminateInstances中的。

`UnmonitorInstances`

下列程式碼範例會示範如何使用`UnmonitorInstances`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import { EC2Client, UnmonitorInstancesCommand } from "@aws-sdk/client-ec2";
import { fileURLToPath } from "url";
import { parseArgs } from "util";

/**
 * Turn off detailed monitoring for the selected instance.
 * @param {{ instanceIds: string[] }} options
 */
export const main = async ({ instanceIds }) => {
  const client = new EC2Client({});
  const command = new UnmonitorInstancesCommand({
    InstanceIds: instanceIds,
  });
```

```
  try {
    const { InstanceMonitorings } = await client.send(command);
    const instanceMonitoringsList = InstanceMonitorings.map(
      (im) =>
        ` • Detailed monitoring state for ${im.InstanceId} is
 ${im.Monitoring.State}.`,
    );
    console.log("Monitoring status:");
    console.log(instanceMonitoringsList.join("\n"));
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "InvalidInstanceID.NotFound"
    ) {
      console.warn(`${caught.message}`);
    } else {
      throw caught;
    }
  }
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考UnmonitorInstances中的。

案例

建置及管理彈性服務

下列程式碼範例會示範如何建立負載平衡的 Web 服務，以傳回書籍、影片和歌曲建議。此範例顯示服務如何回應失故障，以及如何在發生故障時重組服務以提高復原能力。

- 使用 Amazon EC2 Auto Scaling 群組根據啟動範本建立 Amazon 彈性運算雲端 (AmazonEC2) 執行個體，並將執行個體數量保持在指定範圍內。
- 使用 Elastic Load Balancing 處理和分配HTTP請求。
- 監控 Auto Scaling 群組中執行個體的運作狀態，並且只將請求轉送給運作良好的執行個體。
- 在每個執行個EC2體上執行 Python 網頁伺服器來處理HTTP要求。Web 伺服器會回應建議和運作狀態檢查。
- 使用 Amazon DynamoDB 資料表模擬建議服務。
- 透過更新 AWS Systems Manager 參數來控制 Web 伺服器對要求和健康狀態檢查的回應。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

在命令提示中執行互動式案例。

```
#!/usr/bin/env node
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0


import {
  Scenario,
  parseScenarioArgs,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

/**
 * The workflow steps are split into three stages:
 *   - deploy
 *   - demo
 *   - destroy
 *
 * Each of these stages has a corresponding file prefixed with steps-*.
 */
import { deploySteps } from "./steps-deploy.js";
import { demoSteps } from "./steps-demo.js";
import { destroySteps } from "./steps-destroy.js";

/**
 * The context is passed to every scenario. Scenario steps
 * will modify the context.
 */
const context = {};

/**
 * Three Scenarios are created for the workflow. A Scenario is an orchestration
 class
 * that simplifies running a series of steps.
 */
```

```
export const scenarios = {
  // Deploys all resources necessary for the workflow.
  deploy: new Scenario("Resilient Workflow - Deploy", deploySteps, context),
  // Demonstrates how a fragile web service can be made more resilient.
  demo: new Scenario("Resilient Workflow - Demo", demoSteps, context),
  // Destroys the resources created for the workflow.
  destroy: new Scenario("Resilient Workflow - Destroy", destroySteps, context),
};

// Call function if run directly
import { fileURLToPath } from "url";

if (process.argv[1] === fileURLToPath(import.meta.url)) {
  parseScenarioArgs(scenarios);
}
```

建立步驟以部署所有資源。

```
import { join } from "node:path";
import { readFileSync, writeFileSync } from "node:fs";
import axios from "axios";

import {
  BatchWriteItemCommand,
  CreateTableCommand,
  DynamoDBClient,
  waitUntilTableExists,
} from "@aws-sdk/client-dynamodb";
import {
  EC2Client,
  CreateKeyPairCommand,
  CreateLaunchTemplateCommand,
  DescribeAvailabilityZonesCommand,
  DescribeVpcsCommand,
  DescribeSubnetsCommand,
  DescribeSecurityGroupsCommand,
  AuthorizeSecurityGroupIngressCommand,
} from "@aws-sdk/client-ec2";
import {
  IAMClient,
  CreatePolicyCommand,
  CreateRoleCommand,
```

```
  CreateInstanceProfileCommand,
  AddRoleToInstanceProfileCommand,
  AttachRolePolicyCommand,
  waitUntilInstanceProfileExists,
} from "@aws-sdk/client-iam";
import { SSMClient, GetParameterCommand } from "@aws-sdk/client-ssm";
import {
  CreateAutoScalingGroupCommand,
  AutoScalingClient,
  AttachLoadBalancerTargetGroupsCommand,
} from "@aws-sdk/client-auto-scaling";
import {
  CreateListenerCommand,
  CreateLoadBalancerCommand,
  CreateTargetGroupCommand,
  ElasticLoadBalancingV2Client,
  waitUntilLoadBalancerAvailable,
} from "@aws-sdk/client-elastic-load-balancing-v2";

import {
  ScenarioOutput,
  ScenarioInput,
  ScenarioAction,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import { saveState } from "@aws-doc-sdk-examples/lib/scenario/steps-common.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

import { MESSAGES, NAMES, RESOURCES_PATH, ROOT } from "./constants.js";
import { initParamsSteps } from "./steps-reset-params.js";

/**
 * @type {import('@aws-doc-sdk-examples/lib/scenario.js').Step[]}
 */
export const deploySteps = [
  new ScenarioOutput("introduction", MESSAGES.introduction, { header: true }),
  new ScenarioInput("confirmDeployment", MESSAGES.confirmDeployment, {
    type: "confirm",
  }),
  new ScenarioAction(
    "handleConfirmDeployment",
    (c) => c.confirmDeployment === false && process.exit(),
  ),
  new ScenarioOutput(
    "creatingTable",
```

```
    MESSAGES.creatingTable.replace("${TABLE_NAME}", NAMES.tableName),
  ),
  new ScenarioAction("createTable", async () => {
    const client = new DynamoDBClient({});
    await client.send(
      new CreateTableCommand({
        TableName: NAMES.tableName,
        ProvisionedThroughput: {
          ReadCapacityUnits: 5,
          WriteCapacityUnits: 5,
        },
        AttributeDefinitions: [
          {
            AttributeName: "MediaType",
            AttributeType: "S",
          },
          {
            AttributeName: "ItemId",
            AttributeType: "N",
          },
        ],
        KeySchema: [
          {
            AttributeName: "MediaType",
            KeyType: "HASH",
          },
          {
            AttributeName: "ItemId",
            KeyType: "RANGE",
          },
        ],
      }),
    );
    await waitUntilTableExists({ client }, { TableName: NAMES.tableName });
  }),
  new ScenarioOutput(
    "createdTable",
    MESSAGES.createdTable.replace("${TABLE_NAME}", NAMES.tableName),
  ),
  new ScenarioOutput(
    "populatingTable",
    MESSAGES.populatingTable.replace("${TABLE_NAME}", NAMES.tableName),
  ),
  new ScenarioAction("populateTable", () => {
```

```
    const client = new DynamoDBClient({});
    /**
     * @type {{ default: import("@aws-sdk/client-dynamodb").PutRequest['Item'][] }}
     */
    const recommendations = JSON.parse(
      readFileSync(join(RESOURCES_PATH, "recommendations.json")),
    );

    return client.send(
      new BatchWriteItemCommand({
        RequestItems: {
          [NAMES.tableName]: recommendations.map((item) => ({
            PutRequest: { Item: item },
          })),
        },
      }),
    );
  }),
  new ScenarioOutput(
    "populatedTable",
    MESSAGES.populatedTable.replace("${TABLE_NAME}", NAMES.tableName),
  ),
  new ScenarioOutput(
    "creatingKeyPair",
    MESSAGES.creatingKeyPair.replace("${KEY_PAIR_NAME}", NAMES.keyPairName),
  ),
  new ScenarioAction("createKeyPair", async () => {
    const client = new EC2Client({});
    const { KeyMaterial } = await client.send(
      new CreateKeyPairCommand({
        KeyName: NAMES.keyPairName,
      }),
    );

    writeFileSync(`${NAMES.keyPairName}.pem`, KeyMaterial, { mode: 0o600 });
  }),
  new ScenarioOutput(
    "createdKeyPair",
    MESSAGES.createdKeyPair.replace("${KEY_PAIR_NAME}", NAMES.keyPairName),
  ),
  new ScenarioOutput(
    "creatingInstancePolicy",
    MESSAGES.creatingInstancePolicy.replace(
      "${INSTANCE_POLICY_NAME}",
```

```
    NAMES.instancePolicyName,
  ),
),
new ScenarioAction("createInstancePolicy", async (state) => {
  const client = new IAMClient({});
  const {
    Policy: { Arn },
  } = await client.send(
    new CreatePolicyCommand({
      PolicyName: NAMES.instancePolicyName,
      PolicyDocument: readFileSync(
        join(RESOURCES_PATH, "instance_policy.json"),
      ),
    }),
  );
  state.instancePolicyArn = Arn;
}),
new ScenarioOutput("createdInstancePolicy", (state) =>
  MESSAGES.createdInstancePolicy
    .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
    .replace("${INSTANCE_POLICY_ARN}", state.instancePolicyArn),
),
new ScenarioOutput(
  "creatingInstanceRole",
  MESSAGES.creatingInstanceRole.replace(
    "${INSTANCE_ROLE_NAME}",
    NAMES.instanceRoleName,
  ),
),
new ScenarioAction("createInstanceRole", () => {
  const client = new IAMClient({});
  return client.send(
    new CreateRoleCommand({
      RoleName: NAMES.instanceRoleName,
      AssumeRolePolicyDocument: readFileSync(
        join(ROOT, "assume-role-policy.json"),
      ),
    }),
  );
}),
new ScenarioOutput(
  "createdInstanceRole",
  MESSAGES.createdInstanceRole.replace(
    "${INSTANCE_ROLE_NAME}",
```

```
      NAMES.instanceRoleName,
    ),
  ),
  new ScenarioOutput(
    "attachingPolicyToRole",
    MESSAGES.attachingPolicyToRole
      .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName)
      .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName),
  ),
  new ScenarioAction("attachPolicyToRole", async (state) => {
    const client = new IAMClient({});
    await client.send(
      new AttachRolePolicyCommand({
        RoleName: NAMES.instanceRoleName,
        PolicyArn: state.instancePolicyArn,
      }),
    );
  }),
  new ScenarioOutput(
    "attachedPolicyToRole",
    MESSAGES.attachedPolicyToRole
      .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
      .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName),
  ),
  new ScenarioOutput(
    "creatingInstanceProfile",
    MESSAGES.creatingInstanceProfile.replace(
      "${INSTANCE_PROFILE_NAME}",
      NAMES.instanceProfileName,
    ),
  ),
  new ScenarioAction("createInstanceProfile", async (state) => {
    const client = new IAMClient({});
    const {
      InstanceProfile: { Arn },
    } = await client.send(
      new CreateInstanceProfileCommand({
        InstanceProfileName: NAMES.instanceProfileName,
      }),
    );
    state.instanceProfileArn = Arn;

    await waitUntilInstanceProfileExists(
      { client },
```

```
      { InstanceProfileName: NAMES.instanceProfileName },
    );
  }),
  new ScenarioOutput("createdInstanceProfile", (state) =>
    MESSAGES.createdInstanceProfile
      .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
      .replace("${INSTANCE_PROFILE_ARN}", state.instanceProfileArn),
  ),
  new ScenarioOutput(
    "addingRoleToInstanceProfile",
    MESSAGES.addingRoleToInstanceProfile
      .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
      .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName),
  ),
  new ScenarioAction("addRoleToInstanceProfile", () => {
    const client = new IAMClient({});
    return client.send(
      new AddRoleToInstanceProfileCommand({
        RoleName: NAMES.instanceRoleName,
        InstanceProfileName: NAMES.instanceProfileName,
      }),
    );
  }),
  new ScenarioOutput(
    "addedRoleToInstanceProfile",
    MESSAGES.addedRoleToInstanceProfile
      .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
      .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName),
  ),
  ...initParamsSteps,
  new ScenarioOutput("creatingLaunchTemplate", MESSAGES.creatingLaunchTemplate),
  new ScenarioAction("createLaunchTemplate", async () => {
    const ssmClient = new SSMClient({});
    const { Parameter } = await ssmClient.send(
      new GetParameterCommand({
        Name: "/aws/service/ami-amazon-linux-latest/amzn2-ami-hvm-x86_64-gp2",
      }),
    );
    const ec2Client = new EC2Client({});
    await ec2Client.send(
      new CreateLaunchTemplateCommand({
        LaunchTemplateName: NAMES.launchTemplateName,
        LaunchTemplateData: {
          InstanceType: "t3.micro",
```

```
          ImageId: Parameter.Value,
          IamInstanceProfile: { Name: NAMES.instanceProfileName },
          UserData: readFileSync(
            join(RESOURCES_PATH, "server_startup_script.sh"),
          ).toString("base64"),
          KeyName: NAMES.keyPairName,
        },
      }),
    );
  }),
  new ScenarioOutput(
    "createdLaunchTemplate",
    MESSAGES.createdLaunchTemplate.replace(
      "${LAUNCH_TEMPLATE_NAME}",
      NAMES.launchTemplateName,
    ),
  ),
  new ScenarioOutput(
    "creatingAutoScalingGroup",
    MESSAGES.creatingAutoScalingGroup.replace(
      "${AUTO_SCALING_GROUP_NAME}",
      NAMES.autoScalingGroupName,
    ),
  ),
  new ScenarioAction("createAutoScalingGroup", async (state) => {
    const ec2Client = new EC2Client({});
    const { AvailabilityZones } = await ec2Client.send(
      new DescribeAvailabilityZonesCommand({}),
    );
    state.availabilityZoneNames = AvailabilityZones.map((az) => az.ZoneName);
    const autoScalingClient = new AutoScalingClient({});
    await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
      autoScalingClient.send(
        new CreateAutoScalingGroupCommand({
          AvailabilityZones: state.availabilityZoneNames,
          AutoScalingGroupName: NAMES.autoScalingGroupName,
          LaunchTemplate: {
            LaunchTemplateName: NAMES.launchTemplateName,
            Version: "$Default",
          },
          MinSize: 3,
          MaxSize: 3,
        }),
      ),
```

```
    );
  }),
  new ScenarioOutput(
    "createdAutoScalingGroup",
    /**
     * @param {{ availabilityZoneNames: string[] }} state
     */
    (state) =>
      MESSAGES.createdAutoScalingGroup
        .replace("${AUTO_SCALING_GROUP_NAME}", NAMES.autoScalingGroupName)
        .replace(
          "${AVAILABILITY_ZONE_NAMES}",
          state.availabilityZoneNames.join(", "),
        ),
  ),
  new ScenarioInput("confirmContinue", MESSAGES.confirmContinue, {
    type: "confirm",
  }),
  new ScenarioOutput("loadBalancer", MESSAGES.loadBalancer),
  new ScenarioOutput("gettingVpc", MESSAGES.gettingVpc),
  new ScenarioAction("getVpc", async (state) => {
    const client = new EC2Client({});
    const { Vpcs } = await client.send(
      new DescribeVpcsCommand({
        Filters: [{ Name: "is-default", Values: ["true"] }],
      }),
    );
    state.defaultVpc = Vpcs[0].VpcId;
  }),
  new ScenarioOutput("gotVpc", (state) =>
    MESSAGES.gotVpc.replace("${VPC_ID}", state.defaultVpc),
  ),
  new ScenarioOutput("gettingSubnets", MESSAGES.gettingSubnets),
  new ScenarioAction("getSubnets", async (state) => {
    const client = new EC2Client({});
    const { Subnets } = await client.send(
      new DescribeSubnetsCommand({
        Filters: [
          { Name: "vpc-id", Values: [state.defaultVpc] },
          { Name: "availability-zone", Values: state.availabilityZoneNames },
          { Name: "default-for-az", Values: ["true"] },
        ],
      }),
    );
```

```
    state.subnets = Subnets.map((subnet) => subnet.SubnetId);
  }),
  new ScenarioOutput(
    "gotSubnets",
    /**
     * @param {{ subnets: string[] }} state
     */
    (state) =>
      MESSAGES.gotSubnets.replace("${SUBNETS}", state.subnets.join(", ")),
  ),
  new ScenarioOutput(
    "creatingLoadBalancerTargetGroup",
    MESSAGES.creatingLoadBalancerTargetGroup.replace(
      "${TARGET_GROUP_NAME}",
      NAMES.loadBalancerTargetGroupName,
    ),
  ),
  new ScenarioAction("createLoadBalancerTargetGroup", async (state) => {
    const client = new ElasticLoadBalancingV2Client({});
    const { TargetGroups } = await client.send(
      new CreateTargetGroupCommand({
        Name: NAMES.loadBalancerTargetGroupName,
        Protocol: "HTTP",
        Port: 80,
        HealthCheckPath: "/healthcheck",
        HealthCheckIntervalSeconds: 10,
        HealthCheckTimeoutSeconds: 5,
        HealthyThresholdCount: 2,
        UnhealthyThresholdCount: 2,
        VpcId: state.defaultVpc,
      }),
    );
    const targetGroup = TargetGroups[0];
    state.targetGroupArn = targetGroup.TargetGroupArn;
    state.targetGroupProtocol = targetGroup.Protocol;
    state.targetGroupPort = targetGroup.Port;
  }),
  new ScenarioOutput(
    "createdLoadBalancerTargetGroup",
    MESSAGES.createdLoadBalancerTargetGroup.replace(
      "${TARGET_GROUP_NAME}",
      NAMES.loadBalancerTargetGroupName,
    ),
  ),
```

```
  new ScenarioOutput(
    "creatingLoadBalancer",
    MESSAGES.creatingLoadBalancer.replace("${LB_NAME}", NAMES.loadBalancerName),
  ),
  new ScenarioAction("createLoadBalancer", async (state) => {
    const client = new ElasticLoadBalancingV2Client({});
    const { LoadBalancers } = await client.send(
      new CreateLoadBalancerCommand({
        Name: NAMES.loadBalancerName,
        Subnets: state.subnets,
      }),
    );
    state.loadBalancerDns = LoadBalancers[0].DNSName;
    state.loadBalancerArn = LoadBalancers[0].LoadBalancerArn;
    await waitUntilLoadBalancerAvailable(
      { client },
      { Names: [NAMES.loadBalancerName] },
    );
  }),
  new ScenarioOutput("createdLoadBalancer", (state) =>
    MESSAGES.createdLoadBalancer
      .replace("${LB_NAME}", NAMES.loadBalancerName)
      .replace("${DNS_NAME}", state.loadBalancerDns),
  ),
  new ScenarioOutput(
    "creatingListener",
    MESSAGES.creatingLoadBalancerListener
      .replace("${LB_NAME}", NAMES.loadBalancerName)
      .replace("${TARGET_GROUP_NAME}", NAMES.loadBalancerTargetGroupName),
  ),
  new ScenarioAction("createListener", async (state) => {
    const client = new ElasticLoadBalancingV2Client({});
    const { Listeners } = await client.send(
      new CreateListenerCommand({
        LoadBalancerArn: state.loadBalancerArn,
        Protocol: state.targetGroupProtocol,
        Port: state.targetGroupPort,
        DefaultActions: [
          { Type: "forward", TargetGroupArn: state.targetGroupArn },
        ],
      }),
    );
    const listener = Listeners[0];
    state.loadBalancerListenerArn = listener.ListenerArn;
```

```
  }),
  new ScenarioOutput("createdListener", (state) =>
    MESSAGES.createdLoadBalancerListener.replace(
      "${LB_LISTENER_ARN}",
      state.loadBalancerListenerArn,
    ),
  ),
  new ScenarioOutput(
    "attachingLoadBalancerTargetGroup",
    MESSAGES.attachingLoadBalancerTargetGroup
      .replace("${TARGET_GROUP_NAME}", NAMES.loadBalancerTargetGroupName)
      .replace("${AUTO_SCALING_GROUP_NAME}", NAMES.autoScalingGroupName),
  ),
  new ScenarioAction("attachLoadBalancerTargetGroup", async (state) => {
    const client = new AutoScalingClient({});
    await client.send(
      new AttachLoadBalancerTargetGroupsCommand({
        AutoScalingGroupName: NAMES.autoScalingGroupName,
        TargetGroupARNs: [state.targetGroupArn],
      }),
    );
  }),
  new ScenarioOutput(
    "attachedLoadBalancerTargetGroup",
    MESSAGES.attachedLoadBalancerTargetGroup,
  ),
  new ScenarioOutput("verifyingInboundPort", MESSAGES.verifyingInboundPort),
  new ScenarioAction(
    "verifyInboundPort",
    /**
     *
     * @param {{ defaultSecurityGroup: import('@aws-sdk/client-ec2').SecurityGroup}}
state
     */
    async (state) => {
      const client = new EC2Client({});
      const { SecurityGroups } = await client.send(
        new DescribeSecurityGroupsCommand({
          Filters: [{ Name: "group-name", Values: ["default"] }],
        }),
      );
      if (!SecurityGroups) {
        state.verifyInboundPortError = new Error(MESSAGES.noSecurityGroups);
      }
```

```
      state.defaultSecurityGroup = SecurityGroups[0];

      /**
       * @type {string}
       */
      const ipResponse = (await axios.get("http://checkip.amazonaws.com")).data;
      state.myIp = ipResponse.trim();
      const myIpRules = state.defaultSecurityGroup.IpPermissions.filter(
        ({ IpRanges }) =>
          IpRanges.some(
            ({ CidrIp }) =>
              CidrIp.startsWith(state.myIp) || CidrIp === "0.0.0.0/0",
          ),
      )
        .filter(({ IpProtocol }) => IpProtocol === "tcp")
        .filter(({ FromPort }) => FromPort === 80);

      state.myIpRules = myIpRules;
    },
  ),
  new ScenarioOutput(
    "verifiedInboundPort",
    /**
     * @param {{ myIpRules: any[] }} state
     */
    (state) => {
      if (state.myIpRules.length > 0) {
        return MESSAGES.foundIpRules.replace(
          "${IP_RULES}",
          JSON.stringify(state.myIpRules, null, 2),
        );
      } else {
        return MESSAGES.noIpRules;
      }
    },
  ),
  new ScenarioInput(
    "shouldAddInboundRule",
    /**
     * @param {{ myIpRules: any[] }} state
     */
    (state) => {
      if (state.myIpRules.length > 0) {
        return false;
```

```
      } else {
        return MESSAGES.noIpRules;
      }
    },
    { type: "confirm" },
  ),
  new ScenarioAction(
    "addInboundRule",
    /**
     * @param {{ defaultSecurityGroup: import('@aws-sdk/client-
ec2').SecurityGroup }} state
     */
    async (state) => {
      if (!state.shouldAddInboundRule) {
        return;
      }

      const client = new EC2Client({});
      await client.send(
        new AuthorizeSecurityGroupIngressCommand({
          GroupId: state.defaultSecurityGroup.GroupId,
          CidrIp: `${state.myIp}/32`,
          FromPort: 80,
          ToPort: 80,
          IpProtocol: "tcp",
        }),
      );
    },
  ),
  new ScenarioOutput("addedInboundRule", (state) => {
    if (state.shouldAddInboundRule) {
      return MESSAGES.addedInboundRule.replace("${IP_ADDRESS}", state.myIp);
    } else {
      return false;
    }
  }),
  new ScenarioOutput("verifyingEndpoint", (state) =>
    MESSAGES.verifyingEndpoint.replace("${DNS_NAME}", state.loadBalancerDns),
  ),
  new ScenarioAction("verifyEndpoint", async (state) => {
    try {
      const response = await retry({ intervalInMs: 2000, maxRetries: 30 }, () =>
        axios.get(`http://${state.loadBalancerDns}`),
      );
```

```
      state.endpointResponse = JSON.stringify(response.data, null, 2);
    } catch (e) {
      state.verifyEndpointError = e;
    }
  }),
  new ScenarioOutput("verifiedEndpoint", (state) => {
    if (state.verifyEndpointError) {
      console.error(state.verifyEndpointError);
    } else {
      return MESSAGES.verifiedEndpoint.replace(
        "${ENDPOINT_RESPONSE}",
        state.endpointResponse,
      );
    }
  }),
  saveState,
];
```

建立步驟以執行示範。

```
import { readFileSync } from "node:fs";
import { join } from "node:path";

import axios from "axios";

import {
  DescribeTargetGroupsCommand,
  DescribeTargetHealthCommand,
  ElasticLoadBalancingV2Client,
} from "@aws-sdk/client-elastic-load-balancing-v2";
import {
  DescribeInstanceInformationCommand,
  PutParameterCommand,
  SSMClient,
  SendCommandCommand,
} from "@aws-sdk/client-ssm";
import {
  IAMClient,
  CreatePolicyCommand,
  CreateRoleCommand,
  AttachRolePolicyCommand,
  CreateInstanceProfileCommand,
```

```
  AddRoleToInstanceProfileCommand,
  waitUntilInstanceProfileExists,
} from "@aws-sdk/client-iam";
import {
  AutoScalingClient,
  DescribeAutoScalingGroupsCommand,
  TerminateInstanceInAutoScalingGroupCommand,
} from "@aws-sdk/client-auto-scaling";
import {
  DescribeIamInstanceProfileAssociationsCommand,
  EC2Client,
  RebootInstancesCommand,
  ReplaceIamInstanceProfileAssociationCommand,
} from "@aws-sdk/client-ec2";

import {
  ScenarioAction,
  ScenarioInput,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/scenario.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

import { MESSAGES, NAMES, RESOURCES_PATH } from "./constants.js";
import { findLoadBalancer } from "./shared.js";

const getRecommendation = new ScenarioAction(
  "getRecommendation",
  async (state) => {
    const loadBalancer = await findLoadBalancer(NAMES.loadBalancerName);
    if (loadBalancer) {
      state.loadBalancerDnsName = loadBalancer.DNSName;
      try {
        state.recommendation = (
          await axios.get(`http://${state.loadBalancerDnsName}`)
        ).data;
      } catch (e) {
        state.recommendation = e instanceof Error ? e.message : e;
      }
    } else {
      throw new Error(MESSAGES.demoFindLoadBalancerError);
    }
  },
);
```

```
const getRecommendationResult = new ScenarioOutput(
  "getRecommendationResult",
  (state) =>
    `Recommendation:\n${JSON.stringify(state.recommendation, null, 2)}`,
  { preformatted: true },
);

const getHealthCheck = new ScenarioAction("getHealthCheck", async (state) => {
  const client = new ElasticLoadBalancingV2Client({});
  const { TargetGroups } = await client.send(
    new DescribeTargetGroupsCommand({
      Names: [NAMES.loadBalancerTargetGroupName],
    }),
  );

  const { TargetHealthDescriptions } = await client.send(
    new DescribeTargetHealthCommand({
      TargetGroupArn: TargetGroups[0].TargetGroupArn,
    }),
  );
  state.targetHealthDescriptions = TargetHealthDescriptions;
});

const getHealthCheckResult = new ScenarioOutput(
  "getHealthCheckResult",
  /**
   * @param {{ targetHealthDescriptions: import('@aws-sdk/client-elastic-load-
balancing-v2').TargetHealthDescription[]}} state
   */
  (state) => {
    const status = state.targetHealthDescriptions
      .map((th) => `${th.Target.Id}: ${th.TargetHealth.State}`)
      .join("\n");
    return `Health check:\n${status}`;
  },
  { preformatted: true },
);

const loadBalancerLoop = new ScenarioAction(
  "loadBalancerLoop",
  getRecommendation.action,
  {
    whileConfig: {
      whileFn: ({ loadBalancerCheck }) => loadBalancerCheck,
```

```
      input: new ScenarioInput(
        "loadBalancerCheck",
        MESSAGES.demoLoadBalancerCheck,
        {
          type: "confirm",
        },
      ),
      output: getRecommendationResult,
    },
  },
);

const healthCheckLoop = new ScenarioAction(
  "healthCheckLoop",
  getHealthCheck.action,
  {
    whileConfig: {
      whileFn: ({ healthCheck }) => healthCheck,
      input: new ScenarioInput("healthCheck", MESSAGES.demoHealthCheck, {
        type: "confirm",
      }),
      output: getHealthCheckResult,
    },
  },
);

const statusSteps = [
  getRecommendation,
  getRecommendationResult,
  getHealthCheck,
  getHealthCheckResult,
];

/**
 * @type {import('@aws-doc-sdk-examples/lib/scenario.js').Step[]}
 */
export const demoSteps = [
  new ScenarioOutput("header", MESSAGES.demoHeader, { header: true }),
  new ScenarioOutput("sanityCheck", MESSAGES.demoSanityCheck),
  ...statusSteps,
  new ScenarioInput(
    "brokenDependencyConfirmation",
    MESSAGES.demoBrokenDependencyConfirmation,
    { type: "confirm" },
```

```
  ),
  new ScenarioAction("brokenDependency", async (state) => {
    if (!state.brokenDependencyConfirmation) {
      process.exit();
    } else {
      const client = new SSMClient({});
      state.badTableName = `fake-table-${Date.now()}`;
      await client.send(
        new PutParameterCommand({
          Name: NAMES.ssmTableNameKey,
          Value: state.badTableName,
          Overwrite: true,
          Type: "String",
        }),
      );
    }
  }),
  new ScenarioOutput("testBrokenDependency", (state) =>
    MESSAGES.demoTestBrokenDependency.replace(
      "${TABLE_NAME}",
      state.badTableName,
    ),
  ),
  ...statusSteps,
  new ScenarioInput(
    "staticResponseConfirmation",
    MESSAGES.demoStaticResponseConfirmation,
    { type: "confirm" },
  ),
  new ScenarioAction("staticResponse", async (state) => {
    if (!state.staticResponseConfirmation) {
      process.exit();
    } else {
      const client = new SSMClient({});
      await client.send(
        new PutParameterCommand({
          Name: NAMES.ssmFailureResponseKey,
          Value: "static",
          Overwrite: true,
          Type: "String",
        }),
      );
    }
  }),
```

```
  new ScenarioOutput("testStaticResponse", MESSAGES.demoTestStaticResponse),
  ...statusSteps,
  new ScenarioInput(
    "badCredentialsConfirmation",
    MESSAGES.demoBadCredentialsConfirmation,
    { type: "confirm" },
  ),
  new ScenarioAction("badCredentialsExit", (state) => {
    if (!state.badCredentialsConfirmation) {
      process.exit();
    }
  }),
  new ScenarioAction("fixDynamoDBName", async () => {
    const client = new SSMClient({});
    await client.send(
      new PutParameterCommand({
        Name: NAMES.ssmTableNameKey,
        Value: NAMES.tableName,
        Overwrite: true,
        Type: "String",
      }),
    );
  }),
  new ScenarioAction(
    "badCredentials",
    /**
     * @param {{ targetInstance: import('@aws-sdk/client-auto-scaling').Instance }}
state
     */
    async (state) => {
      await createSsmOnlyInstanceProfile();
      const autoScalingClient = new AutoScalingClient({});
      const { AutoScalingGroups } = await autoScalingClient.send(
        new DescribeAutoScalingGroupsCommand({
          AutoScalingGroupNames: [NAMES.autoScalingGroupName],
        }),
      );
      state.targetInstance = AutoScalingGroups[0].Instances[0];
      const ec2Client = new EC2Client({});
      const { IamInstanceProfileAssociations } = await ec2Client.send(
        new DescribeIamInstanceProfileAssociationsCommand({
          Filters: [
            { Name: "instance-id", Values: [state.targetInstance.InstanceId] },
          ],
```

```
        }),
      );
      state.instanceProfileAssociationId =
        IamInstanceProfileAssociations[0].AssociationId;
      await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
        ec2Client.send(
          new ReplaceIamInstanceProfileAssociationCommand({
            AssociationId: state.instanceProfileAssociationId,
            IamInstanceProfile: { Name: NAMES.ssmOnlyInstanceProfileName },
          }),
        ),
      );

      await ec2Client.send(
        new RebootInstancesCommand({
          InstanceIds: [state.targetInstance.InstanceId],
        }),
      );

      const ssmClient = new SSMClient({});
      await retry({ intervalInMs: 20000, maxRetries: 15 }, async () => {
        const { InstanceInformationList } = await ssmClient.send(
          new DescribeInstanceInformationCommand({}),
        );

        const instance = InstanceInformationList.find(
          (info) => info.InstanceId === state.targetInstance.InstanceId,
        );

        if (!instance) {
          throw new Error("Instance not found.");
        }
      });

      await ssmClient.send(
        new SendCommandCommand({
          InstanceIds: [state.targetInstance.InstanceId],
          DocumentName: "AWS-RunShellScript",
          Parameters: { commands: ["cd / && sudo python3 server.py 80"] },
        }),
      );
    },
  ),
  new ScenarioOutput(
```

```
    "testBadCredentials",
    /**
     * @param {{ targetInstance: import('@aws-sdk/client-ssm').InstanceInformation}}
 state
     */
    (state) =>
      MESSAGES.demoTestBadCredentials.replace(
        "${INSTANCE_ID}",
        state.targetInstance.InstanceId,
      ),
  ),
  loadBalancerLoop,
  new ScenarioInput(
    "deepHealthCheckConfirmation",
    MESSAGES.demoDeepHealthCheckConfirmation,
    { type: "confirm" },
  ),
  new ScenarioAction("deepHealthCheckExit", (state) => {
    if (!state.deepHealthCheckConfirmation) {
      process.exit();
    }
  }),
  new ScenarioAction("deepHealthCheck", async () => {
    const client = new SSMClient({});
    await client.send(
      new PutParameterCommand({
        Name: NAMES.ssmHealthCheckKey,
        Value: "deep",
        Overwrite: true,
        Type: "String",
      }),
    );
  }),
  new ScenarioOutput("testDeepHealthCheck", MESSAGES.demoTestDeepHealthCheck),
  healthCheckLoop,
  loadBalancerLoop,
  new ScenarioInput(
    "killInstanceConfirmation",
    /**
     * @param {{ targetInstance: import('@aws-sdk/client-
ssm').InstanceInformation }} state
     */
    (state) =>
      MESSAGES.demoKillInstanceConfirmation.replace(
```

```
        "${INSTANCE_ID}",
        state.targetInstance.InstanceId,
      ),
    { type: "confirm" },
  ),
  new ScenarioAction("killInstanceExit", (state) => {
    if (!state.killInstanceConfirmation) {
      process.exit();
    }
  }),
  new ScenarioAction(
    "killInstance",
    /**
     * @param {{ targetInstance: import('@aws-sdk/client-
ssm').InstanceInformation }} state
     */
    async (state) => {
      const client = new AutoScalingClient({});
      await client.send(
        new TerminateInstanceInAutoScalingGroupCommand({
          InstanceId: state.targetInstance.InstanceId,
          ShouldDecrementDesiredCapacity: false,
        }),
      );
    },
  ),
  new ScenarioOutput("testKillInstance", MESSAGES.demoTestKillInstance),
  healthCheckLoop,
  loadBalancerLoop,
  new ScenarioInput("failOpenConfirmation", MESSAGES.demoFailOpenConfirmation, {
    type: "confirm",
  }),
  new ScenarioAction("failOpenExit", (state) => {
    if (!state.failOpenConfirmation) {
      process.exit();
    }
  }),
  new ScenarioAction("failOpen", () => {
    const client = new SSMClient({});
    return client.send(
      new PutParameterCommand({
        Name: NAMES.ssmTableNameKey,
        Value: `fake-table-${Date.now()}`,
        Overwrite: true,
```

```
        Type: "String",
      }),
    );
  }),
  new ScenarioOutput("testFailOpen", MESSAGES.demoFailOpenTest),
  healthCheckLoop,
  loadBalancerLoop,
  new ScenarioInput(
    "resetTableConfirmation",
    MESSAGES.demoResetTableConfirmation,
    { type: "confirm" },
  ),
  new ScenarioAction("resetTableExit", (state) => {
    if (!state.resetTableConfirmation) {
      process.exit();
    }
  }),
  new ScenarioAction("resetTable", async () => {
    const client = new SSMClient({});
    await client.send(
      new PutParameterCommand({
        Name: NAMES.ssmTableNameKey,
        Value: NAMES.tableName,
        Overwrite: true,
        Type: "String",
      }),
    );
  }),
  new ScenarioOutput("testResetTable", MESSAGES.demoTestResetTable),
  healthCheckLoop,
  loadBalancerLoop,
];

async function createSsmOnlyInstanceProfile() {
  const iamClient = new IAMClient({});
  const { Policy } = await iamClient.send(
    new CreatePolicyCommand({
      PolicyName: NAMES.ssmOnlyPolicyName,
      PolicyDocument: readFileSync(
        join(RESOURCES_PATH, "ssm_only_policy.json"),
      ),
    }),
  );
  await iamClient.send(
```

```
    new CreateRoleCommand({
      RoleName: NAMES.ssmOnlyRoleName,
      AssumeRolePolicyDocument: JSON.stringify({
        Version: "2012-10-17",
        Statement: [
          {
            Effect: "Allow",
            Principal: { Service: "ec2.amazonaws.com" },
            Action: "sts:AssumeRole",
          },
        ],
      }),
    }),
  );
  await iamClient.send(
    new AttachRolePolicyCommand({
      RoleName: NAMES.ssmOnlyRoleName,
      PolicyArn: Policy.Arn,
    }),
  );
  await iamClient.send(
    new AttachRolePolicyCommand({
      RoleName: NAMES.ssmOnlyRoleName,
      PolicyArn: "arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore",
    }),
  );
  const { InstanceProfile } = await iamClient.send(
    new CreateInstanceProfileCommand({
      InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
    }),
  );
  await waitUntilInstanceProfileExists(
    { client: iamClient },
    { InstanceProfileName: NAMES.ssmOnlyInstanceProfileName },
  );
  await iamClient.send(
    new AddRoleToInstanceProfileCommand({
      InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
      RoleName: NAMES.ssmOnlyRoleName,
    }),
  );

  return InstanceProfile;
}
```

建立步驟以銷毀所有資源。

```
import { unlinkSync } from "node:fs";

import { DynamoDBClient, DeleteTableCommand } from "@aws-sdk/client-dynamodb";
import {
  EC2Client,
  DeleteKeyPairCommand,
  DeleteLaunchTemplateCommand,
  RevokeSecurityGroupIngressCommand,
} from "@aws-sdk/client-ec2";
import {
  IAMClient,
  DeleteInstanceProfileCommand,
  RemoveRoleFromInstanceProfileCommand,
  DeletePolicyCommand,
  DeleteRoleCommand,
  DetachRolePolicyCommand,
  paginateListPolicies,
} from "@aws-sdk/client-iam";
import {
  AutoScalingClient,
  DeleteAutoScalingGroupCommand,
  TerminateInstanceInAutoScalingGroupCommand,
  UpdateAutoScalingGroupCommand,
  paginateDescribeAutoScalingGroups,
} from "@aws-sdk/client-auto-scaling";
import {
  DeleteLoadBalancerCommand,
  DeleteTargetGroupCommand,
  DescribeTargetGroupsCommand,
  ElasticLoadBalancingV2Client,
} from "@aws-sdk/client-elastic-load-balancing-v2";

import {
  ScenarioOutput,
  ScenarioInput,
  ScenarioAction,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import { loadState } from "@aws-doc-sdk-examples/lib/scenario/steps-common.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";
```

```
import { MESSAGES, NAMES } from "./constants.js";
import { findLoadBalancer } from "./shared.js";

/**
 * @type {import('@aws-doc-sdk-examples/lib/scenario.js').Step[]}
 */
export const destroySteps = [
  loadState,
  new ScenarioInput("destroy", MESSAGES.destroy, { type: "confirm" }),
  new ScenarioAction(
    "abort",
    (state) => state.destroy === false && process.exit(),
  ),
  new ScenarioAction("deleteTable", async (c) => {
    try {
      const client = new DynamoDBClient({});
      await client.send(new DeleteTableCommand({ TableName: NAMES.tableName }));
    } catch (e) {
      c.deleteTableError = e;
    }
  }),
  new ScenarioOutput("deleteTableResult", (state) => {
    if (state.deleteTableError) {
      console.error(state.deleteTableError);
      return MESSAGES.deleteTableError.replace(
        "${TABLE_NAME}",
        NAMES.tableName,
      );
    } else {
      return MESSAGES.deletedTable.replace("${TABLE_NAME}", NAMES.tableName);
    }
  }),
  new ScenarioAction("deleteKeyPair", async (state) => {
    try {
      const client = new EC2Client({});
      await client.send(
        new DeleteKeyPairCommand({ KeyName: NAMES.keyPairName }),
      );
      unlinkSync(`${NAMES.keyPairName}.pem`);
    } catch (e) {
      state.deleteKeyPairError = e;
    }
  }),
```

```
  new ScenarioOutput("deleteKeyPairResult", (state) => {
    if (state.deleteKeyPairError) {
      console.error(state.deleteKeyPairError);
      return MESSAGES.deleteKeyPairError.replace(
        "${KEY_PAIR_NAME}",
        NAMES.keyPairName,
      );
    } else {
      return MESSAGES.deletedKeyPair.replace(
        "${KEY_PAIR_NAME}",
        NAMES.keyPairName,
      );
    }
  }),
  new ScenarioAction("detachPolicyFromRole", async (state) => {
    try {
      const client = new IAMClient({});
      const policy = await findPolicy(NAMES.instancePolicyName);

      if (!policy) {
        state.detachPolicyFromRoleError = new Error(
          `Policy ${NAMES.instancePolicyName} not found.`,
        );
      } else {
        await client.send(
          new DetachRolePolicyCommand({
            RoleName: NAMES.instanceRoleName,
            PolicyArn: policy.Arn,
          }),
        );
      }
    } catch (e) {
      state.detachPolicyFromRoleError = e;
    }
  }),
  new ScenarioOutput("detachedPolicyFromRole", (state) => {
    if (state.detachPolicyFromRoleError) {
      console.error(state.detachPolicyFromRoleError);
      return MESSAGES.detachPolicyFromRoleError
        .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
        .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
    } else {
      return MESSAGES.detachedPolicyFromRole
        .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
```

```
        .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
    }
  }),
  new ScenarioAction("deleteInstancePolicy", async (state) => {
    const client = new IAMClient({});
    const policy = await findPolicy(NAMES.instancePolicyName);

    if (!policy) {
      state.deletePolicyError = new Error(
        `Policy ${NAMES.instancePolicyName} not found.`,
      );
    } else {
      return client.send(
        new DeletePolicyCommand({
          PolicyArn: policy.Arn,
        }),
      );
    }
  }),
  new ScenarioOutput("deletePolicyResult", (state) => {
    if (state.deletePolicyError) {
      console.error(state.deletePolicyError);
      return MESSAGES.deletePolicyError.replace(
        "${INSTANCE_POLICY_NAME}",
        NAMES.instancePolicyName,
      );
    } else {
      return MESSAGES.deletedPolicy.replace(
        "${INSTANCE_POLICY_NAME}",
        NAMES.instancePolicyName,
      );
    }
  }),
  new ScenarioAction("removeRoleFromInstanceProfile", async (state) => {
    try {
      const client = new IAMClient({});
      await client.send(
        new RemoveRoleFromInstanceProfileCommand({
          RoleName: NAMES.instanceRoleName,
          InstanceProfileName: NAMES.instanceProfileName,
        }),
      );
    } catch (e) {
      state.removeRoleFromInstanceProfileError = e;
```

```
    }
  }),
  new ScenarioOutput("removeRoleFromInstanceProfileResult", (state) => {
    if (state.removeRoleFromInstanceProfile) {
      console.error(state.removeRoleFromInstanceProfileError);
      return MESSAGES.removeRoleFromInstanceProfileError
        .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
        .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
    } else {
      return MESSAGES.removedRoleFromInstanceProfile
        .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
        .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
    }
  }),
  new ScenarioAction("deleteInstanceRole", async (state) => {
    try {
      const client = new IAMClient({});
      await client.send(
        new DeleteRoleCommand({
          RoleName: NAMES.instanceRoleName,
        }),
      );
    } catch (e) {
      state.deleteInstanceRoleError = e;
    }
  }),
  new ScenarioOutput("deleteInstanceRoleResult", (state) => {
    if (state.deleteInstanceRoleError) {
      console.error(state.deleteInstanceRoleError);
      return MESSAGES.deleteInstanceRoleError.replace(
        "${INSTANCE_ROLE_NAME}",
        NAMES.instanceRoleName,
      );
    } else {
      return MESSAGES.deletedInstanceRole.replace(
        "${INSTANCE_ROLE_NAME}",
        NAMES.instanceRoleName,
      );
    }
  }),
  new ScenarioAction("deleteInstanceProfile", async (state) => {
    try {
      const client = new IAMClient({});
      await client.send(
```

```
        new DeleteInstanceProfileCommand({
          InstanceProfileName: NAMES.instanceProfileName,
        }),
      );
    } catch (e) {
      state.deleteInstanceProfileError = e;
    }
  }),
  new ScenarioOutput("deleteInstanceProfileResult", (state) => {
    if (state.deleteInstanceProfileError) {
      console.error(state.deleteInstanceProfileError);
      return MESSAGES.deleteInstanceProfileError.replace(
        "${INSTANCE_PROFILE_NAME}",
        NAMES.instanceProfileName,
      );
    } else {
      return MESSAGES.deletedInstanceProfile.replace(
        "${INSTANCE_PROFILE_NAME}",
        NAMES.instanceProfileName,
      );
    }
  }),
  new ScenarioAction("deleteAutoScalingGroup", async (state) => {
    try {
      await terminateGroupInstances(NAMES.autoScalingGroupName);
      await retry({ intervalInMs: 60000, maxRetries: 60 }, async () => {
        await deleteAutoScalingGroup(NAMES.autoScalingGroupName);
      });
    } catch (e) {
      state.deleteAutoScalingGroupError = e;
    }
  }),
  new ScenarioOutput("deleteAutoScalingGroupResult", (state) => {
    if (state.deleteAutoScalingGroupError) {
      console.error(state.deleteAutoScalingGroupError);
      return MESSAGES.deleteAutoScalingGroupError.replace(
        "${AUTO_SCALING_GROUP_NAME}",
        NAMES.autoScalingGroupName,
      );
    } else {
      return MESSAGES.deletedAutoScalingGroup.replace(
        "${AUTO_SCALING_GROUP_NAME}",
        NAMES.autoScalingGroupName,
      );
```

```
    }
  }),
  new ScenarioAction("deleteLaunchTemplate", async (state) => {
    const client = new EC2Client({});
    try {
      await client.send(
        new DeleteLaunchTemplateCommand({
          LaunchTemplateName: NAMES.launchTemplateName,
        }),
      );
    } catch (e) {
      state.deleteLaunchTemplateError = e;
    }
  }),
  new ScenarioOutput("deleteLaunchTemplateResult", (state) => {
    if (state.deleteLaunchTemplateError) {
      console.error(state.deleteLaunchTemplateError);
      return MESSAGES.deleteLaunchTemplateError.replace(
        "${LAUNCH_TEMPLATE_NAME}",
        NAMES.launchTemplateName,
      );
    } else {
      return MESSAGES.deletedLaunchTemplate.replace(
        "${LAUNCH_TEMPLATE_NAME}",
        NAMES.launchTemplateName,
      );
    }
  }),
  new ScenarioAction("deleteLoadBalancer", async (state) => {
    try {
      const client = new ElasticLoadBalancingV2Client({});
      const loadBalancer = await findLoadBalancer(NAMES.loadBalancerName);
      await client.send(
        new DeleteLoadBalancerCommand({
          LoadBalancerArn: loadBalancer.LoadBalancerArn,
        }),
      );
      await retry({ intervalInMs: 1000, maxRetries: 60 }, async () => {
        const lb = await findLoadBalancer(NAMES.loadBalancerName);
        if (lb) {
          throw new Error("Load balancer still exists.");
        }
      });
    } catch (e) {
```

```
      state.deleteLoadBalancerError = e;
    }
  }),
  new ScenarioOutput("deleteLoadBalancerResult", (state) => {
    if (state.deleteLoadBalancerError) {
      console.error(state.deleteLoadBalancerError);
      return MESSAGES.deleteLoadBalancerError.replace(
        "${LB_NAME}",
        NAMES.loadBalancerName,
      );
    } else {
      return MESSAGES.deletedLoadBalancer.replace(
        "${LB_NAME}",
        NAMES.loadBalancerName,
      );
    }
  }),
  new ScenarioAction("deleteLoadBalancerTargetGroup", async (state) => {
    const client = new ElasticLoadBalancingV2Client({});
    try {
      const { TargetGroups } = await client.send(
        new DescribeTargetGroupsCommand({
          Names: [NAMES.loadBalancerTargetGroupName],
        }),
      );

      await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
        client.send(
          new DeleteTargetGroupCommand({
            TargetGroupArn: TargetGroups[0].TargetGroupArn,
          }),
        ),
      );
    } catch (e) {
      state.deleteLoadBalancerTargetGroupError = e;
    }
  }),
  new ScenarioOutput("deleteLoadBalancerTargetGroupResult", (state) => {
    if (state.deleteLoadBalancerTargetGroupError) {
      console.error(state.deleteLoadBalancerTargetGroupError);
      return MESSAGES.deleteLoadBalancerTargetGroupError.replace(
        "${TARGET_GROUP_NAME}",
        NAMES.loadBalancerTargetGroupName,
      );
```

```
    } else {
      return MESSAGES.deletedLoadBalancerTargetGroup.replace(
        "${TARGET_GROUP_NAME}",
        NAMES.loadBalancerTargetGroupName,
      );
    }
  }),
  new ScenarioAction("detachSsmOnlyRoleFromProfile", async (state) => {
    try {
      const client = new IAMClient({});
      await client.send(
        new RemoveRoleFromInstanceProfileCommand({
          InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
          RoleName: NAMES.ssmOnlyRoleName,
        }),
      );
    } catch (e) {
      state.detachSsmOnlyRoleFromProfileError = e;
    }
  }),
  new ScenarioOutput("detachSsmOnlyRoleFromProfileResult", (state) => {
    if (state.detachSsmOnlyRoleFromProfileError) {
      console.error(state.detachSsmOnlyRoleFromProfileError);
      return MESSAGES.detachSsmOnlyRoleFromProfileError
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
        .replace("${PROFILE_NAME}", NAMES.ssmOnlyInstanceProfileName);
    } else {
      return MESSAGES.detachedSsmOnlyRoleFromProfile
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
        .replace("${PROFILE_NAME}", NAMES.ssmOnlyInstanceProfileName);
    }
  }),
  new ScenarioAction("detachSsmOnlyCustomRolePolicy", async (state) => {
    try {
      const iamClient = new IAMClient({});
      const ssmOnlyPolicy = await findPolicy(NAMES.ssmOnlyPolicyName);
      await iamClient.send(
        new DetachRolePolicyCommand({
          RoleName: NAMES.ssmOnlyRoleName,
          PolicyArn: ssmOnlyPolicy.Arn,
        }),
      );
    } catch (e) {
      state.detachSsmOnlyCustomRolePolicyError = e;
```

```
    }
  }),
  new ScenarioOutput("detachSsmOnlyCustomRolePolicyResult", (state) => {
    if (state.detachSsmOnlyCustomRolePolicyError) {
      console.error(state.detachSsmOnlyCustomRolePolicyError);
      return MESSAGES.detachSsmOnlyCustomRolePolicyError
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
        .replace("${POLICY_NAME}", NAMES.ssmOnlyPolicyName);
    } else {
      return MESSAGES.detachedSsmOnlyCustomRolePolicy
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
        .replace("${POLICY_NAME}", NAMES.ssmOnlyPolicyName);
    }
  }),
  new ScenarioAction("detachSsmOnlyAWSRolePolicy", async (state) => {
    try {
      const iamClient = new IAMClient({});
      await iamClient.send(
        new DetachRolePolicyCommand({
          RoleName: NAMES.ssmOnlyRoleName,
          PolicyArn: "arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore",
        }),
      );
    } catch (e) {
      state.detachSsmOnlyAWSRolePolicyError = e;
    }
  }),
  new ScenarioOutput("detachSsmOnlyAWSRolePolicyResult", (state) => {
    if (state.detachSsmOnlyAWSRolePolicyError) {
      console.error(state.detachSsmOnlyAWSRolePolicyError);
      return MESSAGES.detachSsmOnlyAWSRolePolicyError
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
        .replace("${POLICY_NAME}", "AmazonSSMManagedInstanceCore");
    } else {
      return MESSAGES.detachedSsmOnlyAWSRolePolicy
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
        .replace("${POLICY_NAME}", "AmazonSSMManagedInstanceCore");
    }
  }),
  new ScenarioAction("deleteSsmOnlyInstanceProfile", async (state) => {
    try {
      const iamClient = new IAMClient({});
      await iamClient.send(
        new DeleteInstanceProfileCommand({
```

```
          InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
        }),
      );
    } catch (e) {
      state.deleteSsmOnlyInstanceProfileError = e;
    }
  }),
  new ScenarioOutput("deleteSsmOnlyInstanceProfileResult", (state) => {
    if (state.deleteSsmOnlyInstanceProfileError) {
      console.error(state.deleteSsmOnlyInstanceProfileError);
      return MESSAGES.deleteSsmOnlyInstanceProfileError.replace(
        "${INSTANCE_PROFILE_NAME}",
        NAMES.ssmOnlyInstanceProfileName,
      );
    } else {
      return MESSAGES.deletedSsmOnlyInstanceProfile.replace(
        "${INSTANCE_PROFILE_NAME}",
        NAMES.ssmOnlyInstanceProfileName,
      );
    }
  }),
  new ScenarioAction("deleteSsmOnlyPolicy", async (state) => {
    try {
      const iamClient = new IAMClient({});
      const ssmOnlyPolicy = await findPolicy(NAMES.ssmOnlyPolicyName);
      await iamClient.send(
        new DeletePolicyCommand({
          PolicyArn: ssmOnlyPolicy.Arn,
        }),
      );
    } catch (e) {
      state.deleteSsmOnlyPolicyError = e;
    }
  }),
  new ScenarioOutput("deleteSsmOnlyPolicyResult", (state) => {
    if (state.deleteSsmOnlyPolicyError) {
      console.error(state.deleteSsmOnlyPolicyError);
      return MESSAGES.deleteSsmOnlyPolicyError.replace(
        "${POLICY_NAME}",
        NAMES.ssmOnlyPolicyName,
      );
    } else {
      return MESSAGES.deletedSsmOnlyPolicy.replace(
        "${POLICY_NAME}",
```

```
        NAMES.ssmOnlyPolicyName,
      );
    }
  }),
  new ScenarioAction("deleteSsmOnlyRole", async (state) => {
    try {
      const iamClient = new IAMClient({});
      await iamClient.send(
        new DeleteRoleCommand({
          RoleName: NAMES.ssmOnlyRoleName,
        }),
      );
    } catch (e) {
      state.deleteSsmOnlyRoleError = e;
    }
  }),
  new ScenarioOutput("deleteSsmOnlyRoleResult", (state) => {
    if (state.deleteSsmOnlyRoleError) {
      console.error(state.deleteSsmOnlyRoleError);
      return MESSAGES.deleteSsmOnlyRoleError.replace(
        "${ROLE_NAME}",
        NAMES.ssmOnlyRoleName,
      );
    } else {
      return MESSAGES.deletedSsmOnlyRole.replace(
        "${ROLE_NAME}",
        NAMES.ssmOnlyRoleName,
      );
    }
  }),
  new ScenarioAction(
    "revokeSecurityGroupIngress",
    async (
      /** @type {{ myIp: string, defaultSecurityGroup: { GroupId: string } }} */
state,
    ) => {
      const ec2Client = new EC2Client({});

      try {
        await ec2Client.send(
          new RevokeSecurityGroupIngressCommand({
            GroupId: state.defaultSecurityGroup.GroupId,
            CidrIp: `${state.myIp}/32`,
            FromPort: 80,
```

```
            ToPort: 80,
            IpProtocol: "tcp",
          }),
        );
      } catch (e) {
        state.revokeSecurityGroupIngressError = e;
      }
    },
  ),
  new ScenarioOutput("revokeSecurityGroupIngressResult", (state) => {
    if (state.revokeSecurityGroupIngressError) {
      console.error(state.revokeSecurityGroupIngressError);
      return MESSAGES.revokeSecurityGroupIngressError.replace(
        "${IP}",
        state.myIp,
      );
    } else {
      return MESSAGES.revokedSecurityGroupIngress.replace("${IP}", state.myIp);
    }
  }),
];

/**
 * @param {string} policyName
 */
async function findPolicy(policyName) {
  const client = new IAMClient({});
  const paginatedPolicies = paginateListPolicies({ client }, {});
  for await (const page of paginatedPolicies) {
    const policy = page.Policies.find((p) => p.PolicyName === policyName);
    if (policy) {
      return policy;
    }
  }
}

/**
 * @param {string} groupName
 */
async function deleteAutoScalingGroup(groupName) {
  const client = new AutoScalingClient({});
  try {
    await client.send(
      new DeleteAutoScalingGroupCommand({
```

```
        AutoScalingGroupName: groupName,
      }),
    );
  } catch (err) {
    if (!(err instanceof Error)) {
      throw err;
    } else {
      console.log(err.name);
      throw err;
    }
  }
}

/**
 * @param {string} groupName
 */
async function terminateGroupInstances(groupName) {
  const autoScalingClient = new AutoScalingClient({});
  const group = await findAutoScalingGroup(groupName);
  await autoScalingClient.send(
    new UpdateAutoScalingGroupCommand({
      AutoScalingGroupName: group.AutoScalingGroupName,
      MinSize: 0,
    }),
  );
  for (const i of group.Instances) {
    await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
      autoScalingClient.send(
        new TerminateInstanceInAutoScalingGroupCommand({
          InstanceId: i.InstanceId,
          ShouldDecrementDesiredCapacity: true,
        }),
      ),
    );
  }
}

async function findAutoScalingGroup(groupName) {
  const client = new AutoScalingClient({});
  const paginatedGroups = paginateDescribeAutoScalingGroups({ client }, {});
  for await (const page of paginatedGroups) {
    const group = page.AutoScalingGroups.find(
      (g) => g.AutoScalingGroupName === groupName,
    );
```

```
    if (group) {
      return group;
    }
  }
  throw new Error(`Auto scaling group ${groupName} not found.`);
}
```

- 如需詳API細資訊，請參閱「AWS SDK for JavaScript API參考」中的下列主題。
  - AttachLoadBalancerTargetGroups
  - CreateAutoScalingGroup
  - CreateInstanceProfile
  - CreateLaunchTemplate
  - CreateListener
  - CreateLoadBalancer
  - CreateTargetGroup
  - DeleteAutoScalingGroup
  - DeleteInstanceProfile
  - DeleteLaunchTemplate
  - DeleteLoadBalancer
  - DeleteTargetGroup
  - DescribeAutoScalingGroups
  - DescribeAvailabilityZones
  - DescribeIamInstanceProfileAssociations
  - DescribeInstances
  - DescribeLoadBalancers
  - DescribeSubnets
  - DescribeTargetGroups
  - DescribeTargetHealth
  - DescribeVpcs
  - RebootInstances
  - ReplaceIamInstanceProfileAssociation
  - TerminateInstanceInAutoScalingGroup

- UpdateAutoScalingGroup

Elastic Load Balancing-第 2 版範SDK例使用 in JavaScript (v3)

下列程式碼範例說明如何使用 AWS SDK for JavaScript (v3) 搭配 Elastic Load Balancing-第 2 版，來執行動作及實作常見案例。

Actions 是大型程式的程式碼摘錄，必須在內容中執行。雖然動作會顯示如何呼叫個別服務函數，但您可以在其相關案例中查看內容中的動作。

案例是程式碼範例，向您展示如何透過呼叫服務中的多個函式或與其他函式結合來完成特定工作 AWS 服務。

每個範例都包含完整原始程式碼的連結，您可以在其中找到如何在內容中設定和執行程式碼的指示。

開始使用

你好 Elastic Load Balancing

下列程式碼範例會示範如何開始使用 Elastic Load Balancing。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import {
  ElasticLoadBalancingV2Client,
  DescribeLoadBalancersCommand,
} from "@aws-sdk/client-elastic-load-balancing-v2";

export async function main() {
  const client = new ElasticLoadBalancingV2Client({});
  const { LoadBalancers } = await client.send(
    new DescribeLoadBalancersCommand({}),
```

```
  );
  const loadBalancersList = LoadBalancers.map(
    (lb) => `• ${lb.LoadBalancerName}: ${lb.DNSName}`,
  ).join("\n");
  console.log(
    "Hello, Elastic Load Balancing! Let's list some of your load balancers:\n",
    loadBalancersList,
  );
}

// Call function if run directly
import { fileURLToPath } from "url";
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  main();
}
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考DescribeLoadBalancers中的。

主題

- 動作
- 案例

動作

`CreateListener`

下列程式碼範例會示範如何使用`CreateListener`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    const client = new ElasticLoadBalancingV2Client({});
    const { Listeners } = await client.send(
```

```
    new CreateListenerCommand({
      LoadBalancerArn: state.loadBalancerArn,
      Protocol: state.targetGroupProtocol,
      Port: state.targetGroupPort,
      DefaultActions: [
        { Type: "forward", TargetGroupArn: state.targetGroupArn },
      ],
    }),
  );
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考CreateListener中的。

`CreateLoadBalancer`

下列程式碼範例會示範如何使用`CreateLoadBalancer`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
const client = new ElasticLoadBalancingV2Client({});
const { LoadBalancers } = await client.send(
  new CreateLoadBalancerCommand({
    Name: NAMES.loadBalancerName,
    Subnets: state.subnets,
  }),
);
state.loadBalancerDns = LoadBalancers[0].DNSName;
state.loadBalancerArn = LoadBalancers[0].LoadBalancerArn;
await waitUntilLoadBalancerAvailable(
  { client },
  { Names: [NAMES.loadBalancerName] },
);
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考CreateLoadBalancer中的。

`CreateTargetGroup`

下列程式碼範例會示範如何使用`CreateTargetGroup`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
const client = new ElasticLoadBalancingV2Client({});
const { TargetGroups } = await client.send(
  new CreateTargetGroupCommand({
    Name: NAMES.loadBalancerTargetGroupName,
    Protocol: "HTTP",
    Port: 80,
    HealthCheckPath: "/healthcheck",
    HealthCheckIntervalSeconds: 10,
    HealthCheckTimeoutSeconds: 5,
    HealthyThresholdCount: 2,
    UnhealthyThresholdCount: 2,
    VpcId: state.defaultVpc,
  }),
);
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考CreateTargetGroup中的。

`DeleteLoadBalancer`

下列程式碼範例會示範如何使用`DeleteLoadBalancer`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
  const client = new ElasticLoadBalancingV2Client({});
  const loadBalancer = await findLoadBalancer(NAMES.loadBalancerName);
  await client.send(
    new DeleteLoadBalancerCommand({
      LoadBalancerArn: loadBalancer.LoadBalancerArn,
    }),
  );
  await retry({ intervalInMs: 1000, maxRetries: 60 }, async () => {
    const lb = await findLoadBalancer(NAMES.loadBalancerName);
    if (lb) {
      throw new Error("Load balancer still exists.");
    }
  });
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考DeleteLoadBalancer中的。

`DeleteTargetGroup`

下列程式碼範例會示範如何使用`DeleteTargetGroup`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
const client = new ElasticLoadBalancingV2Client({});
try {
  const { TargetGroups } = await client.send(
    new DescribeTargetGroupsCommand({
      Names: [NAMES.loadBalancerTargetGroupName],
    }),
  );

  await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
    client.send(
      new DeleteTargetGroupCommand({
        TargetGroupArn: TargetGroups[0].TargetGroupArn,
```

```
        }),
      ),
    );
  } catch (e) {
    state.deleteLoadBalancerTargetGroupError = e;
  }
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考DeleteTargetGroup中的。

`DescribeLoadBalancers`

下列程式碼範例會示範如何使用`DescribeLoadBalancers`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import {
  ElasticLoadBalancingV2Client,
  DescribeLoadBalancersCommand,
} from "@aws-sdk/client-elastic-load-balancing-v2";

export async function main() {
  const client = new ElasticLoadBalancingV2Client({});
  const { LoadBalancers } = await client.send(
    new DescribeLoadBalancersCommand({}),
  );
  const loadBalancersList = LoadBalancers.map(
    (lb) => `• ${lb.LoadBalancerName}: ${lb.DNSName}`,
  ).join("\n");
  console.log(
    "Hello, Elastic Load Balancing! Let's list some of your load balancers:\n",
    loadBalancersList,
  );
}
```

```
// Call function if run directly
import { fileURLToPath } from "url";
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  main();
}
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考DescribeLoadBalancers中的。

`DescribeTargetGroups`

下列程式碼範例會示範如何使用`DescribeTargetGroups`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
const client = new ElasticLoadBalancingV2Client({});
const { TargetGroups } = await client.send(
  new DescribeTargetGroupsCommand({
    Names: [NAMES.loadBalancerTargetGroupName],
  }),
);
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考DescribeTargetGroups中的。

`DescribeTargetHealth`

下列程式碼範例會示範如何使用`DescribeTargetHealth`。

SDK對於 JavaScript （ 3 ）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
const { TargetHealthDescriptions } = await client.send(
  new DescribeTargetHealthCommand({
    TargetGroupArn: TargetGroups[0].TargetGroupArn,
  }),
);
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考DescribeTargetHealth中的。

案例

建置及管理彈性服務

下列程式碼範例會示範如何建立負載平衡的 Web 服務，以傳回書籍、影片和歌曲建議。此範例顯示服務如何回應失故障，以及如何在發生故障時重組服務以提高復原能力。

- 使用 Amazon EC2 Auto Scaling 群組根據啟動範本建立 Amazon 彈性運算雲端 (AmazonEC2) 執行個體，並將執行個體數量保持在指定範圍內。
- 使用 Elastic Load Balancing 處理和分配HTTP請求。
- 監控 Auto Scaling 群組中執行個體的運作狀態，並且只將請求轉送給運作良好的執行個體。
- 在每個執行個EC2體上執行 Python 網頁伺服器來處理HTTP要求。Web 伺服器會回應建議和運作狀態檢查。
- 使用 Amazon DynamoDB 資料表模擬建議服務。
- 透過更新 AWS Systems Manager 參數來控制 Web 伺服器對要求和健康狀態檢查的回應。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

在命令提示中執行互動式案例。

```
#!/usr/bin/env node
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0


import {
  Scenario,
  parseScenarioArgs,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

/**
 * The workflow steps are split into three stages:
 *   - deploy
 *   - demo
 *   - destroy
 *
 * Each of these stages has a corresponding file prefixed with steps-*.
 */
import { deploySteps } from "./steps-deploy.js";
import { demoSteps } from "./steps-demo.js";
import { destroySteps } from "./steps-destroy.js";

/**
 * The context is passed to every scenario. Scenario steps
 * will modify the context.
 */
const context = {};

/**
 * Three Scenarios are created for the workflow. A Scenario is an orchestration
 class
 * that simplifies running a series of steps.
 */
```

```
export const scenarios = {
  // Deploys all resources necessary for the workflow.
  deploy: new Scenario("Resilient Workflow - Deploy", deploySteps, context),
  // Demonstrates how a fragile web service can be made more resilient.
  demo: new Scenario("Resilient Workflow - Demo", demoSteps, context),
  // Destroys the resources created for the workflow.
  destroy: new Scenario("Resilient Workflow - Destroy", destroySteps, context),
};

// Call function if run directly
import { fileURLToPath } from "url";

if (process.argv[1] === fileURLToPath(import.meta.url)) {
  parseScenarioArgs(scenarios);
}
```

建立步驟以部署所有資源。

```
import { join } from "node:path";
import { readFileSync, writeFileSync } from "node:fs";
import axios from "axios";

import {
  BatchWriteItemCommand,
  CreateTableCommand,
  DynamoDBClient,
  waitUntilTableExists,
} from "@aws-sdk/client-dynamodb";
import {
  EC2Client,
  CreateKeyPairCommand,
  CreateLaunchTemplateCommand,
  DescribeAvailabilityZonesCommand,
  DescribeVpcsCommand,
  DescribeSubnetsCommand,
  DescribeSecurityGroupsCommand,
  AuthorizeSecurityGroupIngressCommand,
} from "@aws-sdk/client-ec2";
import {
  IAMClient,
  CreatePolicyCommand,
  CreateRoleCommand,
```

```
  CreateInstanceProfileCommand,
  AddRoleToInstanceProfileCommand,
  AttachRolePolicyCommand,
  waitUntilInstanceProfileExists,
} from "@aws-sdk/client-iam";
import { SSMClient, GetParameterCommand } from "@aws-sdk/client-ssm";
import {
  CreateAutoScalingGroupCommand,
  AutoScalingClient,
  AttachLoadBalancerTargetGroupsCommand,
} from "@aws-sdk/client-auto-scaling";
import {
  CreateListenerCommand,
  CreateLoadBalancerCommand,
  CreateTargetGroupCommand,
  ElasticLoadBalancingV2Client,
  waitUntilLoadBalancerAvailable,
} from "@aws-sdk/client-elastic-load-balancing-v2";

import {
  ScenarioOutput,
  ScenarioInput,
  ScenarioAction,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import { saveState } from "@aws-doc-sdk-examples/lib/scenario/steps-common.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

import { MESSAGES, NAMES, RESOURCES_PATH, ROOT } from "./constants.js";
import { initParamsSteps } from "./steps-reset-params.js";

/**
 * @type {import('@aws-doc-sdk-examples/lib/scenario.js').Step[]}
 */
export const deploySteps = [
  new ScenarioOutput("introduction", MESSAGES.introduction, { header: true }),
  new ScenarioInput("confirmDeployment", MESSAGES.confirmDeployment, {
    type: "confirm",
  }),
  new ScenarioAction(
    "handleConfirmDeployment",
    (c) => c.confirmDeployment === false && process.exit(),
  ),
  new ScenarioOutput(
    "creatingTable",
```

```
    MESSAGES.creatingTable.replace("${TABLE_NAME}", NAMES.tableName),
  ),
  new ScenarioAction("createTable", async () => {
    const client = new DynamoDBClient({});
    await client.send(
      new CreateTableCommand({
        TableName: NAMES.tableName,
        ProvisionedThroughput: {
          ReadCapacityUnits: 5,
          WriteCapacityUnits: 5,
        },
        AttributeDefinitions: [
          {
            AttributeName: "MediaType",
            AttributeType: "S",
          },
          {
            AttributeName: "ItemId",
            AttributeType: "N",
          },
        ],
        KeySchema: [
          {
            AttributeName: "MediaType",
            KeyType: "HASH",
          },
          {
            AttributeName: "ItemId",
            KeyType: "RANGE",
          },
        ],
      }),
    );
    await waitUntilTableExists({ client }, { TableName: NAMES.tableName });
  }),
  new ScenarioOutput(
    "createdTable",
    MESSAGES.createdTable.replace("${TABLE_NAME}", NAMES.tableName),
  ),
  new ScenarioOutput(
    "populatingTable",
    MESSAGES.populatingTable.replace("${TABLE_NAME}", NAMES.tableName),
  ),
  new ScenarioAction("populateTable", () => {
```

```
    const client = new DynamoDBClient({});
    /**
     * @type {{ default: import("@aws-sdk/client-dynamodb").PutRequest['Item'][] }}
     */
    const recommendations = JSON.parse(
      readFileSync(join(RESOURCES_PATH, "recommendations.json")),
    );

    return client.send(
      new BatchWriteItemCommand({
        RequestItems: {
          [NAMES.tableName]: recommendations.map((item) => ({
            PutRequest: { Item: item },
          })),
        },
      }),
    );
  }),
  new ScenarioOutput(
    "populatedTable",
    MESSAGES.populatedTable.replace("${TABLE_NAME}", NAMES.tableName),
  ),
  new ScenarioOutput(
    "creatingKeyPair",
    MESSAGES.creatingKeyPair.replace("${KEY_PAIR_NAME}", NAMES.keyPairName),
  ),
  new ScenarioAction("createKeyPair", async () => {
    const client = new EC2Client({});
    const { KeyMaterial } = await client.send(
      new CreateKeyPairCommand({
        KeyName: NAMES.keyPairName,
      }),
    );

    writeFileSync(`${NAMES.keyPairName}.pem`, KeyMaterial, { mode: 0o600 });
  }),
  new ScenarioOutput(
    "createdKeyPair",
    MESSAGES.createdKeyPair.replace("${KEY_PAIR_NAME}", NAMES.keyPairName),
  ),
  new ScenarioOutput(
    "creatingInstancePolicy",
    MESSAGES.creatingInstancePolicy.replace(
      "${INSTANCE_POLICY_NAME}",
```

```
    NAMES.instancePolicyName,
  ),
),
new ScenarioAction("createInstancePolicy", async (state) => {
  const client = new IAMClient({});
  const {
    Policy: { Arn },
  } = await client.send(
    new CreatePolicyCommand({
      PolicyName: NAMES.instancePolicyName,
      PolicyDocument: readFileSync(
        join(RESOURCES_PATH, "instance_policy.json"),
      ),
    }),
  );
  state.instancePolicyArn = Arn;
}),
new ScenarioOutput("createdInstancePolicy", (state) =>
  MESSAGES.createdInstancePolicy
    .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
    .replace("${INSTANCE_POLICY_ARN}", state.instancePolicyArn),
),
new ScenarioOutput(
  "creatingInstanceRole",
  MESSAGES.creatingInstanceRole.replace(
    "${INSTANCE_ROLE_NAME}",
    NAMES.instanceRoleName,
  ),
),
new ScenarioAction("createInstanceRole", () => {
  const client = new IAMClient({});
  return client.send(
    new CreateRoleCommand({
      RoleName: NAMES.instanceRoleName,
      AssumeRolePolicyDocument: readFileSync(
        join(ROOT, "assume-role-policy.json"),
      ),
    }),
  );
}),
new ScenarioOutput(
  "createdInstanceRole",
  MESSAGES.createdInstanceRole.replace(
    "${INSTANCE_ROLE_NAME}",
```

```
      NAMES.instanceRoleName,
    ),
  ),
  new ScenarioOutput(
    "attachingPolicyToRole",
    MESSAGES.attachingPolicyToRole
      .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName)
      .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName),
  ),
  new ScenarioAction("attachPolicyToRole", async (state) => {
    const client = new IAMClient({});
    await client.send(
      new AttachRolePolicyCommand({
        RoleName: NAMES.instanceRoleName,
        PolicyArn: state.instancePolicyArn,
      }),
    );
  }),
  new ScenarioOutput(
    "attachedPolicyToRole",
    MESSAGES.attachedPolicyToRole
      .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
      .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName),
  ),
  new ScenarioOutput(
    "creatingInstanceProfile",
    MESSAGES.creatingInstanceProfile.replace(
      "${INSTANCE_PROFILE_NAME}",
      NAMES.instanceProfileName,
    ),
  ),
  new ScenarioAction("createInstanceProfile", async (state) => {
    const client = new IAMClient({});
    const {
      InstanceProfile: { Arn },
    } = await client.send(
      new CreateInstanceProfileCommand({
        InstanceProfileName: NAMES.instanceProfileName,
      }),
    );
    state.instanceProfileArn = Arn;

    await waitUntilInstanceProfileExists(
      { client },
```

```
      { InstanceProfileName: NAMES.instanceProfileName },
    );
  }),
  new ScenarioOutput("createdInstanceProfile", (state) =>
    MESSAGES.createdInstanceProfile
      .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
      .replace("${INSTANCE_PROFILE_ARN}", state.instanceProfileArn),
  ),
  new ScenarioOutput(
    "addingRoleToInstanceProfile",
    MESSAGES.addingRoleToInstanceProfile
      .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
      .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName),
  ),
  new ScenarioAction("addRoleToInstanceProfile", () => {
    const client = new IAMClient({});
    return client.send(
      new AddRoleToInstanceProfileCommand({
        RoleName: NAMES.instanceRoleName,
        InstanceProfileName: NAMES.instanceProfileName,
      }),
    );
  }),
  new ScenarioOutput(
    "addedRoleToInstanceProfile",
    MESSAGES.addedRoleToInstanceProfile
      .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
      .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName),
  ),
  ...initParamsSteps,
  new ScenarioOutput("creatingLaunchTemplate", MESSAGES.creatingLaunchTemplate),
  new ScenarioAction("createLaunchTemplate", async () => {
    const ssmClient = new SSMClient({});
    const { Parameter } = await ssmClient.send(
      new GetParameterCommand({
        Name: "/aws/service/ami-amazon-linux-latest/amzn2-ami-hvm-x86_64-gp2",
      }),
    );
    const ec2Client = new EC2Client({});
    await ec2Client.send(
      new CreateLaunchTemplateCommand({
        LaunchTemplateName: NAMES.launchTemplateName,
        LaunchTemplateData: {
          InstanceType: "t3.micro",
```

```
          ImageId: Parameter.Value,
          IamInstanceProfile: { Name: NAMES.instanceProfileName },
          UserData: readFileSync(
            join(RESOURCES_PATH, "server_startup_script.sh"),
          ).toString("base64"),
          KeyName: NAMES.keyPairName,
        },
      }),
    );
  }),
  new ScenarioOutput(
    "createdLaunchTemplate",
    MESSAGES.createdLaunchTemplate.replace(
      "${LAUNCH_TEMPLATE_NAME}",
      NAMES.launchTemplateName,
    ),
  ),
  new ScenarioOutput(
    "creatingAutoScalingGroup",
    MESSAGES.creatingAutoScalingGroup.replace(
      "${AUTO_SCALING_GROUP_NAME}",
      NAMES.autoScalingGroupName,
    ),
  ),
  new ScenarioAction("createAutoScalingGroup", async (state) => {
    const ec2Client = new EC2Client({});
    const { AvailabilityZones } = await ec2Client.send(
      new DescribeAvailabilityZonesCommand({}),
    );
    state.availabilityZoneNames = AvailabilityZones.map((az) => az.ZoneName);
    const autoScalingClient = new AutoScalingClient({});
    await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
      autoScalingClient.send(
        new CreateAutoScalingGroupCommand({
          AvailabilityZones: state.availabilityZoneNames,
          AutoScalingGroupName: NAMES.autoScalingGroupName,
          LaunchTemplate: {
            LaunchTemplateName: NAMES.launchTemplateName,
            Version: "$Default",
          },
          MinSize: 3,
          MaxSize: 3,
        }),
      ),
```

```
    );
  }),
  new ScenarioOutput(
    "createdAutoScalingGroup",
    /**
     * @param {{ availabilityZoneNames: string[] }} state
     */
    (state) =>
      MESSAGES.createdAutoScalingGroup
        .replace("${AUTO_SCALING_GROUP_NAME}", NAMES.autoScalingGroupName)
        .replace(
          "${AVAILABILITY_ZONE_NAMES}",
          state.availabilityZoneNames.join(", "),
        ),
  ),
  new ScenarioInput("confirmContinue", MESSAGES.confirmContinue, {
    type: "confirm",
  }),
  new ScenarioOutput("loadBalancer", MESSAGES.loadBalancer),
  new ScenarioOutput("gettingVpc", MESSAGES.gettingVpc),
  new ScenarioAction("getVpc", async (state) => {
    const client = new EC2Client({});
    const { Vpcs } = await client.send(
      new DescribeVpcsCommand({
        Filters: [{ Name: "is-default", Values: ["true"] }],
      }),
    );
    state.defaultVpc = Vpcs[0].VpcId;
  }),
  new ScenarioOutput("gotVpc", (state) =>
    MESSAGES.gotVpc.replace("${VPC_ID}", state.defaultVpc),
  ),
  new ScenarioOutput("gettingSubnets", MESSAGES.gettingSubnets),
  new ScenarioAction("getSubnets", async (state) => {
    const client = new EC2Client({});
    const { Subnets } = await client.send(
      new DescribeSubnetsCommand({
        Filters: [
          { Name: "vpc-id", Values: [state.defaultVpc] },
          { Name: "availability-zone", Values: state.availabilityZoneNames },
          { Name: "default-for-az", Values: ["true"] },
        ],
      }),
    );
```

```
    state.subnets = Subnets.map((subnet) => subnet.SubnetId);
  }),
  new ScenarioOutput(
    "gotSubnets",
    /**
     * @param {{ subnets: string[] }} state
     */
    (state) =>
      MESSAGES.gotSubnets.replace("${SUBNETS}", state.subnets.join(", ")),
  ),
  new ScenarioOutput(
    "creatingLoadBalancerTargetGroup",
    MESSAGES.creatingLoadBalancerTargetGroup.replace(
      "${TARGET_GROUP_NAME}",
      NAMES.loadBalancerTargetGroupName,
    ),
  ),
  new ScenarioAction("createLoadBalancerTargetGroup", async (state) => {
    const client = new ElasticLoadBalancingV2Client({});
    const { TargetGroups } = await client.send(
      new CreateTargetGroupCommand({
        Name: NAMES.loadBalancerTargetGroupName,
        Protocol: "HTTP",
        Port: 80,
        HealthCheckPath: "/healthcheck",
        HealthCheckIntervalSeconds: 10,
        HealthCheckTimeoutSeconds: 5,
        HealthyThresholdCount: 2,
        UnhealthyThresholdCount: 2,
        VpcId: state.defaultVpc,
      }),
    );
    const targetGroup = TargetGroups[0];
    state.targetGroupArn = targetGroup.TargetGroupArn;
    state.targetGroupProtocol = targetGroup.Protocol;
    state.targetGroupPort = targetGroup.Port;
  }),
  new ScenarioOutput(
    "createdLoadBalancerTargetGroup",
    MESSAGES.createdLoadBalancerTargetGroup.replace(
      "${TARGET_GROUP_NAME}",
      NAMES.loadBalancerTargetGroupName,
    ),
  ),
```

```
  new ScenarioOutput(
    "creatingLoadBalancer",
    MESSAGES.creatingLoadBalancer.replace("${LB_NAME}", NAMES.loadBalancerName),
  ),
  new ScenarioAction("createLoadBalancer", async (state) => {
    const client = new ElasticLoadBalancingV2Client({});
    const { LoadBalancers } = await client.send(
      new CreateLoadBalancerCommand({
        Name: NAMES.loadBalancerName,
        Subnets: state.subnets,
      }),
    );
    state.loadBalancerDns = LoadBalancers[0].DNSName;
    state.loadBalancerArn = LoadBalancers[0].LoadBalancerArn;
    await waitUntilLoadBalancerAvailable(
      { client },
      { Names: [NAMES.loadBalancerName] },
    );
  }),
  new ScenarioOutput("createdLoadBalancer", (state) =>
    MESSAGES.createdLoadBalancer
      .replace("${LB_NAME}", NAMES.loadBalancerName)
      .replace("${DNS_NAME}", state.loadBalancerDns),
  ),
  new ScenarioOutput(
    "creatingListener",
    MESSAGES.creatingLoadBalancerListener
      .replace("${LB_NAME}", NAMES.loadBalancerName)
      .replace("${TARGET_GROUP_NAME}", NAMES.loadBalancerTargetGroupName),
  ),
  new ScenarioAction("createListener", async (state) => {
    const client = new ElasticLoadBalancingV2Client({});
    const { Listeners } = await client.send(
      new CreateListenerCommand({
        LoadBalancerArn: state.loadBalancerArn,
        Protocol: state.targetGroupProtocol,
        Port: state.targetGroupPort,
        DefaultActions: [
          { Type: "forward", TargetGroupArn: state.targetGroupArn },
        ],
      }),
    );
    const listener = Listeners[0];
    state.loadBalancerListenerArn = listener.ListenerArn;
```

```
  }),
  new ScenarioOutput("createdListener", (state) =>
    MESSAGES.createdLoadBalancerListener.replace(
      "${LB_LISTENER_ARN}",
      state.loadBalancerListenerArn,
    ),
  ),
  new ScenarioOutput(
    "attachingLoadBalancerTargetGroup",
    MESSAGES.attachingLoadBalancerTargetGroup
      .replace("${TARGET_GROUP_NAME}", NAMES.loadBalancerTargetGroupName)
      .replace("${AUTO_SCALING_GROUP_NAME}", NAMES.autoScalingGroupName),
  ),
  new ScenarioAction("attachLoadBalancerTargetGroup", async (state) => {
    const client = new AutoScalingClient({});
    await client.send(
      new AttachLoadBalancerTargetGroupsCommand({
        AutoScalingGroupName: NAMES.autoScalingGroupName,
        TargetGroupARNs: [state.targetGroupArn],
      }),
    );
  }),
  new ScenarioOutput(
    "attachedLoadBalancerTargetGroup",
    MESSAGES.attachedLoadBalancerTargetGroup,
  ),
  new ScenarioOutput("verifyingInboundPort", MESSAGES.verifyingInboundPort),
  new ScenarioAction(
    "verifyInboundPort",
    /**
     *
     * @param {{ defaultSecurityGroup: import('@aws-sdk/client-ec2').SecurityGroup}}
state
     */
    async (state) => {
      const client = new EC2Client({});
      const { SecurityGroups } = await client.send(
        new DescribeSecurityGroupsCommand({
          Filters: [{ Name: "group-name", Values: ["default"] }],
        }),
      );
      if (!SecurityGroups) {
        state.verifyInboundPortError = new Error(MESSAGES.noSecurityGroups);
      }
```

```
      state.defaultSecurityGroup = SecurityGroups[0];

      /**
       * @type {string}
       */
      const ipResponse = (await axios.get("http://checkip.amazonaws.com")).data;
      state.myIp = ipResponse.trim();
      const myIpRules = state.defaultSecurityGroup.IpPermissions.filter(
        ({ IpRanges }) =>
          IpRanges.some(
            ({ CidrIp }) =>
              CidrIp.startsWith(state.myIp) || CidrIp === "0.0.0.0/0",
          ),
      )
        .filter(({ IpProtocol }) => IpProtocol === "tcp")
        .filter(({ FromPort }) => FromPort === 80);

      state.myIpRules = myIpRules;
    },
  ),
  new ScenarioOutput(
    "verifiedInboundPort",
    /**
     * @param {{ myIpRules: any[] }} state
     */
    (state) => {
      if (state.myIpRules.length > 0) {
        return MESSAGES.foundIpRules.replace(
          "${IP_RULES}",
          JSON.stringify(state.myIpRules, null, 2),
        );
      } else {
        return MESSAGES.noIpRules;
      }
    },
  ),
  new ScenarioInput(
    "shouldAddInboundRule",
    /**
     * @param {{ myIpRules: any[] }} state
     */
    (state) => {
      if (state.myIpRules.length > 0) {
        return false;
```

```
      } else {
        return MESSAGES.noIpRules;
      }
    },
    { type: "confirm" },
  ),
  new ScenarioAction(
    "addInboundRule",
    /**
     * @param {{ defaultSecurityGroup: import('@aws-sdk/client-
ec2').SecurityGroup }} state
     */
    async (state) => {
      if (!state.shouldAddInboundRule) {
        return;
      }

      const client = new EC2Client({});
      await client.send(
        new AuthorizeSecurityGroupIngressCommand({
          GroupId: state.defaultSecurityGroup.GroupId,
          CidrIp: `${state.myIp}/32`,
          FromPort: 80,
          ToPort: 80,
          IpProtocol: "tcp",
        }),
      );
    },
  ),
  new ScenarioOutput("addedInboundRule", (state) => {
    if (state.shouldAddInboundRule) {
      return MESSAGES.addedInboundRule.replace("${IP_ADDRESS}", state.myIp);
    } else {
      return false;
    }
  }),
  new ScenarioOutput("verifyingEndpoint", (state) =>
    MESSAGES.verifyingEndpoint.replace("${DNS_NAME}", state.loadBalancerDns),
  ),
  new ScenarioAction("verifyEndpoint", async (state) => {
    try {
      const response = await retry({ intervalInMs: 2000, maxRetries: 30 }, () =>
        axios.get(`http://${state.loadBalancerDns}`),
      );
```

```
      state.endpointResponse = JSON.stringify(response.data, null, 2);
    } catch (e) {
      state.verifyEndpointError = e;
    }
  }),
  new ScenarioOutput("verifiedEndpoint", (state) => {
    if (state.verifyEndpointError) {
      console.error(state.verifyEndpointError);
    } else {
      return MESSAGES.verifiedEndpoint.replace(
        "${ENDPOINT_RESPONSE}",
        state.endpointResponse,
      );
    }
  }),
  saveState,
];
```

建立步驟以執行示範。

```
import { readFileSync } from "node:fs";
import { join } from "node:path";

import axios from "axios";

import {
  DescribeTargetGroupsCommand,
  DescribeTargetHealthCommand,
  ElasticLoadBalancingV2Client,
} from "@aws-sdk/client-elastic-load-balancing-v2";
import {
  DescribeInstanceInformationCommand,
  PutParameterCommand,
  SSMClient,
  SendCommandCommand,
} from "@aws-sdk/client-ssm";
import {
  IAMClient,
  CreatePolicyCommand,
  CreateRoleCommand,
  AttachRolePolicyCommand,
  CreateInstanceProfileCommand,
```

```
  AddRoleToInstanceProfileCommand,
  waitUntilInstanceProfileExists,
} from "@aws-sdk/client-iam";
import {
  AutoScalingClient,
  DescribeAutoScalingGroupsCommand,
  TerminateInstanceInAutoScalingGroupCommand,
} from "@aws-sdk/client-auto-scaling";
import {
  DescribeIamInstanceProfileAssociationsCommand,
  EC2Client,
  RebootInstancesCommand,
  ReplaceIamInstanceProfileAssociationCommand,
} from "@aws-sdk/client-ec2";

import {
  ScenarioAction,
  ScenarioInput,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/scenario.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

import { MESSAGES, NAMES, RESOURCES_PATH } from "./constants.js";
import { findLoadBalancer } from "./shared.js";

const getRecommendation = new ScenarioAction(
  "getRecommendation",
  async (state) => {
    const loadBalancer = await findLoadBalancer(NAMES.loadBalancerName);
    if (loadBalancer) {
      state.loadBalancerDnsName = loadBalancer.DNSName;
      try {
        state.recommendation = (
          await axios.get(`http://${state.loadBalancerDnsName}`)
        ).data;
      } catch (e) {
        state.recommendation = e instanceof Error ? e.message : e;
      }
    } else {
      throw new Error(MESSAGES.demoFindLoadBalancerError);
    }
  },
);
```

```
const getRecommendationResult = new ScenarioOutput(
  "getRecommendationResult",
  (state) =>
    `Recommendation:\n${JSON.stringify(state.recommendation, null, 2)}`,
  { preformatted: true },
);

const getHealthCheck = new ScenarioAction("getHealthCheck", async (state) => {
  const client = new ElasticLoadBalancingV2Client({});
  const { TargetGroups } = await client.send(
    new DescribeTargetGroupsCommand({
      Names: [NAMES.loadBalancerTargetGroupName],
    }),
  );

  const { TargetHealthDescriptions } = await client.send(
    new DescribeTargetHealthCommand({
      TargetGroupArn: TargetGroups[0].TargetGroupArn,
    }),
  );
  state.targetHealthDescriptions = TargetHealthDescriptions;
});

const getHealthCheckResult = new ScenarioOutput(
  "getHealthCheckResult",
  /**
   * @param {{ targetHealthDescriptions: import('@aws-sdk/client-elastic-load-
balancing-v2').TargetHealthDescription[]}} state
   */
  (state) => {
    const status = state.targetHealthDescriptions
      .map((th) => `${th.Target.Id}: ${th.TargetHealth.State}`)
      .join("\n");
    return `Health check:\n${status}`;
  },
  { preformatted: true },
);

const loadBalancerLoop = new ScenarioAction(
  "loadBalancerLoop",
  getRecommendation.action,
  {
    whileConfig: {
      whileFn: ({ loadBalancerCheck }) => loadBalancerCheck,
```

```
      input: new ScenarioInput(
        "loadBalancerCheck",
        MESSAGES.demoLoadBalancerCheck,
        {
          type: "confirm",
        },
      ),
      output: getRecommendationResult,
    },
  },
);

const healthCheckLoop = new ScenarioAction(
  "healthCheckLoop",
  getHealthCheck.action,
  {
    whileConfig: {
      whileFn: ({ healthCheck }) => healthCheck,
      input: new ScenarioInput("healthCheck", MESSAGES.demoHealthCheck, {
        type: "confirm",
      }),
      output: getHealthCheckResult,
    },
  },
);

const statusSteps = [
  getRecommendation,
  getRecommendationResult,
  getHealthCheck,
  getHealthCheckResult,
];

/**
 * @type {import('@aws-doc-sdk-examples/lib/scenario.js').Step[]}
 */
export const demoSteps = [
  new ScenarioOutput("header", MESSAGES.demoHeader, { header: true }),
  new ScenarioOutput("sanityCheck", MESSAGES.demoSanityCheck),
  ...statusSteps,
  new ScenarioInput(
    "brokenDependencyConfirmation",
    MESSAGES.demoBrokenDependencyConfirmation,
    { type: "confirm" },
```

```
  ),
  new ScenarioAction("brokenDependency", async (state) => {
    if (!state.brokenDependencyConfirmation) {
      process.exit();
    } else {
      const client = new SSMClient({});
      state.badTableName = `fake-table-${Date.now()}`;
      await client.send(
        new PutParameterCommand({
          Name: NAMES.ssmTableNameKey,
          Value: state.badTableName,
          Overwrite: true,
          Type: "String",
        }),
      );
    }
  }),
  new ScenarioOutput("testBrokenDependency", (state) =>
    MESSAGES.demoTestBrokenDependency.replace(
      "${TABLE_NAME}",
      state.badTableName,
    ),
  ),
  ...statusSteps,
  new ScenarioInput(
    "staticResponseConfirmation",
    MESSAGES.demoStaticResponseConfirmation,
    { type: "confirm" },
  ),
  new ScenarioAction("staticResponse", async (state) => {
    if (!state.staticResponseConfirmation) {
      process.exit();
    } else {
      const client = new SSMClient({});
      await client.send(
        new PutParameterCommand({
          Name: NAMES.ssmFailureResponseKey,
          Value: "static",
          Overwrite: true,
          Type: "String",
        }),
      );
    }
  }),
```

```
  new ScenarioOutput("testStaticResponse", MESSAGES.demoTestStaticResponse),
  ...statusSteps,
  new ScenarioInput(
    "badCredentialsConfirmation",
    MESSAGES.demoBadCredentialsConfirmation,
    { type: "confirm" },
  ),
  new ScenarioAction("badCredentialsExit", (state) => {
    if (!state.badCredentialsConfirmation) {
      process.exit();
    }
  }),
  new ScenarioAction("fixDynamoDBName", async () => {
    const client = new SSMClient({});
    await client.send(
      new PutParameterCommand({
        Name: NAMES.ssmTableNameKey,
        Value: NAMES.tableName,
        Overwrite: true,
        Type: "String",
      }),
    );
  }),
  new ScenarioAction(
    "badCredentials",
    /**
     * @param {{ targetInstance: import('@aws-sdk/client-auto-scaling').Instance }}
state
     */
    async (state) => {
      await createSsmOnlyInstanceProfile();
      const autoScalingClient = new AutoScalingClient({});
      const { AutoScalingGroups } = await autoScalingClient.send(
        new DescribeAutoScalingGroupsCommand({
          AutoScalingGroupNames: [NAMES.autoScalingGroupName],
        }),
      );
      state.targetInstance = AutoScalingGroups[0].Instances[0];
      const ec2Client = new EC2Client({});
      const { IamInstanceProfileAssociations } = await ec2Client.send(
        new DescribeIamInstanceProfileAssociationsCommand({
          Filters: [
            { Name: "instance-id", Values: [state.targetInstance.InstanceId] },
          ],
```

```
        }),
      );
      state.instanceProfileAssociationId =
        IamInstanceProfileAssociations[0].AssociationId;
      await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
        ec2Client.send(
          new ReplaceIamInstanceProfileAssociationCommand({
            AssociationId: state.instanceProfileAssociationId,
            IamInstanceProfile: { Name: NAMES.ssmOnlyInstanceProfileName },
          }),
        ),
      );

      await ec2Client.send(
        new RebootInstancesCommand({
          InstanceIds: [state.targetInstance.InstanceId],
        }),
      );

      const ssmClient = new SSMClient({});
      await retry({ intervalInMs: 20000, maxRetries: 15 }, async () => {
        const { InstanceInformationList } = await ssmClient.send(
          new DescribeInstanceInformationCommand({}),
        );

        const instance = InstanceInformationList.find(
          (info) => info.InstanceId === state.targetInstance.InstanceId,
        );

        if (!instance) {
          throw new Error("Instance not found.");
        }
      });

      await ssmClient.send(
        new SendCommandCommand({
          InstanceIds: [state.targetInstance.InstanceId],
          DocumentName: "AWS-RunShellScript",
          Parameters: { commands: ["cd / && sudo python3 server.py 80"] },
        }),
      );
    },
  ),
  new ScenarioOutput(
```

```
    "testBadCredentials",
    /**
     * @param {{ targetInstance: import('@aws-sdk/client-ssm').InstanceInformation}}
 state
     */
    (state) =>
      MESSAGES.demoTestBadCredentials.replace(
        "${INSTANCE_ID}",
        state.targetInstance.InstanceId,
      ),
  ),
  loadBalancerLoop,
  new ScenarioInput(
    "deepHealthCheckConfirmation",
    MESSAGES.demoDeepHealthCheckConfirmation,
    { type: "confirm" },
  ),
  new ScenarioAction("deepHealthCheckExit", (state) => {
    if (!state.deepHealthCheckConfirmation) {
      process.exit();
    }
  }),
  new ScenarioAction("deepHealthCheck", async () => {
    const client = new SSMClient({});
    await client.send(
      new PutParameterCommand({
        Name: NAMES.ssmHealthCheckKey,
        Value: "deep",
        Overwrite: true,
        Type: "String",
      }),
    );
  }),
  new ScenarioOutput("testDeepHealthCheck", MESSAGES.demoTestDeepHealthCheck),
  healthCheckLoop,
  loadBalancerLoop,
  new ScenarioInput(
    "killInstanceConfirmation",
    /**
     * @param {{ targetInstance: import('@aws-sdk/client-
ssm').InstanceInformation }} state
     */
    (state) =>
      MESSAGES.demoKillInstanceConfirmation.replace(
```

```
        "${INSTANCE_ID}",
        state.targetInstance.InstanceId,
      ),
    { type: "confirm" },
  ),
  new ScenarioAction("killInstanceExit", (state) => {
    if (!state.killInstanceConfirmation) {
      process.exit();
    }
  }),
  new ScenarioAction(
    "killInstance",
    /**
     * @param {{ targetInstance: import('@aws-sdk/client-
ssm').InstanceInformation }} state
     */
    async (state) => {
      const client = new AutoScalingClient({});
      await client.send(
        new TerminateInstanceInAutoScalingGroupCommand({
          InstanceId: state.targetInstance.InstanceId,
          ShouldDecrementDesiredCapacity: false,
        }),
      );
    },
  ),
  new ScenarioOutput("testKillInstance", MESSAGES.demoTestKillInstance),
  healthCheckLoop,
  loadBalancerLoop,
  new ScenarioInput("failOpenConfirmation", MESSAGES.demoFailOpenConfirmation, {
    type: "confirm",
  }),
  new ScenarioAction("failOpenExit", (state) => {
    if (!state.failOpenConfirmation) {
      process.exit();
    }
  }),
  new ScenarioAction("failOpen", () => {
    const client = new SSMClient({});
    return client.send(
      new PutParameterCommand({
        Name: NAMES.ssmTableNameKey,
        Value: `fake-table-${Date.now()}`,
        Overwrite: true,
```

```
        Type: "String",
      }),
    );
  }),
  new ScenarioOutput("testFailOpen", MESSAGES.demoFailOpenTest),
  healthCheckLoop,
  loadBalancerLoop,
  new ScenarioInput(
    "resetTableConfirmation",
    MESSAGES.demoResetTableConfirmation,
    { type: "confirm" },
  ),
  new ScenarioAction("resetTableExit", (state) => {
    if (!state.resetTableConfirmation) {
      process.exit();
    }
  }),
  new ScenarioAction("resetTable", async () => {
    const client = new SSMClient({});
    await client.send(
      new PutParameterCommand({
        Name: NAMES.ssmTableNameKey,
        Value: NAMES.tableName,
        Overwrite: true,
        Type: "String",
      }),
    );
  }),
  new ScenarioOutput("testResetTable", MESSAGES.demoTestResetTable),
  healthCheckLoop,
  loadBalancerLoop,
];

async function createSsmOnlyInstanceProfile() {
  const iamClient = new IAMClient({});
  const { Policy } = await iamClient.send(
    new CreatePolicyCommand({
      PolicyName: NAMES.ssmOnlyPolicyName,
      PolicyDocument: readFileSync(
        join(RESOURCES_PATH, "ssm_only_policy.json"),
      ),
    }),
  );
  await iamClient.send(
```

```
    new CreateRoleCommand({
      RoleName: NAMES.ssmOnlyRoleName,
      AssumeRolePolicyDocument: JSON.stringify({
        Version: "2012-10-17",
        Statement: [
          {
            Effect: "Allow",
            Principal: { Service: "ec2.amazonaws.com" },
            Action: "sts:AssumeRole",
          },
        ],
      }),
    }),
  );
  await iamClient.send(
    new AttachRolePolicyCommand({
      RoleName: NAMES.ssmOnlyRoleName,
      PolicyArn: Policy.Arn,
    }),
  );
  await iamClient.send(
    new AttachRolePolicyCommand({
      RoleName: NAMES.ssmOnlyRoleName,
      PolicyArn: "arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore",
    }),
  );
  const { InstanceProfile } = await iamClient.send(
    new CreateInstanceProfileCommand({
      InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
    }),
  );
  await waitUntilInstanceProfileExists(
    { client: iamClient },
    { InstanceProfileName: NAMES.ssmOnlyInstanceProfileName },
  );
  await iamClient.send(
    new AddRoleToInstanceProfileCommand({
      InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
      RoleName: NAMES.ssmOnlyRoleName,
    }),
  );

  return InstanceProfile;
}
```

建立步驟以銷毀所有資源。

```
import { unlinkSync } from "node:fs";

import { DynamoDBClient, DeleteTableCommand } from "@aws-sdk/client-dynamodb";
import {
  EC2Client,
  DeleteKeyPairCommand,
  DeleteLaunchTemplateCommand,
  RevokeSecurityGroupIngressCommand,
} from "@aws-sdk/client-ec2";
import {
  IAMClient,
  DeleteInstanceProfileCommand,
  RemoveRoleFromInstanceProfileCommand,
  DeletePolicyCommand,
  DeleteRoleCommand,
  DetachRolePolicyCommand,
  paginateListPolicies,
} from "@aws-sdk/client-iam";
import {
  AutoScalingClient,
  DeleteAutoScalingGroupCommand,
  TerminateInstanceInAutoScalingGroupCommand,
  UpdateAutoScalingGroupCommand,
  paginateDescribeAutoScalingGroups,
} from "@aws-sdk/client-auto-scaling";
import {
  DeleteLoadBalancerCommand,
  DeleteTargetGroupCommand,
  DescribeTargetGroupsCommand,
  ElasticLoadBalancingV2Client,
} from "@aws-sdk/client-elastic-load-balancing-v2";

import {
  ScenarioOutput,
  ScenarioInput,
  ScenarioAction,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import { loadState } from "@aws-doc-sdk-examples/lib/scenario/steps-common.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";
```

```
import { MESSAGES, NAMES } from "./constants.js";
import { findLoadBalancer } from "./shared.js";

/**
 * @type {import('@aws-doc-sdk-examples/lib/scenario.js').Step[]}
 */
export const destroySteps = [
  loadState,
  new ScenarioInput("destroy", MESSAGES.destroy, { type: "confirm" }),
  new ScenarioAction(
    "abort",
    (state) => state.destroy === false && process.exit(),
  ),
  new ScenarioAction("deleteTable", async (c) => {
    try {
      const client = new DynamoDBClient({});
      await client.send(new DeleteTableCommand({ TableName: NAMES.tableName }));
    } catch (e) {
      c.deleteTableError = e;
    }
  }),
  new ScenarioOutput("deleteTableResult", (state) => {
    if (state.deleteTableError) {
      console.error(state.deleteTableError);
      return MESSAGES.deleteTableError.replace(
        "${TABLE_NAME}",
        NAMES.tableName,
      );
    } else {
      return MESSAGES.deletedTable.replace("${TABLE_NAME}", NAMES.tableName);
    }
  }),
  new ScenarioAction("deleteKeyPair", async (state) => {
    try {
      const client = new EC2Client({});
      await client.send(
        new DeleteKeyPairCommand({ KeyName: NAMES.keyPairName }),
      );
      unlinkSync(`${NAMES.keyPairName}.pem`);
    } catch (e) {
      state.deleteKeyPairError = e;
    }
  }),
```

```
  new ScenarioOutput("deleteKeyPairResult", (state) => {
    if (state.deleteKeyPairError) {
      console.error(state.deleteKeyPairError);
      return MESSAGES.deleteKeyPairError.replace(
        "${KEY_PAIR_NAME}",
        NAMES.keyPairName,
      );
    } else {
      return MESSAGES.deletedKeyPair.replace(
        "${KEY_PAIR_NAME}",
        NAMES.keyPairName,
      );
    }
  }),
  new ScenarioAction("detachPolicyFromRole", async (state) => {
    try {
      const client = new IAMClient({});
      const policy = await findPolicy(NAMES.instancePolicyName);

      if (!policy) {
        state.detachPolicyFromRoleError = new Error(
          `Policy ${NAMES.instancePolicyName} not found.`,
        );
      } else {
        await client.send(
          new DetachRolePolicyCommand({
            RoleName: NAMES.instanceRoleName,
            PolicyArn: policy.Arn,
          }),
        );
      }
    } catch (e) {
      state.detachPolicyFromRoleError = e;
    }
  }),
  new ScenarioOutput("detachedPolicyFromRole", (state) => {
    if (state.detachPolicyFromRoleError) {
      console.error(state.detachPolicyFromRoleError);
      return MESSAGES.detachPolicyFromRoleError
        .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
        .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
    } else {
      return MESSAGES.detachedPolicyFromRole
        .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
```

```
        .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
    }
  }),
  new ScenarioAction("deleteInstancePolicy", async (state) => {
    const client = new IAMClient({});
    const policy = await findPolicy(NAMES.instancePolicyName);

    if (!policy) {
      state.deletePolicyError = new Error(
        `Policy ${NAMES.instancePolicyName} not found.`,
      );
    } else {
      return client.send(
        new DeletePolicyCommand({
          PolicyArn: policy.Arn,
        }),
      );
    }
  }),
  new ScenarioOutput("deletePolicyResult", (state) => {
    if (state.deletePolicyError) {
      console.error(state.deletePolicyError);
      return MESSAGES.deletePolicyError.replace(
        "${INSTANCE_POLICY_NAME}",
        NAMES.instancePolicyName,
      );
    } else {
      return MESSAGES.deletedPolicy.replace(
        "${INSTANCE_POLICY_NAME}",
        NAMES.instancePolicyName,
      );
    }
  }),
  new ScenarioAction("removeRoleFromInstanceProfile", async (state) => {
    try {
      const client = new IAMClient({});
      await client.send(
        new RemoveRoleFromInstanceProfileCommand({
          RoleName: NAMES.instanceRoleName,
          InstanceProfileName: NAMES.instanceProfileName,
        }),
      );
    } catch (e) {
      state.removeRoleFromInstanceProfileError = e;
```

```
    }
  }),
  new ScenarioOutput("removeRoleFromInstanceProfileResult", (state) => {
    if (state.removeRoleFromInstanceProfile) {
      console.error(state.removeRoleFromInstanceProfileError);
      return MESSAGES.removeRoleFromInstanceProfileError
        .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
        .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
    } else {
      return MESSAGES.removedRoleFromInstanceProfile
        .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
        .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
    }
  }),
  new ScenarioAction("deleteInstanceRole", async (state) => {
    try {
      const client = new IAMClient({});
      await client.send(
        new DeleteRoleCommand({
          RoleName: NAMES.instanceRoleName,
        }),
      );
    } catch (e) {
      state.deleteInstanceRoleError = e;
    }
  }),
  new ScenarioOutput("deleteInstanceRoleResult", (state) => {
    if (state.deleteInstanceRoleError) {
      console.error(state.deleteInstanceRoleError);
      return MESSAGES.deleteInstanceRoleError.replace(
        "${INSTANCE_ROLE_NAME}",
        NAMES.instanceRoleName,
      );
    } else {
      return MESSAGES.deletedInstanceRole.replace(
        "${INSTANCE_ROLE_NAME}",
        NAMES.instanceRoleName,
      );
    }
  }),
  new ScenarioAction("deleteInstanceProfile", async (state) => {
    try {
      const client = new IAMClient({});
      await client.send(
```

```
        new DeleteInstanceProfileCommand({
          InstanceProfileName: NAMES.instanceProfileName,
        }),
      );
    } catch (e) {
      state.deleteInstanceProfileError = e;
    }
  }),
  new ScenarioOutput("deleteInstanceProfileResult", (state) => {
    if (state.deleteInstanceProfileError) {
      console.error(state.deleteInstanceProfileError);
      return MESSAGES.deleteInstanceProfileError.replace(
        "${INSTANCE_PROFILE_NAME}",
        NAMES.instanceProfileName,
      );
    } else {
      return MESSAGES.deletedInstanceProfile.replace(
        "${INSTANCE_PROFILE_NAME}",
        NAMES.instanceProfileName,
      );
    }
  }),
  new ScenarioAction("deleteAutoScalingGroup", async (state) => {
    try {
      await terminateGroupInstances(NAMES.autoScalingGroupName);
      await retry({ intervalInMs: 60000, maxRetries: 60 }, async () => {
        await deleteAutoScalingGroup(NAMES.autoScalingGroupName);
      });
    } catch (e) {
      state.deleteAutoScalingGroupError = e;
    }
  }),
  new ScenarioOutput("deleteAutoScalingGroupResult", (state) => {
    if (state.deleteAutoScalingGroupError) {
      console.error(state.deleteAutoScalingGroupError);
      return MESSAGES.deleteAutoScalingGroupError.replace(
        "${AUTO_SCALING_GROUP_NAME}",
        NAMES.autoScalingGroupName,
      );
    } else {
      return MESSAGES.deletedAutoScalingGroup.replace(
        "${AUTO_SCALING_GROUP_NAME}",
        NAMES.autoScalingGroupName,
      );
```

```
    }
  }),
  new ScenarioAction("deleteLaunchTemplate", async (state) => {
    const client = new EC2Client({});
    try {
      await client.send(
        new DeleteLaunchTemplateCommand({
          LaunchTemplateName: NAMES.launchTemplateName,
        }),
      );
    } catch (e) {
      state.deleteLaunchTemplateError = e;
    }
  }),
  new ScenarioOutput("deleteLaunchTemplateResult", (state) => {
    if (state.deleteLaunchTemplateError) {
      console.error(state.deleteLaunchTemplateError);
      return MESSAGES.deleteLaunchTemplateError.replace(
        "${LAUNCH_TEMPLATE_NAME}",
        NAMES.launchTemplateName,
      );
    } else {
      return MESSAGES.deletedLaunchTemplate.replace(
        "${LAUNCH_TEMPLATE_NAME}",
        NAMES.launchTemplateName,
      );
    }
  }),
  new ScenarioAction("deleteLoadBalancer", async (state) => {
    try {
      const client = new ElasticLoadBalancingV2Client({});
      const loadBalancer = await findLoadBalancer(NAMES.loadBalancerName);
      await client.send(
        new DeleteLoadBalancerCommand({
          LoadBalancerArn: loadBalancer.LoadBalancerArn,
        }),
      );
      await retry({ intervalInMs: 1000, maxRetries: 60 }, async () => {
        const lb = await findLoadBalancer(NAMES.loadBalancerName);
        if (lb) {
          throw new Error("Load balancer still exists.");
        }
      });
    } catch (e) {
```

```
      state.deleteLoadBalancerError = e;
    }
  }),
  new ScenarioOutput("deleteLoadBalancerResult", (state) => {
    if (state.deleteLoadBalancerError) {
      console.error(state.deleteLoadBalancerError);
      return MESSAGES.deleteLoadBalancerError.replace(
        "${LB_NAME}",
        NAMES.loadBalancerName,
      );
    } else {
      return MESSAGES.deletedLoadBalancer.replace(
        "${LB_NAME}",
        NAMES.loadBalancerName,
      );
    }
  }),
  new ScenarioAction("deleteLoadBalancerTargetGroup", async (state) => {
    const client = new ElasticLoadBalancingV2Client({});
    try {
      const { TargetGroups } = await client.send(
        new DescribeTargetGroupsCommand({
          Names: [NAMES.loadBalancerTargetGroupName],
        }),
      );

      await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
        client.send(
          new DeleteTargetGroupCommand({
            TargetGroupArn: TargetGroups[0].TargetGroupArn,
          }),
        ),
      );
    } catch (e) {
      state.deleteLoadBalancerTargetGroupError = e;
    }
  }),
  new ScenarioOutput("deleteLoadBalancerTargetGroupResult", (state) => {
    if (state.deleteLoadBalancerTargetGroupError) {
      console.error(state.deleteLoadBalancerTargetGroupError);
      return MESSAGES.deleteLoadBalancerTargetGroupError.replace(
        "${TARGET_GROUP_NAME}",
        NAMES.loadBalancerTargetGroupName,
      );
```

```
    } else {
      return MESSAGES.deletedLoadBalancerTargetGroup.replace(
        "${TARGET_GROUP_NAME}",
        NAMES.loadBalancerTargetGroupName,
      );
    }
  }),
  new ScenarioAction("detachSsmOnlyRoleFromProfile", async (state) => {
    try {
      const client = new IAMClient({});
      await client.send(
        new RemoveRoleFromInstanceProfileCommand({
          InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
          RoleName: NAMES.ssmOnlyRoleName,
        }),
      );
    } catch (e) {
      state.detachSsmOnlyRoleFromProfileError = e;
    }
  }),
  new ScenarioOutput("detachSsmOnlyRoleFromProfileResult", (state) => {
    if (state.detachSsmOnlyRoleFromProfileError) {
      console.error(state.detachSsmOnlyRoleFromProfileError);
      return MESSAGES.detachSsmOnlyRoleFromProfileError
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
        .replace("${PROFILE_NAME}", NAMES.ssmOnlyInstanceProfileName);
    } else {
      return MESSAGES.detachedSsmOnlyRoleFromProfile
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
        .replace("${PROFILE_NAME}", NAMES.ssmOnlyInstanceProfileName);
    }
  }),
  new ScenarioAction("detachSsmOnlyCustomRolePolicy", async (state) => {
    try {
      const iamClient = new IAMClient({});
      const ssmOnlyPolicy = await findPolicy(NAMES.ssmOnlyPolicyName);
      await iamClient.send(
        new DetachRolePolicyCommand({
          RoleName: NAMES.ssmOnlyRoleName,
          PolicyArn: ssmOnlyPolicy.Arn,
        }),
      );
    } catch (e) {
      state.detachSsmOnlyCustomRolePolicyError = e;
```

```
    }
  }),
  new ScenarioOutput("detachSsmOnlyCustomRolePolicyResult", (state) => {
    if (state.detachSsmOnlyCustomRolePolicyError) {
      console.error(state.detachSsmOnlyCustomRolePolicyError);
      return MESSAGES.detachSsmOnlyCustomRolePolicyError
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
        .replace("${POLICY_NAME}", NAMES.ssmOnlyPolicyName);
    } else {
      return MESSAGES.detachedSsmOnlyCustomRolePolicy
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
        .replace("${POLICY_NAME}", NAMES.ssmOnlyPolicyName);
    }
  }),
  new ScenarioAction("detachSsmOnlyAWSRolePolicy", async (state) => {
    try {
      const iamClient = new IAMClient({});
      await iamClient.send(
        new DetachRolePolicyCommand({
          RoleName: NAMES.ssmOnlyRoleName,
          PolicyArn: "arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore",
        }),
      );
    } catch (e) {
      state.detachSsmOnlyAWSRolePolicyError = e;
    }
  }),
  new ScenarioOutput("detachSsmOnlyAWSRolePolicyResult", (state) => {
    if (state.detachSsmOnlyAWSRolePolicyError) {
      console.error(state.detachSsmOnlyAWSRolePolicyError);
      return MESSAGES.detachSsmOnlyAWSRolePolicyError
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
        .replace("${POLICY_NAME}", "AmazonSSMManagedInstanceCore");
    } else {
      return MESSAGES.detachedSsmOnlyAWSRolePolicy
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
        .replace("${POLICY_NAME}", "AmazonSSMManagedInstanceCore");
    }
  }),
  new ScenarioAction("deleteSsmOnlyInstanceProfile", async (state) => {
    try {
      const iamClient = new IAMClient({});
      await iamClient.send(
        new DeleteInstanceProfileCommand({
```

```
          InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
        }),
      );
    } catch (e) {
      state.deleteSsmOnlyInstanceProfileError = e;
    }
  }),
  new ScenarioOutput("deleteSsmOnlyInstanceProfileResult", (state) => {
    if (state.deleteSsmOnlyInstanceProfileError) {
      console.error(state.deleteSsmOnlyInstanceProfileError);
      return MESSAGES.deleteSsmOnlyInstanceProfileError.replace(
        "${INSTANCE_PROFILE_NAME}",
        NAMES.ssmOnlyInstanceProfileName,
      );
    } else {
      return MESSAGES.deletedSsmOnlyInstanceProfile.replace(
        "${INSTANCE_PROFILE_NAME}",
        NAMES.ssmOnlyInstanceProfileName,
      );
    }
  }),
  new ScenarioAction("deleteSsmOnlyPolicy", async (state) => {
    try {
      const iamClient = new IAMClient({});
      const ssmOnlyPolicy = await findPolicy(NAMES.ssmOnlyPolicyName);
      await iamClient.send(
        new DeletePolicyCommand({
          PolicyArn: ssmOnlyPolicy.Arn,
        }),
      );
    } catch (e) {
      state.deleteSsmOnlyPolicyError = e;
    }
  }),
  new ScenarioOutput("deleteSsmOnlyPolicyResult", (state) => {
    if (state.deleteSsmOnlyPolicyError) {
      console.error(state.deleteSsmOnlyPolicyError);
      return MESSAGES.deleteSsmOnlyPolicyError.replace(
        "${POLICY_NAME}",
        NAMES.ssmOnlyPolicyName,
      );
    } else {
      return MESSAGES.deletedSsmOnlyPolicy.replace(
        "${POLICY_NAME}",
```

```
        NAMES.ssmOnlyPolicyName,
      );
    }
  }),
  new ScenarioAction("deleteSsmOnlyRole", async (state) => {
    try {
      const iamClient = new IAMClient({});
      await iamClient.send(
        new DeleteRoleCommand({
          RoleName: NAMES.ssmOnlyRoleName,
        }),
      );
    } catch (e) {
      state.deleteSsmOnlyRoleError = e;
    }
  }),
  new ScenarioOutput("deleteSsmOnlyRoleResult", (state) => {
    if (state.deleteSsmOnlyRoleError) {
      console.error(state.deleteSsmOnlyRoleError);
      return MESSAGES.deleteSsmOnlyRoleError.replace(
        "${ROLE_NAME}",
        NAMES.ssmOnlyRoleName,
      );
    } else {
      return MESSAGES.deletedSsmOnlyRole.replace(
        "${ROLE_NAME}",
        NAMES.ssmOnlyRoleName,
      );
    }
  }),
  new ScenarioAction(
    "revokeSecurityGroupIngress",
    async (
      /** @type {{ myIp: string, defaultSecurityGroup: { GroupId: string } }} */
state,
    ) => {
      const ec2Client = new EC2Client({});

      try {
        await ec2Client.send(
          new RevokeSecurityGroupIngressCommand({
            GroupId: state.defaultSecurityGroup.GroupId,
            CidrIp: `${state.myIp}/32`,
            FromPort: 80,
```

```
            ToPort: 80,
            IpProtocol: "tcp",
          }),
        );
      } catch (e) {
        state.revokeSecurityGroupIngressError = e;
      }
    },
  ),
  new ScenarioOutput("revokeSecurityGroupIngressResult", (state) => {
    if (state.revokeSecurityGroupIngressError) {
      console.error(state.revokeSecurityGroupIngressError);
      return MESSAGES.revokeSecurityGroupIngressError.replace(
        "${IP}",
        state.myIp,
      );
    } else {
      return MESSAGES.revokedSecurityGroupIngress.replace("${IP}", state.myIp);
    }
  }),
];

/**
 * @param {string} policyName
 */
async function findPolicy(policyName) {
  const client = new IAMClient({});
  const paginatedPolicies = paginateListPolicies({ client }, {});
  for await (const page of paginatedPolicies) {
    const policy = page.Policies.find((p) => p.PolicyName === policyName);
    if (policy) {
      return policy;
    }
  }
}

/**
 * @param {string} groupName
 */
async function deleteAutoScalingGroup(groupName) {
  const client = new AutoScalingClient({});
  try {
    await client.send(
      new DeleteAutoScalingGroupCommand({
```

```
        AutoScalingGroupName: groupName,
      }),
    );
  } catch (err) {
    if (!(err instanceof Error)) {
      throw err;
    } else {
      console.log(err.name);
      throw err;
    }
  }
}

/**
 * @param {string} groupName
 */
async function terminateGroupInstances(groupName) {
  const autoScalingClient = new AutoScalingClient({});
  const group = await findAutoScalingGroup(groupName);
  await autoScalingClient.send(
    new UpdateAutoScalingGroupCommand({
      AutoScalingGroupName: group.AutoScalingGroupName,
      MinSize: 0,
    }),
  );
  for (const i of group.Instances) {
    await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
      autoScalingClient.send(
        new TerminateInstanceInAutoScalingGroupCommand({
          InstanceId: i.InstanceId,
          ShouldDecrementDesiredCapacity: true,
        }),
      ),
    );
  }
}

async function findAutoScalingGroup(groupName) {
  const client = new AutoScalingClient({});
  const paginatedGroups = paginateDescribeAutoScalingGroups({ client }, {});
  for await (const page of paginatedGroups) {
    const group = page.AutoScalingGroups.find(
      (g) => g.AutoScalingGroupName === groupName,
    );
```

```
    if (group) {
      return group;
    }
  }
  throw new Error(`Auto scaling group ${groupName} not found.`);
}
```

- 如需詳API細資訊，請參閱「AWS SDK for JavaScript API參考」中的下列主題。
  - AttachLoadBalancerTargetGroups
  - CreateAutoScalingGroup
  - CreateInstanceProfile
  - CreateLaunchTemplate
  - CreateListener
  - CreateLoadBalancer
  - CreateTargetGroup
  - DeleteAutoScalingGroup
  - DeleteInstanceProfile
  - DeleteLaunchTemplate
  - DeleteLoadBalancer
  - DeleteTargetGroup
  - DescribeAutoScalingGroups
  - DescribeAvailabilityZones
  - DescribeIamInstanceProfileAssociations
  - DescribeInstances
  - DescribeLoadBalancers
  - DescribeSubnets
  - DescribeTargetGroups
  - DescribeTargetHealth
  - DescribeVpcs
  - RebootInstances
  - ReplaceIamInstanceProfileAssociation
  - TerminateInstanceInAutoScalingGroup

- UpdateAutoScalingGroup

EventBridge 使用 JavaScript （ v3 ）SDK的示例

下列程式碼範例會示範如何使用 AWS SDK for JavaScript (v3) 搭配使用來執行動作及實作常見案例 EventBridge。

Actions 是大型程式的程式碼摘錄，必須在內容中執行。雖然動作會顯示如何呼叫個別服務函數，但您可以在其相關案例中查看內容中的動作。

案例是程式碼範例，向您展示如何透過呼叫服務中的多個函式或與其他函式結合來完成特定工作 AWS 服務。

每個範例都包含完整原始程式碼的連結，您可以在其中找到如何在內容中設定和執行程式碼的指示。

主題

- 動作
- 案例

動作

`PutEvents`

下列程式碼範例會示範如何使用`PutEvents`。

SDK對於 JavaScript （ 3 ）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

匯入SDK和用戶端模組並呼叫API.

```
import {
  EventBridgeClient,
```

```
  PutEventsCommand,
} from "@aws-sdk/client-eventbridge";

export const putEvents = async (
  source = "eventbridge.integration.test",
  detailType = "greeting",
  resources = [],
) => {
  const client = new EventBridgeClient({});

  const response = await client.send(
    new PutEventsCommand({
      Entries: [
        {
          Detail: JSON.stringify({ greeting: "Hello there." }),
          DetailType: detailType,
          Resources: resources,
          Source: source,
        },
      ],
    }),
  );

  console.log("PutEvents response:");
  console.log(response);
  // PutEvents response:
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '3d0df73d-dcea-4a23-ae0d-f5556a3ac109',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   Entries: [ { EventId: '51620841-5af4-6402-d9bc-b77734991eb5' } ],
  //   FailedEntryCount: 0
  // }

  return response;
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考PutEvents中的。

SDK對於 JavaScript （ 第 2 個 ）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create CloudWatchEvents service object
var ebevents = new AWS.EventBridge({ apiVersion: "2015-10-07" });

var params = {
  Entries: [
    {
      Detail: '{ "key1": "value1", "key2": "value2" }',
      DetailType: "appRequestSubmitted",
      Resources: ["RESOURCE_ARN"],
      Source: "com.company.app",
    },
  ],
};

ebevents.putEvents(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.Entries);
  }
});
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考PutEvents中的。

`PutRule`

下列程式碼範例會示範如何使用`PutRule`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

匯入SDK和用戶端模組並呼叫API.

```
import { EventBridgeClient, PutRuleCommand } from "@aws-sdk/client-eventbridge";

export const putRule = async (
  ruleName = "some-rule",
  source = "some-source",
) => {
  const client = new EventBridgeClient({});

  const response = await client.send(
    new PutRuleCommand({
      Name: ruleName,
      EventPattern: JSON.stringify({ source: [source] }),
      State: "ENABLED",
      EventBusName: "default",
    }),
  );

  console.log("PutRule response:");
  console.log(response);
  // PutRule response:
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'd7292ced-1544-421b-842f-596326bc7072',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   RuleArn: 'arn:aws:events:us-east-1:xxxxxxxxxxxx:rule/
EventBridgeTestRule-1696280037720'
  // }
  return response;
```

```
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考PutRule中的。

SDK對於 JavaScript（第 2 個）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create CloudWatchEvents service object
var ebevents = new AWS.EventBridge({ apiVersion: "2015-10-07" });

var params = {
  Name: "DEMO_EVENT",
  RoleArn: "IAM_ROLE_ARN",
  ScheduleExpression: "rate(5 minutes)",
  State: "ENABLED",
};

ebevents.putRule(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.RuleArn);
  }
});
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考PutRule中的。

`PutTargets`

下列程式碼範例會示範如何使用`PutTargets`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

匯入SDK和用戶端模組並呼叫API.

```
import {
  EventBridgeClient,
  PutTargetsCommand,
} from "@aws-sdk/client-eventbridge";

export const putTarget = async (
  existingRuleName = "some-rule",
  targetArn = "arn:aws:lambda:us-east-1:000000000000:function:test-func",
  uniqueId = Date.now().toString(),
) => {
  const client = new EventBridgeClient({});
  const response = await client.send(
    new PutTargetsCommand({
      Rule: existingRuleName,
      Targets: [
        {
          Arn: targetArn,
          Id: uniqueId,
        },
      ],
    }),
  );

  console.log("PutTargets response:");
  console.log(response);
  // PutTargets response:
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'f5b23b9a-2c17-45c1-ad5c-f926c3692e3d',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
```

```
  //     totalRetryDelay: 0
  //   },
  //   FailedEntries: [],
  //   FailedEntryCount: 0
  // }

  return response;
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考PutTargets中的。

SDK對於 JavaScript（第 2 個）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create CloudWatchEvents service object
var ebevents = new AWS.EventBridge({ apiVersion: "2015-10-07" });

var params = {
  Rule: "DEMO_EVENT",
  Targets: [
    {
      Arn: "LAMBDA_FUNCTION_ARN",
      Id: "myEventBridgeTarget",
    },
  ],
};

ebevents.putTargets(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
```

```
  }
});
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考PutTargets中的。

案例

使用排程事件來調用 Lambda 函數

下列程式碼範例顯示如何建立 Amazon EventBridge 排程事件所叫用的 AWS Lambda 函數。

SDK對於 JavaScript（3）

說明如何建立叫用 AWS Lambda 函數的 Amazon EventBridge 排程事件。設定 EventBridge 為在叫用 Lambda 函數時使用 cron 運算式來排程。在此範例中，您可以使用 Lambda JavaScript 執行階段來建立 Lambda 函數API。此範例會呼叫不同的 AWS 服務來執行特定使用案例。此範例示範如何建立應用程式，將行動裝置文字訊息傳送給員工，在他們的週年紀念日向他們道賀。

有關如何設置和運行的完整源代碼和說明，請參閱中的完整示例GitHub。

此範例也可在 AWS SDK for JavaScript v3 開發人員指南中取得。

此範例中使用的服務

- DynamoDB
- EventBridge
- Lambda
- Amazon SNS

AWS Glue 使用 JavaScript（v3）SDK的示例

下列程式碼範例會示範如何使用 AWS SDK for JavaScript (v3) 搭配使用來執行動作及實作常見案例 AWS Glue。

基本概念是程式碼範例，會示範如何在服務中執行基本作業。

Actions 是大型程式的程式碼摘錄，必須在內容中執行。雖然動作會顯示如何呼叫個別服務函數，但您可以在其相關案例中查看內容中的動作。

每個範例都包含完整原始程式碼的連結，您可以在其中找到如何在內容中設定和執行程式碼的指示。

開始使用

你好 AWS Glue

下列程式碼範例示範如何開始使用 AWS Glue。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import { ListJobsCommand, GlueClient } from "@aws-sdk/client-glue";

const client = new GlueClient({});

export const main = async () => {
  const command = new ListJobsCommand({});

  const { JobNames } = await client.send(command);
  const formattedJobNames = JobNames.join("\n");
  console.log("Job names: ");
  console.log(formattedJobNames);
  return JobNames;
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考ListJobs中的。

主題

- 基本概念
- 動作

基本概念

學習基礎知識

以下程式碼範例顯示做法：

- 建立可編目公用 Amazon S3 儲存貯體的爬蟲程式，並產生格式化中繼資料的CSV資料庫。
- 列出有關 AWS Glue Data Catalog.
- 建立任務以從 S3 儲存貯體擷取CSV資料、轉換資料，並將JSON格式化的輸出載入另一個 S3 儲存貯體。
- 列出任務執行的相關資訊、檢視已轉換的資料以及清除資源。

如需詳細資訊，請參閱教學課程：開始使用 AWS Glue Studio。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

建立並執行編目公用 Amazon Simple Storage Service (Amazon S3) 貯體的爬網程式，並產生描述找到的CSV格式化資料的中繼資料庫。

```
const createCrawler = (name, role, dbName, tablePrefix, s3TargetPath) => {
  const client = new GlueClient({});

  const command = new CreateCrawlerCommand({
    Name: name,
    Role: role,
    DatabaseName: dbName,
    TablePrefix: tablePrefix,
    Targets: {
      S3Targets: [{ Path: s3TargetPath }],
    },
  });

  return client.send(command);
};

const getCrawler = (name) => {
  const client = new GlueClient({});

  const command = new GetCrawlerCommand({
    Name: name,
```

```
  });

  return client.send(command);
};

const startCrawler = (name) => {
  const client = new GlueClient({});

  const command = new StartCrawlerCommand({
    Name: name,
  });

  return client.send(command);
};

const crawlerExists = async ({ getCrawler }, crawlerName) => {
  try {
    await getCrawler(crawlerName);
    return true;
  } catch {
    return false;
  }
};

/**
 * @param {{ createCrawler: import('../../../actions/create-
crawler.js').createCrawler}} actions
 */
const makeCreateCrawlerStep = (actions) => async (context) => {
  if (await crawlerExists(actions, process.env.CRAWLER_NAME)) {
    log("Crawler already exists. Skipping creation.");
  } else {
    await actions.createCrawler(
      process.env.CRAWLER_NAME,
      process.env.ROLE_NAME,
      process.env.DATABASE_NAME,
      process.env.TABLE_PREFIX,
      process.env.S3_TARGET_PATH,
    );

    log("Crawler created successfully.", { type: "success" });
  }

  return { ...context };
```

```
};

/**
 * @param {(name: string) => Promise<import('@aws-sdk/client-
glue').GetCrawlerCommandOutput>} getCrawler
 * @param {string} crawlerName
 */
const waitForCrawler = async (getCrawler, crawlerName) => {
  const waitTimeInSeconds = 30;
  const { Crawler } = await getCrawler(crawlerName);

  if (!Crawler) {
    throw new Error(`Crawler with name ${crawlerName} not found.`);
  }

  if (Crawler.State === "READY") {
    return;
  }

  log(`Crawler is ${Crawler.State}. Waiting ${waitTimeInSeconds} seconds...`);
  await wait(waitTimeInSeconds);
  return waitForCrawler(getCrawler, crawlerName);
};

const makeStartCrawlerStep =
  ({ startCrawler, getCrawler }) =>
  async (context) => {
    log("Starting crawler.");
    await startCrawler(process.env.CRAWLER_NAME);
    log("Crawler started.", { type: "success" });

    log("Waiting for crawler to finish running. This can take a while.");
    await waitForCrawler(getCrawler, process.env.CRAWLER_NAME);
    log("Crawler ready.", { type: "success" });

    return { ...context };
  };
```

列出有關 AWS Glue Data Catalog.

```
const getDatabase = (name) => {
  const client = new GlueClient({});
```

```
  const command = new GetDatabaseCommand({
    Name: name,
  });

  return client.send(command);
};

const getTables = (databaseName) => {
  const client = new GlueClient({});

  const command = new GetTablesCommand({
    DatabaseName: databaseName,
  });

  return client.send(command);
};

const makeGetDatabaseStep =
  ({ getDatabase }) =>
  async (context) => {
    const {
      Database: { Name },
    } = await getDatabase(process.env.DATABASE_NAME);
    log(`Database: ${Name}`);
    return { ...context };
  };

/**
 * @param {{ getTables: () => Promise<import('@aws-sdk/client-
glue').GetTablesCommandOutput}} config
 */
const makeGetTablesStep =
  ({ getTables }) =>
  async (context) => {
    const { TableList } = await getTables(process.env.DATABASE_NAME);
    log("Tables:");
    log(TableList.map((table) => `  • ${table.Name}\n`));
    return { ...context };
  };
```

建立並執行從來源 Amazon S3 儲存貯體擷取CSV資料的任務、移除和重新命名欄位進行轉換，以及將JSON格式化的輸出載入另一個 Amazon S3 儲存貯體。

```
const createJob = (name, role, scriptBucketName, scriptKey) => {
  const client = new GlueClient({});

  const command = new CreateJobCommand({
    Name: name,
    Role: role,
    Command: {
      Name: "glueetl",
      PythonVersion: "3",
      ScriptLocation: `s3://${scriptBucketName}/${scriptKey}`,
    },
    GlueVersion: "3.0",
  });

  return client.send(command);
};

const startJobRun = (jobName, dbName, tableName, bucketName) => {
  const client = new GlueClient({});

  const command = new StartJobRunCommand({
    JobName: jobName,
    Arguments: {
      "--input_database": dbName,
      "--input_table": tableName,
      "--output_bucket_url": `s3://${bucketName}/`,
    },
  });

  return client.send(command);
};

const makeCreateJobStep =
  ({ createJob }) =>
  async (context) => {
    log("Creating Job.");
    await createJob(
      process.env.JOB_NAME,
      process.env.ROLE_NAME,
      process.env.BUCKET_NAME,
```

```
      process.env.PYTHON_SCRIPT_KEY,
    );
    log("Job created.", { type: "success" });

    return { ...context };
  };

/**
 * @param {(name: string, runId: string) => Promise<import('@aws-sdk/client-
glue').GetJobRunCommandOutput> }  getJobRun
 * @param {string} jobName
 * @param {string} jobRunId
 */
const waitForJobRun = async (getJobRun, jobName, jobRunId) => {
  const waitTimeInSeconds = 30;
  const { JobRun } = await getJobRun(jobName, jobRunId);

  if (!JobRun) {
    throw new Error(`Job run with id ${jobRunId} not found.`);
  }

  switch (JobRun.JobRunState) {
    case "FAILED":
    case "TIMEOUT":
    case "STOPPED":
      throw new Error(
        `Job ${JobRun.JobRunState}. Error: ${JobRun.ErrorMessage}`,
      );
    case "RUNNING":
      break;
    case "SUCCEEDED":
      return;
    default:
      throw new Error(`Unknown job run state: ${JobRun.JobRunState}`);
  }

  log(
    `Job ${JobRun.JobRunState}. Waiting ${waitTimeInSeconds} more seconds...`,
  );
  await wait(waitTimeInSeconds);
  return waitForJobRun(getJobRun, jobName, jobRunId);
};

/**
```

```
 * @param {{ prompter: { prompt: () => Promise<{ shouldOpen: boolean }>} }} context
 */
const promptToOpen = async (context) => {
  const { shouldOpen } = await context.prompter.prompt({
    name: "shouldOpen",
    type: "confirm",
    message: "Open the output bucket in your browser?",
  });

  if (shouldOpen) {
    return open(
      `https://s3.console.aws.amazon.com/s3/buckets/${process.env.BUCKET_NAME} to
 view the output.`,
    );
  }
};

const makeStartJobRunStep =
  ({ startJobRun, getJobRun }) =>
  async (context) => {
    log("Starting job.");
    const { JobRunId } = await startJobRun(
      process.env.JOB_NAME,
      process.env.DATABASE_NAME,
      process.env.TABLE_NAME,
      process.env.BUCKET_NAME,
    );
    log("Job started.", { type: "success" });

    log("Waiting for job to finish running. This can take a while.");
    await waitForJobRun(getJobRun, process.env.JOB_NAME, JobRunId);
    log("Job run succeeded.", { type: "success" });

    await promptToOpen(context);

    return { ...context };
  };
```

列出任務執行的相關資訊，並檢視部分轉換的資料。

```
const getJobRuns = (jobName) => {
  const client = new GlueClient({});
```

```
  const command = new GetJobRunsCommand({
    JobName: jobName,
  });

  return client.send(command);
};

const getJobRun = (jobName, jobRunId) => {
  const client = new GlueClient({});
  const command = new GetJobRunCommand({
    JobName: jobName,
    RunId: jobRunId,
  });

  return client.send(command);
};

/**
 * @typedef {{ prompter: { prompt: () => Promise<{jobName: string}> } }} Context
 */

/**
 * @typedef {() => Promise<import('@aws-sdk/client-glue').GetJobRunCommandOutput>}
 getJobRun
 */

/**
 * @typedef {() => Promise<import('@aws-sdk/client-glue').GetJobRunsCommandOutput}
 getJobRuns
 */

/**
 *
 * @param {getJobRun} getJobRun
 * @param {string} jobName
 * @param {string} jobRunId
 */
const logJobRunDetails = async (getJobRun, jobName, jobRunId) => {
  const { JobRun } = await getJobRun(jobName, jobRunId);
  log(JobRun, { type: "object" });
};

/**
 *
```

```
 * @param {{getJobRuns: getJobRuns, getJobRun: getJobRun }} funcs
 */
const makePickJobRunStep =
  ({ getJobRuns, getJobRun }) =>
  async (/** @type { Context } */ context) => {
    if (context.selectedJobName) {
      const { JobRuns } = await getJobRuns(context.selectedJobName);

      const { jobRunId } = await context.prompter.prompt({
        name: "jobRunId",
        type: "list",
        message: "Select a job run to see details.",
        choices: JobRuns.map((run) => run.Id),
      });

      logJobRunDetails(getJobRun, context.selectedJobName, jobRunId);
    }

    return { ...context };
  };
```

刪除透過示範建立的所有資源。

```
const deleteJob = (jobName) => {
  const client = new GlueClient({});

  const command = new DeleteJobCommand({
    JobName: jobName,
  });

  return client.send(command);
};

const deleteTable = (databaseName, tableName) => {
  const client = new GlueClient({});

  const command = new DeleteTableCommand({
    DatabaseName: databaseName,
    Name: tableName,
  });

  return client.send(command);
```

```
};

const deleteDatabase = (databaseName) => {
  const client = new GlueClient({});

  const command = new DeleteDatabaseCommand({
    Name: databaseName,
  });

  return client.send(command);
};

const deleteCrawler = (crawlerName) => {
  const client = new GlueClient({});

  const command = new DeleteCrawlerCommand({
    Name: crawlerName,
  });

  return client.send(command);
};

/**
 *
 * @param {import('../../../actions/delete-job.js').deleteJob} deleteJobFn
 * @param {string[]} jobNames
 * @param {{ prompter: { prompt: () => Promise<any> }}} context
 */
const handleDeleteJobs = async (deleteJobFn, jobNames, context) => {
  /**
   * @type {{ selectedJobNames: string[] }}
   */
  const { selectedJobNames } = await context.prompter.prompt({
    name: "selectedJobNames",
    type: "checkbox",
    message: "Let's clean up jobs. Select jobs to delete.",
    choices: jobNames,
  });

  if (selectedJobNames.length === 0) {
    log("No jobs selected.");
  } else {
    log("Deleting jobs.");
    await Promise.all(
```

```
      selectedJobNames.map((n) => deleteJobFn(n).catch(console.error)),
    );
    log("Jobs deleted.", { type: "success" });
  }
};

/**
 * @param {{
 *   listJobs: import('../../../actions/list-jobs.js').listJobs,
 *   deleteJob: import('../../../actions/delete-job.js').deleteJob
 * }} config
 */
const makeCleanUpJobsStep =
  ({ listJobs, deleteJob }) =>
  async (context) => {
    const { JobNames } = await listJobs();
    if (JobNames.length > 0) {
      await handleDeleteJobs(deleteJob, JobNames, context);
    }

    return { ...context };
  };

/**
 * @param {import('../../../actions/delete-table.js').deleteTable} deleteTable
 * @param {string} databaseName
 * @param {string[]} tableNames
 */
const deleteTables = (deleteTable, databaseName, tableNames) =>
  Promise.all(
    tableNames.map((tableName) =>
      deleteTable(databaseName, tableName).catch(console.error),
    ),
  );

/**
 * @param {{
 *  getTables: import('../../../actions/get-tables.js').getTables,
 *  deleteTable: import('../../../actions/delete-table.js').deleteTable
 * }} config
 */
const makeCleanUpTablesStep =
  ({ getTables, deleteTable }) =>
  /**
```

```
   * @param {{ prompter: { prompt: () => Promise<any>}}} context
   */
  async (context) => {
    const { TableList } = await getTables(process.env.DATABASE_NAME).catch(
      () => ({ TableList: null }),
    );

    if (TableList && TableList.length > 0) {
      /**
       * @type {{ tableNames: string[] }}
       */
      const { tableNames } = await context.prompter.prompt({
        name: "tableNames",
        type: "checkbox",
        message: "Let's clean up tables. Select tables to delete.",
        choices: TableList.map((t) => t.Name),
      });

      if (tableNames.length === 0) {
        log("No tables selected.");
      } else {
        log("Deleting tables.");
        await deleteTables(deleteTable, process.env.DATABASE_NAME, tableNames);
        log("Tables deleted.", { type: "success" });
      }
    }

    return { ...context };
  };

/**
 * @param {import('../../../actions/delete-database.js').deleteDatabase}
 deleteDatabase
 * @param {string[]} databaseNames
 */
const deleteDatabases = (deleteDatabase, databaseNames) =>
  Promise.all(
    databaseNames.map((dbName) => deleteDatabase(dbName).catch(console.error)),
  );

/**
 * @param {{
 *   getDatabases: import('../../../actions/get-databases.js').getDatabases
 *   deleteDatabase: import('../../../actions/delete-database.js').deleteDatabase
```

```
 * }} config
 */
const makeCleanUpDatabasesStep =
  ({ getDatabases, deleteDatabase }) =>
  /**
   * @param {{ prompter: { prompt: () => Promise<any>}} context
   */
  async (context) => {
    const { DatabaseList } = await getDatabases();

    if (DatabaseList.length > 0) {
      /** @type {{ dbNames: string[] }} */
      const { dbNames } = await context.prompter.prompt({
        name: "dbNames",
        type: "checkbox",
        message: "Let's clean up databases. Select databases to delete.",
        choices: DatabaseList.map((db) => db.Name),
      });

      if (dbNames.length === 0) {
        log("No databases selected.");
      } else {
        log("Deleting databases.");
        await deleteDatabases(deleteDatabase, dbNames);
        log("Databases deleted.", { type: "success" });
      }
    }

    return { ...context };
  };

const cleanUpCrawlerStep = async (context) => {
  log(`Deleting crawler.`);

  try {
    await deleteCrawler(process.env.CRAWLER_NAME);
    log("Crawler deleted.", { type: "success" });
  } catch (err) {
    if (err.name === "EntityNotFoundException") {
      log(`Crawler is already deleted.`);
    } else {
      throw err;
    }
  }
```

```
  return { ...context };
};
```

- 如需詳API細資訊，請參閱「AWS SDK for JavaScript API參考」中的下列主題。
  - CreateCrawler
  - CreateJob
  - DeleteCrawler
  - DeleteDatabase
  - DeleteJob
  - DeleteTable
  - GetCrawler
  - GetDatabase
  - GetDatabases
  - GetJob
  - GetJobRun
  - GetJobRuns
  - GetTables
  - ListJobs
  - StartCrawler
  - StartJobRun

動作

`CreateCrawler`

下列程式碼範例會示範如何使用`CreateCrawler`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
const createCrawler = (name, role, dbName, tablePrefix, s3TargetPath) => {
  const client = new GlueClient({});

  const command = new CreateCrawlerCommand({
    Name: name,
    Role: role,
    DatabaseName: dbName,
    TablePrefix: tablePrefix,
    Targets: {
      S3Targets: [{ Path: s3TargetPath }],
    },
  });

  return client.send(command);
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考CreateCrawler中的。

`CreateJob`

下列程式碼範例會示範如何使用`CreateJob`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
const createJob = (name, role, scriptBucketName, scriptKey) => {
```

```
  const client = new GlueClient({});

  const command = new CreateJobCommand({
    Name: name,
    Role: role,
    Command: {
      Name: "glueetl",
      PythonVersion: "3",
      ScriptLocation: `s3://${scriptBucketName}/${scriptKey}`,
    },
    GlueVersion: "3.0",
  });

  return client.send(command);
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考CreateJob中的。

`DeleteCrawler`

下列程式碼範例會示範如何使用`DeleteCrawler`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
const deleteCrawler = (crawlerName) => {
  const client = new GlueClient({});

  const command = new DeleteCrawlerCommand({
    Name: crawlerName,
  });

  return client.send(command);
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考DeleteCrawler中的。

`DeleteDatabase`

下列程式碼範例會示範如何使用`DeleteDatabase`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
const deleteDatabase = (databaseName) => {
  const client = new GlueClient({});

  const command = new DeleteDatabaseCommand({
    Name: databaseName,
  });

  return client.send(command);
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考DeleteDatabase中的。

`DeleteJob`

下列程式碼範例會示範如何使用`DeleteJob`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
const deleteJob = (jobName) => {
```

```
  const client = new GlueClient({});

  const command = new DeleteJobCommand({
    JobName: jobName,
  });

  return client.send(command);
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考DeleteJob中的。

`DeleteTable`

下列程式碼範例會示範如何使用`DeleteTable`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
const deleteTable = (databaseName, tableName) => {
  const client = new GlueClient({});

  const command = new DeleteTableCommand({
    DatabaseName: databaseName,
    Name: tableName,
  });

  return client.send(command);
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考DeleteTable中的。

`GetCrawler`

下列程式碼範例會示範如何使用`GetCrawler`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
const getCrawler = (name) => {
  const client = new GlueClient({});

  const command = new GetCrawlerCommand({
    Name: name,
  });

  return client.send(command);
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考GetCrawler中的。

`GetDatabase`

下列程式碼範例會示範如何使用`GetDatabase`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
const getDatabase = (name) => {
  const client = new GlueClient({});

  const command = new GetDatabaseCommand({
    Name: name,
  });
```

```
  return client.send(command);
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考GetDatabase中的。

`GetDatabases`

下列程式碼範例會示範如何使用`GetDatabases`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
const getDatabases = () => {
  const client = new GlueClient({});

  const command = new GetDatabasesCommand({});

  return client.send(command);
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考GetDatabases中的。

`GetJob`

下列程式碼範例會示範如何使用`GetJob`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
const getJob = (jobName) => {
  const client = new GlueClient({});

  const command = new GetJobCommand({
    JobName: jobName,
  });

  return client.send(command);
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考GetJob中的。

`GetJobRun`

下列程式碼範例會示範如何使用`GetJobRun`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
const getJobRun = (jobName, jobRunId) => {
  const client = new GlueClient({});
  const command = new GetJobRunCommand({
    JobName: jobName,
    RunId: jobRunId,
  });

  return client.send(command);
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考GetJobRun中的。

`GetJobRuns`

下列程式碼範例會示範如何使用`GetJobRuns`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
const getJobRuns = (jobName) => {
  const client = new GlueClient({});
  const command = new GetJobRunsCommand({
    JobName: jobName,
  });

  return client.send(command);
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考GetJobRuns中的。

`GetTables`

下列程式碼範例會示範如何使用`GetTables`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
const getTables = (databaseName) => {
  const client = new GlueClient({});
```

```
  const command = new GetTablesCommand({
    DatabaseName: databaseName,
  });

  return client.send(command);
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考GetTables中的。

`ListJobs`

下列程式碼範例會示範如何使用`ListJobs`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
const listJobs = () => {
  const client = new GlueClient({});

  const command = new ListJobsCommand({});

  return client.send(command);
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考ListJobs中的。

`StartCrawler`

下列程式碼範例會示範如何使用`StartCrawler`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
const startCrawler = (name) => {
  const client = new GlueClient({});

  const command = new StartCrawlerCommand({
    Name: name,
  });

  return client.send(command);
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考StartCrawler中的。

`StartJobRun`

下列程式碼範例會示範如何使用`StartJobRun`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
const startJobRun = (jobName, dbName, tableName, bucketName) => {
  const client = new GlueClient({});

  const command = new StartJobRunCommand({
    JobName: jobName,
    Arguments: {
      "--input_database": dbName,
```

```
      "--input_table": tableName,
      "--output_bucket_url": `s3://${bucketName}/`,
    },
  });

  return client.send(command);
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考StartJobRun中的。

HealthImaging 使用 JavaScript （ v3 ）SDK的示例

下列程式碼範例會示範如何使用 AWS SDK for JavaScript (v3) 搭配使用來執行動作及實作常見案例 HealthImaging。

Actions 是大型程式的程式碼摘錄，必須在內容中執行。雖然動作會顯示如何呼叫個別服務函數，但您可以在其相關案例中查看內容中的動作。

案例是程式碼範例，向您展示如何透過呼叫服務中的多個函式或與其他函式結合來完成特定工作 AWS 服務。

每個範例都包含完整原始程式碼的連結，您可以在其中找到如何在內容中設定和執行程式碼的指示。

開始使用

你好 HealthImaging

下列程式碼範例會示範如何開始使用 HealthImaging。

SDK對於 JavaScript （ 3 ）

```
import {
  ListDatastoresCommand,
  MedicalImagingClient,
} from "@aws-sdk/client-medical-imaging";

// When no region or credentials are provided, the SDK will use the
// region and credentials from the local AWS config.
const client = new MedicalImagingClient({});

export const helloMedicalImaging = async () => {
  const command = new ListDatastoresCommand({});
```

```
  const { datastoreSummaries } = await client.send(command);
  console.log("Datastores: ");
  console.log(datastoreSummaries.map((item) => item.datastoreName).join("\n"));
  return datastoreSummaries;
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考ListDatastores中的。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

主題

- 動作
- 案例

動作

`CopyImageSet`

下列程式碼範例會示範如何使用CopyImageSet。

SDK對於 JavaScript（3）

複製影像集的實用程式功能。

```
import {CopyImageSetCommand} from "@aws-sdk/client-medical-imaging";
import {medicalImagingClient} from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreId - The ID of the data store.
 * @param {string} imageSetId - The source image set ID.
 * @param {string} sourceVersionId - The source version ID.
 * @param {string} destinationImageSetId - The optional ID of the destination image
 set.
 * @param {string} destinationVersionId - The optional version ID of the destination
 image set.
```

```
 * @param {boolean} force - Force the copy action.
 * @param {[string]} copySubsets - A subset of instance IDs to copy.
 */
export const copyImageSet = async (
    datastoreId = "xxxxxxxxxx",
    imageSetId = "xxxxxxxxxxx",
    sourceVersionId = "1",
    destinationImageSetId = "",
    destinationVersionId = "",
    force = false,
    copySubsets = []
) => {
    try {
        const params = {
            datastoreId: datastoreId,
            sourceImageSetId: imageSetId,
            copyImageSetInformation: {
                sourceImageSet: {latestVersionId: sourceVersionId},
            },
            force: force
        };
        if (destinationImageSetId !== "" && destinationVersionId !== "") {
            params.copyImageSetInformation.destinationImageSet = {
                imageSetId: destinationImageSetId,
                latestVersionId: destinationVersionId,
            };
        }

        if (copySubsets.length > 0) {
            let copySubsetsJson;
            copySubsetsJson = {
                SchemaVersion: 1.1,
                Study: {
                    Series: {
                        imageSetId: {
                            Instances: {}
                        }
                    }
                }
            };

            for (let i = 0; i < copySubsets.length; i++) {
                copySubsetsJson.Study.Series.imageSetId.Instances[
                    copySubsets[i]
```

```
                    ] = {};
            }

            params.copyImageSetInformation.dicomCopies = copySubsetsJson;
        }

        const response = await medicalImagingClient.send(
            new CopyImageSetCommand(params)
        );
        console.log(response);
        // {
        //     '$metadata': {
        //         httpStatusCode: 200,
        //         requestId: 'd9b219ce-cc48-4a44-a5b2-c5c3068f1ee8',
        //         extendedRequestId: undefined,
        //         cfId: undefined,
        //         attempts: 1,
        //         totalRetryDelay: 0
        //      },
        //       datastoreId: 'xxxxxxxxxxxxx',
        //       destinationImageSetProperties: {
        //             createdAt: 2023-09-27T19:46:21.824Z,
        //             imageSetArn: 'arn:aws:medical-imaging:us-
east-1:xxxxxxxxxxx:datastore/xxxxxxxxxxxxx/imageset/xxxxxxxxxxxxxxxxx',
        //             imageSetId: 'xxxxxxxxxxxxxx',
        //             imageSetState: 'LOCKED',
        //             imageSetWorkflowStatus: 'COPYING',
        //             latestVersionId: '1',
        //             updatedAt: 2023-09-27T19:46:21.824Z
        //       },
        //       sourceImageSetProperties: {
        //             createdAt: 2023-09-22T14:49:26.427Z,
        //             imageSetArn: 'arn:aws:medical-imaging:us-
east-1:xxxxxxxxxxx:datastore/xxxxxxxxxxxxx/imageset/xxxxxxxxxxxxxxx',
        //             imageSetId: 'xxxxxxxxxxxxxxx',
        //             imageSetState: 'LOCKED',
        //             imageSetWorkflowStatus: 'COPYING_WITH_READ_ONLY_ACCESS',
        //             latestVersionId: '4',
        //             updatedAt: 2023-09-27T19:46:21.824Z
        //      }
        // }
        return response;
    } catch (err) {
        console.error(err);
```

```
    }
};
```

複製沒有目的地的影像集。

```
    await copyImageSet(
        "1234567890123456789012345678901 2",
        "1234567890123456789012345678901 2",
        "1"
    );
```

複製含有目標的影像集。

```
    await copyImageSet(
        "1234567890123456789012345678901 2",
        "1234567890123456789012345678901 2",
        "1",
        "1234567890123456789012345678901 2",
        "1",
        false,
    );
```

複製具有目標的影像集子集，然後強制複製。

```
    await copyImageSet(
        "1234567890123456789012345678901 2",
        "1234567890123456789012345678901 2",
        "1",
        "1234567890123456789012345678901 2",
        "1",
        true,
        ["1234567890123456789012345678901 2", "11223344556677889900112233445566"]
    );
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考CopyImageSet中的。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

`CreateDatastore`

下列程式碼範例會示範如何使用`CreateDatastore`。

SDK對於 JavaScript（3）

```
import { CreateDatastoreCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreName - The name of the data store to create.
 */
export const createDatastore = async (datastoreName = "DATASTORE_NAME") => {
  const response = await medicalImagingClient.send(
    new CreateDatastoreCommand({ datastoreName: datastoreName })
  );
  console.log(response);
  // {
  //   '$metadata': {
  //       httpStatusCode: 200,
  //       requestId: 'a71cd65f-2382-49bf-b682-f9209d8d399b',
  //       extendedRequestId: undefined,
  //       cfId: undefined,
  //       attempts: 1,
  //       totalRetryDelay: 0
  //    },
  //    datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //    datastoreStatus: 'CREATING'
  // }
  return response;
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考CreateDatastore中的。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

`DeleteDatastore`

下列程式碼範例會示範如何使用`DeleteDatastore`。

SDK對於 JavaScript（3）

```
import { DeleteDatastoreCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreId - The ID of the data store to delete.
 */
export const deleteDatastore = async (datastoreId = "DATASTORE_ID") => {
  const response = await medicalImagingClient.send(
    new DeleteDatastoreCommand({ datastoreId })
  );
  console.log(response);
  // {
  //   '$metadata': {
  //           httpStatusCode: 200,
  //           requestId: 'f5beb409-678d-48c9-9173-9a001ee1ebb1',
  //           extendedRequestId: undefined,
  //           cfId: undefined,
  //           attempts: 1,
  //           totalRetryDelay: 0
  //         },
  //     datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //     datastoreStatus: 'DELETING'
  // }

  return response;
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考DeleteDatastore中的。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

`DeleteImageSet`

下列程式碼範例會示範如何使用`DeleteImageSet`。

SDK對於 JavaScript（3）

```
import { DeleteImageSetCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreId - The data store ID.
 * @param {string} imageSetId - The image set ID.
 */
export const deleteImageSet = async (
  datastoreId = "xxxxxxxxxxxxxxx",
  imageSetId = "xxxxxxxxxxxxxxx"
) => {
  const response = await medicalImagingClient.send(
    new DeleteImageSetCommand({
      datastoreId: datastoreId,
      imageSetId: imageSetId,
    })
  );
  console.log(response);
  // {
  //    '$metadata': {
  //         httpStatusCode: 200,
  //         requestId: '6267bbd2-eaa5-4a50-8ee8-8fddf535cf73',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  //     },
  //     datastoreId: 'xxxxxxxxxxxxxxx',
  //     imageSetId: 'xxxxxxxxxxxxxxx',
  //     imageSetState: 'LOCKED',
  //     imageSetWorkflowStatus: 'DELETING'
```

```
  // }
  return response;
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考DeleteImageSet中的。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

`GetDICOMImportJob`

下列程式碼範例會示範如何使用`GetDICOMImportJob`。

SDK對於 JavaScript（3）

```
import { GetDICOMImportJobCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreId - The ID of the data store.
 * @param {string} jobId - The ID of the import job.
 */
export const getDICOMImportJob = async (
  datastoreId = "xxxxxxxxxxxxxxxxxxxx",
  jobId = "xxxxxxxxxxxxxxxxxxxx"
) => {
  const response = await medicalImagingClient.send(
    new GetDICOMImportJobCommand({ datastoreId: datastoreId, jobId: jobId })
  );
  console.log(response);
  // {
  //     '$metadata': {
  //     httpStatusCode: 200,
  //         requestId: 'a2637936-78ea-44e7-98b8-7a87d95dfaee',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
```

```
  // },
  //     jobProperties: {
  //         dataAccessRoleArn: 'arn:aws:iam::xxxxxxxxxxxx:role/dicom_import',
  //             datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxx',
  //             endedAt: 2023-09-19T17:29:21.753Z,
  //             inputS3Uri: 's3://healthimaging-source/CTStudy/',
  //             jobId: ''xxxxxxxxxxxxxxxxxxxxxxxx'',
  //             jobName: 'job_1',
  //             jobStatus: 'COMPLETED',
  //             outputS3Uri: 's3://health-imaging-dest/
ouput_ct/'xxxxxxxxxxxxxxxxxxxxxxxx'-DicomImport-'xxxxxxxxxxxxxxxxxxxxxxxx'/',
  //             submittedAt: 2023-09-19T17:27:25.143Z
  //     }
  // }

  return response;
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參照中的 G etDICOMImport Job。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

`GetDatastore`

下列程式碼範例會示範如何使用`GetDatastore`。

SDK對於 JavaScript（3）

```
import { GetDatastoreCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreID - The ID of the data store.
 */
export const getDatastore = async (datastoreID = "DATASTORE_ID") => {
  const response = await medicalImagingClient.send(
    new GetDatastoreCommand({ datastoreId: datastoreID })
```

```
  );
  console.log(response);
  // {
  //   '$metadata': {
  //       httpStatusCode: 200,
  //       requestId: '55ea7d2e-222c-4a6a-871e-4f591f40cadb',
  //       extendedRequestId: undefined,
  //       cfId: undefined,
  //       attempts: 1,
  //       totalRetryDelay: 0
  //    },
  //   datastoreProperties: {
  //        createdAt: 2023-08-04T18:50:36.239Z,
  //         datastoreArn: 'arn:aws:medical-imaging:us-east-1:xxxxxxxxx:datastore/
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //         datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //         datastoreName: 'my_datastore',
  //         datastoreStatus: 'ACTIVE',
  //         updatedAt: 2023-08-04T18:50:36.239Z
  //   }
  // }
  return response["datastoreProperties"];
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考GetDatastore中的。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

`GetImageFrame`

下列程式碼範例會示範如何使用`GetImageFrame`。

SDK對於 JavaScript（3）

```
import { GetImageFrameCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";
```

```
/**
 * @param {string} imageFrameFileName - The name of the file for the HTJ2K-encoded
 image frame.
 * @param {string} datastoreID - The data store's ID.
 * @param {string} imageSetID - The image set's ID.
 * @param {string} imageFrameID - The image frame's ID.
 */
export const getImageFrame = async (
  imageFrameFileName = "image.jph",
  datastoreID = "DATASTORE_ID",
  imageSetID = "IMAGE_SET_ID",
  imageFrameID = "IMAGE_FRAME_ID"
) => {
  const response = await medicalImagingClient.send(
    new GetImageFrameCommand({
      datastoreId: datastoreID,
      imageSetId: imageSetID,
      imageFrameInformation: { imageFrameId: imageFrameID },
    })
  );
  const buffer = await response.imageFrameBlob.transformToByteArray();
  writeFileSync(imageFrameFileName, buffer);

  console.log(response);
  // {
  //     '$metadata': {
  //         httpStatusCode: 200,
  //         requestId: 'e4ab42a5-25a3-4377-873f-374ecf4380e1',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  //     },
  //     contentType: 'application/octet-stream',
  //     imageFrameBlob: <ref *1> IncomingMessage {}
  // }
  return response;
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考GetImageFrame中的。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

`GetImageSet`

下列程式碼範例會示範如何使用`GetImageSet`。

SDK對於 JavaScript（3）

```
import { GetImageSetCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreId - The ID of the data store.
 * @param {string} imageSetId - The ID of the image set.
 * @param {string} imageSetVersion - The optional version of the image set.
 *
 */
export const getImageSet = async (
  datastoreId = "xxxxxxxxxxxxxxx",
  imageSetId = "xxxxxxxxxxxxxxx",
  imageSetVersion = ""
) => {
  let params = { datastoreId: datastoreId, imageSetId: imageSetId };
  if (imageSetVersion !== "") {
    params.imageSetVersion = imageSetVersion;
  }
  const response = await medicalImagingClient.send(
    new GetImageSetCommand(params)
  );
  console.log(response);
  // {
  //     '$metadata': {
  //     httpStatusCode: 200,
  //         requestId: '0615c161-410d-4d06-9d8c-6e1241bb0a5a',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  // },
```

```
  //     createdAt: 2023-09-22T14:49:26.427Z,
  //     datastoreId: 'xxxxxxxxxxxxxx',
  //     imageSetArn: 'arn:aws:medical-imaging:us-east-1:xxxxxxxxxx:datastore/
xxxxxxxxxxxxxxxxxxx/imageset/xxxxxxxxxxxxxxxxxx',
  //     imageSetId: 'xxxxxxxxxxxxxx',
  //     imageSetState: 'ACTIVE',
  //     imageSetWorkflowStatus: 'CREATED',
  //     updatedAt: 2023-09-22T14:49:26.427Z,
  //     versionId: '1'
  // }

  return response;
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考GetImageSet中的。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

`GetImageSetMetadata`

下列程式碼範例會示範如何使用`GetImageSetMetadata`。

SDK對於 JavaScript（3）

獲取圖像集元數據的實用程序功能。

```
import { GetImageSetMetadataCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";
import { writeFileSync } from "fs";

/**
 * @param {string} metadataFileName - The name of the file for the gzipped metadata.
 * @param {string} datastoreId - The ID of the data store.
 * @param {string} imagesetId - The ID of the image set.
 * @param {string} versionID - The optional version ID of the image set.
 */
export const getImageSetMetadata = async (
```

```
  metadataFileName = "metadata.json.gzip",
  datastoreId = "xxxxxxxxxxxxx",
  imagesetId = "xxxxxxxxxxxxx",
  versionID = ""
) => {
  const params = { datastoreId: datastoreId, imageSetId: imagesetId };

  if (versionID) {
    params.versionID = versionID;
  }

  const response = await medicalImagingClient.send(
    new GetImageSetMetadataCommand(params)
  );
  const buffer = await response.imageSetMetadataBlob.transformToByteArray();
  writeFileSync(metadataFileName, buffer);

  console.log(response);
  // {
  //     '$metadata': {
  //     httpStatusCode: 200,
  //         requestId: '5219b274-30ff-4986-8cab-48753de3a599',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  // },
  //     contentType: 'application/json',
  //     contentEncoding: 'gzip',
  //     imageSetMetadataBlob: <ref *1> IncomingMessage {}
  // }

  return response;
};
```

獲取沒有版本的圖像集元數據。

```
  try {
    await getImageSetMetadata(
      "metadata.json.gzip",
      "12345678901234567890123456789012",
```

```
      "12345678901234567890123456789012"
    );
  } catch (err) {
    console.log("Error", err);
  }
```

獲取具有版本的圖像集元數據。

```
  try {
    await getImageSetMetadata(
      "metadata2.json.gzip",
      "12345678901234567890123456789012",
      "12345678901234567890123456789012",
      "1"
    );
  } catch (err) {
    console.log("Error", err);
  }
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考GetImageSetMetadata中的。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

`ListDICOMImportJobs`

下列程式碼範例會示範如何使用`ListDICOMImportJobs`。

SDK對於 JavaScript（3）

```
import { paginateListDICOMImportJobs } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreId - The ID of the data store.
 */
```

```
export const listDICOMImportJobs = async (
  datastoreId = "xxxxxxxxxxxxxxxxx"
) => {
  const paginatorConfig = {
    client: medicalImagingClient,
    pageSize: 50,
  };

  const commandParams = { datastoreId: datastoreId };
  const paginator = paginateListDICOMImportJobs(paginatorConfig, commandParams);

  let jobSummaries = [];
  for await (const page of paginator) {
    // Each page contains a list of `jobSummaries`. The list is truncated if is
 larger than `pageSize`.
    jobSummaries.push(...page["jobSummaries"]);
    console.log(page);
  }
  // {
  //     '$metadata': {
  //     httpStatusCode: 200,
  //         requestId: '3c20c66e-0797-446a-a1d8-91b742fd15a0',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  // },
  //     jobSummaries: [
  //         {
  //             dataAccessRoleArn: 'arn:aws:iam::xxxxxxxxxxxx:role/dicom_import',
  //             datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxx',
  //             endedAt: 2023-09-22T14:49:51.351Z,
  //             jobId: 'xxxxxxxxxxxxxxxxxxxxxxxx',
  //             jobName: 'test-1',
  //             jobStatus: 'COMPLETED',
  //             submittedAt: 2023-09-22T14:48:45.767Z
  // }
  // ]}

  return jobSummaries;
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考中的 L istDICOMImport 工作。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

`ListDatastores`

下列程式碼範例會示範如何使用`ListDatastores`。

SDK對於 JavaScript（3）

```
import { paginateListDatastores } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

export const listDatastores = async () => {
  const paginatorConfig = {
    client: medicalImagingClient,
    pageSize: 50,
  };

  const commandParams = {};
  const paginator = paginateListDatastores(paginatorConfig, commandParams);

  /**
   * @type {import("@aws-sdk/client-medical-imaging").DatastoreSummary[]}
   */
  const datastoreSummaries = [];
  for await (const page of paginator) {
    // Each page contains a list of `jobSummaries`. The list is truncated if is
 larger than `pageSize`.
    datastoreSummaries.push(...page["datastoreSummaries"]);
    console.log(page);
  }
  // {
  //   '$metadata': {
  //       httpStatusCode: 200,
  //       requestId: '6aa99231-d9c2-4716-a46e-edb830116fa3',
  //       extendedRequestId: undefined,
  //       cfId: undefined,
  //       attempts: 1,
  //       totalRetryDelay: 0
  //   },
```

```
  //   datastoreSummaries: [
  //     {
  //       createdAt: 2023-08-04T18:49:54.429Z,
  //       datastoreArn: 'arn:aws:medical-imaging:us-east-1:xxxxxxxxx:datastore/
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //       datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //       datastoreName: 'my_datastore',
  //       datastoreStatus: 'ACTIVE',
  //       updatedAt: 2023-08-04T18:49:54.429Z
  //     }
  //     ...
  //   ]
  // }

  return datastoreSummaries;
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考ListDatastores中的。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

`ListImageSetVersions`

下列程式碼範例會示範如何使用`ListImageSetVersions`。

SDK對於 JavaScript（3）

```
import { paginateListImageSetVersions } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreId - The ID of the data store.
 * @param {string} imageSetId - The ID of the image set.
 */
export const listImageSetVersions = async (
  datastoreId = "xxxxxxxxxxxx",
  imageSetId = "xxxxxxxxxxxx"
```

```
) => {
  const paginatorConfig = {
    client: medicalImagingClient,
    pageSize: 50,
  };

  const commandParams = { datastoreId, imageSetId };
  const paginator = paginateListImageSetVersions(
    paginatorConfig,
    commandParams
  );

  let imageSetPropertiesList = [];
  for await (const page of paginator) {
    // Each page contains a list of `jobSummaries`. The list is truncated if is
 larger than `pageSize`.
    imageSetPropertiesList.push(...page["imageSetPropertiesList"]);
    console.log(page);
  }
  // {
  //     '$metadata': {
  //         httpStatusCode: 200,
  //         requestId: '74590b37-a002-4827-83f2-3c590279c742',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  //     },
  //     imageSetPropertiesList: [
  //         {
  //             ImageSetWorkflowStatus: 'CREATED',
  //             createdAt: 2023-09-22T14:49:26.427Z,
  //             imageSetId: 'xxxxxxxxxxxxxxxxxxxxxx',
  //             imageSetState: 'ACTIVE',
  //             versionId: '1'
  //         }]
  // }
  return imageSetPropertiesList;
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考ListImageSetVersions中的。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

`ListTagsForResource`

下列程式碼範例會示範如何使用`ListTagsForResource`。

SDK對於 JavaScript（3）

```
import { ListTagsForResourceCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} resourceArn - The Amazon Resource Name (ARN) for the data store
 or image set.
 */
export const listTagsForResource = async (
  resourceArn = "arn:aws:medical-imaging:us-east-1:abc:datastore/def/imageset/ghi"
) => {
  const response = await medicalImagingClient.send(
    new ListTagsForResourceCommand({ resourceArn: resourceArn })
  );
  console.log(response);
  // {
  //     '$metadata': {
  //         httpStatusCode: 200,
  //         requestId: '008fc6d3-abec-4870-a155-20fa3631e645',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  //     },
  //     tags: { Deployment: 'Development' }
  // }

  return response;
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考ListTagsForResource中的。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

`SearchImageSets`

下列程式碼範例會示範如何使用`SearchImageSets`。

SDK對於 JavaScript（3）

用於搜索圖像集的實用程序功能。

```
import {paginateSearchImageSets} from "@aws-sdk/client-medical-imaging";
import {medicalImagingClient} from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreId - The data store's ID.
 * @param { import('@aws-sdk/client-medical-imaging').SearchFilter[] } filters - The
 search criteria filters.
 * @param { import('@aws-sdk/client-medical-imaging').Sort } sort - The search
 criteria sort.
 */
export const searchImageSets = async (
    datastoreId = "xxxxxxxx",
    searchCriteria = {}
) => {
    const paginatorConfig = {
        client: medicalImagingClient,
        pageSize: 50,
    };

    const commandParams = {
        datastoreId: datastoreId,
        searchCriteria: searchCriteria,
    };

    const paginator = paginateSearchImageSets(paginatorConfig, commandParams);

    const imageSetsMetadataSummaries = [];
```

```
    for await (const page of paginator) {
        // Each page contains a list of `jobSummaries`. The list is truncated if is
 larger than `pageSize`.
        imageSetsMetadataSummaries.push(...page["imageSetsMetadataSummaries"]);
        console.log(page);
    }
    // {
    //     '$metadata': {
    //         httpStatusCode: 200,
    //         requestId: 'f009ea9c-84ca-4749-b5b6-7164f00a5ada',
    //         extendedRequestId: undefined,
    //         cfId: undefined,
    //         attempts: 1,
    //         totalRetryDelay: 0
    //     },
    //     imageSetsMetadataSummaries: [
    //         {
    //             DICOMTags: [Object],
    //             createdAt: "2023-09-19T16:59:40.551Z",
    //             imageSetId: '7f75e1b5c0f40eac2b24cf712f485f50',
    //             updatedAt: "2023-09-19T16:59:40.551Z",
    //             version: 1
    //         }]
    // }

    return imageSetsMetadataSummaries;
};
```

使用案例 #1: EQUAL 運算子。

```
    const datastoreId = "12345678901234567890123456789012";

    try {
        const searchCriteria = {
            filters: [
                {
                    values: [{DICOMPatientId: "1234567"}],
                    operator: "EQUAL",
                },
            ]
        };
```

```
    await searchImageSets(datastoreId, searchCriteria);
} catch (err) {
    console.error(err);
}
```

使用案例 #2: BETWEEN 運算子使用DICOMStudyDate和DICOMStudyTime。

```
const datastoreId = "12345678901234567890123456789012";

try {
    const searchCriteria = {
        filters: [
            {
                values: [
                    {
                        DICOMStudyDateAndTime: {
                            DICOMStudyDate: "19900101",
                            DICOMStudyTime: "000000",
                        },
                    },
                    {
                        DICOMStudyDateAndTime: {
                            DICOMStudyDate: "20230901",
                            DICOMStudyTime: "000000",
                        },
                    },
                ],
                operator: "BETWEEN",
            },
        ]
    };

    await searchImageSets(datastoreId, searchCriteria);
} catch (err) {
    console.error(err);
}
```

使用案例 #3: BETWEEN 運算子使用createdAt. 時間研究以前持續存在。

```
const datastoreId = "12345678901234567890123456789012";
```

```
    try {
        const searchCriteria = {
            filters: [
                {
                    values: [
                        {createdAt: new Date("1985-04-12T23:20:50.52Z")},
                        {createdAt: new Date()},
                    ],
                    operator: "BETWEEN",
                },
            ]
        };

        await searchImageSets(datastoreId, searchCriteria);
    } catch (err) {
        console.error(err);
    }
```

使用案例 #4: 開啟DICOMSeriesInstanceUID和BETWEEN開啟EQUAL運算子， updatedAt 並按 updatedAt 欄位ASC順序排序回應。

```
    const datastoreId = "12345678901234567890123456789012";

    try {
        const searchCriteria = {
            filters: [
                {
                    values: [
                        {updatedAt: new Date("1985-04-12T23:20:50.52Z")},
                        {updatedAt: new Date()},
                    ],
                    operator: "BETWEEN",
                },
                {
                    values: [
                        {DICOMSeriesInstanceUID:
"1.1.123.123456.1.12.1.1234567890.1234.12345678.123"},
                    ],
                    operator: "EQUAL",
                },
            ],
            sort: {
```

```
                sortOrder: "ASC",
                sortField: "updatedAt",
            }
        };

        await searchImageSets(datastoreId, searchCriteria);
    } catch (err) {
        console.error(err);
    }
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考SearchImageSets中的。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

`StartDICOMImportJob`

下列程式碼範例會示範如何使用`StartDICOMImportJob`。

SDK對於 JavaScript（3）

```
import { StartDICOMImportJobCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} jobName - The name of the import job.
 * @param {string} datastoreId - The ID of the data store.
 * @param {string} dataAccessRoleArn - The Amazon Resource Name (ARN) of the role
 that grants permission.
 * @param {string} inputS3Uri - The URI of the S3 bucket containing the input files.
 * @param {string} outputS3Uri - The URI of the S3 bucket where the output files are
 stored.
 */
export const startDicomImportJob = async (
  jobName = "test-1",
  datastoreId = "12345678901234567890123456789012",
  dataAccessRoleArn = "arn:aws:iam::xxxxxxxxxxxx:role/ImportJobDataAccessRole",
  inputS3Uri = "s3://medical-imaging-dicom-input/dicom_input/",
```

```
  outputS3Uri = "s3://medical-imaging-output/job_output/"
) => {
  const response = await medicalImagingClient.send(
    new StartDICOMImportJobCommand({
      jobName: jobName,
      datastoreId: datastoreId,
      dataAccessRoleArn: dataAccessRoleArn,
      inputS3Uri: inputS3Uri,
      outputS3Uri: outputS3Uri,
    })
  );
  console.log(response);
  // {
  //     '$metadata': {
  //     httpStatusCode: 200,
  //         requestId: '6e81d191-d46b-4e48-a08a-cdcc7e11eb79',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  // },
  //     datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //     jobId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //     jobStatus: 'SUBMITTED',
  //     submittedAt: 2023-09-22T14:48:45.767Z
  // }
  return response;
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參照中的 S tartDICOMImport Job。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

`TagResource`

下列程式碼範例會示範如何使用TagResource。

SDK對於 JavaScript（3）

```
import { TagResourceCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} resourceArn - The Amazon Resource Name (ARN) for the data store
 or image set.
 * @param {Record<string,string>} tags - The tags to add to the resource as JSON.
 *                      - For example: {"Deployment" : "Development"}
 */
export const tagResource = async (
  resourceArn = "arn:aws:medical-imaging:us-east-1:xxxxxx:datastore/xxxxx/imageset/
xxx",
  tags = {}
) => {
  const response = await medicalImagingClient.send(
    new TagResourceCommand({ resourceArn: resourceArn, tags: tags })
  );
  console.log(response);
  // {
  //     '$metadata': {
  //        httpStatusCode: 204,
  //         requestId: '8a6de9a3-ec8e-47ef-8643-473518b19d45',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  //    }
  // }

  return response;
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考TagResource中的。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

`UntagResource`

下列程式碼範例會示範如何使用`UntagResource`。

SDK對於 JavaScript（3）

```
import { UntagResourceCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} resourceArn - The Amazon Resource Name (ARN) for the data store
 or image set.
 * @param {string[]} tagKeys - The keys of the tags to remove.
 */
export const untagResource = async (
  resourceArn = "arn:aws:medical-imaging:us-east-1:xxxxxx:datastore/xxxxx/imageset/
xxx",
  tagKeys = []
) => {
  const response = await medicalImagingClient.send(
    new UntagResourceCommand({ resourceArn: resourceArn, tagKeys: tagKeys })
  );
  console.log(response);
  // {
  //     '$metadata': {
  //        httpStatusCode: 204,
  //         requestId: '8a6de9a3-ec8e-47ef-8643-473518b19d45',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  //    }
  // }

  return response;
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考UntagResource中的。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

`UpdateImageSetMetadata`

下列程式碼範例會示範如何使用UpdateImageSetMetadata。

SDK對於 JavaScript（3）

```
import {UpdateImageSetMetadataCommand} from "@aws-sdk/client-medical-imaging";
import {medicalImagingClient} from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreId - The ID of the HealthImaging data store.
 * @param {string} imageSetId - The ID of the HealthImaging image set.
 * @param {string} latestVersionId - The ID of the HealthImaging image set version.
 * @param {{}} updateMetadata - The metadata to update.
 * @param {boolean} force - Force the update.
 */
export const updateImageSetMetadata = async (datastoreId = "xxxxxxxxxx",
                                             imageSetId = "xxxxxxxxxx",
                                             latestVersionId = "1",
                                             updateMetadata = '{}',
                                             force = false) => {
    try {
        const response = await medicalImagingClient.send(
            new UpdateImageSetMetadataCommand({
                datastoreId: datastoreId,
                imageSetId: imageSetId,
                latestVersionId: latestVersionId,
                updateImageSetMetadataUpdates: updateMetadata,
                force: force,
            })
        );
        console.log(response);
        // {
        //     '$metadata': {
        //     httpStatusCode: 200,
        //         requestId: '7966e869-e311-4bff-92ec-56a61d3003ea',
        //         extendedRequestId: undefined,
```

```
        //         cfId: undefined,
        //         attempts: 1,
        //         totalRetryDelay: 0
        // },
        //     createdAt: 2023-09-22T14:49:26.427Z,
        //     datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
        //     imageSetId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
        //     imageSetState: 'LOCKED',
        //     imageSetWorkflowStatus: 'UPDATING',
        //     latestVersionId: '4',
        //     updatedAt: 2023-09-27T19:41:43.494Z
        // }
        return response;
    } catch (err) {
        console.error(err);
    }
};
```

使用案例 #1: 插入或更新屬性並強制更新。

```
        const insertAttributes =
            JSON.stringify({
                "SchemaVersion": 1.1,
                "Study": {
                    "DICOM": {
                        "StudyDescription": "CT CHEST"
                    }
                }
            });

        const updateMetadata = {
            "DICOMUpdates": {
                "updatableAttributes":
                    new TextEncoder().encode(insertAttributes)
            }
        };

        await updateImageSetMetadata(datastoreID, imageSetID,
            versionID, updateMetadata, true);
```

使用案例 #2: 移除屬性。

```
        // Attribute key and value must match the existing attribute.
        const remove_attribute =
            JSON.stringify({
                "SchemaVersion": 1.1,
                "Study": {
                    "DICOM": {
                        "StudyDescription": "CT CHEST"
                    }
                }
            });

        const updateMetadata = {
            "DICOMUpdates": {
                "removableAttributes":
                    new TextEncoder().encode(remove_attribute)
            }
        };

        await updateImageSetMetadata(datastoreID, imageSetID,
            versionID, updateMetadata);
```

使用案例 #3: 移除執行個體。

```
        const remove_instance =
            JSON.stringify({
                "SchemaVersion": 1.1,
                "Study": {
                    "Series": {
                        "1.1.1.1.1.1.12345.123456789012.123.12345678901234.1": {
                            "Instances": {

"1.1.1.1.1.1.12345.123456789012.123.12345678901234.1": {}
                            }
                        }
                    }
                }
            });

        const updateMetadata = {
            "DICOMUpdates": {
                "removableAttributes":
                    new TextEncoder().encode(remove_instance)
```

```
            }
        };

        await updateImageSetMetadata(datastoreID, imageSetID,
            versionID, updateMetadata);
```

使用案例 #4: 還原為較早的版本。

```
    const updateMetadata = {
        "revertToVersionId": "1"
    };

    await updateImageSetMetadata(datastoreID, imageSetID,
        versionID, updateMetadata);
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考UpdateImageSetMetadata中的。

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

案例

開始使用影像集和影像框

下列程式碼範例示範如何在中匯入DICOM檔案和下載影像框 HealthImaging。

實作結構為工作流程命令列應用程式。

- 設定DICOM匯入的資源。
- 將DICOM檔案匯入資料倉庫。
- 擷取匯入工IDs作的影像集。
- 擷取影像集IDs的影像框。
- 下載，解碼和驗證圖像幀。
- 清理資源。

SDK對於 JavaScript（3）

index.js-協調步驟。

```
import {
  parseScenarioArgs,
  Scenario,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import {
  saveState,
  loadState,
} from "@aws-doc-sdk-examples/lib/scenario/steps-common.js";

import {
  createStack,
  deployStack,
  getAccountId,
  getDatastoreName,
  getStackName,
  outputState,
  waitForStackCreation,
} from "./deploy-steps.js";
import {
  doCopy,
  selectDataset,
  copyDataset,
  outputCopiedObjects,
} from "./dataset-steps.js";
import {
  doImport,
  outputImportJobStatus,
  startDICOMImport,
  waitForImportJobCompletion,
} from "./import-steps.js";
import {
  getManifestFile,
  outputImageSetIds,
  parseManifestFile,
} from "./image-set-steps.js";
import {
  getImageSetMetadata,
  outputImageFrameIds,
} from "./image-frame-steps.js";
```

```
import { decodeAndVerifyImages, doVerify } from "./verify-steps.js";
import {
  confirmCleanup,
  deleteImageSets,
  deleteStack,
} from "./clean-up-steps.js";

const context = {};

const scenarios = {
  deploy: new Scenario(
    "Deploy Resources",
    [
      deployStack,
      getStackName,
      getDatastoreName,
      getAccountId,
      createStack,
      waitForStackCreation,
      outputState,
      saveState,
    ],
    context,
  ),
  demo: new Scenario(
    "Run Demo",
    [
      loadState,
      doCopy,
      selectDataset,
      copyDataset,
      outputCopiedObjects,
      doImport,
      startDICOMImport,
      waitForImportJobCompletion,
      outputImportJobStatus,
      getManifestFile,
      parseManifestFile,
      outputImageSetIds,
      getImageSetMetadata,
      outputImageFrameIds,
      doVerify,
      decodeAndVerifyImages,
      saveState,
```

```
    ],
    context,
  ),
  destroy: new Scenario(
    "Clean Up Resources",
    [loadState, confirmCleanup, deleteImageSets, deleteStack],
    context,
  ),
};

// Call function if run directly
import { fileURLToPath } from "url";
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  parseScenarioArgs(scenarios);
}
```

deploy-steps.js-部署資源。

```
import fs from "node:fs/promises";
import path from "node:path";

import {
  CloudFormationClient,
  CreateStackCommand,
  DescribeStacksCommand,
} from "@aws-sdk/client-cloudformation";
import { STSClient, GetCallerIdentityCommand } from "@aws-sdk/client-sts";

import {
  ScenarioAction,
  ScenarioInput,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

const cfnClient = new CloudFormationClient({});
const stsClient = new STSClient({});

const __dirname = path.dirname(new URL(import.meta.url).pathname);
const cfnTemplatePath = path.join(
  __dirname,
  "../../../../../workflows/healthimaging_image_sets/resources/cfn_template.yaml",
```

```
);

export const deployStack = new ScenarioInput(
  "deployStack",
  "Do you want to deploy the CloudFormation stack?",
  { type: "confirm" },
);

export const getStackName = new ScenarioInput(
  "getStackName",
  "Enter a name for the CloudFormation stack:",
  { type: "input", skipWhen: (/** @type {{}} */ state) => !state.deployStack },
);

export const getDatastoreName = new ScenarioInput(
  "getDatastoreName",
  "Enter a name for the HealthImaging datastore:",
  { type: "input", skipWhen: (/** @type {{}} */ state) => !state.deployStack },
);

export const getAccountId = new ScenarioAction(
  "getAccountId",
  async (/** @type {{}} */ state) => {
    const command = new GetCallerIdentityCommand({});
    const response = await stsClient.send(command);
    state.accountId = response.Account;
  },
  {
    skipWhen: (/** @type {{}} */ state) => !state.deployStack,
  },
);

export const createStack = new ScenarioAction(
  "createStack",
  async (/** @type {{}} */ state) => {
    const stackName = state.getStackName;
    const datastoreName = state.getDatastoreName;
    const accountId = state.accountId;

    const command = new CreateStackCommand({
      StackName: stackName,
      TemplateBody: await fs.readFile(cfnTemplatePath, "utf8"),
      Capabilities: ["CAPABILITY_IAM"],
      Parameters: [
```

```
        {
          ParameterKey: "datastoreName",
          ParameterValue: datastoreName,
        },
        {
          ParameterKey: "userAccountID",
          ParameterValue: accountId,
        },
      ],
    });

    const response = await cfnClient.send(command);
    state.stackId = response.StackId;
  },
  { skipWhen: (/** @type {{}} */ state) => !state.deployStack },
);

export const waitForStackCreation = new ScenarioAction(
  "waitForStackCreation",
  async (/** @type {{}} */ state) => {
    const command = new DescribeStacksCommand({
      StackName: state.stackId,
    });

    await retry({ intervalInMs: 10000, maxRetries: 60 }, async () => {
      const response = await cfnClient.send(command);
      const stack = response.Stacks?.find(
        (s) => s.StackName == state.getStackName,
      );
      if (!stack || stack.StackStatus === "CREATE_IN_PROGRESS") {
        throw new Error("Stack creation is still in progress");
      }
      if (stack.StackStatus === "CREATE_COMPLETE") {
        state.stackOutputs = stack.Outputs?.reduce((acc, output) => {
          acc[output.OutputKey] = output.OutputValue;
          return acc;
        }, {});
      } else {
        throw new Error(
          `Stack creation failed with status: ${stack.StackStatus}`,
        );
      }
    });
  },
```

```
  {
    skipWhen: (/** @type {{}} */ state) => !state.deployStack,
  },
);

export const outputState = new ScenarioOutput(
  "outputState",
  (/** @type {{}} */ state) => {
    /**
     * @type {{ stackOutputs: { DatastoreID: string, BucketName: string, RoleArn:
 string }}}
     */
    const { stackOutputs } = state;
    return `Stack creation completed. Output values:
Datastore ID: ${stackOutputs?.DatastoreID}
Bucket Name: ${stackOutputs?.BucketName}
Role ARN: ${stackOutputs?.RoleArn}
    `;
  },
  { skipWhen: (/** @type {{}} */ state) => !state.deployStack },
);
```

dataset-steps.js-複製DICOM文件。

```
import {
  S3Client,
  CopyObjectCommand,
  ListObjectsV2Command,
} from "@aws-sdk/client-s3";

import {
  ScenarioAction,
  ScenarioInput,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

const s3Client = new S3Client({});

const datasetOptions = [
  {
    name: "CT of chest (2 images)",
    value: "00029d25-fb18-4d42-aaa5-a0897d1ac8f7",
```

```
  },
  {
    name: "CT of pelvis (57 images)",
    value: "00025d30-ef8f-4135-a35a-d83eff264fc1",
  },
  {
    name: "MRI of head (192 images)",
    value: "0002d261-8a5d-4e63-8e2e-0cbfac87b904",
  },
  {
    name: "MRI of breast (92 images)",
    value: "0002dd07-0b7f-4a68-a655-44461ca34096",
  },
];

/**
 * @typedef {{ stackOutputs: {
 *   BucketName: string,
 *   DatastoreID: string,
 *   doCopy: boolean
 * }}} State
 */

export const selectDataset = new ScenarioInput(
  "selectDataset",
  (state) => {
    if (!state.doCopy) {
      process.exit(0);
    }
    return "Select a DICOM dataset to import:";
  },
  {
    type: "select",
    choices: datasetOptions,
  },
);

export const doCopy = new ScenarioInput(
  "doCopy",
  "Do you want to copy images from the public dataset into your bucket?",
  {
    type: "confirm",
  },
);
```

```
export const copyDataset = new ScenarioAction(
  "copyDataset",
  async (/** @type { State } */ state) => {
    const inputBucket = state.stackOutputs.BucketName;
    const inputPrefix = `input/`;
    const selectedDatasetId = state.selectDataset;

    const sourceBucket = "idc-open-data";
    const sourcePrefix = `${selectedDatasetId}`;

    const listObjectsCommand = new ListObjectsV2Command({
      Bucket: sourceBucket,
      Prefix: sourcePrefix,
    });

    const objects = await s3Client.send(listObjectsCommand);

    const copyPromises = objects.Contents.map((object) => {
      const sourceKey = object.Key;
      const destinationKey = `${inputPrefix}${sourceKey
        .split("/")
        .slice(1)
        .join("/")}`;

      const copyCommand = new CopyObjectCommand({
        Bucket: inputBucket,
        CopySource: `/${sourceBucket}/${sourceKey}`,
        Key: destinationKey,
      });

      return s3Client.send(copyCommand);
    });

    const results = await Promise.all(copyPromises);
    state.copiedObjects = results.length;
  },
);

export const outputCopiedObjects = new ScenarioOutput(
  "outputCopiedObjects",
  (state) => `${state.copiedObjects} DICOM files were copied.`,
);
```

import-steps.js-開始匯入資料存放區。

```
import {
  MedicalImagingClient,
  StartDICOMImportJobCommand,
  GetDICOMImportJobCommand,
} from "@aws-sdk/client-medical-imaging";

import {
  ScenarioAction,
  ScenarioOutput,
  ScenarioInput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

/**
 * @typedef {{ stackOutputs: {
 *   BucketName: string,
 *   DatastoreID: string,
 *   RoleArn: string
 * }}} State
 */

export const doImport = new ScenarioInput(
  "doImport",
  "Do you want to import DICOM images into your datastore?",
  {
    type: "confirm",
    default: true,
  },
);

export const startDICOMImport = new ScenarioAction(
  "startDICOMImport",
  async (/** @type {State} */ state) => {
    if (!state.doImport) {
      process.exit(0);
    }
    const medicalImagingClient = new MedicalImagingClient({});
    const inputS3Uri = `s3://${state.stackOutputs.BucketName}/input/`;
    const outputS3Uri = `s3://${state.stackOutputs.BucketName}/output/`;
```

```
    const command = new StartDICOMImportJobCommand({
      dataAccessRoleArn: state.stackOutputs.RoleArn,
      datastoreId: state.stackOutputs.DatastoreID,
      inputS3Uri,
      outputS3Uri,
    });

    const response = await medicalImagingClient.send(command);
    state.importJobId = response.jobId;
  },
);

export const waitForImportJobCompletion = new ScenarioAction(
  "waitForImportJobCompletion",
  async (/** @type {State} */ state) => {
    const medicalImagingClient = new MedicalImagingClient({});
    const command = new GetDICOMImportJobCommand({
      datastoreId: state.stackOutputs.DatastoreID,
      jobId: state.importJobId,
    });

    await retry({ intervalInMs: 10000, maxRetries: 60 }, async () => {
      const response = await medicalImagingClient.send(command);
      const jobStatus = response.jobProperties?.jobStatus;
      if (!jobStatus || jobStatus === "IN_PROGRESS") {
        throw new Error("Import job is still in progress");
      }
      if (jobStatus === "COMPLETED") {
        state.importJobOutputS3Uri = response.jobProperties.outputS3Uri;
      } else {
        throw new Error(`Import job failed with status: ${jobStatus}`);
      }
    });
  },
);

export const outputImportJobStatus = new ScenarioOutput(
  "outputImportJobStatus",
  (state) =>
    `DICOM import job completed. Output location: ${state.importJobOutputS3Uri}`,
);
```

image-set-steps.js-獲取圖像集IDs。

```
import { S3Client, GetObjectCommand } from "@aws-sdk/client-s3";

import {
  ScenarioAction,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

/**
 * @typedef {{ stackOutputs: {
 *   BucketName: string,
 *   DatastoreID: string,
 *   RoleArn: string
 * }, importJobId: string,
 * importJobOutputS3Uri: string,
 * imageSetIds: string[],
 * manifestContent: { jobSummary: { imageSetsSummary: { imageSetId: string }[] } }
 * }} State
 */

const s3Client = new S3Client({});

export const getManifestFile = new ScenarioAction(
  "getManifestFile",
  async (/** @type {State} */ state) => {
    const bucket = state.stackOutputs.BucketName;
    const prefix = `output/${state.stackOutputs.DatastoreID}-DicomImport-
${state.importJobId}/`;
    const key = `${prefix}job-output-manifest.json`;

    const command = new GetObjectCommand({
      Bucket: bucket,
      Key: key,
    });

    const response = await s3Client.send(command);
    const manifestContent = await response.Body.transformToString();
    state.manifestContent = JSON.parse(manifestContent);
  },
);

export const parseManifestFile = new ScenarioAction(
  "parseManifestFile",
```

```
  (/** @type {State} */ state) => {
    const imageSetIds =
      state.manifestContent.jobSummary.imageSetsSummary.reduce(
        (imageSetIds, next) => {
          return { ...imageSetIds, [next.imageSetId]: next.imageSetId };
        },
        {},
      );
    state.imageSetIds = Object.keys(imageSetIds);
  },
);

export const outputImageSetIds = new ScenarioOutput(
  "outputImageSetIds",
  (/** @type {State} */ state) =>
    `The image sets created by this import job are: \n${state.imageSetIds
      .map((id) => `Image set: ${id}`)
      .join("\n")}`,
);
```

image-frame-steps.js-獲取圖像幀IDs。

```
import {
  MedicalImagingClient,
  GetImageSetMetadataCommand,
} from "@aws-sdk/client-medical-imaging";
import { gunzip } from "zlib";
import { promisify } from "util";

import {
  ScenarioAction,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

const gunzipAsync = promisify(gunzip);

/**
 * @typedef {Object} DICOMValueRepresentation
 * @property {string} name
 * @property {string} type
 * @property {string} value
 */
```

```
/**
 * @typedef {Object} ImageFrameInformation
 * @property {string} ID
 * @property {Array<{ Checksum: number, Height: number, Width: number }>}
 PixelDataChecksumFromBaseToFullResolution
 * @property {number} MinPixelValue
 * @property {number} MaxPixelValue
 * @property {number} FrameSizeInBytes
 */

/**
 * @typedef {Object} DICOMMetadata
 * @property {Object} DICOM
 * @property {DICOMValueRepresentation[]} DICOMVRs
 * @property {ImageFrameInformation[]} ImageFrames
 */

/**
 * @typedef {Object} Series
 * @property {{ [key: string]: DICOMMetadata }} Instances
 */

/**
 * @typedef {Object} Study
 * @property {Object} DICOM
 * @property {Series[]} Series
 */

/**
 * @typedef {Object} Patient
 * @property {Object} DICOM
 */

/**
 * @typedef {{
 *  SchemaVersion: string,
 *  DatastoreID: string,
 *  ImageSetID: string,
 *  Patient: Patient,
 *  Study: Study
 * }} ImageSetMetadata
 */
```

```
/**
 * @typedef {{ stackOutputs: {
 *   BucketName: string,
 *   DatastoreID: string,
 *   RoleArn: string
 * }, imageSetIds: string[] }} State
 */

const medicalImagingClient = new MedicalImagingClient({});

export const getImageSetMetadata = new ScenarioAction(
  "getImageSetMetadata",
  async (/** @type {State} */ state) => {
    const outputMetadata = [];

    for (const imageSetId of state.imageSetIds) {
      const command = new GetImageSetMetadataCommand({
        datastoreId: state.stackOutputs.DatastoreID,
        imageSetId,
      });

      const response = await medicalImagingClient.send(command);
      const compressedMetadataBlob =
        await response.imageSetMetadataBlob.transformToByteArray();
      const decompressedMetadata = await gunzipAsync(compressedMetadataBlob);
      const imageSetMetadata = JSON.parse(decompressedMetadata.toString());

      outputMetadata.push(imageSetMetadata);
    }

    state.imageSetMetadata = outputMetadata;
  },
);

export const outputImageFrameIds = new ScenarioOutput(
  "outputImageFrameIds",
  (/** @type {State & { imageSetMetadata: ImageSetMetadata[] }} */ state) => {
    let output = "";

    for (const metadata of state.imageSetMetadata) {
      const imageSetId = metadata.ImageSetID;
      /** @type {DICOMMetadata[]} */
      const instances = Object.values(metadata.Study.Series).flatMap(
        (series) => {
```

```
          return Object.values(series.Instances);
        },
      );
      const imageFrameIds = instances.flatMap((instance) =>
        instance.ImageFrames.map((frame) => frame.ID),
      );

      output += `Image set ID: ${imageSetId}\nImage frame IDs:\n
${imageFrameIds.join(
        "\n",
      )}\n\n`;
    }

    return output;
  },
  { slow: false },
);
```

verify-steps.js-驗證圖像幀。使用「AWS HealthImaging 像素資料驗證」程式庫進行驗證。

```
import { spawn } from "node:child_process";

import {
  ScenarioAction,
  ScenarioInput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

/**
 * @typedef {Object} DICOMValueRepresentation
 * @property {string} name
 * @property {string} type
 * @property {string} value
 */

/**
 * @typedef {Object} ImageFrameInformation
 * @property {string} ID
 * @property {Array<{ Checksum: number, Height: number, Width: number }>}
 PixelDataChecksumFromBaseToFullResolution
 * @property {number} MinPixelValue
 * @property {number} MaxPixelValue
 * @property {number} FrameSizeInBytes
```

```
 */

/**
 * @typedef {Object} DICOMMetadata
 * @property {Object} DICOM
 * @property {DICOMValueRepresentation[]} DICOMVRs
 * @property {ImageFrameInformation[]} ImageFrames
 */

/**
 * @typedef {Object} Series
 * @property {{ [key: string]: DICOMMetadata }} Instances
 */

/**
 * @typedef {Object} Study
 * @property {Object} DICOM
 * @property {Series[]} Series
 */

/**
 * @typedef {Object} Patient
 * @property {Object} DICOM
 */

/**
 * @typedef {{
 *  SchemaVersion: string,
 *  DatastoreID: string,
 *  ImageSetID: string,
 *  Patient: Patient,
 *  Study: Study
 * }} ImageSetMetadata
 */

/**
 * @typedef {{ stackOutputs: {
 *   BucketName: string,
 *   DatastoreID: string,
 *   RoleArn: string
 * }, imageSetMetadata: ImageSetMetadata[] }} State
 */

export const doVerify = new ScenarioInput(
```

```
  "doVerify",
  "Do you want to verify the imported images?",
  {
    type: "confirm",
    default: true,
  },
);

export const decodeAndVerifyImages = new ScenarioAction(
  "decodeAndVerifyImages",
  async (/** @type {State} */ state) => {
    if (!state.doVerify) {
      process.exit(0);
    }
    const verificationTool = "./pixel-data-verification/index.js";

    for (const metadata of state.imageSetMetadata) {
      const datastoreId = state.stackOutputs.DatastoreID;
      const imageSetId = metadata.ImageSetID;

      for (const [seriesInstanceUid, series] of Object.entries(
        metadata.Study.Series,
      )) {
        for (const [sopInstanceUid, _] of Object.entries(series.Instances)) {
          console.log(
            `Verifying image set ${imageSetId} with series ${seriesInstanceUid} and
 sop ${sopInstanceUid}`,
          );
          const child = spawn(
            "node",
            [
              verificationTool,
              datastoreId,
              imageSetId,
              seriesInstanceUid,
              sopInstanceUid,
            ],
            { stdio: "inherit" },
          );

          await new Promise((resolve, reject) => {
            child.on("exit", (code) => {
              if (code === 0) {
                resolve();
```

```
              } else {
                reject(
                  new Error(
                    `Verification tool exited with code ${code} for image set
 ${imageSetId}`,
                  ),
                );
              }
            });
          });
        }
      }
    }
  },
);
```

clean-up-steps.js-摧毀資源。

```
import {
  CloudFormationClient,
  DeleteStackCommand,
} from "@aws-sdk/client-cloudformation";
import {
  MedicalImagingClient,
  DeleteImageSetCommand,
} from "@aws-sdk/client-medical-imaging";

import {
  ScenarioAction,
  ScenarioInput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

/**
 * @typedef {Object} DICOMValueRepresentation
 * @property {string} name
 * @property {string} type
 * @property {string} value
 */

/**
 * @typedef {Object} ImageFrameInformation
 * @property {string} ID
```

```
 * @property {Array<{ Checksum: number, Height: number, Width: number }>}
 PixelDataChecksumFromBaseToFullResolution
 * @property {number} MinPixelValue
 * @property {number} MaxPixelValue
 * @property {number} FrameSizeInBytes
 */

/**
 * @typedef {Object} DICOMMetadata
 * @property {Object} DICOM
 * @property {DICOMValueRepresentation[]} DICOMVRs
 * @property {ImageFrameInformation[]} ImageFrames
 */

/**
 * @typedef {Object} Series
 * @property {{ [key: string]: DICOMMetadata }} Instances
 */

/**
 * @typedef {Object} Study
 * @property {Object} DICOM
 * @property {Series[]} Series
 */

/**
 * @typedef {Object} Patient
 * @property {Object} DICOM
 */

/**
 * @typedef {{
 *  SchemaVersion: string,
 *  DatastoreID: string,
 *  ImageSetID: string,
 *  Patient: Patient,
 *  Study: Study
 * }} ImageSetMetadata
 */

/**
 * @typedef {{ stackOutputs: {
 *   BucketName: string,
 *   DatastoreID: string,
```

```
 *   RoleArn: string
 * }, imageSetMetadata: ImageSetMetadata[] }} State
 */

const cfnClient = new CloudFormationClient({});
const medicalImagingClient = new MedicalImagingClient({});

export const confirmCleanup = new ScenarioInput(
  "confirmCleanup",
  "Do you want to delete the created resources?",
  { type: "confirm" },
);

export const deleteImageSets = new ScenarioAction(
  "deleteImageSets",
  async (/** @type {State} */ state) => {
    const datastoreId = state.stackOutputs.DatastoreID;

    for (const metadata of state.imageSetMetadata) {
      const command = new DeleteImageSetCommand({
        datastoreId,
        imageSetId: metadata.ImageSetID,
      });

      try {
        await medicalImagingClient.send(command);
        console.log(`Successfully deleted image set ${metadata.ImageSetID}`);
      } catch (e) {
        if (e instanceof Error) {
          if (e.name === "ConflictException") {
            console.log(`Image set ${metadata.ImageSetID} already deleted`);
          }
        }
      }
    }
  },
  {
    skipWhen: (/** @type {{}} */ state) => !state.confirmCleanup,
  },
);

export const deleteStack = new ScenarioAction(
  "deleteStack",
  async (/** @type {State} */ state) => {
```

```
    const stackName = state.getStackName;

    const command = new DeleteStackCommand({
      StackName: stackName,
    });

    await cfnClient.send(command);
    console.log(`Stack ${stackName} deletion initiated`);
  },
  {
    skipWhen: (/** @type {{}} */ state) => !state.confirmCleanup,
  },
);
```

- 如需詳API細資訊，請參閱「AWS SDK for JavaScript API參考」中的下列主題。
  - DeleteImageSet
  - G etDICOMImport Job 機會
  - GetImageFrame
  - GetImageSetMetadata
  - SearchImageSets
  - S tartDICOMImport Job 機會

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

為資料倉庫加標籤

下列程式碼範例示範如何標記 HealthImaging 資料倉庫。

SDK對於 JavaScript（3）

為資料倉庫加上標籤。

```
  try {
    const datastoreArn =
```

```
      "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012";
    const tags = {
      Deployment: "Development",
    };
    await tagResource(datastoreArn, tags);
  } catch (e) {
    console.log(e);
  }
```

用於標記資源的實用程序功能。

```
import { TagResourceCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} resourceArn - The Amazon Resource Name (ARN) for the data store
 or image set.
 * @param {Record<string,string>} tags - The tags to add to the resource as JSON.
 *                     - For example: {"Deployment" : "Development"}
 */
export const tagResource = async (
  resourceArn = "arn:aws:medical-imaging:us-east-1:xxxxxx:datastore/xxxxx/imageset/
xxx",
  tags = {}
) => {
  const response = await medicalImagingClient.send(
    new TagResourceCommand({ resourceArn: resourceArn, tags: tags })
  );
  console.log(response);
  // {
  //     '$metadata': {
  //        httpStatusCode: 204,
  //         requestId: '8a6de9a3-ec8e-47ef-8643-473518b19d45',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  //    }
  // }

  return response;
```

```
};
```

列示資料倉庫的標籤。

```
  try {
    const datastoreArn =
      "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012";
    const { tags } = await listTagsForResource(datastoreArn);
    console.log(tags);
  } catch (e) {
    console.log(e);
  }
```

列出資源標籤的公用程式功能。

```
import { ListTagsForResourceCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} resourceArn - The Amazon Resource Name (ARN) for the data store
 or image set.
 */
export const listTagsForResource = async (
  resourceArn = "arn:aws:medical-imaging:us-east-1:abc:datastore/def/imageset/ghi"
) => {
  const response = await medicalImagingClient.send(
    new ListTagsForResourceCommand({ resourceArn: resourceArn })
  );
  console.log(response);
  // {
  //     '$metadata': {
  //         httpStatusCode: 200,
  //         requestId: '008fc6d3-abec-4870-a155-20fa3631e645',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  //     },
  //     tags: { Deployment: 'Development' }
  // }
```

```
  return response;
};
```

取消標籤資料倉庫。

```
  try {
    const datastoreArn =
      "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012";
    const keys = ["Deployment"];
    await untagResource(datastoreArn, keys);
  } catch (e) {
    console.log(e);
  }
```

取消標記資源的公用程式功能。

```
import { UntagResourceCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} resourceArn - The Amazon Resource Name (ARN) for the data store
 or image set.
 * @param {string[]} tagKeys - The keys of the tags to remove.
 */
export const untagResource = async (
  resourceArn = "arn:aws:medical-imaging:us-east-1:xxxxxx:datastore/xxxxx/imageset/
xxx",
  tagKeys = []
) => {
  const response = await medicalImagingClient.send(
    new UntagResourceCommand({ resourceArn: resourceArn, tagKeys: tagKeys })
  );
  console.log(response);
  // {
  //     '$metadata': {
  //        httpStatusCode: 204,
  //         requestId: '8a6de9a3-ec8e-47ef-8643-473518b19d45',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
```

```
  //         attempts: 1,
  //         totalRetryDelay: 0
  //     }
  // }

  return response;
};
```

- 如需詳API細資訊，請參閱「AWS SDK for JavaScript API參考」中的下列主題。
  - ListTagsForResource
  - TagResource
  - UntagResource

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

標記影像集

下列程式碼範例會示範如何標記 HealthImaging 影像集。

SDK對於 JavaScript（3）

標記影像集。

```
  try {
    const imagesetArn =
      "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012/
imageset/12345678901234567890123456789012";
    const tags = {
      Deployment: "Development",
    };
    await tagResource(imagesetArn, tags);
  } catch (e) {
    console.log(e);
  }
```

用於標記資源的實用程序功能。

```
import { TagResourceCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} resourceArn - The Amazon Resource Name (ARN) for the data store
 or image set.
 * @param {Record<string,string>} tags - The tags to add to the resource as JSON.
 *                      - For example: {"Deployment" : "Development"}
 */
export const tagResource = async (
  resourceArn = "arn:aws:medical-imaging:us-east-1:xxxxxx:datastore/xxxxx/imageset/
xxx",
  tags = {}
) => {
  const response = await medicalImagingClient.send(
    new TagResourceCommand({ resourceArn: resourceArn, tags: tags })
  );
  console.log(response);
  // {
  //     '$metadata': {
  //        httpStatusCode: 204,
  //         requestId: '8a6de9a3-ec8e-47ef-8643-473518b19d45',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  //    }
  // }

  return response;
};
```

列出影像集的標籤。

```
  try {
    const imagesetArn =
      "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012/
imageset/12345678901234567890123456789012";
    const { tags } = await listTagsForResource(imagesetArn);
```

```
    console.log(tags);
  } catch (e) {
    console.log(e);
  }
```

列出資源標籤的公用程式功能。

```
import { ListTagsForResourceCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} resourceArn - The Amazon Resource Name (ARN) for the data store
 or image set.
 */
export const listTagsForResource = async (
  resourceArn = "arn:aws:medical-imaging:us-east-1:abc:datastore/def/imageset/ghi"
) => {
  const response = await medicalImagingClient.send(
    new ListTagsForResourceCommand({ resourceArn: resourceArn })
  );
  console.log(response);
  // {
  //     '$metadata': {
  //         httpStatusCode: 200,
  //         requestId: '008fc6d3-abec-4870-a155-20fa3631e645',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  //     },
  //     tags: { Deployment: 'Development' }
  // }

  return response;
};
```

取消標記影像集。

```
  try {
    const imagesetArn =
```

```
      "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012/
imageset/12345678901234567890123456789012";
    const keys = ["Deployment"];
    await untagResource(imagesetArn, keys);
  } catch (e) {
    console.log(e);
  }
```

取消標記資源的公用程式功能。

```
import { UntagResourceCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} resourceArn - The Amazon Resource Name (ARN) for the data store
 or image set.
 * @param {string[]} tagKeys - The keys of the tags to remove.
 */
export const untagResource = async (
  resourceArn = "arn:aws:medical-imaging:us-east-1:xxxxxx:datastore/xxxxx/imageset/
xxx",
  tagKeys = []
) => {
  const response = await medicalImagingClient.send(
    new UntagResourceCommand({ resourceArn: resourceArn, tagKeys: tagKeys })
  );
  console.log(response);
  // {
  //     '$metadata': {
  //        httpStatusCode: 204,
  //         requestId: '8a6de9a3-ec8e-47ef-8643-473518b19d45',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  //    }
  // }

  return response;
};
```

- 如需詳API細資訊，請參閱「AWS SDK for JavaScript API參考」中的下列主題。
  - ListTagsForResource
  - TagResource
  - UntagResource

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

IAM使用 JavaScript （ v3 ）SDK的示例

下列程式碼範例會示範如何使用 AWS SDK for JavaScript (v3) 搭配使用來執行動作及實作常見案例 IAM。

Actions 是大型程式的程式碼摘錄，必須在內容中執行。雖然動作會顯示如何呼叫個別服務函數，但您可以在其相關案例中查看內容中的動作。

案例是程式碼範例，向您展示如何透過呼叫服務中的多個函式或與其他函式結合來完成特定工作 AWS 服務。

每個範例都包含完整原始程式碼的連結，您可以在其中找到如何在內容中設定和執行程式碼的指示。

開始使用

Hello IAM

下列程式碼範例會示範如何開始使用IAM。

SDK對於 JavaScript （ 3 ）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import { IAMClient, paginateListPolicies } from "@aws-sdk/client-iam";
```

```
const client = new IAMClient({});

export const listLocalPolicies = async () => {
  /**
   * In v3, the clients expose paginateOperationName APIs that are written using
 async generators so that you can use async iterators in a for await..of loop.
   * https://docs.aws.amazon.com/AWSJavaScriptSDK/v3/latest/index.html#paginators
   */
  const paginator = paginateListPolicies(
    { client, pageSize: 10 },
    // List only customer managed policies.
    { Scope: "Local" },
  );

  console.log("IAM policies defined in your account:");
  let policyCount = 0;
  for await (const page of paginator) {
    if (page.Policies) {
      page.Policies.forEach((p) => {
        console.log(`${p.PolicyName}`);
        policyCount++;
      });
    }
  }
  console.log(`Found ${policyCount} policies.`);
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考ListPolicies中的。

主題

- 動作
- 案例

動作

`AttachRolePolicy`

下列程式碼範例會示範如何使用`AttachRolePolicy`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

連接政策。

```
import { AttachRolePolicyCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} policyArn
 * @param {string} roleName
 */
export const attachRolePolicy = (policyArn, roleName) => {
  const command = new AttachRolePolicyCommand({
    PolicyArn: policyArn,
    RoleName: roleName,
  });

  return client.send(command);
};
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考AttachRolePolicy中的。

SDK對於 JavaScript（第 2 個）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
// Load the AWS SDK for Node.js
```

```
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var paramsRoleList = {
  RoleName: process.argv[2],
};

iam.listAttachedRolePolicies(paramsRoleList, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    var myRolePolicies = data.AttachedPolicies;
    myRolePolicies.forEach(function (val, index, array) {
      if (myRolePolicies[index].PolicyName === "AmazonDynamoDBFullAccess") {
        console.log(
          "AmazonDynamoDBFullAccess is already attached to this role."
        );
        process.exit();
      }
    });
    var params = {
      PolicyArn: "arn:aws:iam::aws:policy/AmazonDynamoDBFullAccess",
      RoleName: process.argv[2],
    };
    iam.attachRolePolicy(params, function (err, data) {
      if (err) {
        console.log("Unable to attach policy to role", err);
      } else {
        console.log("Role attached successfully");
      }
    });
  }
});
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考AttachRolePolicy中的。

`CreateAccessKey`

下列程式碼範例會示範如何使用`CreateAccessKey`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

建立存取金鑰。

```
import { CreateAccessKeyCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} userName
 */
export const createAccessKey = (userName) => {
  const command = new CreateAccessKeyCommand({ UserName: userName });
  return client.send(command);
};
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考CreateAccessKey中的。

SDK對於 JavaScript（第 2 個）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
// Load the AWS SDK for Node.js
```

```
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

iam.createAccessKey({ UserName: "IAM_USER_NAME" }, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.AccessKey);
  }
});
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考CreateAccessKey中的。

`CreateAccountAlias`

下列程式碼範例會示範如何使用`CreateAccountAlias`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

建立帳戶別名。

```
import { CreateAccountAliasCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} alias - A unique name for the account alias.
 * @returns
```

```
 */
export const createAccountAlias = (alias) => {
  const command = new CreateAccountAliasCommand({
    AccountAlias: alias,
  });

  return client.send(command);
};
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考CreateAccountAlias中的。

SDK對於 JavaScript（第 2 個）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

iam.createAccountAlias({ AccountAlias: process.argv[2] }, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考CreateAccountAlias中的。

CreateGroup

下列程式碼範例會示範如何使用CreateGroup。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import { CreateGroupCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} groupName
 */
export const createGroup = async (groupName) => {
  const command = new CreateGroupCommand({ GroupName: groupName });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考CreateGroup中的。

CreateInstanceProfile

下列程式碼範例會示範如何使用CreateInstanceProfile。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
const { InstanceProfile } = await iamClient.send(
  new CreateInstanceProfileCommand({
    InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
  }),
);
await waitUntilInstanceProfileExists(
  { client: iamClient },
  { InstanceProfileName: NAMES.ssmOnlyInstanceProfileName },
);
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考CreateInstanceProfile中的。

`CreatePolicy`

下列程式碼範例會示範如何使用`CreatePolicy`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

建立政策。

```
import { CreatePolicyCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
```

```
 * @param {string} policyName
 */
export const createPolicy = (policyName) => {
  const command = new CreatePolicyCommand({
    PolicyDocument: JSON.stringify({
      Version: "2012-10-17",
      Statement: [
        {
          Effect: "Allow",
          Action: "*",
          Resource: "*",
        },
      ],
    }),
    PolicyName: policyName,
  });

  return client.send(command);
};
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考CreatePolicy中的。

SDK對於 JavaScript（第 2 個）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var myManagedPolicy = {
  Version: "2012-10-17",
```

```
  Statement: [
    {
      Effect: "Allow",
      Action: "logs:CreateLogGroup",
      Resource: "RESOURCE_ARN",
    },
    {
      Effect: "Allow",
      Action: [
        "dynamodb:DeleteItem",
        "dynamodb:GetItem",
        "dynamodb:PutItem",
        "dynamodb:Scan",
        "dynamodb:UpdateItem",
      ],
      Resource: "RESOURCE_ARN",
    },
  ],
};

var params = {
  PolicyDocument: JSON.stringify(myManagedPolicy),
  PolicyName: "myDynamoDBPolicy",
};

iam.createPolicy(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考CreatePolicy中的。

`CreateRole`

下列程式碼範例會示範如何使用`CreateRole`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

建立角色。

```
import { CreateRoleCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} roleName
 */
export const createRole = (roleName) => {
  const command = new CreateRoleCommand({
    AssumeRolePolicyDocument: JSON.stringify({
      Version: "2012-10-17",
      Statement: [
        {
          Effect: "Allow",
          Principal: {
            Service: "lambda.amazonaws.com",
          },
          Action: "sts:AssumeRole",
        },
      ],
    }),
    RoleName: roleName,
  });

  return client.send(command);
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考CreateRole中的。

`CreateSAMLProvider`

下列程式碼範例會示範如何使用`CreateSAMLProvider`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import { CreateSAMLProviderCommand, IAMClient } from "@aws-sdk/client-iam";
import { readFileSync } from "fs";
import * as path from "path";
import { dirnameFromMetaUrl } from "@aws-doc-sdk-examples/lib/utils/util-fs.js";

const client = new IAMClient({});

/**
 * This sample document was generated using Auth0.
 * For more information on generating this document,
 see https://docs.aws.amazon.com/IAM/latest/UserGuide/
id_roles_providers_create_saml.html#samlstep1.
 */
const sampleMetadataDocument = readFileSync(
  path.join(
    dirnameFromMetaUrl(import.meta.url),
    "../../../../resources/sample_files/sample_saml_metadata.xml",
  ),
);

/**
 *
 * @param {*} providerName
 * @returns
 */
export const createSAMLProvider = async (providerName) => {
  const command = new CreateSAMLProviderCommand({
    Name: providerName,
    SAMLMetadataDocument: sampleMetadataDocument.toString(),
  });
```

```
  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參照reateSAMLProvider中的 C。

`CreateServiceLinkedRole`

下列程式碼範例會示範如何使用`CreateServiceLinkedRole`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

建立服務連結角色。

```
import {
  CreateServiceLinkedRoleCommand,
  GetRoleCommand,
  IAMClient,
} from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} serviceName
 */
export const createServiceLinkedRole = async (serviceName) => {
  const command = new CreateServiceLinkedRoleCommand({
    // For a list of AWS services that support service-linked roles,
    // see https://docs.aws.amazon.com/IAM/latest/UserGuide/reference_aws-services-
that-work-with-iam.html.
    //
    // For a list of AWS service endpoints, see https://docs.aws.amazon.com/general/
latest/gr/aws-service-information.html.
```

```
    AWSServiceName: serviceName,
  });
  try {
    const response = await client.send(command);
    console.log(response);
    return response;
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "InvalidInputException" &&
      caught.message.includes(
        "Service role name AWSServiceRoleForElasticBeanstalk has been taken in this
 account",
      )
    ) {
      console.warn(caught.message);
      return client.send(
        new GetRoleCommand({ RoleName: "AWSServiceRoleForElasticBeanstalk" }),
      );
    } else {
      throw caught;
    }
  }
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考CreateServiceLinkedRole中的。

`CreateUser`

下列程式碼範例會示範如何使用`CreateUser`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

建立使用者。

```
import { CreateUserCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} name
 */
export const createUser = (name) => {
  const command = new CreateUserCommand({ UserName: name });
  return client.send(command);
};
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考CreateUser中的。

SDK對於 JavaScript（第 2 個）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var params = {
  UserName: process.argv[2],
};

iam.getUser(params, function (err, data) {
  if (err && err.code === "NoSuchEntity") {
    iam.createUser(params, function (err, data) {
      if (err) {
        console.log("Error", err);
```

```
      } else {
        console.log("Success", data);
      }
    });
  } else {
    console.log(
      "User " + process.argv[2] + " already exists",
      data.User.UserId
    );
  }
});
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考CreateUser中的。

`DeleteAccessKey`

下列程式碼範例會示範如何使用`DeleteAccessKey`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

刪除存取金鑰。

```
import { DeleteAccessKeyCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} userName
 * @param {string} accessKeyId
 */
export const deleteAccessKey = (userName, accessKeyId) => {
  const command = new DeleteAccessKeyCommand({
    AccessKeyId: accessKeyId,
```

```
    UserName: userName,
  });

  return client.send(command);
};
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考DeleteAccessKey中的。

SDK對於 JavaScript（第 2 個）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var params = {
  AccessKeyId: "ACCESS_KEY_ID",
  UserName: "USER_NAME",
};

iam.deleteAccessKey(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考DeleteAccessKey中的。

`DeleteAccountAlias`

下列程式碼範例會示範如何使用`DeleteAccountAlias`。

SDK對於 JavaScript （ 3 ）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

刪除帳戶別名。

```
import { DeleteAccountAliasCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} alias
 */
export const deleteAccountAlias = (alias) => {
  const command = new DeleteAccountAliasCommand({ AccountAlias: alias });

  return client.send(command);
};
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳細API資訊，請參閱AWS SDK for JavaScript API參考DeleteAccountAlias中的。

SDK對於 JavaScript （ 第 2 個 ）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
// Load the AWS SDK for Node.js
```

```
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

iam.deleteAccountAlias({ AccountAlias: process.argv[2] }, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考DeleteAccountAlias中的。

`DeleteGroup`

下列程式碼範例會示範如何使用`DeleteGroup`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import { DeleteGroupCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} groupName
 */
export const deleteGroup = async (groupName) => {
  const command = new DeleteGroupCommand({
```

```
    GroupName: groupName,
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考DeleteGroup中的。

`DeleteInstanceProfile`

下列程式碼範例會示範如何使用`DeleteInstanceProfile`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    const client = new IAMClient({});
    await client.send(
      new DeleteInstanceProfileCommand({
        InstanceProfileName: NAMES.instanceProfileName,
      }),
    );
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考DeleteInstanceProfile中的。

`DeletePolicy`

下列程式碼範例會示範如何使用`DeletePolicy`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

刪除策略。

```
import { DeletePolicyCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} policyArn
 */
export const deletePolicy = (policyArn) => {
  const command = new DeletePolicyCommand({ PolicyArn: policyArn });
  return client.send(command);
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考DeletePolicy中的。

`DeleteRole`

下列程式碼範例會示範如何使用`DeleteRole`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

刪除角色。

```
import { DeleteRoleCommand, IAMClient } from "@aws-sdk/client-iam";
```

```
const client = new IAMClient({});

/**
 *
 * @param {string} roleName
 */
export const deleteRole = (roleName) => {
  const command = new DeleteRoleCommand({ RoleName: roleName });
  return client.send(command);
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考DeleteRole中的。

`DeleteRolePolicy`

下列程式碼範例會示範如何使用`DeleteRolePolicy`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import { DeleteRolePolicyCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} roleName
 * @param {string} policyName
 */
export const deleteRolePolicy = (roleName, policyName) => {
  const command = new DeleteRolePolicyCommand({
    RoleName: roleName,
    PolicyName: policyName,
  });
```

```
  return client.send(command);
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考DeleteRolePolicy中的。

`DeleteSAMLProvider`

下列程式碼範例會示範如何使用`DeleteSAMLProvider`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import { DeleteSAMLProviderCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} providerArn
 * @returns
 */
export const deleteSAMLProvider = async (providerArn) => {
  const command = new DeleteSAMLProviderCommand({
    SAMLProviderArn: providerArn,
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考eleteSAMLProvider中的 D。

`DeleteServerCertificate`

下列程式碼範例會示範如何使用`DeleteServerCertificate`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

刪除伺服器憑證。

```
import { DeleteServerCertificateCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} certName
 */
export const deleteServerCertificate = (certName) => {
  const command = new DeleteServerCertificateCommand({
    ServerCertificateName: certName,
  });

  return client.send(command);
};
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳細API資訊，請參閱AWS SDK for JavaScript API參考DeleteServerCertificate中的。

SDK對於 JavaScript（第 2 個）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

iam.deleteServerCertificate(
  { ServerCertificateName: "CERTIFICATE_NAME" },
  function (err, data) {
    if (err) {
      console.log("Error", err);
    } else {
      console.log("Success", data);
    }
  }
);
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考DeleteServerCertificate中的。

`DeleteServiceLinkedRole`

下列程式碼範例會示範如何使用`DeleteServiceLinkedRole`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import { DeleteServiceLinkedRoleCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
```

```
 *
 * @param {string} roleName
 */
export const deleteServiceLinkedRole = (roleName) => {
  const command = new DeleteServiceLinkedRoleCommand({ RoleName: roleName });
  return client.send(command);
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考DeleteServiceLinkedRole中的。

`DeleteUser`

下列程式碼範例會示範如何使用`DeleteUser`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

刪除使用者。

```
import { DeleteUserCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} name
 */
export const deleteUser = (name) => {
  const command = new DeleteUserCommand({ UserName: name });
  return client.send(command);
};
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考DeleteUser中的。

SDK對於 JavaScript （ 第 2 個 ）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var params = {
  UserName: process.argv[2],
};

iam.getUser(params, function (err, data) {
  if (err && err.code === "NoSuchEntity") {
    console.log("User " + process.argv[2] + " does not exist.");
  } else {
    iam.deleteUser(params, function (err, data) {
      if (err) {
        console.log("Error", err);
      } else {
        console.log("Success", data);
      }
    });
  }
});
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考DeleteUser中的。

`DetachRolePolicy`

下列程式碼範例會示範如何使用`DetachRolePolicy`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

分離政策。

```
import { DetachRolePolicyCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} policyArn
 * @param {string} roleName
 */
export const detachRolePolicy = (policyArn, roleName) => {
  const command = new DetachRolePolicyCommand({
    PolicyArn: policyArn,
    RoleName: roleName,
  });

  return client.send(command);
};
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考DetachRolePolicy中的。

SDK對於 JavaScript（第 2 個）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
// Load the AWS SDK for Node.js
```

```
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var paramsRoleList = {
  RoleName: process.argv[2],
};

iam.listAttachedRolePolicies(paramsRoleList, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    var myRolePolicies = data.AttachedPolicies;
    myRolePolicies.forEach(function (val, index, array) {
      if (myRolePolicies[index].PolicyName === "AmazonDynamoDBFullAccess") {
        var params = {
          PolicyArn: "arn:aws:iam::aws:policy/AmazonDynamoDBFullAccess",
          RoleName: process.argv[2],
        };
        iam.detachRolePolicy(params, function (err, data) {
          if (err) {
            console.log("Unable to detach policy from role", err);
          } else {
            console.log("Policy detached from role successfully");
            process.exit();
          }
        });
      }
    });
  }
});
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考DetachRolePolicy中的。

GetAccessKeyLastUsed

下列程式碼範例會示範如何使用GetAccessKeyLastUsed。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

獲取存取金鑰。

```
import { GetAccessKeyLastUsedCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} accessKeyId
 */
export const getAccessKeyLastUsed = async (accessKeyId) => {
  const command = new GetAccessKeyLastUsedCommand({
    AccessKeyId: accessKeyId,
  });

  const response = await client.send(command);

  if (response.AccessKeyLastUsed?.LastUsedDate) {
    console.log(`
    ${accessKeyId} was last used by ${response.UserName} via
    the ${response.AccessKeyLastUsed.ServiceName} service on
    ${response.AccessKeyLastUsed.LastUsedDate.toISOString()}
    `);
  }

  return response;
};
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳細API資訊，請參閱AWS SDK for JavaScript API參考GetAccessKeyLastUsed中的。

SDK對於 JavaScript （ 第 2 個 ）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

iam.getAccessKeyLastUsed(
  { AccessKeyId: "ACCESS_KEY_ID" },
  function (err, data) {
    if (err) {
      console.log("Error", err);
    } else {
      console.log("Success", data.AccessKeyLastUsed);
    }
  }
);
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考GetAccessKeyLastUsed中的。

`GetAccountPasswordPolicy`

下列程式碼範例會示範如何使用`GetAccountPasswordPolicy`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

取得帳戶密碼政策。

```
import {
  GetAccountPasswordPolicyCommand,
  IAMClient,
} from "@aws-sdk/client-iam";

const client = new IAMClient({});

export const getAccountPasswordPolicy = async () => {
  const command = new GetAccountPasswordPolicyCommand({});

  const response = await client.send(command);
  console.log(response.PasswordPolicy);
  return response;
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考GetAccountPasswordPolicy中的。

`GetPolicy`

下列程式碼範例會示範如何使用`GetPolicy`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

取得政策。

```
import { GetPolicyCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} policyArn
 */
export const getPolicy = (policyArn) => {
  const command = new GetPolicyCommand({
    PolicyArn: policyArn,
  });

  return client.send(command);
};
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考GetPolicy中的。

SDK對於 JavaScript （ 第 2 個 ）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var params = {
  PolicyArn: "arn:aws:iam::aws:policy/AWSLambdaExecute",
};

iam.getPolicy(params, function (err, data) {
  if (err) {
```

```
    console.log("Error", err);
  } else {
    console.log("Success", data.Policy.Description);
  }
});
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考GetPolicy中的。

`GetRole`

下列程式碼範例會示範如何使用`GetRole`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

取得角色。

```
import { GetRoleCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} roleName
 */
export const getRole = (roleName) => {
  const command = new GetRoleCommand({
    RoleName: roleName,
  });

  return client.send(command);
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考GetRole中的。

`GetServerCertificate`

下列程式碼範例會示範如何使用`GetServerCertificate`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

獲取伺服器憑證。

```
import { GetServerCertificateCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} certName
 * @returns
 */
export const getServerCertificate = async (certName) => {
  const command = new GetServerCertificateCommand({
    ServerCertificateName: certName,
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考GetServerCertificate中的。

SDK對於 JavaScript （ 第 2 個 ）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

iam.getServerCertificate(
  { ServerCertificateName: "CERTIFICATE_NAME" },
  function (err, data) {
    if (err) {
      console.log("Error", err);
    } else {
      console.log("Success", data);
    }
  }
);
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考GetServerCertificate中的。

`GetServiceLinkedRoleDeletionStatus`

下列程式碼範例會示範如何使用`GetServiceLinkedRoleDeletionStatus`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import {
  GetServiceLinkedRoleDeletionStatusCommand,
  IAMClient,
} from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} deletionTaskId
 */
export const getServiceLinkedRoleDeletionStatus = (deletionTaskId) => {
  const command = new GetServiceLinkedRoleDeletionStatusCommand({
    DeletionTaskId: deletionTaskId,
  });

  return client.send(command);
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考GetServiceLinkedRoleDeletionStatus中的。

`ListAccessKeys`

下列程式碼範例會示範如何使用`ListAccessKeys`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

列出存取金鑰。

```
import { ListAccessKeysCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 * A generator function that handles paginated results.
 * The AWS SDK for JavaScript (v3) provides {@link https://docs.aws.amazon.com/
AWSJavaScriptSDK/v3/latest/index.html#paginators | paginator} functions to simplify
 this.
 *
 * @param {string} userName
 */
export async function* listAccessKeys(userName) {
  const command = new ListAccessKeysCommand({
    MaxItems: 5,
    UserName: userName,
  });

  /**
   * @type {import("@aws-sdk/client-iam").ListAccessKeysCommandOutput | undefined}
   */
  let response = await client.send(command);

  while (response?.AccessKeyMetadata?.length) {
    for (const key of response.AccessKeyMetadata) {
      yield key;
    }

    if (response.IsTruncated) {
      response = await client.send(
        new ListAccessKeysCommand({
          Marker: response.Marker,
        }),
```

```
      );
    } else {
      break;
    }
  }
}
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考ListAccessKeys中的。

SDK對於 JavaScript （ 第 2 個 ）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var params = {
  MaxItems: 5,
  UserName: "IAM_USER_NAME",
};

iam.listAccessKeys(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考ListAccessKeys中的。

`ListAccountAliases`

下列程式碼範例會示範如何使用`ListAccountAliases`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

列出帳戶別名。

```
import { ListAccountAliasesCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 * A generator function that handles paginated results.
 * The AWS SDK for JavaScript (v3) provides {@link https://docs.aws.amazon.com/
AWSJavaScriptSDK/v3/latest/index.html#paginators | paginator} functions to simplify
 this.
 */
export async function* listAccountAliases() {
  const command = new ListAccountAliasesCommand({ MaxItems: 5 });

  let response = await client.send(command);

  while (response.AccountAliases?.length) {
    for (const alias of response.AccountAliases) {
      yield alias;
    }

    if (response.IsTruncated) {
      response = await client.send(
        new ListAccountAliasesCommand({
          Marker: response.Marker,
          MaxItems: 5,
        }),
```

```
      );
    } else {
      break;
    }
  }
}
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考ListAccountAliases中的。

SDK對於 JavaScript （ 第 2 個 ）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

iam.listAccountAliases({ MaxItems: 10 }, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考ListAccountAliases中的。

`ListAttachedRolePolicies`

下列程式碼範例會示範如何使用`ListAttachedRolePolicies`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

列出連接至角色的政策。

```
import {
  ListAttachedRolePoliciesCommand,
  IAMClient,
} from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 * A generator function that handles paginated results.
 * The AWS SDK for JavaScript (v3) provides {@link https://docs.aws.amazon.com/
AWSJavaScriptSDK/v3/latest/index.html#paginators | paginator} functions to simplify
 this.
 * @param {string} roleName
 */
export async function* listAttachedRolePolicies(roleName) {
  const command = new ListAttachedRolePoliciesCommand({
    RoleName: roleName,
  });

  let response = await client.send(command);

  while (response.AttachedPolicies?.length) {
    for (const policy of response.AttachedPolicies) {
      yield policy;
    }

    if (response.IsTruncated) {
      response = await client.send(
        new ListAttachedRolePoliciesCommand({
```

```
          RoleName: roleName,
          Marker: response.Marker,
        }),
      );
    } else {
      break;
    }
  }
}
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考ListAttachedRolePolicies中的。

`ListGroups`

下列程式碼範例會示範如何使用`ListGroups`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

列出群組。

```
import { ListGroupsCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 * A generator function that handles paginated results.
 * The AWS SDK for JavaScript (v3) provides {@link https://docs.aws.amazon.com/
AWSJavaScriptSDK/v3/latest/index.html#paginators | paginator} functions to simplify
 this.
 */
export async function* listGroups() {
  const command = new ListGroupsCommand({
    MaxItems: 10,
  });
```

```
  let response = await client.send(command);

  while (response.Groups?.length) {
    for (const group of response.Groups) {
      yield group;
    }

    if (response.IsTruncated) {
      response = await client.send(
        new ListGroupsCommand({
          Marker: response.Marker,
          MaxItems: 10,
        }),
      );
    } else {
      break;
    }
  }
}
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考ListGroups中的。

`ListPolicies`

下列程式碼範例會示範如何使用`ListPolicies`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

列出政策。

```
import { ListPoliciesCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});
```

```
/**
 * A generator function that handles paginated results.
 * The AWS SDK for JavaScript (v3) provides {@link https://docs.aws.amazon.com/
AWSJavaScriptSDK/v3/latest/index.html#paginators | paginator} functions to simplify
 this.
 *
 */
export async function* listPolicies() {
  const command = new ListPoliciesCommand({
    MaxItems: 10,
    OnlyAttached: false,
    // List only the customer managed policies in your Amazon Web Services account.
    Scope: "Local",
  });

  let response = await client.send(command);

  while (response.Policies?.length) {
    for (const policy of response.Policies) {
      yield policy;
    }

    if (response.IsTruncated) {
      response = await client.send(
        new ListPoliciesCommand({
          Marker: response.Marker,
          MaxItems: 10,
          OnlyAttached: false,
          Scope: "Local",
        }),
      );
    } else {
      break;
    }
  }
}
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考ListPolicies中的。

`ListRolePolicies`

下列程式碼範例會示範如何使用`ListRolePolicies`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

列出政策。

```
import { ListRolePoliciesCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 * A generator function that handles paginated results.
 * The AWS SDK for JavaScript (v3) provides {@link https://docs.aws.amazon.com/
AWSJavaScriptSDK/v3/latest/index.html#paginators | paginator} functions to simplify
 this.
 *
 * @param {string} roleName
 */
export async function* listRolePolicies(roleName) {
  const command = new ListRolePoliciesCommand({
    RoleName: roleName,
    MaxItems: 10,
  });

  let response = await client.send(command);

  while (response.PolicyNames?.length) {
    for (const policyName of response.PolicyNames) {
      yield policyName;
    }

    if (response.IsTruncated) {
      response = await client.send(
        new ListRolePoliciesCommand({
          RoleName: roleName,
          MaxItems: 10,
          Marker: response.Marker,
        }),
      );
```

```
    } else {
      break;
    }
  }
}
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考ListRolePolicies中的。

`ListRoles`

下列程式碼範例會示範如何使用`ListRoles`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

列出角色。

```
import { ListRolesCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 * A generator function that handles paginated results.
 * The AWS SDK for JavaScript (v3) provides {@link https://docs.aws.amazon.com/
AWSJavaScriptSDK/v3/latest/index.html#paginators | paginator} functions to simplify
 this.
 *
 */
export async function* listRoles() {
  const command = new ListRolesCommand({
    MaxItems: 10,
  });

  /**
   * @type {import("@aws-sdk/client-iam").ListRolesCommandOutput | undefined}
   */
```

```
  let response = await client.send(command);

  while (response?.Roles?.length) {
    for (const role of response.Roles) {
      yield role;
    }

    if (response.IsTruncated) {
      response = await client.send(
        new ListRolesCommand({
          Marker: response.Marker,
        }),
      );
    } else {
      break;
    }
  }
}
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考ListRoles中的。

`ListSAMLProviders`

下列程式碼範例會示範如何使用`ListSAMLProviders`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

列出提SAML供者。

```
import { ListSAMLProvidersCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

export const listSamlProviders = async () => {
```

```
  const command = new ListSAMLProvidersCommand({});

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考istSAMLProviders中的 L。

`ListServerCertificates`

下列程式碼範例會示範如何使用ListServerCertificates。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

列出憑證。

```
import { ListServerCertificatesCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 * A generator function that handles paginated results.
 * The AWS SDK for JavaScript (v3) provides {@link https://docs.aws.amazon.com/
AWSJavaScriptSDK/v3/latest/index.html#paginators | paginator} functions to simplify
 this.
 *
 */
export async function* listServerCertificates() {
  const command = new ListServerCertificatesCommand({});
  let response = await client.send(command);

  while (response.ServerCertificateMetadataList?.length) {
    for await (const cert of response.ServerCertificateMetadataList) {
      yield cert;
```

```
    }

    if (response.IsTruncated) {
      response = await client.send(new ListServerCertificatesCommand({}));
    } else {
      break;
    }
  }
}
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考ListServerCertificates中的。

SDK對於 JavaScript （ 第 2 個 ）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

iam.listServerCertificates({}, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考ListServerCertificates中的。

`ListUsers`

下列程式碼範例會示範如何使用`ListUsers`。

SDK對於 JavaScript （ 3 ）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

列出使用者。

```
import { ListUsersCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

export const listUsers = async () => {
  const command = new ListUsersCommand({ MaxItems: 10 });

  const response = await client.send(command);
  response.Users?.forEach(({ UserName, CreateDate }) => {
    console.log(`${UserName} created on: ${CreateDate}`);
  });
  return response;
};
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考ListUsers中的。

SDK對於 JavaScript （ 第 2 個 ）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
// Load the AWS SDK for Node.js
```

```
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var params = {
  MaxItems: 10,
};

iam.listUsers(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    var users = data.Users || [];
    users.forEach(function (user) {
      console.log("User " + user.UserName + " created", user.CreateDate);
    });
  }
});
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考ListUsers中的。

`PutRolePolicy`

下列程式碼範例會示範如何使用`PutRolePolicy`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import { PutRolePolicyCommand, IAMClient } from "@aws-sdk/client-iam";

const examplePolicyDocument = JSON.stringify({
```

```
  Version: "2012-10-17",
  Statement: [
    {
      Sid: "VisualEditor0",
      Effect: "Allow",
      Action: [
        "s3:ListBucketMultipartUploads",
        "s3:ListBucketVersions",
        "s3:ListBucket",
        "s3:ListMultipartUploadParts",
      ],
      Resource: "arn:aws:s3:::some-test-bucket",
    },
    {
      Sid: "VisualEditor1",
      Effect: "Allow",
      Action: [
        "s3:ListStorageLensConfigurations",
        "s3:ListAccessPointsForObjectLambda",
        "s3:ListAllMyBuckets",
        "s3:ListAccessPoints",
        "s3:ListJobs",
        "s3:ListMultiRegionAccessPoints",
      ],
      Resource: "*",
    },
  ],
});

const client = new IAMClient({});

/**
 *
 * @param {string} roleName
 * @param {string} policyName
 * @param {string} policyDocument
 */
export const putRolePolicy = async (roleName, policyName, policyDocument) => {
  const command = new PutRolePolicyCommand({
    RoleName: roleName,
    PolicyName: policyName,
    PolicyDocument: policyDocument,
  });
```

```
  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考PutRolePolicy中的。

`UpdateAccessKey`

下列程式碼範例會示範如何使用`UpdateAccessKey`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

更新存取金鑰。

```
import {
  UpdateAccessKeyCommand,
  IAMClient,
  StatusType,
} from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} userName
 * @param {string} accessKeyId
 */
export const updateAccessKey = (userName, accessKeyId) => {
  const command = new UpdateAccessKeyCommand({
    AccessKeyId: accessKeyId,
    Status: StatusType.Inactive,
    UserName: userName,
  });
```

```
  return client.send(command);
};
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考UpdateAccessKey中的。

SDK對於 JavaScript（第 2 個）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var params = {
  AccessKeyId: "ACCESS_KEY_ID",
  Status: "Active",
  UserName: "USER_NAME",
};

iam.updateAccessKey(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考UpdateAccessKey中的。

`UpdateServerCertificate`

下列程式碼範例會示範如何使用`UpdateServerCertificate`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

更新伺服器憑證。

```
import { UpdateServerCertificateCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} currentName
 * @param {string} newName
 */
export const updateServerCertificate = (currentName, newName) => {
  const command = new UpdateServerCertificateCommand({
    ServerCertificateName: currentName,
    NewServerCertificateName: newName,
  });

  return client.send(command);
};
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考UpdateServerCertificate中的。

SDK對於 JavaScript （ 第 2 個 ）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var params = {
  ServerCertificateName: "CERTIFICATE_NAME",
  NewServerCertificateName: "NEW_CERTIFICATE_NAME",
};

iam.updateServerCertificate(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考UpdateServerCertificate中的。

`UpdateUser`

下列程式碼範例會示範如何使用`UpdateUser`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

更新使用者。

```
import { UpdateUserCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} currentUserName
 * @param {string} newUserName
 */
export const updateUser = (currentUserName, newUserName) => {
  const command = new UpdateUserCommand({
    UserName: currentUserName,
    NewUserName: newUserName,
  });

  return client.send(command);
};
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考UpdateUser中的。

SDK對於 JavaScript（第 2 個）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
// Load the AWS SDK for Node.js
```

```
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var params = {
  UserName: process.argv[2],
  NewUserName: process.argv[3],
};

iam.updateUser(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考UpdateUser中的。

`UploadServerCertificate`

下列程式碼範例會示範如何使用`UploadServerCertificate`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import { UploadServerCertificateCommand, IAMClient } from "@aws-sdk/client-iam";
import { readFileSync } from "fs";
import { dirnameFromMetaUrl } from "@aws-doc-sdk-examples/lib/utils/util-fs.js";
import * as path from "path";
```

```
const client = new IAMClient({});

const certMessage = `Generate a certificate and key with the following command, or
 the equivalent for your system.

openssl req -x509 -newkey rsa:4096 -sha256 -days 3650 -nodes \
-keyout example.key -out example.crt -subj "/CN=example.com" \
-addext "subjectAltName=DNS:example.com,DNS:www.example.net,IP:10.0.0.1"
`;

const getCertAndKey = () => {
  try {
    const cert = readFileSync(
      path.join(dirnameFromMetaUrl(import.meta.url), "./example.crt"),
    );
    const key = readFileSync(
      path.join(dirnameFromMetaUrl(import.meta.url), "./example.key"),
    );
    return { cert, key };
  } catch (err) {
    if (err.code === "ENOENT") {
      throw new Error(
        `Certificate and/or private key not found. ${certMessage}`,
      );
    }

    throw err;
  }
};

/**
 *
 * @param {string} certificateName
 */
export const uploadServerCertificate = (certificateName) => {
  const { cert, key } = getCertAndKey();
  const command = new UploadServerCertificateCommand({
    ServerCertificateName: certificateName,
    CertificateBody: cert.toString(),
    PrivateKey: key.toString(),
  });

  return client.send(command);
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考UploadServerCertificate中的。

案例

建置及管理彈性服務

下列程式碼範例會示範如何建立負載平衡的 Web 服務，以傳回書籍、影片和歌曲建議。此範例顯示服務如何回應失故障，以及如何在發生故障時重組服務以提高復原能力。

- 使用 Amazon EC2 Auto Scaling 群組根據啟動範本建立 Amazon 彈性運算雲端 (AmazonEC2) 執行個體，並將執行個體數量保持在指定範圍內。
- 使用 Elastic Load Balancing 處理和分配HTTP請求。
- 監控 Auto Scaling 群組中執行個體的運作狀態，並且只將請求轉送給運作良好的執行個體。
- 在每個執行個EC2體上執行 Python 網頁伺服器來處理HTTP要求。Web 伺服器會回應建議和運作狀態檢查。
- 使用 Amazon DynamoDB 資料表模擬建議服務。
- 透過更新 AWS Systems Manager 參數來控制 Web 伺服器對要求和健康狀態檢查的回應。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

在命令提示中執行互動式案例。

```
#!/usr/bin/env node
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0


import {
  Scenario,
  parseScenarioArgs,
```

```
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

/**
 * The workflow steps are split into three stages:
 *   - deploy
 *   - demo
 *   - destroy
 *
 * Each of these stages has a corresponding file prefixed with steps-*.
 */
import { deploySteps } from "./steps-deploy.js";
import { demoSteps } from "./steps-demo.js";
import { destroySteps } from "./steps-destroy.js";

/**
 * The context is passed to every scenario. Scenario steps
 * will modify the context.
 */
const context = {};

/**
 * Three Scenarios are created for the workflow. A Scenario is an orchestration
 class
 * that simplifies running a series of steps.
 */
export const scenarios = {
  // Deploys all resources necessary for the workflow.
  deploy: new Scenario("Resilient Workflow - Deploy", deploySteps, context),
  // Demonstrates how a fragile web service can be made more resilient.
  demo: new Scenario("Resilient Workflow - Demo", demoSteps, context),
  // Destroys the resources created for the workflow.
  destroy: new Scenario("Resilient Workflow - Destroy", destroySteps, context),
};

// Call function if run directly
import { fileURLToPath } from "url";

if (process.argv[1] === fileURLToPath(import.meta.url)) {
  parseScenarioArgs(scenarios);
}
```

建立步驟以部署所有資源。

```
import { join } from "node:path";
import { readFileSync, writeFileSync } from "node:fs";
import axios from "axios";

import {
  BatchWriteItemCommand,
  CreateTableCommand,
  DynamoDBClient,
  waitUntilTableExists,
} from "@aws-sdk/client-dynamodb";
import {
  EC2Client,
  CreateKeyPairCommand,
  CreateLaunchTemplateCommand,
  DescribeAvailabilityZonesCommand,
  DescribeVpcsCommand,
  DescribeSubnetsCommand,
  DescribeSecurityGroupsCommand,
  AuthorizeSecurityGroupIngressCommand,
} from "@aws-sdk/client-ec2";
import {
  IAMClient,
  CreatePolicyCommand,
  CreateRoleCommand,
  CreateInstanceProfileCommand,
  AddRoleToInstanceProfileCommand,
  AttachRolePolicyCommand,
  waitUntilInstanceProfileExists,
} from "@aws-sdk/client-iam";
import { SSMClient, GetParameterCommand } from "@aws-sdk/client-ssm";
import {
  CreateAutoScalingGroupCommand,
  AutoScalingClient,
  AttachLoadBalancerTargetGroupsCommand,
} from "@aws-sdk/client-auto-scaling";
import {
  CreateListenerCommand,
  CreateLoadBalancerCommand,
  CreateTargetGroupCommand,
  ElasticLoadBalancingV2Client,
  waitUntilLoadBalancerAvailable,
} from "@aws-sdk/client-elastic-load-balancing-v2";
```

```
import {
  ScenarioOutput,
  ScenarioInput,
  ScenarioAction,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import { saveState } from "@aws-doc-sdk-examples/lib/scenario/steps-common.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

import { MESSAGES, NAMES, RESOURCES_PATH, ROOT } from "./constants.js";
import { initParamsSteps } from "./steps-reset-params.js";

/**
 * @type {import('@aws-doc-sdk-examples/lib/scenario.js').Step[]}
 */
export const deploySteps = [
  new ScenarioOutput("introduction", MESSAGES.introduction, { header: true }),
  new ScenarioInput("confirmDeployment", MESSAGES.confirmDeployment, {
    type: "confirm",
  }),
  new ScenarioAction(
    "handleConfirmDeployment",
    (c) => c.confirmDeployment === false && process.exit(),
  ),
  new ScenarioOutput(
    "creatingTable",
    MESSAGES.creatingTable.replace("${TABLE_NAME}", NAMES.tableName),
  ),
  new ScenarioAction("createTable", async () => {
    const client = new DynamoDBClient({});
    await client.send(
      new CreateTableCommand({
        TableName: NAMES.tableName,
        ProvisionedThroughput: {
          ReadCapacityUnits: 5,
          WriteCapacityUnits: 5,
        },
        AttributeDefinitions: [
          {
            AttributeName: "MediaType",
            AttributeType: "S",
          },
          {
            AttributeName: "ItemId",
            AttributeType: "N",
```

```
          },
        ],
        KeySchema: [
          {
            AttributeName: "MediaType",
            KeyType: "HASH",
          },
          {
            AttributeName: "ItemId",
            KeyType: "RANGE",
          },
        ],
      }),
    );
    await waitUntilTableExists({ client }, { TableName: NAMES.tableName });
  }),
  new ScenarioOutput(
    "createdTable",
    MESSAGES.createdTable.replace("${TABLE_NAME}", NAMES.tableName),
  ),
  new ScenarioOutput(
    "populatingTable",
    MESSAGES.populatingTable.replace("${TABLE_NAME}", NAMES.tableName),
  ),
  new ScenarioAction("populateTable", () => {
    const client = new DynamoDBClient({});
    /**
     * @type {{ default: import("@aws-sdk/client-dynamodb").PutRequest['Item'][] }}
     */
    const recommendations = JSON.parse(
      readFileSync(join(RESOURCES_PATH, "recommendations.json")),
    );

    return client.send(
      new BatchWriteItemCommand({
        RequestItems: {
          [NAMES.tableName]: recommendations.map((item) => ({
            PutRequest: { Item: item },
          })),
        },
      }),
    );
  }),
  new ScenarioOutput(
```

```
    "populatedTable",
    MESSAGES.populatedTable.replace("${TABLE_NAME}", NAMES.tableName),
  ),
  new ScenarioOutput(
    "creatingKeyPair",
    MESSAGES.creatingKeyPair.replace("${KEY_PAIR_NAME}", NAMES.keyPairName),
  ),
  new ScenarioAction("createKeyPair", async () => {
    const client = new EC2Client({});
    const { KeyMaterial } = await client.send(
      new CreateKeyPairCommand({
        KeyName: NAMES.keyPairName,
      }),
    );

    writeFileSync(`${NAMES.keyPairName}.pem`, KeyMaterial, { mode: 0o600 });
  }),
  new ScenarioOutput(
    "createdKeyPair",
    MESSAGES.createdKeyPair.replace("${KEY_PAIR_NAME}", NAMES.keyPairName),
  ),
  new ScenarioOutput(
    "creatingInstancePolicy",
    MESSAGES.creatingInstancePolicy.replace(
      "${INSTANCE_POLICY_NAME}",
      NAMES.instancePolicyName,
    ),
  ),
  new ScenarioAction("createInstancePolicy", async (state) => {
    const client = new IAMClient({});
    const {
      Policy: { Arn },
    } = await client.send(
      new CreatePolicyCommand({
        PolicyName: NAMES.instancePolicyName,
        PolicyDocument: readFileSync(
          join(RESOURCES_PATH, "instance_policy.json"),
        ),
      }),
    );
    state.instancePolicyArn = Arn;
  }),
  new ScenarioOutput("createdInstancePolicy", (state) =>
    MESSAGES.createdInstancePolicy
```

```
    .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
    .replace("${INSTANCE_POLICY_ARN}", state.instancePolicyArn),
),
new ScenarioOutput(
  "creatingInstanceRole",
  MESSAGES.creatingInstanceRole.replace(
    "${INSTANCE_ROLE_NAME}",
    NAMES.instanceRoleName,
  ),
),
new ScenarioAction("createInstanceRole", () => {
  const client = new IAMClient({});
  return client.send(
    new CreateRoleCommand({
      RoleName: NAMES.instanceRoleName,
      AssumeRolePolicyDocument: readFileSync(
        join(ROOT, "assume-role-policy.json"),
      ),
    }),
  );
}),
new ScenarioOutput(
  "createdInstanceRole",
  MESSAGES.createdInstanceRole.replace(
    "${INSTANCE_ROLE_NAME}",
    NAMES.instanceRoleName,
  ),
),
new ScenarioOutput(
  "attachingPolicyToRole",
  MESSAGES.attachingPolicyToRole
    .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName)
    .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName),
),
new ScenarioAction("attachPolicyToRole", async (state) => {
  const client = new IAMClient({});
  await client.send(
    new AttachRolePolicyCommand({
      RoleName: NAMES.instanceRoleName,
      PolicyArn: state.instancePolicyArn,
    }),
  );
}),
new ScenarioOutput(
```

```
    "attachedPolicyToRole",
    MESSAGES.attachedPolicyToRole
      .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
      .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName),
  ),
  new ScenarioOutput(
    "creatingInstanceProfile",
    MESSAGES.creatingInstanceProfile.replace(
      "${INSTANCE_PROFILE_NAME}",
      NAMES.instanceProfileName,
    ),
  ),
  new ScenarioAction("createInstanceProfile", async (state) => {
    const client = new IAMClient({});
    const {
      InstanceProfile: { Arn },
    } = await client.send(
      new CreateInstanceProfileCommand({
        InstanceProfileName: NAMES.instanceProfileName,
      }),
    );
    state.instanceProfileArn = Arn;

    await waitUntilInstanceProfileExists(
      { client },
      { InstanceProfileName: NAMES.instanceProfileName },
    );
  }),
  new ScenarioOutput("createdInstanceProfile", (state) =>
    MESSAGES.createdInstanceProfile
      .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
      .replace("${INSTANCE_PROFILE_ARN}", state.instanceProfileArn),
  ),
  new ScenarioOutput(
    "addingRoleToInstanceProfile",
    MESSAGES.addingRoleToInstanceProfile
      .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
      .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName),
  ),
  new ScenarioAction("addRoleToInstanceProfile", () => {
    const client = new IAMClient({});
    return client.send(
      new AddRoleToInstanceProfileCommand({
        RoleName: NAMES.instanceRoleName,
```

```
      InstanceProfileName: NAMES.instanceProfileName,
    }),
  );
}),
new ScenarioOutput(
  "addedRoleToInstanceProfile",
  MESSAGES.addedRoleToInstanceProfile
    .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
    .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName),
),
...initParamsSteps,
new ScenarioOutput("creatingLaunchTemplate", MESSAGES.creatingLaunchTemplate),
new ScenarioAction("createLaunchTemplate", async () => {
  const ssmClient = new SSMClient({});
  const { Parameter } = await ssmClient.send(
    new GetParameterCommand({
      Name: "/aws/service/ami-amazon-linux-latest/amzn2-ami-hvm-x86_64-gp2",
    }),
  );
  const ec2Client = new EC2Client({});
  await ec2Client.send(
    new CreateLaunchTemplateCommand({
      LaunchTemplateName: NAMES.launchTemplateName,
      LaunchTemplateData: {
        InstanceType: "t3.micro",
        ImageId: Parameter.Value,
        IamInstanceProfile: { Name: NAMES.instanceProfileName },
        UserData: readFileSync(
          join(RESOURCES_PATH, "server_startup_script.sh"),
        ).toString("base64"),
        KeyName: NAMES.keyPairName,
      },
    }),
  );
}),
new ScenarioOutput(
  "createdLaunchTemplate",
  MESSAGES.createdLaunchTemplate.replace(
    "${LAUNCH_TEMPLATE_NAME}",
    NAMES.launchTemplateName,
  ),
),
new ScenarioOutput(
  "creatingAutoScalingGroup",
```

```
    MESSAGES.creatingAutoScalingGroup.replace(
      "${AUTO_SCALING_GROUP_NAME}",
      NAMES.autoScalingGroupName,
    ),
  ),
  new ScenarioAction("createAutoScalingGroup", async (state) => {
    const ec2Client = new EC2Client({});
    const { AvailabilityZones } = await ec2Client.send(
      new DescribeAvailabilityZonesCommand({}),
    );
    state.availabilityZoneNames = AvailabilityZones.map((az) => az.ZoneName);
    const autoScalingClient = new AutoScalingClient({});
    await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
      autoScalingClient.send(
        new CreateAutoScalingGroupCommand({
          AvailabilityZones: state.availabilityZoneNames,
          AutoScalingGroupName: NAMES.autoScalingGroupName,
          LaunchTemplate: {
            LaunchTemplateName: NAMES.launchTemplateName,
            Version: "$Default",
          },
          MinSize: 3,
          MaxSize: 3,
        }),
      ),
    );
  }),
  new ScenarioOutput(
    "createdAutoScalingGroup",
    /**
     * @param {{ availabilityZoneNames: string[] }} state
     */
    (state) =>
      MESSAGES.createdAutoScalingGroup
        .replace("${AUTO_SCALING_GROUP_NAME}", NAMES.autoScalingGroupName)
        .replace(
          "${AVAILABILITY_ZONE_NAMES}",
          state.availabilityZoneNames.join(", "),
        ),
  ),
  new ScenarioInput("confirmContinue", MESSAGES.confirmContinue, {
    type: "confirm",
  }),
  new ScenarioOutput("loadBalancer", MESSAGES.loadBalancer),
```

```
  new ScenarioOutput("gettingVpc", MESSAGES.gettingVpc),
  new ScenarioAction("getVpc", async (state) => {
    const client = new EC2Client({});
    const { Vpcs } = await client.send(
      new DescribeVpcsCommand({
        Filters: [{ Name: "is-default", Values: ["true"] }],
      }),
    );
    state.defaultVpc = Vpcs[0].VpcId;
  }),
  new ScenarioOutput("gotVpc", (state) =>
    MESSAGES.gotVpc.replace("${VPC_ID}", state.defaultVpc),
  ),
  new ScenarioOutput("gettingSubnets", MESSAGES.gettingSubnets),
  new ScenarioAction("getSubnets", async (state) => {
    const client = new EC2Client({});
    const { Subnets } = await client.send(
      new DescribeSubnetsCommand({
        Filters: [
          { Name: "vpc-id", Values: [state.defaultVpc] },
          { Name: "availability-zone", Values: state.availabilityZoneNames },
          { Name: "default-for-az", Values: ["true"] },
        ],
      }),
    );
    state.subnets = Subnets.map((subnet) => subnet.SubnetId);
  }),
  new ScenarioOutput(
    "gotSubnets",
    /**
     * @param {{ subnets: string[] }} state
     */
    (state) =>
      MESSAGES.gotSubnets.replace("${SUBNETS}", state.subnets.join(", ")),
  ),
  new ScenarioOutput(
    "creatingLoadBalancerTargetGroup",
    MESSAGES.creatingLoadBalancerTargetGroup.replace(
      "${TARGET_GROUP_NAME}",
      NAMES.loadBalancerTargetGroupName,
    ),
  ),
  new ScenarioAction("createLoadBalancerTargetGroup", async (state) => {
    const client = new ElasticLoadBalancingV2Client({});
```

```
    const { TargetGroups } = await client.send(
      new CreateTargetGroupCommand({
        Name: NAMES.loadBalancerTargetGroupName,
        Protocol: "HTTP",
        Port: 80,
        HealthCheckPath: "/healthcheck",
        HealthCheckIntervalSeconds: 10,
        HealthCheckTimeoutSeconds: 5,
        HealthyThresholdCount: 2,
        UnhealthyThresholdCount: 2,
        VpcId: state.defaultVpc,
      }),
    );
    const targetGroup = TargetGroups[0];
    state.targetGroupArn = targetGroup.TargetGroupArn;
    state.targetGroupProtocol = targetGroup.Protocol;
    state.targetGroupPort = targetGroup.Port;
  }),
  new ScenarioOutput(
    "createdLoadBalancerTargetGroup",
    MESSAGES.createdLoadBalancerTargetGroup.replace(
      "${TARGET_GROUP_NAME}",
      NAMES.loadBalancerTargetGroupName,
    ),
  ),
  new ScenarioOutput(
    "creatingLoadBalancer",
    MESSAGES.creatingLoadBalancer.replace("${LB_NAME}", NAMES.loadBalancerName),
  ),
  new ScenarioAction("createLoadBalancer", async (state) => {
    const client = new ElasticLoadBalancingV2Client({});
    const { LoadBalancers } = await client.send(
      new CreateLoadBalancerCommand({
        Name: NAMES.loadBalancerName,
        Subnets: state.subnets,
      }),
    );
    state.loadBalancerDns = LoadBalancers[0].DNSName;
    state.loadBalancerArn = LoadBalancers[0].LoadBalancerArn;
    await waitUntilLoadBalancerAvailable(
      { client },
      { Names: [NAMES.loadBalancerName] },
    );
  }),
```

```
  new ScenarioOutput("createdLoadBalancer", (state) =>
    MESSAGES.createdLoadBalancer
      .replace("${LB_NAME}", NAMES.loadBalancerName)
      .replace("${DNS_NAME}", state.loadBalancerDns),
  ),
  new ScenarioOutput(
    "creatingListener",
    MESSAGES.creatingLoadBalancerListener
      .replace("${LB_NAME}", NAMES.loadBalancerName)
      .replace("${TARGET_GROUP_NAME}", NAMES.loadBalancerTargetGroupName),
  ),
  new ScenarioAction("createListener", async (state) => {
    const client = new ElasticLoadBalancingV2Client({});
    const { Listeners } = await client.send(
      new CreateListenerCommand({
        LoadBalancerArn: state.loadBalancerArn,
        Protocol: state.targetGroupProtocol,
        Port: state.targetGroupPort,
        DefaultActions: [
          { Type: "forward", TargetGroupArn: state.targetGroupArn },
        ],
      }),
    );
    const listener = Listeners[0];
    state.loadBalancerListenerArn = listener.ListenerArn;
  }),
  new ScenarioOutput("createdListener", (state) =>
    MESSAGES.createdLoadBalancerListener.replace(
      "${LB_LISTENER_ARN}",
      state.loadBalancerListenerArn,
    ),
  ),
  new ScenarioOutput(
    "attachingLoadBalancerTargetGroup",
    MESSAGES.attachingLoadBalancerTargetGroup
      .replace("${TARGET_GROUP_NAME}", NAMES.loadBalancerTargetGroupName)
      .replace("${AUTO_SCALING_GROUP_NAME}", NAMES.autoScalingGroupName),
  ),
  new ScenarioAction("attachLoadBalancerTargetGroup", async (state) => {
    const client = new AutoScalingClient({});
    await client.send(
      new AttachLoadBalancerTargetGroupsCommand({
        AutoScalingGroupName: NAMES.autoScalingGroupName,
        TargetGroupARNs: [state.targetGroupArn],
```

```
      }),
    );
  }),
  new ScenarioOutput(
    "attachedLoadBalancerTargetGroup",
    MESSAGES.attachedLoadBalancerTargetGroup,
  ),
  new ScenarioOutput("verifyingInboundPort", MESSAGES.verifyingInboundPort),
  new ScenarioAction(
    "verifyInboundPort",
    /**
     *
     * @param {{ defaultSecurityGroup: import('@aws-sdk/client-ec2').SecurityGroup}}
state
     */
    async (state) => {
      const client = new EC2Client({});
      const { SecurityGroups } = await client.send(
        new DescribeSecurityGroupsCommand({
          Filters: [{ Name: "group-name", Values: ["default"] }],
        }),
      );
      if (!SecurityGroups) {
        state.verifyInboundPortError = new Error(MESSAGES.noSecurityGroups);
      }
      state.defaultSecurityGroup = SecurityGroups[0];

      /**
       * @type {string}
       */
      const ipResponse = (await axios.get("http://checkip.amazonaws.com")).data;
      state.myIp = ipResponse.trim();
      const myIpRules = state.defaultSecurityGroup.IpPermissions.filter(
        ({ IpRanges }) =>
          IpRanges.some(
            ({ CidrIp }) =>
              CidrIp.startsWith(state.myIp) || CidrIp === "0.0.0.0/0",
          ),
      )
        .filter(({ IpProtocol }) => IpProtocol === "tcp")
        .filter(({ FromPort }) => FromPort === 80);

      state.myIpRules = myIpRules;
    },
```

```
  ),
  new ScenarioOutput(
    "verifiedInboundPort",
    /**
     * @param {{ myIpRules: any[] }} state
     */
    (state) => {
      if (state.myIpRules.length > 0) {
        return MESSAGES.foundIpRules.replace(
          "${IP_RULES}",
          JSON.stringify(state.myIpRules, null, 2),
        );
      } else {
        return MESSAGES.noIpRules;
      }
    },
  ),
  new ScenarioInput(
    "shouldAddInboundRule",
    /**
     * @param {{ myIpRules: any[] }} state
     */
    (state) => {
      if (state.myIpRules.length > 0) {
        return false;
      } else {
        return MESSAGES.noIpRules;
      }
    },
    { type: "confirm" },
  ),
  new ScenarioAction(
    "addInboundRule",
    /**
     * @param {{ defaultSecurityGroup: import('@aws-sdk/client-
ec2').SecurityGroup }} state
     */
    async (state) => {
      if (!state.shouldAddInboundRule) {
        return;
      }

      const client = new EC2Client({});
      await client.send(
```

```
        new AuthorizeSecurityGroupIngressCommand({
          GroupId: state.defaultSecurityGroup.GroupId,
          CidrIp: `${state.myIp}/32`,
          FromPort: 80,
          ToPort: 80,
          IpProtocol: "tcp",
        }),
      );
    },
  ),
  new ScenarioOutput("addedInboundRule", (state) => {
    if (state.shouldAddInboundRule) {
      return MESSAGES.addedInboundRule.replace("${IP_ADDRESS}", state.myIp);
    } else {
      return false;
    }
  }),
  new ScenarioOutput("verifyingEndpoint", (state) =>
    MESSAGES.verifyingEndpoint.replace("${DNS_NAME}", state.loadBalancerDns),
  ),
  new ScenarioAction("verifyEndpoint", async (state) => {
    try {
      const response = await retry({ intervalInMs: 2000, maxRetries: 30 }, () =>
        axios.get(`http://${state.loadBalancerDns}`),
      );
      state.endpointResponse = JSON.stringify(response.data, null, 2);
    } catch (e) {
      state.verifyEndpointError = e;
    }
  }),
  new ScenarioOutput("verifiedEndpoint", (state) => {
    if (state.verifyEndpointError) {
      console.error(state.verifyEndpointError);
    } else {
      return MESSAGES.verifiedEndpoint.replace(
        "${ENDPOINT_RESPONSE}",
        state.endpointResponse,
      );
    }
  }),
  saveState,
];
```

建立步驟以執行示範。

```
import { readFileSync } from "node:fs";
import { join } from "node:path";

import axios from "axios";

import {
  DescribeTargetGroupsCommand,
  DescribeTargetHealthCommand,
  ElasticLoadBalancingV2Client,
} from "@aws-sdk/client-elastic-load-balancing-v2";
import {
  DescribeInstanceInformationCommand,
  PutParameterCommand,
  SSMClient,
  SendCommandCommand,
} from "@aws-sdk/client-ssm";
import {
  IAMClient,
  CreatePolicyCommand,
  CreateRoleCommand,
  AttachRolePolicyCommand,
  CreateInstanceProfileCommand,
  AddRoleToInstanceProfileCommand,
  waitUntilInstanceProfileExists,
} from "@aws-sdk/client-iam";
import {
  AutoScalingClient,
  DescribeAutoScalingGroupsCommand,
  TerminateInstanceInAutoScalingGroupCommand,
} from "@aws-sdk/client-auto-scaling";
import {
  DescribeIamInstanceProfileAssociationsCommand,
  EC2Client,
  RebootInstancesCommand,
  ReplaceIamInstanceProfileAssociationCommand,
} from "@aws-sdk/client-ec2";

import {
  ScenarioAction,
  ScenarioInput,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/scenario.js";
```

```
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

import { MESSAGES, NAMES, RESOURCES_PATH } from "./constants.js";
import { findLoadBalancer } from "./shared.js";

const getRecommendation = new ScenarioAction(
  "getRecommendation",
  async (state) => {
    const loadBalancer = await findLoadBalancer(NAMES.loadBalancerName);
    if (loadBalancer) {
      state.loadBalancerDnsName = loadBalancer.DNSName;
      try {
        state.recommendation = (
          await axios.get(`http://${state.loadBalancerDnsName}`)
        ).data;
      } catch (e) {
        state.recommendation = e instanceof Error ? e.message : e;
      }
    } else {
      throw new Error(MESSAGES.demoFindLoadBalancerError);
    }
  },
);

const getRecommendationResult = new ScenarioOutput(
  "getRecommendationResult",
  (state) =>
    `Recommendation:\n${JSON.stringify(state.recommendation, null, 2)}`,
  { preformatted: true },
);

const getHealthCheck = new ScenarioAction("getHealthCheck", async (state) => {
  const client = new ElasticLoadBalancingV2Client({});
  const { TargetGroups } = await client.send(
    new DescribeTargetGroupsCommand({
      Names: [NAMES.loadBalancerTargetGroupName],
    }),
  );

  const { TargetHealthDescriptions } = await client.send(
    new DescribeTargetHealthCommand({
      TargetGroupArn: TargetGroups[0].TargetGroupArn,
    }),
  );
```

```
  state.targetHealthDescriptions = TargetHealthDescriptions;
});

const getHealthCheckResult = new ScenarioOutput(
  "getHealthCheckResult",
  /**
   * @param {{ targetHealthDescriptions: import('@aws-sdk/client-elastic-load-
balancing-v2').TargetHealthDescription[]}} state
   */
  (state) => {
    const status = state.targetHealthDescriptions
      .map((th) => `${th.Target.Id}: ${th.TargetHealth.State}`)
      .join("\n");
    return `Health check:\n${status}`;
  },
  { preformatted: true },
);

const loadBalancerLoop = new ScenarioAction(
  "loadBalancerLoop",
  getRecommendation.action,
  {
    whileConfig: {
      whileFn: ({ loadBalancerCheck }) => loadBalancerCheck,
      input: new ScenarioInput(
        "loadBalancerCheck",
        MESSAGES.demoLoadBalancerCheck,
        {
          type: "confirm",
        },
      ),
      output: getRecommendationResult,
    },
  },
);

const healthCheckLoop = new ScenarioAction(
  "healthCheckLoop",
  getHealthCheck.action,
  {
    whileConfig: {
      whileFn: ({ healthCheck }) => healthCheck,
      input: new ScenarioInput("healthCheck", MESSAGES.demoHealthCheck, {
        type: "confirm",
```

```
      }),
      output: getHealthCheckResult,
    },
  },
);

const statusSteps = [
  getRecommendation,
  getRecommendationResult,
  getHealthCheck,
  getHealthCheckResult,
];

/**
 * @type {import('@aws-doc-sdk-examples/lib/scenario.js').Step[]}
 */
export const demoSteps = [
  new ScenarioOutput("header", MESSAGES.demoHeader, { header: true }),
  new ScenarioOutput("sanityCheck", MESSAGES.demoSanityCheck),
  ...statusSteps,
  new ScenarioInput(
    "brokenDependencyConfirmation",
    MESSAGES.demoBrokenDependencyConfirmation,
    { type: "confirm" },
  ),
  new ScenarioAction("brokenDependency", async (state) => {
    if (!state.brokenDependencyConfirmation) {
      process.exit();
    } else {
      const client = new SSMClient({});
      state.badTableName = `fake-table-${Date.now()}`;
      await client.send(
        new PutParameterCommand({
          Name: NAMES.ssmTableNameKey,
          Value: state.badTableName,
          Overwrite: true,
          Type: "String",
        }),
      );
    }
  }),
  new ScenarioOutput("testBrokenDependency", (state) =>
    MESSAGES.demoTestBrokenDependency.replace(
      "${TABLE_NAME}",
```

```
      state.badTableName,
    ),
  ),
  ...statusSteps,
  new ScenarioInput(
    "staticResponseConfirmation",
    MESSAGES.demoStaticResponseConfirmation,
    { type: "confirm" },
  ),
  new ScenarioAction("staticResponse", async (state) => {
    if (!state.staticResponseConfirmation) {
      process.exit();
    } else {
      const client = new SSMClient({});
      await client.send(
        new PutParameterCommand({
          Name: NAMES.ssmFailureResponseKey,
          Value: "static",
          Overwrite: true,
          Type: "String",
        }),
      );
    }
  }),
  new ScenarioOutput("testStaticResponse", MESSAGES.demoTestStaticResponse),
  ...statusSteps,
  new ScenarioInput(
    "badCredentialsConfirmation",
    MESSAGES.demoBadCredentialsConfirmation,
    { type: "confirm" },
  ),
  new ScenarioAction("badCredentialsExit", (state) => {
    if (!state.badCredentialsConfirmation) {
      process.exit();
    }
  }),
  new ScenarioAction("fixDynamoDBName", async () => {
    const client = new SSMClient({});
    await client.send(
      new PutParameterCommand({
        Name: NAMES.ssmTableNameKey,
        Value: NAMES.tableName,
        Overwrite: true,
        Type: "String",
```

```
      }),
    );
  }),
  new ScenarioAction(
    "badCredentials",
    /**
     * @param {{ targetInstance: import('@aws-sdk/client-auto-scaling').Instance }}
state
     */
    async (state) => {
      await createSsmOnlyInstanceProfile();
      const autoScalingClient = new AutoScalingClient({});
      const { AutoScalingGroups } = await autoScalingClient.send(
        new DescribeAutoScalingGroupsCommand({
          AutoScalingGroupNames: [NAMES.autoScalingGroupName],
        }),
      );
      state.targetInstance = AutoScalingGroups[0].Instances[0];
      const ec2Client = new EC2Client({});
      const { IamInstanceProfileAssociations } = await ec2Client.send(
        new DescribeIamInstanceProfileAssociationsCommand({
          Filters: [
            { Name: "instance-id", Values: [state.targetInstance.InstanceId] },
          ],
        }),
      );
      state.instanceProfileAssociationId =
        IamInstanceProfileAssociations[0].AssociationId;
      await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
        ec2Client.send(
          new ReplaceIamInstanceProfileAssociationCommand({
            AssociationId: state.instanceProfileAssociationId,
            IamInstanceProfile: { Name: NAMES.ssmOnlyInstanceProfileName },
          }),
        ),
      );

      await ec2Client.send(
        new RebootInstancesCommand({
          InstanceIds: [state.targetInstance.InstanceId],
        }),
      );

      const ssmClient = new SSMClient({});
```

```
      await retry({ intervalInMs: 20000, maxRetries: 15 }, async () => {
        const { InstanceInformationList } = await ssmClient.send(
          new DescribeInstanceInformationCommand({}),
        );

        const instance = InstanceInformationList.find(
          (info) => info.InstanceId === state.targetInstance.InstanceId,
        );

        if (!instance) {
          throw new Error("Instance not found.");
        }
      });

      await ssmClient.send(
        new SendCommandCommand({
          InstanceIds: [state.targetInstance.InstanceId],
          DocumentName: "AWS-RunShellScript",
          Parameters: { commands: ["cd / && sudo python3 server.py 80"] },
        }),
      );
    },
  ),
  new ScenarioOutput(
    "testBadCredentials",
    /**
     * @param {{ targetInstance: import('@aws-sdk/client-ssm').InstanceInformation}}
state
     */
    (state) =>
      MESSAGES.demoTestBadCredentials.replace(
        "${INSTANCE_ID}",
        state.targetInstance.InstanceId,
      ),
  ),
  loadBalancerLoop,
  new ScenarioInput(
    "deepHealthCheckConfirmation",
    MESSAGES.demoDeepHealthCheckConfirmation,
    { type: "confirm" },
  ),
  new ScenarioAction("deepHealthCheckExit", (state) => {
    if (!state.deepHealthCheckConfirmation) {
      process.exit();
```

```
    }
  }),
  new ScenarioAction("deepHealthCheck", async () => {
    const client = new SSMClient({});
    await client.send(
      new PutParameterCommand({
        Name: NAMES.ssmHealthCheckKey,
        Value: "deep",
        Overwrite: true,
        Type: "String",
      }),
    );
  }),
  new ScenarioOutput("testDeepHealthCheck", MESSAGES.demoTestDeepHealthCheck),
  healthCheckLoop,
  loadBalancerLoop,
  new ScenarioInput(
    "killInstanceConfirmation",
    /**
     * @param {{ targetInstance: import('@aws-sdk/client-
ssm').InstanceInformation }} state
     */
    (state) =>
      MESSAGES.demoKillInstanceConfirmation.replace(
        "${INSTANCE_ID}",
        state.targetInstance.InstanceId,
      ),
    { type: "confirm" },
  ),
  new ScenarioAction("killInstanceExit", (state) => {
    if (!state.killInstanceConfirmation) {
      process.exit();
    }
  }),
  new ScenarioAction(
    "killInstance",
    /**
     * @param {{ targetInstance: import('@aws-sdk/client-
ssm').InstanceInformation }} state
     */
    async (state) => {
      const client = new AutoScalingClient({});
      await client.send(
        new TerminateInstanceInAutoScalingGroupCommand({
```

```
        InstanceId: state.targetInstance.InstanceId,
        ShouldDecrementDesiredCapacity: false,
      }),
    );
  },
),
new ScenarioOutput("testKillInstance", MESSAGES.demoTestKillInstance),
healthCheckLoop,
loadBalancerLoop,
new ScenarioInput("failOpenConfirmation", MESSAGES.demoFailOpenConfirmation, {
  type: "confirm",
}),
new ScenarioAction("failOpenExit", (state) => {
  if (!state.failOpenConfirmation) {
    process.exit();
  }
}),
new ScenarioAction("failOpen", () => {
  const client = new SSMClient({});
  return client.send(
    new PutParameterCommand({
      Name: NAMES.ssmTableNameKey,
      Value: `fake-table-${Date.now()}`,
      Overwrite: true,
      Type: "String",
    }),
  );
}),
new ScenarioOutput("testFailOpen", MESSAGES.demoFailOpenTest),
healthCheckLoop,
loadBalancerLoop,
new ScenarioInput(
  "resetTableConfirmation",
  MESSAGES.demoResetTableConfirmation,
  { type: "confirm" },
),
new ScenarioAction("resetTableExit", (state) => {
  if (!state.resetTableConfirmation) {
    process.exit();
  }
}),
new ScenarioAction("resetTable", async () => {
  const client = new SSMClient({});
  await client.send(
```

```
      new PutParameterCommand({
        Name: NAMES.ssmTableNameKey,
        Value: NAMES.tableName,
        Overwrite: true,
        Type: "String",
      }),
    );
  }),
  new ScenarioOutput("testResetTable", MESSAGES.demoTestResetTable),
  healthCheckLoop,
  loadBalancerLoop,
];

async function createSsmOnlyInstanceProfile() {
  const iamClient = new IAMClient({});
  const { Policy } = await iamClient.send(
    new CreatePolicyCommand({
      PolicyName: NAMES.ssmOnlyPolicyName,
      PolicyDocument: readFileSync(
        join(RESOURCES_PATH, "ssm_only_policy.json"),
      ),
    }),
  );
  await iamClient.send(
    new CreateRoleCommand({
      RoleName: NAMES.ssmOnlyRoleName,
      AssumeRolePolicyDocument: JSON.stringify({
        Version: "2012-10-17",
        Statement: [
          {
            Effect: "Allow",
            Principal: { Service: "ec2.amazonaws.com" },
            Action: "sts:AssumeRole",
          },
        ],
      }),
    }),
  );
  await iamClient.send(
    new AttachRolePolicyCommand({
      RoleName: NAMES.ssmOnlyRoleName,
      PolicyArn: Policy.Arn,
    }),
  );
```

```
  await iamClient.send(
    new AttachRolePolicyCommand({
      RoleName: NAMES.ssmOnlyRoleName,
      PolicyArn: "arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore",
    }),
  );
  const { InstanceProfile } = await iamClient.send(
    new CreateInstanceProfileCommand({
      InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
    }),
  );
  await waitUntilInstanceProfileExists(
    { client: iamClient },
    { InstanceProfileName: NAMES.ssmOnlyInstanceProfileName },
  );
  await iamClient.send(
    new AddRoleToInstanceProfileCommand({
      InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
      RoleName: NAMES.ssmOnlyRoleName,
    }),
  );

  return InstanceProfile;
}
```

建立步驟以銷毀所有資源。

```
import { unlinkSync } from "node:fs";

import { DynamoDBClient, DeleteTableCommand } from "@aws-sdk/client-dynamodb";
import {
  EC2Client,
  DeleteKeyPairCommand,
  DeleteLaunchTemplateCommand,
  RevokeSecurityGroupIngressCommand,
} from "@aws-sdk/client-ec2";
import {
  IAMClient,
  DeleteInstanceProfileCommand,
  RemoveRoleFromInstanceProfileCommand,
  DeletePolicyCommand,
  DeleteRoleCommand,
```

```
  DetachRolePolicyCommand,
  paginateListPolicies,
} from "@aws-sdk/client-iam";
import {
  AutoScalingClient,
  DeleteAutoScalingGroupCommand,
  TerminateInstanceInAutoScalingGroupCommand,
  UpdateAutoScalingGroupCommand,
  paginateDescribeAutoScalingGroups,
} from "@aws-sdk/client-auto-scaling";
import {
  DeleteLoadBalancerCommand,
  DeleteTargetGroupCommand,
  DescribeTargetGroupsCommand,
  ElasticLoadBalancingV2Client,
} from "@aws-sdk/client-elastic-load-balancing-v2";

import {
  ScenarioOutput,
  ScenarioInput,
  ScenarioAction,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import { loadState } from "@aws-doc-sdk-examples/lib/scenario/steps-common.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

import { MESSAGES, NAMES } from "./constants.js";
import { findLoadBalancer } from "./shared.js";

/**
 * @type {import('@aws-doc-sdk-examples/lib/scenario.js').Step[]}
 */
export const destroySteps = [
  loadState,
  new ScenarioInput("destroy", MESSAGES.destroy, { type: "confirm" }),
  new ScenarioAction(
    "abort",
    (state) => state.destroy === false && process.exit(),
  ),
  new ScenarioAction("deleteTable", async (c) => {
    try {
      const client = new DynamoDBClient({});
      await client.send(new DeleteTableCommand({ TableName: NAMES.tableName }));
    } catch (e) {
      c.deleteTableError = e;
```

```
    }
  }),
  new ScenarioOutput("deleteTableResult", (state) => {
    if (state.deleteTableError) {
      console.error(state.deleteTableError);
      return MESSAGES.deleteTableError.replace(
        "${TABLE_NAME}",
        NAMES.tableName,
      );
    } else {
      return MESSAGES.deletedTable.replace("${TABLE_NAME}", NAMES.tableName);
    }
  }),
  new ScenarioAction("deleteKeyPair", async (state) => {
    try {
      const client = new EC2Client({});
      await client.send(
        new DeleteKeyPairCommand({ KeyName: NAMES.keyPairName }),
      );
      unlinkSync(`${NAMES.keyPairName}.pem`);
    } catch (e) {
      state.deleteKeyPairError = e;
    }
  }),
  new ScenarioOutput("deleteKeyPairResult", (state) => {
    if (state.deleteKeyPairError) {
      console.error(state.deleteKeyPairError);
      return MESSAGES.deleteKeyPairError.replace(
        "${KEY_PAIR_NAME}",
        NAMES.keyPairName,
      );
    } else {
      return MESSAGES.deletedKeyPair.replace(
        "${KEY_PAIR_NAME}",
        NAMES.keyPairName,
      );
    }
  }),
  new ScenarioAction("detachPolicyFromRole", async (state) => {
    try {
      const client = new IAMClient({});
      const policy = await findPolicy(NAMES.instancePolicyName);

      if (!policy) {
```

```
        state.detachPolicyFromRoleError = new Error(
          `Policy ${NAMES.instancePolicyName} not found.`,
        );
      } else {
        await client.send(
          new DetachRolePolicyCommand({
            RoleName: NAMES.instanceRoleName,
            PolicyArn: policy.Arn,
          }),
        );
      }
    } catch (e) {
      state.detachPolicyFromRoleError = e;
    }
  }),
  new ScenarioOutput("detachedPolicyFromRole", (state) => {
    if (state.detachPolicyFromRoleError) {
      console.error(state.detachPolicyFromRoleError);
      return MESSAGES.detachPolicyFromRoleError
        .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
        .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
    } else {
      return MESSAGES.detachedPolicyFromRole
        .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
        .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
    }
  }),
  new ScenarioAction("deleteInstancePolicy", async (state) => {
    const client = new IAMClient({});
    const policy = await findPolicy(NAMES.instancePolicyName);

    if (!policy) {
      state.deletePolicyError = new Error(
        `Policy ${NAMES.instancePolicyName} not found.`,
      );
    } else {
      return client.send(
        new DeletePolicyCommand({
          PolicyArn: policy.Arn,
        }),
      );
    }
  }),
  new ScenarioOutput("deletePolicyResult", (state) => {
```

```
    if (state.deletePolicyError) {
      console.error(state.deletePolicyError);
      return MESSAGES.deletePolicyError.replace(
        "${INSTANCE_POLICY_NAME}",
        NAMES.instancePolicyName,
      );
    } else {
      return MESSAGES.deletedPolicy.replace(
        "${INSTANCE_POLICY_NAME}",
        NAMES.instancePolicyName,
      );
    }
  }),
  new ScenarioAction("removeRoleFromInstanceProfile", async (state) => {
    try {
      const client = new IAMClient({});
      await client.send(
        new RemoveRoleFromInstanceProfileCommand({
          RoleName: NAMES.instanceRoleName,
          InstanceProfileName: NAMES.instanceProfileName,
        }),
      );
    } catch (e) {
      state.removeRoleFromInstanceProfileError = e;
    }
  }),
  new ScenarioOutput("removeRoleFromInstanceProfileResult", (state) => {
    if (state.removeRoleFromInstanceProfile) {
      console.error(state.removeRoleFromInstanceProfileError);
      return MESSAGES.removeRoleFromInstanceProfileError
        .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
        .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
    } else {
      return MESSAGES.removedRoleFromInstanceProfile
        .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
        .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
    }
  }),
  new ScenarioAction("deleteInstanceRole", async (state) => {
    try {
      const client = new IAMClient({});
      await client.send(
        new DeleteRoleCommand({
          RoleName: NAMES.instanceRoleName,
```

```
      }),
    );
  } catch (e) {
    state.deleteInstanceRoleError = e;
  }
}),
new ScenarioOutput("deleteInstanceRoleResult", (state) => {
  if (state.deleteInstanceRoleError) {
    console.error(state.deleteInstanceRoleError);
    return MESSAGES.deleteInstanceRoleError.replace(
      "${INSTANCE_ROLE_NAME}",
      NAMES.instanceRoleName,
    );
  } else {
    return MESSAGES.deletedInstanceRole.replace(
      "${INSTANCE_ROLE_NAME}",
      NAMES.instanceRoleName,
    );
  }
}),
new ScenarioAction("deleteInstanceProfile", async (state) => {
  try {
    const client = new IAMClient({});
    await client.send(
      new DeleteInstanceProfileCommand({
        InstanceProfileName: NAMES.instanceProfileName,
      }),
    );
  } catch (e) {
    state.deleteInstanceProfileError = e;
  }
}),
new ScenarioOutput("deleteInstanceProfileResult", (state) => {
  if (state.deleteInstanceProfileError) {
    console.error(state.deleteInstanceProfileError);
    return MESSAGES.deleteInstanceProfileError.replace(
      "${INSTANCE_PROFILE_NAME}",
      NAMES.instanceProfileName,
    );
  } else {
    return MESSAGES.deletedInstanceProfile.replace(
      "${INSTANCE_PROFILE_NAME}",
      NAMES.instanceProfileName,
    );
```

```
    }
  }),
  new ScenarioAction("deleteAutoScalingGroup", async (state) => {
    try {
      await terminateGroupInstances(NAMES.autoScalingGroupName);
      await retry({ intervalInMs: 60000, maxRetries: 60 }, async () => {
        await deleteAutoScalingGroup(NAMES.autoScalingGroupName);
      });
    } catch (e) {
      state.deleteAutoScalingGroupError = e;
    }
  }),
  new ScenarioOutput("deleteAutoScalingGroupResult", (state) => {
    if (state.deleteAutoScalingGroupError) {
      console.error(state.deleteAutoScalingGroupError);
      return MESSAGES.deleteAutoScalingGroupError.replace(
        "${AUTO_SCALING_GROUP_NAME}",
        NAMES.autoScalingGroupName,
      );
    } else {
      return MESSAGES.deletedAutoScalingGroup.replace(
        "${AUTO_SCALING_GROUP_NAME}",
        NAMES.autoScalingGroupName,
      );
    }
  }),
  new ScenarioAction("deleteLaunchTemplate", async (state) => {
    const client = new EC2Client({});
    try {
      await client.send(
        new DeleteLaunchTemplateCommand({
          LaunchTemplateName: NAMES.launchTemplateName,
        }),
      );
    } catch (e) {
      state.deleteLaunchTemplateError = e;
    }
  }),
  new ScenarioOutput("deleteLaunchTemplateResult", (state) => {
    if (state.deleteLaunchTemplateError) {
      console.error(state.deleteLaunchTemplateError);
      return MESSAGES.deleteLaunchTemplateError.replace(
        "${LAUNCH_TEMPLATE_NAME}",
        NAMES.launchTemplateName,
```

```
      );
    } else {
      return MESSAGES.deletedLaunchTemplate.replace(
        "${LAUNCH_TEMPLATE_NAME}",
        NAMES.launchTemplateName,
      );
    }
  }),
  new ScenarioAction("deleteLoadBalancer", async (state) => {
    try {
      const client = new ElasticLoadBalancingV2Client({});
      const loadBalancer = await findLoadBalancer(NAMES.loadBalancerName);
      await client.send(
        new DeleteLoadBalancerCommand({
          LoadBalancerArn: loadBalancer.LoadBalancerArn,
        }),
      );
      await retry({ intervalInMs: 1000, maxRetries: 60 }, async () => {
        const lb = await findLoadBalancer(NAMES.loadBalancerName);
        if (lb) {
          throw new Error("Load balancer still exists.");
        }
      });
    } catch (e) {
      state.deleteLoadBalancerError = e;
    }
  }),
  new ScenarioOutput("deleteLoadBalancerResult", (state) => {
    if (state.deleteLoadBalancerError) {
      console.error(state.deleteLoadBalancerError);
      return MESSAGES.deleteLoadBalancerError.replace(
        "${LB_NAME}",
        NAMES.loadBalancerName,
      );
    } else {
      return MESSAGES.deletedLoadBalancer.replace(
        "${LB_NAME}",
        NAMES.loadBalancerName,
      );
    }
  }),
  new ScenarioAction("deleteLoadBalancerTargetGroup", async (state) => {
    const client = new ElasticLoadBalancingV2Client({});
    try {
```

```
      const { TargetGroups } = await client.send(
        new DescribeTargetGroupsCommand({
          Names: [NAMES.loadBalancerTargetGroupName],
        }),
      );

      await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
        client.send(
          new DeleteTargetGroupCommand({
            TargetGroupArn: TargetGroups[0].TargetGroupArn,
          }),
        ),
      );
    } catch (e) {
      state.deleteLoadBalancerTargetGroupError = e;
    }
  }),
  new ScenarioOutput("deleteLoadBalancerTargetGroupResult", (state) => {
    if (state.deleteLoadBalancerTargetGroupError) {
      console.error(state.deleteLoadBalancerTargetGroupError);
      return MESSAGES.deleteLoadBalancerTargetGroupError.replace(
        "${TARGET_GROUP_NAME}",
        NAMES.loadBalancerTargetGroupName,
      );
    } else {
      return MESSAGES.deletedLoadBalancerTargetGroup.replace(
        "${TARGET_GROUP_NAME}",
        NAMES.loadBalancerTargetGroupName,
      );
    }
  }),
  new ScenarioAction("detachSsmOnlyRoleFromProfile", async (state) => {
    try {
      const client = new IAMClient({});
      await client.send(
        new RemoveRoleFromInstanceProfileCommand({
          InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
          RoleName: NAMES.ssmOnlyRoleName,
        }),
      );
    } catch (e) {
      state.detachSsmOnlyRoleFromProfileError = e;
    }
  }),
```

```
  new ScenarioOutput("detachSsmOnlyRoleFromProfileResult", (state) => {
    if (state.detachSsmOnlyRoleFromProfileError) {
      console.error(state.detachSsmOnlyRoleFromProfileError);
      return MESSAGES.detachSsmOnlyRoleFromProfileError
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
        .replace("${PROFILE_NAME}", NAMES.ssmOnlyInstanceProfileName);
    } else {
      return MESSAGES.detachedSsmOnlyRoleFromProfile
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
        .replace("${PROFILE_NAME}", NAMES.ssmOnlyInstanceProfileName);
    }
  }),
  new ScenarioAction("detachSsmOnlyCustomRolePolicy", async (state) => {
    try {
      const iamClient = new IAMClient({});
      const ssmOnlyPolicy = await findPolicy(NAMES.ssmOnlyPolicyName);
      await iamClient.send(
        new DetachRolePolicyCommand({
          RoleName: NAMES.ssmOnlyRoleName,
          PolicyArn: ssmOnlyPolicy.Arn,
        }),
      );
    } catch (e) {
      state.detachSsmOnlyCustomRolePolicyError = e;
    }
  }),
  new ScenarioOutput("detachSsmOnlyCustomRolePolicyResult", (state) => {
    if (state.detachSsmOnlyCustomRolePolicyError) {
      console.error(state.detachSsmOnlyCustomRolePolicyError);
      return MESSAGES.detachSsmOnlyCustomRolePolicyError
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
        .replace("${POLICY_NAME}", NAMES.ssmOnlyPolicyName);
    } else {
      return MESSAGES.detachedSsmOnlyCustomRolePolicy
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
        .replace("${POLICY_NAME}", NAMES.ssmOnlyPolicyName);
    }
  }),
  new ScenarioAction("detachSsmOnlyAWSRolePolicy", async (state) => {
    try {
      const iamClient = new IAMClient({});
      await iamClient.send(
        new DetachRolePolicyCommand({
          RoleName: NAMES.ssmOnlyRoleName,
```

```
          PolicyArn: "arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore",
        }),
      );
    } catch (e) {
      state.detachSsmOnlyAWSRolePolicyError = e;
    }
  }),
  new ScenarioOutput("detachSsmOnlyAWSRolePolicyResult", (state) => {
    if (state.detachSsmOnlyAWSRolePolicyError) {
      console.error(state.detachSsmOnlyAWSRolePolicyError);
      return MESSAGES.detachSsmOnlyAWSRolePolicyError
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
        .replace("${POLICY_NAME}", "AmazonSSMManagedInstanceCore");
    } else {
      return MESSAGES.detachedSsmOnlyAWSRolePolicy
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
        .replace("${POLICY_NAME}", "AmazonSSMManagedInstanceCore");
    }
  }),
  new ScenarioAction("deleteSsmOnlyInstanceProfile", async (state) => {
    try {
      const iamClient = new IAMClient({});
      await iamClient.send(
        new DeleteInstanceProfileCommand({
          InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
        }),
      );
    } catch (e) {
      state.deleteSsmOnlyInstanceProfileError = e;
    }
  }),
  new ScenarioOutput("deleteSsmOnlyInstanceProfileResult", (state) => {
    if (state.deleteSsmOnlyInstanceProfileError) {
      console.error(state.deleteSsmOnlyInstanceProfileError);
      return MESSAGES.deleteSsmOnlyInstanceProfileError.replace(
        "${INSTANCE_PROFILE_NAME}",
        NAMES.ssmOnlyInstanceProfileName,
      );
    } else {
      return MESSAGES.deletedSsmOnlyInstanceProfile.replace(
        "${INSTANCE_PROFILE_NAME}",
        NAMES.ssmOnlyInstanceProfileName,
      );
    }
```

```
  }),
  new ScenarioAction("deleteSsmOnlyPolicy", async (state) => {
    try {
      const iamClient = new IAMClient({});
      const ssmOnlyPolicy = await findPolicy(NAMES.ssmOnlyPolicyName);
      await iamClient.send(
        new DeletePolicyCommand({
          PolicyArn: ssmOnlyPolicy.Arn,
        }),
      );
    } catch (e) {
      state.deleteSsmOnlyPolicyError = e;
    }
  }),
  new ScenarioOutput("deleteSsmOnlyPolicyResult", (state) => {
    if (state.deleteSsmOnlyPolicyError) {
      console.error(state.deleteSsmOnlyPolicyError);
      return MESSAGES.deleteSsmOnlyPolicyError.replace(
        "${POLICY_NAME}",
        NAMES.ssmOnlyPolicyName,
      );
    } else {
      return MESSAGES.deletedSsmOnlyPolicy.replace(
        "${POLICY_NAME}",
        NAMES.ssmOnlyPolicyName,
      );
    }
  }),
  new ScenarioAction("deleteSsmOnlyRole", async (state) => {
    try {
      const iamClient = new IAMClient({});
      await iamClient.send(
        new DeleteRoleCommand({
          RoleName: NAMES.ssmOnlyRoleName,
        }),
      );
    } catch (e) {
      state.deleteSsmOnlyRoleError = e;
    }
  }),
  new ScenarioOutput("deleteSsmOnlyRoleResult", (state) => {
    if (state.deleteSsmOnlyRoleError) {
      console.error(state.deleteSsmOnlyRoleError);
      return MESSAGES.deleteSsmOnlyRoleError.replace(
```

```
        "${ROLE_NAME}",
        NAMES.ssmOnlyRoleName,
      );
    } else {
      return MESSAGES.deletedSsmOnlyRole.replace(
        "${ROLE_NAME}",
        NAMES.ssmOnlyRoleName,
      );
    }
  }),
  new ScenarioAction(
    "revokeSecurityGroupIngress",
    async (
      /** @type {{ myIp: string, defaultSecurityGroup: { GroupId: string } }} */
state,
    ) => {
      const ec2Client = new EC2Client({});

      try {
        await ec2Client.send(
          new RevokeSecurityGroupIngressCommand({
            GroupId: state.defaultSecurityGroup.GroupId,
            CidrIp: `${state.myIp}/32`,
            FromPort: 80,
            ToPort: 80,
            IpProtocol: "tcp",
          }),
        );
      } catch (e) {
        state.revokeSecurityGroupIngressError = e;
      }
    },
  ),
  new ScenarioOutput("revokeSecurityGroupIngressResult", (state) => {
    if (state.revokeSecurityGroupIngressError) {
      console.error(state.revokeSecurityGroupIngressError);
      return MESSAGES.revokeSecurityGroupIngressError.replace(
        "${IP}",
        state.myIp,
      );
    } else {
      return MESSAGES.revokedSecurityGroupIngress.replace("${IP}", state.myIp);
    }
  }),
```

```
];

/**
 * @param {string} policyName
 */
async function findPolicy(policyName) {
  const client = new IAMClient({});
  const paginatedPolicies = paginateListPolicies({ client }, {});
  for await (const page of paginatedPolicies) {
    const policy = page.Policies.find((p) => p.PolicyName === policyName);
    if (policy) {
      return policy;
    }
  }
}

/**
 * @param {string} groupName
 */
async function deleteAutoScalingGroup(groupName) {
  const client = new AutoScalingClient({});
  try {
    await client.send(
      new DeleteAutoScalingGroupCommand({
        AutoScalingGroupName: groupName,
      }),
    );
  } catch (err) {
    if (!(err instanceof Error)) {
      throw err;
    } else {
      console.log(err.name);
      throw err;
    }
  }
}

/**
 * @param {string} groupName
 */
async function terminateGroupInstances(groupName) {
  const autoScalingClient = new AutoScalingClient({});
  const group = await findAutoScalingGroup(groupName);
  await autoScalingClient.send(
```

```
    new UpdateAutoScalingGroupCommand({
      AutoScalingGroupName: group.AutoScalingGroupName,
      MinSize: 0,
    }),
  );
  for (const i of group.Instances) {
    await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
      autoScalingClient.send(
        new TerminateInstanceInAutoScalingGroupCommand({
          InstanceId: i.InstanceId,
          ShouldDecrementDesiredCapacity: true,
        }),
      ),
    );
  }
}

async function findAutoScalingGroup(groupName) {
  const client = new AutoScalingClient({});
  const paginatedGroups = paginateDescribeAutoScalingGroups({ client }, {});
  for await (const page of paginatedGroups) {
    const group = page.AutoScalingGroups.find(
      (g) => g.AutoScalingGroupName === groupName,
    );
    if (group) {
      return group;
    }
  }
  throw new Error(`Auto scaling group ${groupName} not found.`);
}
```

- 如需詳API細資訊，請參閱「AWS SDK for JavaScript API參考」中的下列主題。
  - AttachLoadBalancerTargetGroups
  - CreateAutoScalingGroup
  - CreateInstanceProfile
  - CreateLaunchTemplate
  - CreateListener
  - CreateLoadBalancer
  - CreateTargetGroup

- DeleteAutoScalingGroup
- DeleteInstanceProfile
- DeleteLaunchTemplate
- DeleteLoadBalancer
- DeleteTargetGroup
- DescribeAutoScalingGroups
- DescribeAvailabilityZones
- DescribeIamInstanceProfileAssociations
- DescribeInstances
- DescribeLoadBalancers
- DescribeSubnets
- DescribeTargetGroups
- DescribeTargetHealth
- DescribeVpcs
- RebootInstances
- ReplaceIamInstanceProfileAssociation
- TerminateInstanceInAutoScalingGroup
- UpdateAutoScalingGroup

建立使用者並擔任角色

下列程式碼範例示範如何建立使用者並擔任角色。

> **Warning**
>
> 為避免安全風險，在開發專用軟件或使IAM用真實數據時，請勿使用用戶進行身份驗證。相反地，搭配使用聯合功能和身分提供者，例如 AWS IAM Identity Center。

- 建立沒有許可的使用者。
- 建立一個可授予許可的角色，以列出帳戶的 Amazon S3 儲存貯體。
- 新增政策，讓使用者擔任該角色。
- 使用暫時憑證，擔任角色並列出 Amazon S3 儲存貯體，然後清理資源。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

建立IAM使用者和角色，以授與列出 Amazon S3 儲存貯體的權限。使用者只有擔任該角色的權利。擔任角色後，請使用暫時性憑證列出該帳戶的儲存貯體。

```
import {
  CreateUserCommand,
  GetUserCommand,
  CreateAccessKeyCommand,
  CreatePolicyCommand,
  CreateRoleCommand,
  AttachRolePolicyCommand,
  DeleteAccessKeyCommand,
  DeleteUserCommand,
  DeleteRoleCommand,
  DeletePolicyCommand,
  DetachRolePolicyCommand,
  IAMClient,
} from "@aws-sdk/client-iam";
import { ListBucketsCommand, S3Client } from "@aws-sdk/client-s3";
import { AssumeRoleCommand, STSClient } from "@aws-sdk/client-sts";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";
import { ScenarioInput } from "@aws-doc-sdk-examples/lib/scenario/index.js";

// Set the parameters.
const iamClient = new IAMClient({});
const userName = "test_name";
const policyName = "test_policy";
const roleName = "test_role";

/**
 * Create a new IAM user. If the user already exists, give
 * the option to delete and re-create it.
 * @param {string} name
 */
export const createUser = async (name, confirmAll = false) => {
```

```
  try {
    const { User } = await iamClient.send(
      new GetUserCommand({ UserName: name }),
    );
    const input = new ScenarioInput(
      "deleteUser",
      "Do you want to delete and remake this user?",
      { type: "confirm" },
    );
    const deleteUser = await input.handle({}, { confirmAll });
    // If the user exists, and you want to delete it, delete the user
    // and then create it again.
    if (deleteUser) {
      await iamClient.send(new DeleteUserCommand({ UserName: User.UserName }));
      await iamClient.send(new CreateUserCommand({ UserName: name }));
    } else {
      console.warn(
        `${name} already exists. The scenario may not work as expected.`,
      );
      return User;
    }
  } catch (caught) {
    // If there is no user by that name, create one.
    if (caught instanceof Error && caught.name === "NoSuchEntityException") {
      const { User } = await iamClient.send(
        new CreateUserCommand({ UserName: name }),
      );
      return User;
    } else {
      throw caught;
    }
  }
};

export const main = async (confirmAll = false) => {
  // Create a user. The user has no permissions by default.
  const User = await createUser(userName, confirmAll);

  if (!User) {
    throw new Error("User not created");
  }

  // Create an access key. This key is used to authenticate the new user to
```

```
  // Amazon Simple Storage Service (Amazon S3) and AWS Security Token Service (AWS
 STS).
  // It's not best practice to use access keys. For more information, see https://
aws.amazon.com/iam/resources/best-practices/.
  const createAccessKeyResponse = await iamClient.send(
    new CreateAccessKeyCommand({ UserName: userName }),
  );

  if (
    !createAccessKeyResponse.AccessKey?.AccessKeyId ||
    !createAccessKeyResponse.AccessKey?.SecretAccessKey
  ) {
    throw new Error("Access key not created");
  }

  const {
    AccessKey: { AccessKeyId, SecretAccessKey },
  } = createAccessKeyResponse;

  let s3Client = new S3Client({
    credentials: {
      accessKeyId: AccessKeyId,
      secretAccessKey: SecretAccessKey,
    },
  });

  // Retry the list buckets operation until it succeeds. InvalidAccessKeyId is
  // thrown while the user and access keys are still stabilizing.
  await retry({ intervalInMs: 1000, maxRetries: 300 }, async () => {
    try {
      return await listBuckets(s3Client);
    } catch (err) {
      if (err instanceof Error && err.name === "InvalidAccessKeyId") {
        throw err;
      }
    }
  });

  // Retry the create role operation until it succeeds. A MalformedPolicyDocument
 error
  // is thrown while the user and access keys are still stabilizing.
  const { Role } = await retry(
    {
      intervalInMs: 2000,
```

```
    maxRetries: 60,
  },
  () =>
    iamClient.send(
      new CreateRoleCommand({
        AssumeRolePolicyDocument: JSON.stringify({
          Version: "2012-10-17",
          Statement: [
            {
              Effect: "Allow",
              Principal: {
                // Allow the previously created user to assume this role.
                AWS: User.Arn,
              },
              Action: "sts:AssumeRole",
            },
          ],
        }),
        RoleName: roleName,
      }),
    ),
);

if (!Role) {
  throw new Error("Role not created");
}

// Create a policy that allows the user to list S3 buckets.
const { Policy: listBucketPolicy } = await iamClient.send(
  new CreatePolicyCommand({
    PolicyDocument: JSON.stringify({
      Version: "2012-10-17",
      Statement: [
        {
          Effect: "Allow",
          Action: ["s3:ListAllMyBuckets"],
          Resource: "*",
        },
      ],
    }),
    PolicyName: policyName,
  }),
);
```

```
if (!listBucketPolicy) {
  throw new Error("Policy not created");
}

// Attach the policy granting the 's3:ListAllMyBuckets' action to the role.
await iamClient.send(
  new AttachRolePolicyCommand({
    PolicyArn: listBucketPolicy.Arn,
    RoleName: Role.RoleName,
  }),
);

// Assume the role.
const stsClient = new STSClient({
  credentials: {
    accessKeyId: AccessKeyId,
    secretAccessKey: SecretAccessKey,
  },
});

// Retry the assume role operation until it succeeds.
const { Credentials } = await retry(
  { intervalInMs: 2000, maxRetries: 60 },
  () =>
    stsClient.send(
      new AssumeRoleCommand({
        RoleArn: Role.Arn,
        RoleSessionName: `iamBasicScenarioSession-${Math.floor(
          Math.random() * 1000000,
        )}`,
        DurationSeconds: 900,
      }),
    ),
);

if (!Credentials?.AccessKeyId || !Credentials?.SecretAccessKey) {
  throw new Error("Credentials not created");
}

s3Client = new S3Client({
  credentials: {
    accessKeyId: Credentials.AccessKeyId,
    secretAccessKey: Credentials.SecretAccessKey,
    sessionToken: Credentials.SessionToken,
```

```
    },
  });

  // List the S3 buckets again.
  // Retry the list buckets operation until it succeeds. AccessDenied might
  // be thrown while the role policy is still stabilizing.
  await retry({ intervalInMs: 2000, maxRetries: 60 }, () =>
    listBuckets(s3Client),
  );

  // Clean up.
  await iamClient.send(
    new DetachRolePolicyCommand({
      PolicyArn: listBucketPolicy.Arn,
      RoleName: Role.RoleName,
    }),
  );

  await iamClient.send(
    new DeletePolicyCommand({
      PolicyArn: listBucketPolicy.Arn,
    }),
  );

  await iamClient.send(
    new DeleteRoleCommand({
      RoleName: Role.RoleName,
    }),
  );

  await iamClient.send(
    new DeleteAccessKeyCommand({
      UserName: userName,
      AccessKeyId,
    }),
  );

  await iamClient.send(
    new DeleteUserCommand({
      UserName: userName,
    }),
  );
};
```

```
/**
 *
 * @param {S3Client} s3Client
 */
const listBuckets = async (s3Client) => {
  const { Buckets } = await s3Client.send(new ListBucketsCommand({}));

  if (!Buckets) {
    throw new Error("Buckets not listed");
  }

  console.log(Buckets.map((bucket) => bucket.Name).join("\n"));
};
```

- 如需詳API細資訊，請參閱「AWS SDK for JavaScript API參考」中的下列主題。
  - AttachRolePolicy
  - CreateAccessKey
  - CreatePolicy
  - CreateRole
  - CreateUser
  - DeleteAccessKey
  - DeletePolicy
  - DeleteRole
  - DeleteUser
  - DeleteUserPolicy
  - DetachRolePolicy
  - PutUserPolicy

Kinesis 示例使SDK用 JavaScript (v3)

下列程式碼範例說明如何使用 AWS SDK for JavaScript (v3) 搭配 Kinesis 來執行動作和實作常見案例。

每個範例都包含完整原始程式碼的連結，您可以在其中找到如何在內容中設定和執行程式碼的指示。

主題

無伺服器範例

使用 Kinesis 觸發條件調用 Lambda 函數

下列程式碼範例示範如何實作 Lambda 函數，該函數會接收從 Kinesis 串流接收記錄而觸發的事件。此函數會擷取 Kinesis 承載、從 Base64 解碼，並記錄記錄內容。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在無伺服器範例儲存庫中設定和執行。

使用 Lambda 使用 JavaScript的 Kinesis 事件。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
  for (const record of event.Records) {
    try {
      console.log(`Processed Kinesis Event - EventID: ${record.eventID}`);
      const recordData = await getRecordDataAsync(record.kinesis);
      console.log(`Record Data: ${recordData}`);
      // TODO: Do interesting work based on the new data
    } catch (err) {
      console.error(`An error occurred ${err}`);
      throw err;
    }
  }
  console.log(`Successfully processed ${event.Records.length} records.`);
};

async function getRecordDataAsync(payload) {
  var data = Buffer.from(payload.data, "base64").toString("utf-8");
  await Promise.resolve(1); //Placeholder for actual async work
  return data;
}
```

使用 Lambda 使用 TypeScript的 Kinesis 事件。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import {
  KinesisStreamEvent,
  Context,
  KinesisStreamHandler,
  KinesisStreamRecordPayload,
} from "aws-lambda";
import { Buffer } from "buffer";
import { Logger } from "@aws-lambda-powertools/logger";

const logger = new Logger({
  logLevel: "INFO",
  serviceName: "kinesis-stream-handler-sample",
});

export const functionHandler: KinesisStreamHandler = async (
  event: KinesisStreamEvent,
  context: Context
): Promise<void> => {
  for (const record of event.Records) {
    try {
      logger.info(`Processed Kinesis Event - EventID: ${record.eventID}`);
      const recordData = await getRecordDataAsync(record.kinesis);
      logger.info(`Record Data: ${recordData}`);
      // TODO: Do interesting work based on the new data
    } catch (err) {
      logger.error(`An error occurred ${err}`);
      throw err;
    }
    logger.info(`Successfully processed ${event.Records.length} records.`);
  }
};

async function getRecordDataAsync(
  payload: KinesisStreamRecordPayload
): Promise<string> {
  var data = Buffer.from(payload.data, "base64").toString("utf-8");
  await Promise.resolve(1); //Placeholder for actual async work
```

```
  return data;
}
```

使用 Kinesis 觸發條件報告 Lambda 函數的批次項目失敗

下列程式碼範例顯示如何針對接收來自 Kinesis 串流之事件的 Lambda 函數實作部分批次回應。此函數會在回應中報告批次項目失敗，指示 Lambda 稍後重試這些訊息。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在無伺服器範例儲存庫中設定和執行。

使用 Javascript 搭配 Lambda 報告 Kinesis 批次項目失敗。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
  for (const record of event.Records) {
    try {
      console.log(`Processed Kinesis Event - EventID: ${record.eventID}`);
      const recordData = await getRecordDataAsync(record.kinesis);
      console.log(`Record Data: ${recordData}`);
      // TODO: Do interesting work based on the new data
    } catch (err) {
      console.error(`An error occurred ${err}`);
      /* Since we are working with streams, we can return the failed item
 immediately.
           Lambda will immediately begin to retry processing from this failed item
 onwards. */
      return {
        batchItemFailures: [{ itemIdentifier: record.kinesis.sequenceNumber }],
      };
    }
  }
  console.log(`Successfully processed ${event.Records.length} records.`);
  return { batchItemFailures: [] };
};
```

```
async function getRecordDataAsync(payload) {
  var data = Buffer.from(payload.data, "base64").toString("utf-8");
  await Promise.resolve(1); //Placeholder for actual async work
  return data;
}
```

使用 Lambda 使用 TypeScript報告 Kinesis 批次項目失敗。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import {
  KinesisStreamEvent,
  Context,
  KinesisStreamHandler,
  KinesisStreamRecordPayload,
  KinesisStreamBatchResponse,
} from "aws-lambda";
import { Buffer } from "buffer";
import { Logger } from "@aws-lambda-powertools/logger";

const logger = new Logger({
  logLevel: "INFO",
  serviceName: "kinesis-stream-handler-sample",
});

export const functionHandler: KinesisStreamHandler = async (
  event: KinesisStreamEvent,
  context: Context
): Promise<KinesisStreamBatchResponse> => {
  for (const record of event.Records) {
    try {
      logger.info(`Processed Kinesis Event - EventID: ${record.eventID}`);
      const recordData = await getRecordDataAsync(record.kinesis);
      logger.info(`Record Data: ${recordData}`);
      // TODO: Do interesting work based on the new data
    } catch (err) {
      logger.error(`An error occurred ${err}`);
      /* Since we are working with streams, we can return the failed item
 immediately.
           Lambda will immediately begin to retry processing from this failed item
 onwards. */
      return {
```

```
        batchItemFailures: [{ itemIdentifier: record.kinesis.sequenceNumber }],
      };
    }
  }
  logger.info(`Successfully processed ${event.Records.length} records.`);
  return { batchItemFailures: [] };
};

async function getRecordDataAsync(
  payload: KinesisStreamRecordPayload
): Promise<string> {
  var data = Buffer.from(payload.data, "base64").toString("utf-8");
  await Promise.resolve(1); //Placeholder for actual async work
  return data;
}
```

使用SDK的 Lambda 示例 JavaScript (v3)

下列程式碼範例說明如何透過使用 AWS SDK for JavaScript (v3) 搭配 Lambda 來執行動作和實作常見案例。

Actions 是大型程式的程式碼摘錄，必須在內容中執行。雖然動作會顯示如何呼叫個別服務函數，但您可以在其相關案例中查看內容中的動作。

案例是程式碼範例，向您展示如何透過呼叫服務中的多個函式或與其他函式結合來完成特定工作 AWS 服務。

每個範例都包含完整原始程式碼的連結，您可以在其中找到如何在內容中設定和執行程式碼的指示。

開始使用

Hello Lambda

下列程式碼範例示範如何開始使用 Lambda。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import { LambdaClient, paginateListFunctions } from "@aws-sdk/client-lambda";

const client = new LambdaClient({});

export const helloLambda = async () => {
  const paginator = paginateListFunctions({ client }, {});
  const functions = [];

  for await (const page of paginator) {
    const funcNames = page.Functions.map((f) => f.FunctionName);
    functions.push(...funcNames);
  }

  console.log("Functions:");
  console.log(functions.join("\n"));
  return functions;
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考ListFunctions中的。

主題

- 動作
- 案例
- 無伺服器範例

動作

`CreateFunction`

下列程式碼範例會示範如何使用`CreateFunction`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
const createFunction = async (funcName, roleArn) => {
  const client = new LambdaClient({});
  const code = await readFile(`${dirname}../functions/${funcName}.zip`);

  const command = new CreateFunctionCommand({
    Code: { ZipFile: code },
    FunctionName: funcName,
    Role: roleArn,
    Architectures: [Architecture.arm64],
    Handler: "index.handler", // Required when sending a .zip file
    PackageType: PackageType.Zip, // Required when sending a .zip file
    Runtime: Runtime.nodejs16x, // Required when sending a .zip file
  });

  return client.send(command);
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考CreateFunction中的。

`DeleteFunction`

下列程式碼範例會示範如何使用`DeleteFunction`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
/**
 * @param {string} funcName
 */
const deleteFunction = (funcName) => {
  const client = new LambdaClient({});
  const command = new DeleteFunctionCommand({ FunctionName: funcName });
  return client.send(command);
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考DeleteFunction中的。

`GetFunction`

下列程式碼範例會示範如何使用`GetFunction`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
const getFunction = (funcName) => {
  const client = new LambdaClient({});
  const command = new GetFunctionCommand({ FunctionName: funcName });
  return client.send(command);
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考GetFunction中的。

`Invoke`

下列程式碼範例會示範如何使用`Invoke`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
const invoke = async (funcName, payload) => {
  const client = new LambdaClient({});
  const command = new InvokeCommand({
    FunctionName: funcName,
    Payload: JSON.stringify(payload),
    LogType: LogType.Tail,
  });

  const { Payload, LogResult } = await client.send(command);
  const result = Buffer.from(Payload).toString();
  const logs = Buffer.from(LogResult, "base64").toString();
  return { logs, result };
};
```

- 如需詳API細資訊，請參閱在AWS SDK for JavaScript API參考中呼叫。

`ListFunctions`

下列程式碼範例會示範如何使用`ListFunctions`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
const listFunctions = () => {
  const client = new LambdaClient({});
  const command = new ListFunctionsCommand({});
```

```
  return client.send(command);
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考ListFunctions中的。

`UpdateFunctionCode`

下列程式碼範例會示範如何使用`UpdateFunctionCode`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
const updateFunctionCode = async (funcName, newFunc) => {
  const client = new LambdaClient({});
  const code = await readFile(`${dirname}../functions/${newFunc}.zip`);
  const command = new UpdateFunctionCodeCommand({
    ZipFile: code,
    FunctionName: funcName,
    Architectures: [Architecture.arm64],
    Handler: "index.handler", // Required when sending a .zip file
    PackageType: PackageType.Zip, // Required when sending a .zip file
    Runtime: Runtime.nodejs16x, // Required when sending a .zip file
  });

  return client.send(command);
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考UpdateFunctionCode中的。

`UpdateFunctionConfiguration`

下列程式碼範例會示範如何使用`UpdateFunctionConfiguration`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
const updateFunctionConfiguration = (funcName) => {
  const client = new LambdaClient({});
  const config = readFileSync(`${dirname}../functions/config.json`).toString();
  const command = new UpdateFunctionConfigurationCommand({
    ...JSON.parse(config),
    FunctionName: funcName,
  });
  return client.send(command);
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考UpdateFunctionConfiguration中的。

案例

建立無伺服器應用程式來管理相片

下列程式碼範例示範如何建立無伺服器應用程式，讓使用者以標籤管理相片。

SDK對於 JavaScript（3）

顯示如何開發照片資產管理應用程式，以便使用 Amazon Rekognition 偵測圖片中的標籤，並將其儲存以供日後擷取。

有關如何設置和運行的完整源代碼和說明，請參閱中的完整示例 GitHub。

如要深入探索此範例的來源，請參閱 AWS 社群上的文章。

此範例中使用的服務

- API閘道
- DynamoDB

- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

建立應用程式以分析客戶意見回饋

下列程式碼範例會示範如何建立可分析客戶評論卡、從其原始語言進行翻譯、判斷對方情緒，以及透過翻譯後的文字產生音訊檔案的應用程式。

SDK對於 JavaScript（3）

此範例應用程式會分析和存儲客戶的意見回饋卡。具體來說，它滿足了紐約市一家虛構飯店的需求。飯店以實體評論卡的形式收到賓客以各種語言撰寫的意見回饋。這些意見回饋透過 Web 用戶端上傳至應用程式。評論卡的影像上傳後，系統會執行下列步驟：

- 文字內容是使用 Amazon Textract 從影像中擷取。
- Amazon Comprehend 會決定擷取文字及其用語的情感。
- 擷取的文字內容會使用 Amazon Translate 翻譯成英文。
- Amazon Polly 會使用擷取的文字內容合成音訊檔案。

完整的應用程式可透過 AWS CDK 部署。如需原始程式碼和部署指示，請參閱中的專案 GitHub。下列摘錄顯示如何在 AWS SDK for JavaScript Lambda 函數內部使用。

```
import {
  ComprehendClient,
  DetectDominantLanguageCommand,
  DetectSentimentCommand,
} from "@aws-sdk/client-comprehend";

/**
 * Determine the language and sentiment of the extracted text.
 *
 * @param {{ source_text: string}} extractTextOutput
 */
export const handler = async (extractTextOutput) => {
  const comprehendClient = new ComprehendClient({});

  const detectDominantLanguageCommand = new DetectDominantLanguageCommand({
    Text: extractTextOutput.source_text,
```

```
  });

  // The source language is required for sentiment analysis and
  // translation in the next step.
  const { Languages } = await comprehendClient.send(
    detectDominantLanguageCommand,
  );

  const languageCode = Languages[0].LanguageCode;

  const detectSentimentCommand = new DetectSentimentCommand({
    Text: extractTextOutput.source_text,
    LanguageCode: languageCode,
  });

  const { Sentiment } = await comprehendClient.send(detectSentimentCommand);

  return {
    sentiment: Sentiment,
    language_code: languageCode,
  };
};
```

```
import {
  DetectDocumentTextCommand,
  TextractClient,
} from "@aws-sdk/client-textract";

/**
 * Fetch the S3 object from the event and analyze it using Amazon Textract.
 *
 * @param {import("@types/aws-lambda").EventBridgeEvent<"Object Created">}
 eventBridgeS3Event
 */
export const handler = async (eventBridgeS3Event) => {
  const textractClient = new TextractClient();

  const detectDocumentTextCommand = new DetectDocumentTextCommand({
    Document: {
      S3Object: {
        Bucket: eventBridgeS3Event.bucket,
        Name: eventBridgeS3Event.object,
      },
```

```
    },
  });

  // Textract returns a list of blocks. A block can be a line, a page, word, etc.
  // Each block also contains geometry of the detected text.
  // For more information on the Block type, see https://docs.aws.amazon.com/
textract/latest/dg/API_Block.html.
  const { Blocks } = await textractClient.send(detectDocumentTextCommand);

  // For the purpose of this example, we are only interested in words.
  const extractedWords = Blocks.filter((b) => b.BlockType === "WORD").map(
    (b) => b.Text,
  );

  return extractedWords.join(" ");
};
```

```
import { PollyClient, SynthesizeSpeechCommand } from "@aws-sdk/client-polly";
import { S3Client } from "@aws-sdk/client-s3";
import { Upload } from "@aws-sdk/lib-storage";

/**
 * Synthesize an audio file from text.
 *
 * @param {{ bucket: string, translated_text: string, object: string}}
 sourceDestinationConfig
 */
export const handler = async (sourceDestinationConfig) => {
  const pollyClient = new PollyClient({});

  const synthesizeSpeechCommand = new SynthesizeSpeechCommand({
    Engine: "neural",
    Text: sourceDestinationConfig.translated_text,
    VoiceId: "Ruth",
    OutputFormat: "mp3",
  });

  const { AudioStream } = await pollyClient.send(synthesizeSpeechCommand);

  const audioKey = `${sourceDestinationConfig.object}.mp3`;

  // Store the audio file in S3.
  const s3Client = new S3Client();
```

```
  const upload = new Upload({
    client: s3Client,
    params: {
      Bucket: sourceDestinationConfig.bucket,
      Key: audioKey,
      Body: AudioStream,
      ContentType: "audio/mp3",
    },
  });

  await upload.done();
  return audioKey;
};
```

```
import {
  TranslateClient,
  TranslateTextCommand,
} from "@aws-sdk/client-translate";

/**
 * Translate the extracted text to English.
 *
 * @param {{ extracted_text: string, source_language_code: string}}
 textAndSourceLanguage
 */
export const handler = async (textAndSourceLanguage) => {
  const translateClient = new TranslateClient({});

  const translateCommand = new TranslateTextCommand({
    SourceLanguageCode: textAndSourceLanguage.source_language_code,
    TargetLanguageCode: "en",
    Text: textAndSourceLanguage.extracted_text,
  });

  const { TranslatedText } = await translateClient.send(translateCommand);

  return { translated_text: TranslatedText };
};
```

此範例中使用的服務

- Amazon Comprehend
- Lambda

- Amazon Polly
- Amazon Textract
- Amazon Translate

開始使用函數

以下程式碼範例顯示做法：

- 建立IAM角色和 Lambda 函數，然後上傳處理常式程式碼。
- 調用具有單一參數的函數並取得結果。
- 更新函數程式碼並使用環境變數進行設定。
- 調用具有新參數的函數並取得結果。顯示傳回的執行日誌。
- 列出您帳戶的函數，然後清理相關資源。

如需詳細資訊，請參閱使用主控台建立 Lambda 函數。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

建立 AWS Identity and Access Management (IAM) 角色，以授與 Lambda 寫入記錄的權限。

```
    log(`Creating role (${NAME_ROLE_LAMBDA})...`);
    const response = await createRole(NAME_ROLE_LAMBDA);

import { AttachRolePolicyCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} policyArn
 * @param {string} roleName
 */
export const attachRolePolicy = (policyArn, roleName) => {
```

```
  const command = new AttachRolePolicyCommand({
    PolicyArn: policyArn,
    RoleName: roleName,
  });

  return client.send(command);
};
```

建立 Lambda 函數並上傳處理常式程式碼。

```
const createFunction = async (funcName, roleArn) => {
  const client = new LambdaClient({});
  const code = await readFile(`${dirname}../functions/${funcName}.zip`);

  const command = new CreateFunctionCommand({
    Code: { ZipFile: code },
    FunctionName: funcName,
    Role: roleArn,
    Architectures: [Architecture.arm64],
    Handler: "index.handler", // Required when sending a .zip file
    PackageType: PackageType.Zip, // Required when sending a .zip file
    Runtime: Runtime.nodejs16x, // Required when sending a .zip file
  });

  return client.send(command);
};
```

調用具有單一參數的函數並取得結果。

```
const invoke = async (funcName, payload) => {
  const client = new LambdaClient({});
  const command = new InvokeCommand({
    FunctionName: funcName,
    Payload: JSON.stringify(payload),
    LogType: LogType.Tail,
  });

  const { Payload, LogResult } = await client.send(command);
  const result = Buffer.from(Payload).toString();
  const logs = Buffer.from(LogResult, "base64").toString();
  return { logs, result };
```

```
};
```

更新函數程式碼並設定具有環境變數的 Lambda 環境。

```
const updateFunctionCode = async (funcName, newFunc) => {
  const client = new LambdaClient({});
  const code = await readFile(`${dirname}../functions/${newFunc}.zip`);
  const command = new UpdateFunctionCodeCommand({
    ZipFile: code,
    FunctionName: funcName,
    Architectures: [Architecture.arm64],
    Handler: "index.handler", // Required when sending a .zip file
    PackageType: PackageType.Zip, // Required when sending a .zip file
    Runtime: Runtime.nodejs16x, // Required when sending a .zip file
  });

  return client.send(command);
};

const updateFunctionConfiguration = (funcName) => {
  const client = new LambdaClient({});
  const config = readFileSync(`${dirname}../functions/config.json`).toString();
  const command = new UpdateFunctionConfigurationCommand({
    ...JSON.parse(config),
    FunctionName: funcName,
  });
  return client.send(command);
};
```

列出您帳戶的函數。

```
const listFunctions = () => {
  const client = new LambdaClient({});
  const command = new ListFunctionsCommand({});

  return client.send(command);
};
```

刪除IAM角色和 Lambda 函數。

```
import { DeleteRoleCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} roleName
 */
export const deleteRole = (roleName) => {
  const command = new DeleteRoleCommand({ RoleName: roleName });
  return client.send(command);
};

/**
 * @param {string} funcName
 */
const deleteFunction = (funcName) => {
  const client = new LambdaClient({});
  const command = new DeleteFunctionCommand({ FunctionName: funcName });
  return client.send(command);
};
```

- 如需詳API細資訊，請參閱「AWS SDK for JavaScript API參考」中的下列主題。
  - CreateFunction
  - DeleteFunction
  - GetFunction
  - Invoke
  - ListFunctions
  - UpdateFunctionCode
  - UpdateFunctionConfiguration

從瀏覽器調用 Lambda 函數

下列程式碼範例會示範如何從瀏覽器叫用 AWS Lambda 函數。

SDK對於 JavaScript（第 2 個）

您可以建立以瀏覽器為基礎的應用程式，使用 AWS Lambda 函數更新具有使用者選擇的 Amazon DynamoDB 表格。

有關如何設置和運行的完整源代碼和說明，請參閱中的完整示例GitHub。

此範例中使用的服務

- DynamoDB
- Lambda

SDK對於 JavaScript（3）

您可以建立以瀏覽器為基礎的應用程式，使用 AWS Lambda 函數更新具有使用者選擇的 Amazon DynamoDB 表格。此應用程序使用 AWS SDK for JavaScript v3。

有關如何設置和運行的完整源代碼和說明，請參閱中的完整示例GitHub。

此範例中使用的服務

- DynamoDB
- Lambda

使用API閘道來叫用 Lambda 函數

下列程式碼範例顯示如何建立 Amazon API 閘道叫用的 AWS Lambda 函數。

SDK對於 JavaScript（3）

示範如何使用 Lambda JavaScript 執行階段建立 AWS Lambda 函數API。此範例會呼叫不同的 AWS 服務來執行特定使用案例。此範例示範如何建立 Amazon API 閘道叫用的 Lambda 函數，以掃描 Amazon DynamoDB 表是否有工作週年紀念日，並使用 Amazon 簡單通知服務 (AmazonSNS) 向您的員工傳送文字訊息，並在一年週年紀念日祝賀他們。

有關如何設置和運行的完整源代碼和說明，請參閱中的完整示例GitHub。

此範例也可在 AWS SDK for JavaScript v3 開發人員指南中取得。

此範例中使用的服務

- API閘道

- DynamoDB
- Lambda
- Amazon SNS

使用排程事件來調用 Lambda 函數

下列程式碼範例顯示如何建立 Amazon EventBridge 排程事件所叫用的 AWS Lambda 函數。

SDK對於 JavaScript（3）

說明如何建立叫用 AWS Lambda 函數的 Amazon EventBridge 排程事件。設定 EventBridge 為在叫用 Lambda 函數時使用 cron 運算式來排程。在此範例中，您可以使用 Lambda JavaScript 執行階段來建立 Lambda 函數API。此範例會呼叫不同的 AWS 服務來執行特定使用案例。此範例示範如何建立應用程式，將行動裝置文字訊息傳送給員工，在他們的週年紀念日向他們道賀。

有關如何設置和運行的完整源代碼和說明，請參閱中的完整示例GitHub。

此範例也可在 AWS SDK for JavaScript v3 開發人員指南中取得。

此範例中使用的服務

- DynamoDB
- EventBridge
- Lambda
- Amazon SNS

無伺服器範例

在 Lambda 函RDS數中連接到 Amazon 數據庫

下列程式碼範例示範如何實作連線至RDS資料庫的 Lambda 函數。該函數提出了一個簡單的數據庫請求，並返回結果。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在無伺服器範例儲存庫中設定和執行。

使用 Lambda 函RDS數連接到 Amazon 數據庫 JavaScript。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
/*
Node.js code here.
*/
// ES6+ example
import { Signer } from "@aws-sdk/rds-signer";
import mysql from 'mysql2/promise';

async function createAuthToken() {
  // Define connection authentication parameters
  const dbinfo = {

    hostname: process.env.ProxyHostName,
    port: process.env.Port,
    username: process.env.DBUserName,
    region: process.env.AWS_REGION,

  }

  // Create RDS Signer object
  const signer = new Signer(dbinfo);

  // Request authorization token from RDS, specifying the username
  const token = await signer.getAuthToken();
  return token;
}

async function dbOps() {

  // Obtain auth token
  const token = await createAuthToken();
  // Define connection configuration
  let connectionConfig = {
    host: process.env.ProxyHostName,
    user: process.env.DBUserName,
    password: token,
    database: process.env.DBName,
    ssl: 'Amazon RDS'
  }
  // Create the connection to the DB
  const conn = await mysql.createConnection(connectionConfig);
```

```
  // Obtain the result of the query
  const [res,] = await conn.execute('select ?+? as sum', [3, 2]);
  return res;

}

export const handler = async (event) => {
  // Execute database flow
  const result = await dbOps();
  // Return result
  return {
    statusCode: 200,
    body: JSON.stringify("The selected sum is: " + result[0].sum)
  }
};
```

使用 Lambda 函RDS數連接到 Amazon 數據庫 TypeScript。

```
import { Signer } from "@aws-sdk/rds-signer";
import mysql from 'mysql2/promise';

// RDS settings
// Using '!' (non-null assertion operator) to tell the TypeScript compiler that the
 DB settings are not null or undefined,
const proxy_host_name = process.env.PROXY_HOST_NAME!
const port = parseInt(process.env.PORT!)
const db_name = process.env.DB_NAME!
const db_user_name = process.env.DB_USER_NAME!
const aws_region = process.env.AWS_REGION!


async function createAuthToken(): Promise<string> {

    // Create RDS Signer object
    const signer = new Signer({
        hostname: proxy_host_name,
        port: port,
        region: aws_region,
        username: db_user_name
    });
```

```
    // Request authorization token from RDS, specifying the username
    const token = await signer.getAuthToken();
    return token;
}

async function dbOps(): Promise<mysql.QueryResult | undefined> {
    try {
        // Obtain auth token
        const token = await createAuthToken();
        const conn = await mysql.createConnection({
            host: proxy_host_name,
            user: db_user_name,
            password: token,
            database: db_name,
            ssl: 'Amazon RDS' // Ensure you have the CA bundle for SSL connection
        });
        const [rows, fields] = await conn.execute('SELECT ? + ? AS sum', [3, 2]);
        console.log('result:', rows);
        return rows;
    }
    catch (err) {
        console.log(err);
    }
}

export const lambdaHandler = async (event: any): Promise<{ statusCode: number; body:
 string }> => {
    // Execute database flow
    const result = await dbOps();

    // Return error is result is undefined
    if (result == undefined)
        return {
            statusCode: 500,
            body: JSON.stringify(`Error with connection to DB host`)
        }

    // Return result
    return {
        statusCode: 200,
        body: JSON.stringify(`The selected sum is: ${result[0].sum}`)
    };
};
```

使用 Kinesis 觸發條件調用 Lambda 函數

下列程式碼範例示範如何實作 Lambda 函數，該函數會接收從 Kinesis 串流接收記錄而觸發的事件。此函數會擷取 Kinesis 承載、從 Base64 解碼，並記錄記錄內容。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在無伺服器範例儲存庫中設定和執行。

使用 Lambda 使用 JavaScript的 Kinesis 事件。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
  for (const record of event.Records) {
    try {
      console.log(`Processed Kinesis Event - EventID: ${record.eventID}`);
      const recordData = await getRecordDataAsync(record.kinesis);
      console.log(`Record Data: ${recordData}`);
      // TODO: Do interesting work based on the new data
    } catch (err) {
      console.error(`An error occurred ${err}`);
      throw err;
    }
  }
  console.log(`Successfully processed ${event.Records.length} records.`);
};

async function getRecordDataAsync(payload) {
  var data = Buffer.from(payload.data, "base64").toString("utf-8");
  await Promise.resolve(1); //Placeholder for actual async work
  return data;
}
```

使用 Lambda 使用 TypeScript的 Kinesis 事件。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
```

```
import {
  KinesisStreamEvent,
  Context,
  KinesisStreamHandler,
  KinesisStreamRecordPayload,
} from "aws-lambda";
import { Buffer } from "buffer";
import { Logger } from "@aws-lambda-powertools/logger";

const logger = new Logger({
  logLevel: "INFO",
  serviceName: "kinesis-stream-handler-sample",
});

export const functionHandler: KinesisStreamHandler = async (
  event: KinesisStreamEvent,
  context: Context
): Promise<void> => {
  for (const record of event.Records) {
    try {
      logger.info(`Processed Kinesis Event - EventID: ${record.eventID}`);
      const recordData = await getRecordDataAsync(record.kinesis);
      logger.info(`Record Data: ${recordData}`);
      // TODO: Do interesting work based on the new data
    } catch (err) {
      logger.error(`An error occurred ${err}`);
      throw err;
    }
    logger.info(`Successfully processed ${event.Records.length} records.`);
  }
};

async function getRecordDataAsync(
  payload: KinesisStreamRecordPayload
): Promise<string> {
  var data = Buffer.from(payload.data, "base64").toString("utf-8");
  await Promise.resolve(1); //Placeholder for actual async work
  return data;
}
```

從 DynamoDB 觸發程序叫用 Lambda 函數

下列程式碼範例示範如何實作 Lambda 函數，該函數會接收透過從 DynamoDB 串流接收記錄而觸發的事件。此函數會擷取 DynamoDB 承載並記錄記錄內容。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在無伺服器範例儲存庫中設定和執行。

使用 Lambda 使用. JavaScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
    console.log(JSON.stringify(event, null, 2));
    event.Records.forEach(record => {
        logDynamoDBRecord(record);
    });
};

const logDynamoDBRecord = (record) => {
    console.log(record.eventID);
    console.log(record.eventName);
    console.log(`DynamoDB Record: ${JSON.stringify(record.dynamodb)}`);
};
```

使用 Lambda 使用. TypeScript

```
export const handler = async (event, context) => {
    console.log(JSON.stringify(event, null, 2));
    event.Records.forEach(record => {
        logDynamoDBRecord(record);
    });
}
const logDynamoDBRecord = (record) => {
    console.log(record.eventID);
    console.log(record.eventName);
    console.log(`DynamoDB Record: ${JSON.stringify(record.dynamodb)}`);
```

```
};
```

從 Amazon DocumentDB 觸發器調用 Lambda 函數

下列程式碼範例示範如何實作 Lambda 函數，此函數會接收由 DocumentDB 變更串流接收記錄所觸發的事件。該函數檢索 DocumentDB 有效載荷和記錄的內容。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在無伺服器範例儲存庫中設定和執行。

使用 Amazon DocumentDB 事件與 Lambda 使用. JavaScript

```
console.log('Loading function');
exports.handler = async (event, context) => {
    event.events.forEach(record => {
        logDocumentDBEvent(record);
    });
    return 'OK';
};

const logDocumentDBEvent = (record) => {
    console.log('Operation type: ' + record.event.operationType);
    console.log('db: ' + record.event.ns.db);
    console.log('collection: ' + record.event.ns.coll);
    console.log('Full document:', JSON.stringify(record.event.fullDocument, null,
 2));
};
```

使用使用 Lambda 使用 Amazon DocumentDB 事件 TypeScript

```
import { DocumentDBEventRecord, DocumentDBEventSubscriptionContext } from 'aws-
lambda';

console.log('Loading function');
```

```
export const handler = async (
  event: DocumentDBEventSubscriptionContext,
  context: any
): Promise<string> => {
  event.events.forEach((record: DocumentDBEventRecord) => {
    logDocumentDBEvent(record);
  });
  return 'OK';
};

const logDocumentDBEvent = (record: DocumentDBEventRecord): void => {
  console.log('Operation type: ' + record.event.operationType);
  console.log('db: ' + record.event.ns.db);
  console.log('collection: ' + record.event.ns.coll);
  console.log('Full document:', JSON.stringify(record.event.fullDocument, null, 2));
};
```

從 Amazon MSK 觸發器調用 Lambda 函數

下列程式碼範例示範如何實作 Lambda 函數，該函數會接收透過從 Amazon MSK 叢集接收記錄而觸發的事件。該函數檢索MSK有效負載並記錄記錄內容。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在無伺服器範例儲存庫中設定和執行。

消費 Amazon MSK 事件 Lambda 使用 JavaScript.

```
exports.handler = async (event) => {
    // Iterate through keys
    for (let key in event.records) {
      console.log('Key: ', key)
      // Iterate through records
      event.records[key].map((record) => {
        console.log('Record: ', record)
```

```
        // Decode base64
        const msg = Buffer.from(record.value, 'base64').toString()
        console.log('Message:', msg)
      })
    }
}
```

使用 Amazon S3 觸發條件調用 Lambda 函數

下列程式碼範例示範如何實作 Lambda 函數，該函數會接收透過將物件上傳至 S3 儲存貯體而觸發的事件。函數會從事件參數擷取 S3 儲存貯體名稱和物件金鑰，並呼叫 Amazon S3 API 以擷取和記錄物件的內容類型。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在無伺服器範例儲存庫中設定和執行。

使用使 Lambda JavaScript.

```
import { S3Client, HeadObjectCommand } from "@aws-sdk/client-s3";

const client = new S3Client();

export const handler = async (event, context) => {

    // Get the object from the event and show its content type
    const bucket = event.Records[0].s3.bucket.name;
    const key = decodeURIComponent(event.Records[0].s3.object.key.replace(/\+/g, '
 '));

    try {
        const { ContentType } = await client.send(new HeadObjectCommand({
            Bucket: bucket,
            Key: key,
        }));

        console.log('CONTENT TYPE:', ContentType);
        return ContentType;
```

```
    } catch (err) {
        console.log(err);
        const message = `Error getting object ${key} from bucket ${bucket}. Make
 sure they exist and your bucket is in the same region as this function.`;
        console.log(message);
        throw new Error(message);
    }
};
```

使用使 Lambda TypeScript.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { S3Event } from 'aws-lambda';
import { S3Client, HeadObjectCommand } from '@aws-sdk/client-s3';

const s3 = new S3Client({ region: process.env.AWS_REGION });

export const handler = async (event: S3Event): Promise<string | undefined> => {
  // Get the object from the event and show its content type
  const bucket = event.Records[0].s3.bucket.name;
  const key = decodeURIComponent(event.Records[0].s3.object.key.replace(/\+/g, '
 '));
  const params = {
    Bucket: bucket,
    Key: key,
  };
  try {
    const { ContentType } = await s3.send(new HeadObjectCommand(params));
    console.log('CONTENT TYPE:', ContentType);
    return ContentType;
  } catch (err) {
    console.log(err);
    const message = `Error getting object ${key} from bucket ${bucket}. Make sure
 they exist and your bucket is in the same region as this function.`;
    console.log(message);
    throw new Error(message);
  }
};
```

從 Amazon SNS 觸發器調用 Lambda 函數

下列程式碼範例示範如何實作 Lambda 函數，該函數會接收由接收來自SNS主題的訊息而觸發的事件。函數會從事件參數擷取訊息，並記錄每一則訊息的內容。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在無伺服器範例儲存庫中設定和執行。

使 Lambda JavaScript. SNS

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
  for (const record of event.Records) {
    await processMessageAsync(record);
  }
  console.info("done");
};

async function processMessageAsync(record) {
  try {
    const message = JSON.stringify(record.Sns.Message);
    console.log(`Processed message ${message}`);
    await Promise.resolve(1); //Placeholder for actual async work
  } catch (err) {
    console.error("An error occurred");
    throw err;
  }
}
```

使 Lambda TypeScript. SNS

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { SNSEvent, Context, SNSHandler, SNSEventRecord } from "aws-lambda";

export const functionHandler: SNSHandler = async (
```

```
  event: SNSEvent,
  context: Context
): Promise<void> => {
  for (const record of event.Records) {
    await processMessageAsync(record);
  }
  console.info("done");
};

async function processMessageAsync(record: SNSEventRecord): Promise<any> {
  try {
    const message: string = JSON.stringify(record.Sns.Message);
    console.log(`Processed message ${message}`);
    await Promise.resolve(1); //Placeholder for actual async work
  } catch (err) {
    console.error("An error occurred");
    throw err;
  }
}
```

從 Amazon SQS 觸發器調用 Lambda 函數

下列程式碼範例示範如何實作 Lambda 函數，該函數會接收透過從SQS佇列接收訊息觸發的事件。函數會從事件參數擷取訊息，並記錄每一則訊息的內容。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在無伺服器範例儲存庫中設定和執行。

使 Lambda JavaScript. SQS

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
  for (const message of event.Records) {
    await processMessageAsync(message);
  }
```

```
  console.info("done");
};

async function processMessageAsync(message) {
  try {
    console.log(`Processed message ${message.body}`);
    // TODO: Do interesting work based on the new message
    await Promise.resolve(1); //Placeholder for actual async work
  } catch (err) {
    console.error("An error occurred");
    throw err;
  }
}
```

使 Lambda TypeScript. SQS

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { SQSEvent, Context, SQSHandler, SQSRecord } from "aws-lambda";

export const functionHandler: SQSHandler = async (
  event: SQSEvent,
  context: Context
): Promise<void> => {
  for (const message of event.Records) {
    await processMessageAsync(message);
  }
  console.info("done");
};

async function processMessageAsync(message: SQSRecord): Promise<any> {
  try {
    console.log(`Processed message ${message.body}`);
    // TODO: Do interesting work based on the new message
    await Promise.resolve(1); //Placeholder for actual async work
  } catch (err) {
    console.error("An error occurred");
    throw err;
  }
}
```

使用 Kinesis 觸發條件報告 Lambda 函數的批次項目失敗

下列程式碼範例顯示如何針對接收來自 Kinesis 串流之事件的 Lambda 函數實作部分批次回應。此函數會在回應中報告批次項目失敗，指示 Lambda 稍後重試這些訊息。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在無伺服器範例儲存庫中設定和執行。

使用 Javascript 搭配 Lambda 報告 Kinesis 批次項目失敗。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
  for (const record of event.Records) {
    try {
      console.log(`Processed Kinesis Event - EventID: ${record.eventID}`);
      const recordData = await getRecordDataAsync(record.kinesis);
      console.log(`Record Data: ${recordData}`);
      // TODO: Do interesting work based on the new data
    } catch (err) {
      console.error(`An error occurred ${err}`);
      /* Since we are working with streams, we can return the failed item
 immediately.
           Lambda will immediately begin to retry processing from this failed item
 onwards. */
      return {
        batchItemFailures: [{ itemIdentifier: record.kinesis.sequenceNumber }],
      };
    }
  }
  console.log(`Successfully processed ${event.Records.length} records.`);
  return { batchItemFailures: [] };
};

async function getRecordDataAsync(payload) {
  var data = Buffer.from(payload.data, "base64").toString("utf-8");
  await Promise.resolve(1); //Placeholder for actual async work
  return data;
}
```

使用 Lambda 使用 TypeScript報告 Kinesis 批次項目失敗。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import {
  KinesisStreamEvent,
  Context,
  KinesisStreamHandler,
  KinesisStreamRecordPayload,
  KinesisStreamBatchResponse,
} from "aws-lambda";
import { Buffer } from "buffer";
import { Logger } from "@aws-lambda-powertools/logger";

const logger = new Logger({
  logLevel: "INFO",
  serviceName: "kinesis-stream-handler-sample",
});

export const functionHandler: KinesisStreamHandler = async (
  event: KinesisStreamEvent,
  context: Context
): Promise<KinesisStreamBatchResponse> => {
  for (const record of event.Records) {
    try {
      logger.info(`Processed Kinesis Event - EventID: ${record.eventID}`);
      const recordData = await getRecordDataAsync(record.kinesis);
      logger.info(`Record Data: ${recordData}`);
      // TODO: Do interesting work based on the new data
    } catch (err) {
      logger.error(`An error occurred ${err}`);
      /* Since we are working with streams, we can return the failed item
 immediately.
           Lambda will immediately begin to retry processing from this failed item
 onwards. */
      return {
        batchItemFailures: [{ itemIdentifier: record.kinesis.sequenceNumber }],
      };
    }
  }
  logger.info(`Successfully processed ${event.Records.length} records.`);
```

```
  return { batchItemFailures: [] };
};

async function getRecordDataAsync(
  payload: KinesisStreamRecordPayload
): Promise<string> {
  var data = Buffer.from(payload.data, "base64").toString("utf-8");
  await Promise.resolve(1); //Placeholder for actual async work
  return data;
}
```

使用 DynamoDB 觸發程序報告 Lambda 函數的批次項目失敗

下列程式碼範例示範如何針對接收來自 DynamoDB 串流之事件的 Lambda 函數實作部分批次回應。此函數會在回應中報告批次項目失敗，指示 Lambda 稍後重試這些訊息。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在無伺服器範例儲存庫中設定和執行。

使用 Lambda 使用報告批次項目失敗 JavaScript

```
export const handler = async (event) => {
  const records = event.Records;
  let curRecordSequenceNumber = "";

  for (const record of records) {
    try {
      // Process your record
      curRecordSequenceNumber = record.dynamodb.SequenceNumber;
    } catch (e) {
      // Return failed record's sequence number
      return { batchItemFailures: [{ itemIdentifier: curRecordSequenceNumber }] };
    }
  }

  return { batchItemFailures: [] };
```

```
};
```

使用 Lambda 使用報告批次項目失敗 TypeScript

```
import {
  DynamoDBBatchResponse,
  DynamoDBBatchItemFailure,
  DynamoDBStreamEvent,
} from "aws-lambda";

export const handler = async (
  event: DynamoDBStreamEvent
): Promise<DynamoDBBatchResponse> => {
  const batchItemFailures: DynamoDBBatchItemFailure[] = [];
  let curRecordSequenceNumber;

  for (const record of event.Records) {
    curRecordSequenceNumber = record.dynamodb?.SequenceNumber;

    if (curRecordSequenceNumber) {
      batchItemFailures.push({
        itemIdentifier: curRecordSequenceNumber,
      });
    }
  }

  return { batchItemFailures: batchItemFailures };
};
```

使用 Amazon SQS 觸發器報告 Lambda 函數的批次項目失敗

下列程式碼範例示範如何針對接收SQS佇列事件的 Lambda 函數實作部分批次回應。此函數會在回應中報告批次項目失敗，指示 Lambda 稍後重試這些訊息。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在無伺服器範例儲存庫中設定和執行。

使用 Lambda 使用 JavaScript. SQS

```
// Node.js 20.x Lambda runtime, AWS SDK for Javascript V3
export const handler = async (event, context) => {
    const batchItemFailures = [];
    for (const record of event.Records) {
        try {
            await processMessageAsync(record, context);
        } catch (error) {
            batchItemFailures.push({ itemIdentifier: record.messageId });
        }
    }
    return { batchItemFailures };
};

async function processMessageAsync(record, context) {
    if (record.body && record.body.includes("error")) {
        throw new Error("There is an error in the SQS Message.");
    }
    console.log(`Processed message: ${record.body}`);
}
```

使用 Lambda 使用 TypeScript. SQS

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { SQSEvent, SQSBatchResponse, Context, SQSBatchItemFailure, SQSRecord } from
 'aws-lambda';

export const handler = async (event: SQSEvent, context: Context):
 Promise<SQSBatchResponse> => {
    const batchItemFailures: SQSBatchItemFailure[] = [];

    for (const record of event.Records) {
        try {
            await processMessageAsync(record);
        } catch (error) {
            batchItemFailures.push({ itemIdentifier: record.messageId });
        }
    }

    return {batchItemFailures: batchItemFailures};
};
```

```
async function processMessageAsync(record: SQSRecord): Promise<void> {
    if (record.body && record.body.includes("error")) {
        throw new Error('There is an error in the SQS Message.');
    }
    console.log(`Processed message ${record.body}`);
}
```

Amazon Lex 示SDK例使用 JavaScript （ v3 ）

下列程式碼範例說明如何透過使用 AWS SDK for JavaScript (v3) 搭配 Amazon Lex 執行動作和實作常見案例。

案例是程式碼範例，向您展示如何透過呼叫服務中的多個函式或與其他函式結合來完成特定工作 AWS 服務。

每個範例都包含完整原始程式碼的連結，您可以在其中找到如何在內容中設定和執行程式碼的指示。

主題

案例

建立 Amazon Lex 聊天機器人

以下代碼示例演示瞭如何創建聊天機器人以吸引您的網站訪問者。

SDK對於 JavaScript （ 3 ）

示範如何使用 Amazon Lex 在 Web 應API用程式中建立 Chatbot，以吸引您的網站訪客。

如需有關如何設定和執行的完整原始程式碼和指示，請參閱開發人 AWS SDK for JavaScript 員指南中的建立 Amazon Lex 聊天機器人的完整範例。

此範例中使用的服務

- Amazon Comprehend
- Amazon Lex
- Amazon Translate

Amazon MSK 示例使用 SDK for JavaScript (v3)

下列程式碼範例說明如何透過使用 AWS SDK for JavaScript (v3) 搭配 Amazon 來執行動作和實作常見案例MSK。

每個範例都包含完整原始程式碼的連結，您可以在其中找到如何在內容中設定和執行程式碼的指示。

主題

無伺服器範例

從 Amazon MSK 觸發器調用 Lambda 函數

下列程式碼範例示範如何實作 Lambda 函數，該函數會接收透過從 Amazon MSK 叢集接收記錄而觸發的事件。該函數檢索MSK有效負載並記錄記錄內容。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在無伺服器範例儲存庫中設定和執行。

消費 Amazon MSK 事件 Lambda 使用 JavaScript.

```
exports.handler = async (event) => {
    // Iterate through keys
    for (let key in event.records) {
      console.log('Key: ', key)
      // Iterate through records
      event.records[key].map((record) => {
        console.log('Record: ', record)
        // Decode base64
        const msg = Buffer.from(record.value, 'base64').toString()
        console.log('Message:', msg)
      })
    }
}
```

Amazon Personalize 化示例使用 SDK for JavaScript (v3)

下列程式碼範例說明如何透過將 AWS SDK for JavaScript (v3) 與 Amazon Personalize 搭配使用，以執行動作和實作常見案例。

Actions 是大型程式的程式碼摘錄，必須在內容中執行。雖然動作會顯示如何呼叫個別服務函數，但您可以在其相關案例中查看內容中的動作。

每個範例都包含完整原始程式碼的連結，您可以在其中找到如何在內容中設定和執行程式碼的指示。

主題

動作

`CreateBatchInferenceJob`

下列程式碼範例會示範如何使用`CreateBatchInferenceJob`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
// Get service clients module and commands using ES6 syntax.
import { CreateBatchInferenceJobCommand } from
  "@aws-sdk/client-personalize";
import { personalizeClient } from "./libs/personalizeClients.js";

// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

// Set the batch inference job's parameters.

export const createBatchInferenceJobParam = {
  jobName: 'JOB_NAME',
  jobInput: {         /* required */
    s3DataSource: {   /* required */
```

```
      path: 'INPUT_PATH', /* required */
      // kmsKeyArn: 'INPUT_KMS_KEY_ARN' /* optional */'
    }
  },
  jobOutput: {         /* required */
    s3DataDestination: {   /* required */
      path: 'OUTPUT_PATH', /* required */
      // kmsKeyArn: 'OUTPUT_KMS_KEY_ARN' /* optional */'
    }
  },
  roleArn: 'ROLE_ARN', /* required */
  solutionVersionArn: 'SOLUTION_VERSION_ARN', /* required */
  numResults: 20 /* optional integer*/
};

export const run = async () => {
  try {
    const response = await personalizeClient.send(new
 CreateBatchInferenceJobCommand(createBatchInferenceJobParam));
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考CreateBatchInferenceJob中的。

`CreateBatchSegmentJob`

下列程式碼範例會示範如何使用`CreateBatchSegmentJob`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
// Get service clients module and commands using ES6 syntax.
import { CreateBatchSegmentJobCommand } from
  "@aws-sdk/client-personalize";
import { personalizeClient } from "./libs/personalizeClients.js";

// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

// Set the batch segment job's parameters.

export const createBatchSegmentJobParam = {
  jobName: 'NAME',
  jobInput: {         /* required */
    s3DataSource: {   /* required */
      path: 'INPUT_PATH', /* required */
      // kmsKeyArn: 'INPUT_KMS_KEY_ARN' /* optional */'
    }
  },
  jobOutput: {         /* required */
    s3DataDestination: {   /* required */
      path: 'OUTPUT_PATH', /* required */
      // kmsKeyArn: 'OUTPUT_KMS_KEY_ARN' /* optional */'
    }
  },
  roleArn: 'ROLE_ARN', /* required */
  solutionVersionArn: 'SOLUTION_VERSION_ARN', /* required */
  numResults: 20 /* optional */
};

export const run = async () => {
  try {
    const response = await personalizeClient.send(new
 CreateBatchSegmentJobCommand(createBatchSegmentJobParam));
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考CreateBatchSegmentJob中的。

`CreateCampaign`

下列程式碼範例會示範如何使用`CreateCampaign`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
// Get service clients module and commands using ES6 syntax.

import { CreateCampaignCommand } from
  "@aws-sdk/client-personalize";
import { personalizeClient } from "./libs/personalizeClients.js";

// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

// Set the campaign's parameters.
export const createCampaignParam = {
  solutionVersionArn: 'SOLUTION_VERSION_ARN', /* required */
  name: 'NAME',  /* required */
  minProvisionedTPS: 1    /* optional integer */
}

export const run = async () => {
  try {
    const response = await personalizeClient.send(new
 CreateCampaignCommand(createCampaignParam));
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考CreateCampaign中的。

`CreateDataset`

下列程式碼範例會示範如何使用`CreateDataset`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
// Get service clients module and commands using ES6 syntax.
import { CreateDatasetCommand } from
  "@aws-sdk/client-personalize";
import { personalizeClient } from "./libs/personalizeClients.js";

// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

// Set the dataset's parameters.
export const createDatasetParam = {
  datasetGroupArn: 'DATASET_GROUP_ARN', /* required */
  datasetType: 'DATASET_TYPE', /* required */
  name: 'NAME', /* required */
  schemaArn: 'SCHEMA_ARN' /* required */
}

export const run = async () => {
  try {
    const response = await personalizeClient.send(new
 CreateDatasetCommand(createDatasetParam));
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考CreateDataset中的。

`CreateDatasetExportJob`

下列程式碼範例會示範如何使用`CreateDatasetExportJob`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
// Get service clients module and commands using ES6 syntax.
import { CreateDatasetExportJobCommand } from
  "@aws-sdk/client-personalize";
import { personalizeClient } from "./libs/personalizeClients.js";

// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

// Set the export job parameters.
export const datasetExportJobParam = {
  datasetArn: 'DATASET_ARN', /* required */
  jobOutput: {
    s3DataDestination: {
        path: 'S3_DESTINATION_PATH' /* required */
        //kmsKeyArn: 'ARN'  /* include if your bucket uses AWS KMS for encryption
    }
  },
  jobName: 'NAME',/* required */
  roleArn: 'ROLE_ARN' /* required */
}

export const run = async () => {
  try {
    const response = await personalizeClient.send(new
 CreateDatasetExportJobCommand(datasetExportJobParam));
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
```

```
run();
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考CreateDatasetExportJob中的。

`CreateDatasetGroup`

下列程式碼範例會示範如何使用`CreateDatasetGroup`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
// Get service clients module and commands using ES6 syntax.

import { CreateDatasetGroupCommand } from "@aws-sdk/client-personalize";
import { personalizeClient } from "./libs/personalizeClients.js";

// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

// Set the dataset group parameters.
export const createDatasetGroupParam = {
  name: "NAME" /* required */,
};

export const run = async (createDatasetGroupParam) => {
  try {
    const response = await personalizeClient.send(
      new CreateDatasetGroupCommand(createDatasetGroupParam),
    );
    console.log("Success", response);
    return "Run successfully"; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
```

```
run(createDatasetGroupParam);
```

建立網域資料集群組。

```
// Get service clients module and commands using ES6 syntax.
import {  CreateDatasetGroupCommand } from
  "@aws-sdk/client-personalize";
import { personalizeClient } from "./libs/personalizeClients.js";

// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

// Set the domain dataset group parameters.
export const domainDatasetGroupParams = {
  name: 'NAME',  /* required */
  domain: 'DOMAIN'   /* required for a domain dsg, specify ECOMMERCE or
 VIDEO_ON_DEMAND */
}

export const run = async () => {
  try {
    const response = await personalizeClient.send(new
 CreateDatasetGroupCommand(domainDatasetGroupParams));
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考CreateDatasetGroup中的。

`CreateDatasetImportJob`

下列程式碼範例會示範如何使用`CreateDatasetImportJob`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
// Get service clients module and commands using ES6 syntax.
import {CreateDatasetImportJobCommand } from
  "@aws-sdk/client-personalize";
import { personalizeClient } from "./libs/personalizeClients.js";

// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

// Set the dataset import job parameters.
export const datasetImportJobParam = {
  datasetArn: 'DATASET_ARN', /* required */
  dataSource: {  /* required */
    dataLocation: 'S3_PATH'
  },
  jobName: 'NAME',/* required */
  roleArn: 'ROLE_ARN' /* required */
}

export const run = async () => {
  try {
    const response = await personalizeClient.send(new
 CreateDatasetImportJobCommand(datasetImportJobParam));
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考CreateDatasetImportJob中的。

`CreateEventTracker`

下列程式碼範例會示範如何使用`CreateEventTracker`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
// Get service clients module and commands using ES6 syntax.
import { CreateEventTrackerCommand } from
  "@aws-sdk/client-personalize";
import { personalizeClient } from "./libs/personalizeClients.js";

// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

// Set the event tracker's parameters.
export const createEventTrackerParam = {
  datasetGroupArn: 'DATASET_GROUP_ARN', /* required */
  name: 'NAME', /* required */
}

export const run = async () => {
  try {
    const response = await personalizeClient.send(new
 CreateEventTrackerCommand(createEventTrackerParam));
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考CreateEventTracker中的。

`CreateFilter`

下列程式碼範例會示範如何使用`CreateFilter`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
// Get service clients module and commands using ES6 syntax.
import { CreateFilterCommand } from
  "@aws-sdk/client-personalize";
import { personalizeClient } from "./libs/personalizeClients.js";
// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

// Set the filter's parameters.
export const createFilterParam = {
  datasetGroupArn: 'DATASET_GROUP_ARN', /* required */
  name: 'NAME', /* required */
  filterExpression: 'FILTER_EXPRESSION' /*required */
}

export const run = async () => {
  try {
    const response = await personalizeClient.send(new
 CreateFilterCommand(createFilterParam));
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考CreateFilter中的。

`CreateRecommender`

下列程式碼範例會示範如何使用`CreateRecommender`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
// Get service clients module and commands using ES6 syntax.
import { CreateRecommenderCommand } from
  "@aws-sdk/client-personalize";
import { personalizeClient } from "./libs/personalizeClients.js";

// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

// Set the recommender's parameters.
export const createRecommenderParam = {
  name: 'NAME', /* required */
  recipeArn: 'RECIPE_ARN', /* required */
  datasetGroupArn: 'DATASET_GROUP_ARN'  /* required */
}

export const run = async () => {
  try {
    const response = await personalizeClient.send(new
 CreateRecommenderCommand(createRecommenderParam));
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考CreateRecommender中的。

`CreateSchema`

下列程式碼範例會示範如何使用`CreateSchema`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
// Get service clients module and commands using ES6 syntax.
import { CreateSchemaCommand } from
  "@aws-sdk/client-personalize";
import { personalizeClient } from "./libs/personalizeClients.js";

// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

import fs from 'fs';

let schemaFilePath = "SCHEMA_PATH";
let mySchema = "";

try {
  mySchema = fs.readFileSync(schemaFilePath).toString();
} catch (err) {
  mySchema = 'TEST' // For unit tests.
}
// Set the schema parameters.
export const createSchemaParam = {
  name: 'NAME', /* required */
  schema: mySchema /* required */
};

export const run = async () => {
  try {
    const response = await personalizeClient.send(new
 CreateSchemaCommand(createSchemaParam));
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
```

```
    console.log("Error", err);
  }
};
run();
```

使用網域建立結構描述。

```
// Get service clients module and commands using ES6 syntax.
import { CreateSchemaCommand } from
  "@aws-sdk/client-personalize";
import { personalizeClient } from "./libs/personalizeClients.js";

// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

import fs from 'fs';

let schemaFilePath = "SCHEMA_PATH";
let mySchema = "";

try {
  mySchema = fs.readFileSync(schemaFilePath).toString();
} catch (err) {
  mySchema = 'TEST' // for unit tests.
}

// Set the domain schema parameters.
export const createDomainSchemaParam = {
  name: 'NAME', /* required */
  schema: mySchema, /* required */
  domain: 'DOMAIN'   /* required for a domain dataset group, specify ECOMMERCE or
 VIDEO_ON_DEMAND */
};

export const run = async () => {
  try {
    const response = await personalizeClient.send(new
 CreateSchemaCommand(createDomainSchemaParam));
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
```

```
  }
};
run();
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考CreateSchema中的。

`CreateSolution`

下列程式碼範例會示範如何使用`CreateSolution`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
// Get service clients module and commands using ES6 syntax.
import { CreateSolutionCommand } from
  "@aws-sdk/client-personalize";
import { personalizeClient } from "./libs/personalizeClients.js";
// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

// Set the solution parameters.
export const createSolutionParam = {
  datasetGroupArn: 'DATASET_GROUP_ARN', /* required */
  recipeArn: 'RECIPE_ARN', /* required */
  name: 'NAME' /* required */
}

export const run = async () => {
  try {
    const response = await personalizeClient.send(new
 CreateSolutionCommand(createSolutionParam));
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
```

```
  }
};
run();
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考CreateSolution中的。

`CreateSolutionVersion`

下列程式碼範例會示範如何使用`CreateSolutionVersion`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
// Get service clients module and commands using ES6 syntax.
import { CreateSolutionVersionCommand } from
  "@aws-sdk/client-personalize";
import { personalizeClient } from "./libs/personalizeClients.js";
// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

// Set the solution version parameters.
export const solutionVersionParam = {
  solutionArn: 'SOLUTION_ARN' /* required */
}

export const run = async () => {
  try {
    const response = await personalizeClient.send(new
 CreateSolutionVersionCommand(solutionVersionParam));
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考CreateSolutionVersion中的。

Amazon Personalize 化事件示SDK例使用 JavaScript in (v3)

下列程式碼範例說明如何透過使用 AWS SDK for JavaScript (v3) 搭配 Amazon Personalize 事件來執行動作和實作常見案例。

Actions 是大型程式的程式碼摘錄，必須在內容中執行。雖然動作會顯示如何呼叫個別服務函數，但您可以在其相關案例中查看內容中的動作。

每個範例都包含完整原始程式碼的連結，您可以在其中找到如何在內容中設定和執行程式碼的指示。

主題

- 動作

動作

`PutEvents`

下列程式碼範例會示範如何使用`PutEvents`。

SDK對於 JavaScript （ 3 ）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
// Get service clients module and commands using ES6 syntax.
import { PutEventsCommand } from "@aws-sdk/client-personalize-events";
import { personalizeEventsClient } from "./libs/personalizeClients.js";
// Or, create the client here.
// const personalizeEventsClient = new PersonalizeEventsClient({ region: "REGION"});

// Convert your UNIX timestamp to a Date.
const sentAtDate = new Date(1613443801 * 1000); // 1613443801 is a testing value.
 Replace it with your sentAt timestamp in UNIX format.
```

```
// Set put events parameters.
var putEventsParam = {
  eventList: [
    /* required */
    {
      eventType: "EVENT_TYPE" /* required */,
      sentAt: sentAtDate /* required, must be a Date with js */,
      eventId: "EVENT_ID" /* optional */,
      itemId: "ITEM_ID" /* optional */,
    },
  ],
  sessionId: "SESSION_ID" /* required */,
  trackingId: "TRACKING_ID" /* required */,
  userId: "USER_ID" /* required */,
};
export const run = async () => {
  try {
    const response = await personalizeEventsClient.send(
      new PutEventsCommand(putEventsParam),
    );
    console.log("Success!", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考PutEvents中的。

`PutItems`

下列程式碼範例會示範如何使用`PutItems`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
// Get service clients module and commands using ES6 syntax.
import { PutItemsCommand } from "@aws-sdk/client-personalize-events";
import { personalizeEventsClient } from "./libs/personalizeClients.js";
// Or, create the client here.
// const personalizeEventsClient = new PersonalizeEventsClient({ region: "REGION"});

// Set the put items parameters. For string properties and values, use the \
 character to escape quotes.
var putItemsParam = {
  datasetArn: "DATASET_ARN" /* required */,
  items: [
    /* required */
    {
      itemId: "ITEM_ID" /*  required */,
      properties:
        '{"PROPERTY1_NAME": "PROPERTY1_VALUE", "PROPERTY2_NAME": "PROPERTY2_VALUE",
 "PROPERTY3_NAME": "PROPERTY3_VALUE"}' /* optional */,
    },
  ],
};
export const run = async () => {
  try {
    const response = await personalizeEventsClient.send(
      new PutItemsCommand(putItemsParam),
    );
    console.log("Success!", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考PutItems中的。

PutUsers

下列程式碼範例會示範如何使用PutUsers。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
// Get service clients module and commands using ES6 syntax.
import { PutUsersCommand } from "@aws-sdk/client-personalize-events";
import { personalizeEventsClient } from "./libs/personalizeClients.js";
// Or, create the client here.
// const personalizeEventsClient = new PersonalizeEventsClient({ region: "REGION"});

// Set the put users parameters. For string properties and values, use the \
 character to escape quotes.
var putUsersParam = {
  datasetArn: "DATASET_ARN",
  users: [
    {
      userId: "USER_ID",
      properties: '{"PROPERTY1_NAME": "PROPERTY1_VALUE"}',
    },
  ],
};
export const run = async () => {
  try {
    const response = await personalizeEventsClient.send(
      new PutUsersCommand(putUsersParam),
    );
    console.log("Success!", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考PutUsers中的。

Amazon Personalize 化運行時示SDK例使用 JavaScript in (v3)

下列程式碼範例說明如何透過使用 AWS SDK for JavaScript (v3) 搭配 Amazon Personalize 執行階段來執行動作和實作常見案例。

Actions 是大型程式的程式碼摘錄，必須在內容中執行。雖然動作會顯示如何呼叫個別服務函數，但您可以在其相關案例中查看內容中的動作。

每個範例都包含完整原始程式碼的連結，您可以在其中找到如何在內容中設定和執行程式碼的指示。

主題

動作

`GetPersonalizedRanking`

下列程式碼範例會示範如何使用`GetPersonalizedRanking`。

SDK對於 JavaScript （ 3 ）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
// Get service clients module and commands using ES6 syntax.
import { GetPersonalizedRankingCommand } from
  "@aws-sdk/client-personalize-runtime";
import { personalizeRuntimeClient } from "./libs/personalizeClients.js";
// Or, create the client here.
// const personalizeRuntimeClient = new PersonalizeRuntimeClient({ region:
 "REGION"});

// Set the ranking request parameters.
export const getPersonalizedRankingParam = {
  campaignArn: "CAMPAIGN_ARN", /* required */
  userId: 'USER_ID',      /* required */
  inputList: ["ITEM_ID_1", "ITEM_ID_2", "ITEM_ID_3", "ITEM_ID_4"]
```

```
}

export const run = async () => {
  try {
    const response = await personalizeRuntimeClient.send(new
 GetPersonalizedRankingCommand(getPersonalizedRankingParam));
    console.log("Success!", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考GetPersonalizedRanking中的。

`GetRecommendations`

下列程式碼範例會示範如何使用`GetRecommendations`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
// Get service clients module and commands using ES6 syntax.
import { GetRecommendationsCommand } from
  "@aws-sdk/client-personalize-runtime";

import { personalizeRuntimeClient } from "./libs/personalizeClients.js";
// Or, create the client here.
// const personalizeRuntimeClient = new PersonalizeRuntimeClient({ region:
 "REGION"});

// Set the recommendation request parameters.
export const getRecommendationsParam = {
  campaignArn: 'CAMPAIGN_ARN', /* required */
```

```
  userId: 'USER_ID',      /* required */
  numResults: 15    /* optional */
}

export const run = async () => {
  try {
    const response = await personalizeRuntimeClient.send(new
 GetRecommendationsCommand(getRecommendationsParam));
    console.log("Success!", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

使用篩選器 (自訂資料集群組) 取得建議。

```
// Get service clients module and commands using ES6 syntax.
import { GetRecommendationsCommand } from
  "@aws-sdk/client-personalize-runtime";
import { personalizeRuntimeClient } from "./libs/personalizeClients.js";
// Or, create the client here.
// const personalizeRuntimeClient = new PersonalizeRuntimeClient({ region:
 "REGION"});

// Set the recommendation request parameters.
export const getRecommendationsParam = {
  recommenderArn: 'RECOMMENDER_ARN', /* required */
  userId: 'USER_ID',       /* required */
  numResults: 15    /* optional */
}

export const run = async () => {
  try {
    const response = await personalizeRuntimeClient.send(new
 GetRecommendationsCommand(getRecommendationsParam));
    console.log("Success!", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
```

```
  }
};
run();
```

從網域資料集群組中建立的推薦人取得篩選建議。

```
// Get service clients module and commands using ES6 syntax.
import { GetRecommendationsCommand } from
  "@aws-sdk/client-personalize-runtime";
import { personalizeRuntimeClient } from "./libs/personalizeClients.js";
// Or, create the client here:
// const personalizeRuntimeClient = new PersonalizeRuntimeClient({ region:
 "REGION"});

// Set recommendation request parameters.
export const getRecommendationsParam = {
  campaignArn: 'CAMPAIGN_ARN', /* required */
  userId: 'USER_ID',      /* required */
  numResults: 15,    /* optional */
  filterArn: 'FILTER_ARN',   /* required to filter recommendations */
  filterValues: {
    "PROPERTY": "\"VALUE\""  /* Only required if your filter has a placeholder
 parameter */
  }
}

export const run = async () => {
  try {
    const response = await personalizeRuntimeClient.send(new
 GetRecommendationsCommand(getRecommendationsParam));
    console.log("Success!", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考GetRecommendations中的。

Amazon Pinpoint 示例使用 SDK for JavaScript (v3)

下列程式碼範例說明如何透過使用 AWS SDK for JavaScript (v3) 搭配 Amazon Pinpoint 來執行動作和實作常見案例。

Actions 是大型程式的程式碼摘錄，必須在內容中執行。雖然動作會顯示如何呼叫個別服務函數，但您可以在其相關案例中查看內容中的動作。

每個範例都包含完整原始程式碼的連結，您可以在其中找到如何在內容中設定和執行程式碼的指示。

主題

動作

`SendMessages`

下列程式碼範例會示範如何使用`SendMessages`。

SDK對於 JavaScript （ 3 ）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

在單獨的模組中建立用戶端並將其匯出。

```
import { PinpointClient } from "@aws-sdk/client-pinpoint";
// Set the AWS Region.
const REGION = "us-east-1";
export const pinClient = new PinpointClient({ region: REGION });
```

傳送電子郵件訊息。

```
// Import required AWS SDK clients and commands for Node.js
import { SendMessagesCommand } from "@aws-sdk/client-pinpoint";
import { pinClient } from "./libs/pinClient.js";
```

```
// The FromAddress must be verified in SES.
const fromAddress = "FROM_ADDRESS";
const toAddress = "TO_ADDRESS";
const projectId = "PINPOINT_PROJECT_ID";

// The subject line of the email.
var subject = "Amazon Pinpoint Test (AWS SDK for JavaScript in Node.js)";

// The email body for recipients with non-HTML email clients.
var body_text = `Amazon Pinpoint Test (SDK for JavaScript in Node.js)
-----------------------------------------------------
This email was sent with Amazon Pinpoint using the AWS SDK for JavaScript in
 Node.js.
For more information, see https://aws.amazon.com/sdk-for-node-js/`;

// The body of the email for recipients whose email clients support HTML content.
var body_html = `<html>
<head></head>
<body>
  <h1>Amazon Pinpoint Test (SDK for JavaScript in Node.js)</h1>
  <p>This email was sent with
    <a href='https://aws.amazon.com/pinpoint/'>the Amazon Pinpoint Email API</a>
 using the
    <a href='https://aws.amazon.com/sdk-for-node-js/'>
      AWS SDK for JavaScript in Node.js</a>.</p>
</body>
</html>`;

// The character encoding for the subject line and message body of the email.
var charset = "UTF-8";

const params = {
  ApplicationId: projectId,
  MessageRequest: {
    Addresses: {
      [toAddress]: {
        ChannelType: "EMAIL",
      },
    },
    MessageConfiguration: {
      EmailMessage: {
        FromAddress: fromAddress,
        SimpleEmail: {
```

```
          Subject: {
            Charset: charset,
            Data: subject,
          },
          HtmlPart: {
            Charset: charset,
            Data: body_html,
          },
          TextPart: {
            Charset: charset,
            Data: body_text,
          },
        },
      },
    },
  },
};

const run = async () => {
  try {
    const { MessageResponse } = await pinClient.send(
      new SendMessagesCommand(params),
    );

    if (!MessageResponse) {
      throw new Error("No message response.");
    }

    if (!MessageResponse.Result) {
      throw new Error("No message result.");
    }

    const recipientResult = MessageResponse.Result[toAddress];

    if (recipientResult.StatusCode !== 200) {
      throw new Error(recipientResult.StatusMessage);
    } else {
      console.log(recipientResult.MessageId);
    }
  } catch (err) {
    console.log(err.message);
  }
};
```

```
run();
```

傳送SMS訊息。

```
// Import required AWS SDK clients and commands for Node.js
import { SendMessagesCommand } from "@aws-sdk/client-pinpoint";
import { pinClient } from "./libs/pinClient.js";

/* The phone number or short code to send the message from. The phone number
 or short code that you specify has to be associated with your Amazon Pinpoint
account. For best results, specify long codes in E.164 format. */
const originationNumber = "SENDER_NUMBER"; //e.g., +1XXXXXXXXXX

// The recipient's phone number.  For best results, you should specify the phone
 number in E.164 format.
const destinationNumber = "RECEIVER_NUMBER"; //e.g., +1XXXXXXXXXX

// The content of the SMS message.
const message =
  "This message was sent through Amazon Pinpoint " +
  "using the AWS SDK for JavaScript in Node.js. Reply STOP to " +
  "opt out.";

/*The Amazon Pinpoint project/application ID to use when you send this message.
Make sure that the SMS channel is enabled for the project or application
that you choose.*/
const projectId = "PINPOINT_PROJECT_ID"; //e.g., XXXXXXXX66e4e9986478cXXXXXXXXX

/* The type of SMS message that you want to send. If you plan to send
time-sensitive content, specify TRANSACTIONAL. If you plan to send
marketing-related content, specify PROMOTIONAL.*/
var messageType = "TRANSACTIONAL";

// The registered keyword associated with the originating short code.
var registeredKeyword = "myKeyword";

/* The sender ID to use when sending the message. Support for sender ID
// varies by country or region. For more information, see
https://docs.aws.amazon.com/pinpoint/latest/userguide/channels-sms-countries.html.*/

var senderId = "MySenderID";
```

```
// Specify the parameters to pass to the API.
var params = {
  ApplicationId: projectId,
  MessageRequest: {
    Addresses: {
      [destinationNumber]: {
        ChannelType: "SMS",
      },
    },
    MessageConfiguration: {
      SMSMessage: {
        Body: message,
        Keyword: registeredKeyword,
        MessageType: messageType,
        OriginationNumber: originationNumber,
        SenderId: senderId,
      },
    },
  },
};

const run = async () => {
  try {
    const data = await pinClient.send(new SendMessagesCommand(params));
    console.log(
      "Message sent! " +
        data["MessageResponse"]["Result"][destinationNumber]["StatusMessage"],
    );
  } catch (err) {
    console.log(err);
  }
};
run();
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考SendMessages中的。

SDK對於 JavaScript （第 2 個）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

傳送電子郵件訊息。

```
"use strict";

const AWS = require("aws-sdk");

// The AWS Region that you want to use to send the email. For a list of
// AWS Regions where the Amazon Pinpoint API is available, see
// https://docs.aws.amazon.com/pinpoint/latest/apireference/
const aws_region = "us-west-2";

// The "From" address. This address has to be verified in Amazon Pinpoint
// in the region that you use to send email.
const senderAddress = "sender@example.com";

// The address on the "To" line. If your Amazon Pinpoint account is in
// the sandbox, this address also has to be verified.
var toAddress = "recipient@example.com";

// The Amazon Pinpoint project/application ID to use when you send this message.
// Make sure that the SMS channel is enabled for the project or application
// that you choose.
const appId = "ce796be37f32f178af652b26eexample";

// The subject line of the email.
var subject = "Amazon Pinpoint (AWS SDK for JavaScript in Node.js)";

// The email body for recipients with non-HTML email clients.
var body_text = `Amazon Pinpoint Test (SDK for JavaScript in Node.js)
-----------------------------------------------------
This email was sent with Amazon Pinpoint using the AWS SDK for JavaScript in
 Node.js.
For more information, see https:\/\/aws.amazon.com/sdk-for-node-js/`;
```

```
// The body of the email for recipients whose email clients support HTML content.
var body_html = `<html>
<head></head>
<body>
  <h1>Amazon Pinpoint Test (SDK for JavaScript in Node.js)</h1>
  <p>This email was sent with
    <a href='https://aws.amazon.com/pinpoint/'>the Amazon Pinpoint API</a> using the
    <a href='https://aws.amazon.com/sdk-for-node-js/'>
      AWS SDK for JavaScript in Node.js</a>.</p>
</body>
</html>`;

// The character encoding the you want to use for the subject line and
// message body of the email.
var charset = "UTF-8";

// Specify that you're using a shared credentials file.
var credentials = new AWS.SharedIniFileCredentials({ profile: "default" });
AWS.config.credentials = credentials;

// Specify the region.
AWS.config.update({ region: aws_region });

//Create a new Pinpoint object.
var pinpoint = new AWS.Pinpoint();

// Specify the parameters to pass to the API.
var params = {
  ApplicationId: appId,
  MessageRequest: {
    Addresses: {
      [toAddress]: {
        ChannelType: "EMAIL",
      },
    },
    MessageConfiguration: {
      EmailMessage: {
        FromAddress: senderAddress,
        SimpleEmail: {
          Subject: {
            Charset: charset,
            Data: subject,
          },
          HtmlPart: {
```

```
            Charset: charset,
            Data: body_html,
          },
          TextPart: {
            Charset: charset,
            Data: body_text,
          },
        },
      },
    },
  },
};

//Try to send the email.
pinpoint.sendMessages(params, function (err, data) {
  // If something goes wrong, print an error message.
  if (err) {
    console.log(err.message);
  } else {
    console.log(
      "Email sent! Message ID: ",
      data["MessageResponse"]["Result"][toAddress]["MessageId"]
    );
  }
});
```

傳送SMS訊息。

```
"use strict";

var AWS = require("aws-sdk");

// The AWS Region that you want to use to send the message. For a list of
// AWS Regions where the Amazon Pinpoint API is available, see
// https://docs.aws.amazon.com/pinpoint/latest/apireference/.
var aws_region = "us-east-1";

// The phone number or short code to send the message from. The phone number
// or short code that you specify has to be associated with your Amazon Pinpoint
// account. For best results, specify long codes in E.164 format.
```

```
var originationNumber = "+12065550199";

// The recipient's phone number.  For best results, you should specify the
// phone number in E.164 format.
var destinationNumber = "+14255550142";

// The content of the SMS message.
var message =
  "This message was sent through Amazon Pinpoint " +
  "using the AWS SDK for JavaScript in Node.js. Reply STOP to " +
  "opt out.";

// The Amazon Pinpoint project/application ID to use when you send this message.
// Make sure that the SMS channel is enabled for the project or application
// that you choose.
var applicationId = "ce796be37f32f178af652b26eexample";

// The type of SMS message that you want to send. If you plan to send
// time-sensitive content, specify TRANSACTIONAL. If you plan to send
// marketing-related content, specify PROMOTIONAL.
var messageType = "TRANSACTIONAL";

// The registered keyword associated with the originating short code.
var registeredKeyword = "myKeyword";

// The sender ID to use when sending the message. Support for sender ID
// varies by country or region. For more information, see
// https://docs.aws.amazon.com/pinpoint/latest/userguide/channels-sms-countries.html
var senderId = "MySenderID";

// Specify that you're using a shared credentials file, and optionally specify
// the profile that you want to use.
var credentials = new AWS.SharedIniFileCredentials({ profile: "default" });
AWS.config.credentials = credentials;

// Specify the region.
AWS.config.update({ region: aws_region });

//Create a new Pinpoint object.
var pinpoint = new AWS.Pinpoint();

// Specify the parameters to pass to the API.
var params = {
  ApplicationId: applicationId,
```

```
  MessageRequest: {
    Addresses: {
      [destinationNumber]: {
        ChannelType: "SMS",
      },
    },
    MessageConfiguration: {
      SMSMessage: {
        Body: message,
        Keyword: registeredKeyword,
        MessageType: messageType,
        OriginationNumber: originationNumber,
        SenderId: senderId,
      },
    },
  },
};

//Try to send the message.
pinpoint.sendMessages(params, function (err, data) {
  // If something goes wrong, print an error message.
  if (err) {
    console.log(err.message);
    // Otherwise, show the unique ID for the message.
  } else {
    console.log(
      "Message sent! " +
        data["MessageResponse"]["Result"][destinationNumber]["StatusMessage"]
    );
  }
});
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考SendMessages中的。

Amazon Polly 示例使SDK用 JavaScript (v3)

下列程式碼範例說明如何透過 Amazon Polly 使用 AWS SDK for JavaScript (v3) 來執行動作和實作常見案例。

案例是程式碼範例，向您展示如何透過呼叫服務中的多個函式或與其他函式結合來完成特定工作 AWS 服務。

每個範例都包含完整原始程式碼的連結，您可以在其中找到如何在內容中設定和執行程式碼的指示。

主題

案例

建立應用程式以分析客戶意見回饋

下列程式碼範例會示範如何建立可分析客戶評論卡、從其原始語言進行翻譯、判斷對方情緒，以及透過翻譯後的文字產生音訊檔案的應用程式。

SDK對於 JavaScript（3）

此範例應用程式會分析和存儲客戶的意見回饋卡。具體來說，它滿足了紐約市一家虛構飯店的需求。飯店以實體評論卡的形式收到賓客以各種語言撰寫的意見回饋。這些意見回饋透過 Web 用戶端上傳至應用程式。評論卡的影像上傳後，系統會執行下列步驟：

- 文字內容是使用 Amazon Textract 從影像中擷取。
- Amazon Comprehend 會決定擷取文字及其用語的情感。
- 擷取的文字內容會使用 Amazon Translate 翻譯成英文。
- Amazon Polly 會使用擷取的文字內容合成音訊檔案。

完整的應用程式可透過 AWS CDK 部署。如需原始程式碼和部署指示，請參閱中的專案 GitHub。下列摘錄顯示如何在 AWS SDK for JavaScript Lambda 函數內部使用。

```
import {
  ComprehendClient,
  DetectDominantLanguageCommand,
  DetectSentimentCommand,
} from "@aws-sdk/client-comprehend";

/**
 * Determine the language and sentiment of the extracted text.
 *
 * @param {{ source_text: string}} extractTextOutput
 */
```

```
export const handler = async (extractTextOutput) => {
  const comprehendClient = new ComprehendClient({});

  const detectDominantLanguageCommand = new DetectDominantLanguageCommand({
    Text: extractTextOutput.source_text,
  });

  // The source language is required for sentiment analysis and
  // translation in the next step.
  const { Languages } = await comprehendClient.send(
    detectDominantLanguageCommand,
  );

  const languageCode = Languages[0].LanguageCode;

  const detectSentimentCommand = new DetectSentimentCommand({
    Text: extractTextOutput.source_text,
    LanguageCode: languageCode,
  });

  const { Sentiment } = await comprehendClient.send(detectSentimentCommand);

  return {
    sentiment: Sentiment,
    language_code: languageCode,
  };
};
```

```
import {
  DetectDocumentTextCommand,
  TextractClient,
} from "@aws-sdk/client-textract";

/**
 * Fetch the S3 object from the event and analyze it using Amazon Textract.
 *
 * @param {import("@types/aws-lambda").EventBridgeEvent<"Object Created">}
 eventBridgeS3Event
 */
export const handler = async (eventBridgeS3Event) => {
  const textractClient = new TextractClient();

  const detectDocumentTextCommand = new DetectDocumentTextCommand({
```

```
    Document: {
      S3Object: {
        Bucket: eventBridgeS3Event.bucket,
        Name: eventBridgeS3Event.object,
      },
    },
  });

  // Textract returns a list of blocks. A block can be a line, a page, word, etc.
  // Each block also contains geometry of the detected text.
  // For more information on the Block type, see https://docs.aws.amazon.com/
textract/latest/dg/API_Block.html.
  const { Blocks } = await textractClient.send(detectDocumentTextCommand);

  // For the purpose of this example, we are only interested in words.
  const extractedWords = Blocks.filter((b) => b.BlockType === "WORD").map(
    (b) => b.Text,
  );

  return extractedWords.join(" ");
};
```

```
import { PollyClient, SynthesizeSpeechCommand } from "@aws-sdk/client-polly";
import { S3Client } from "@aws-sdk/client-s3";
import { Upload } from "@aws-sdk/lib-storage";

/**
 * Synthesize an audio file from text.
 *
 * @param {{ bucket: string, translated_text: string, object: string}}
 sourceDestinationConfig
 */
export const handler = async (sourceDestinationConfig) => {
  const pollyClient = new PollyClient({});

  const synthesizeSpeechCommand = new SynthesizeSpeechCommand({
    Engine: "neural",
    Text: sourceDestinationConfig.translated_text,
    VoiceId: "Ruth",
    OutputFormat: "mp3",
  });

  const { AudioStream } = await pollyClient.send(synthesizeSpeechCommand);
```

```
  const audioKey = `${sourceDestinationConfig.object}.mp3`;

  // Store the audio file in S3.
  const s3Client = new S3Client();
  const upload = new Upload({
    client: s3Client,
    params: {
      Bucket: sourceDestinationConfig.bucket,
      Key: audioKey,
      Body: AudioStream,
      ContentType: "audio/mp3",
    },
  });

  await upload.done();
  return audioKey;
};
```

```
import {
  TranslateClient,
  TranslateTextCommand,
} from "@aws-sdk/client-translate";

/**
 * Translate the extracted text to English.
 *
 * @param {{ extracted_text: string, source_language_code: string}}
 textAndSourceLanguage
 */
export const handler = async (textAndSourceLanguage) => {
  const translateClient = new TranslateClient({});

  const translateCommand = new TranslateTextCommand({
    SourceLanguageCode: textAndSourceLanguage.source_language_code,
    TargetLanguageCode: "en",
    Text: textAndSourceLanguage.extracted_text,
  });

  const { TranslatedText } = await translateClient.send(translateCommand);

  return { translated_text: TranslatedText };
};
```

此範例中使用的服務

- Amazon Comprehend
- Lambda
- Amazon Polly
- Amazon Textract
- Amazon Translate

Amazon RDS 示例使用 SDK for JavaScript (v3)

下列程式碼範例說明如何透過使用 AWS SDK for JavaScript (v3) 搭配 Amazon 來執行動作和實作常見案例RDS。

案例是程式碼範例，向您展示如何透過呼叫服務中的多個函式或與其他函式結合來完成特定工作 AWS 服務。

每個範例都包含完整原始程式碼的連結，您可以在其中找到如何在內容中設定和執行程式碼的指示。

主題

案例

建立 Aurora 無伺服器工作項目追蹤器

下列程式碼範例示範如何建立可追蹤 Amazon Aurora 無伺服器資料庫中工作項目的 Web 應用程式，並使用 Amazon 簡單電子郵件服務 (AmazonSES) 傳送報告。

SDK對於 JavaScript （ 3 ）

示範如何使用 AWS SDK for JavaScript (v3) 建立 Web 應用程式，以追蹤 Amazon Aurora 資料庫中的工作項目，以及使用 Amazon 簡單電子郵件服務 (AmazonSES) 傳送電子郵件報告的 Web 應用程式。這個範例使用以 React.js 建置的前端與 Express Node.js 後端互動。

- 將 React.js 網路應用程式與 AWS 服務.
- 列出、新增和更新 Aurora 資料表中的項目。
- 使用 Amazon 傳送篩選工作項目的電子郵件報告SES。

- 使用隨附的 AWS CloudFormation 指令碼部署和管理範例資源。

有關如何設置和運行的完整源代碼和說明，請參閱中的完整示例GitHub。

此範例中使用的服務

- Aurora
- Amazon RDS
- Amazon RDS 數據服務
- Amazon SES

無伺服器範例

在 Lambda 函RDS數中連接到 Amazon 數據庫

下列程式碼範例示範如何實作連線至RDS資料庫的 Lambda 函數。該函數提出了一個簡單的數據庫請求，並返回結果。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在無伺服器範例儲存庫中設定和執行。

使用 Lambda 函RDS數連接到 Amazon 數據庫 JavaScript。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
/*
Node.js code here.
*/
// ES6+ example
import { Signer } from "@aws-sdk/rds-signer";
import mysql from 'mysql2/promise';

async function createAuthToken() {
  // Define connection authentication parameters
  const dbinfo = {

    hostname: process.env.ProxyHostName,
```

```
    port: process.env.Port,
    username: process.env.DBUserName,
    region: process.env.AWS_REGION,

  }

  // Create RDS Signer object
  const signer = new Signer(dbinfo);

  // Request authorization token from RDS, specifying the username
  const token = await signer.getAuthToken();
  return token;
}

async function dbOps() {

  // Obtain auth token
  const token = await createAuthToken();
  // Define connection configuration
  let connectionConfig = {
    host: process.env.ProxyHostName,
    user: process.env.DBUserName,
    password: token,
    database: process.env.DBName,
    ssl: 'Amazon RDS'
  }
  // Create the connection to the DB
  const conn = await mysql.createConnection(connectionConfig);
  // Obtain the result of the query
  const [res,] = await conn.execute('select ?+? as sum', [3, 2]);
  return res;

}

export const handler = async (event) => {
  // Execute database flow
  const result = await dbOps();
  // Return result
  return {
    statusCode: 200,
    body: JSON.stringify("The selected sum is: " + result[0].sum)
  }
};
```

使用 Lambda 函RDS數連接到 Amazon 數據庫 TypeScript。

```
import { Signer } from "@aws-sdk/rds-signer";
import mysql from 'mysql2/promise';

// RDS settings
// Using '!' (non-null assertion operator) to tell the TypeScript compiler that the
 DB settings are not null or undefined,
const proxy_host_name = process.env.PROXY_HOST_NAME!
const port = parseInt(process.env.PORT!)
const db_name = process.env.DB_NAME!
const db_user_name = process.env.DB_USER_NAME!
const aws_region = process.env.AWS_REGION!


async function createAuthToken(): Promise<string> {

    // Create RDS Signer object
    const signer = new Signer({
        hostname: proxy_host_name,
        port: port,
        region: aws_region,
        username: db_user_name
    });

    // Request authorization token from RDS, specifying the username
    const token = await signer.getAuthToken();
    return token;
}

async function dbOps(): Promise<mysql.QueryResult | undefined> {
    try {
        // Obtain auth token
        const token = await createAuthToken();
        const conn = await mysql.createConnection({
            host: proxy_host_name,
            user: db_user_name,
            password: token,
            database: db_name,
            ssl: 'Amazon RDS' // Ensure you have the CA bundle for SSL connection
```

```
        });
        const [rows, fields] = await conn.execute('SELECT ? + ? AS sum', [3, 2]);
        console.log('result:', rows);
        return rows;
    }
    catch (err) {
        console.log(err);
    }
}

export const lambdaHandler = async (event: any): Promise<{ statusCode: number; body:
 string }> => {
    // Execute database flow
    const result = await dbOps();

    // Return error is result is undefined
    if (result == undefined)
        return {
            statusCode: 500,
            body: JSON.stringify(`Error with connection to DB host`)
        }

    // Return result
    return {
        statusCode: 200,
        body: JSON.stringify(`The selected sum is: ${result[0].sum}`)
    };
};
```

Amazon RDS 數據服務示SDK例使用 JavaScript (v3)

下列程式碼範例說明如何透過使用 AWS SDK for JavaScript (v3) 搭配 Amazon RDS 資料服務來執行動作和實作常見案例。

案例是程式碼範例，向您展示如何透過呼叫服務中的多個函式或與其他函式結合來完成特定工作 AWS 服務。

每個範例都包含完整原始程式碼的連結，您可以在其中找到如何在內容中設定和執行程式碼的指示。

主題

- 案例

案例

建立 Aurora 無伺服器工作項目追蹤器

下列程式碼範例示範如何建立可追蹤 Amazon Aurora 無伺服器資料庫中工作項目的 Web 應用程式，並使用 Amazon 簡單電子郵件服務 (AmazonSES) 傳送報告。

SDK對於 JavaScript （ 3 ）

示範如何使用 AWS SDK for JavaScript (v3) 建立 Web 應用程式，以追蹤 Amazon Aurora 資料庫中的工作項目，以及使用 Amazon 簡單電子郵件服務 (AmazonSES) 傳送電子郵件報告的 Web 應用程式。這個範例使用以 React.js 建置的前端與 Express Node.js 後端互動。

- 將 React.js 網路應用程式與 AWS 服務.
- 列出、新增和更新 Aurora 資料表中的項目。
- 使用 Amazon 傳送篩選工作項目的電子郵件報告SES。
- 使用隨附的 AWS CloudFormation 指令碼部署和管理範例資源。

有關如何設置和運行的完整源代碼和說明，請參閱中的完整示例GitHub。

此範例中使用的服務

- Aurora
- Amazon RDS
- Amazon RDS 數據服務
- Amazon SES

Amazon Redshift 示例使用 SDK for JavaScript (v3)

下列程式碼範例說明如何透過使用 AWS SDK for JavaScript (v3) 搭配 Amazon Redshift 來執行動作和實作常見案例。

Actions 是大型程式的程式碼摘錄，必須在內容中執行。雖然動作會顯示如何呼叫個別服務函數，但您可以在其相關案例中查看內容中的動作。

每個範例都包含完整原始程式碼的連結，您可以在其中找到如何在內容中設定和執行程式碼的指示。

主題

- 動作

動作

`CreateCluster`

下列程式碼範例會示範如何使用`CreateCluster`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

建立用戶端。

```
import { RedshiftClient } from "@aws-sdk/client-redshift";
// Set the AWS Region.
const REGION = "REGION";
//Set the Redshift Service Object
const redshiftClient = new RedshiftClient({ region: REGION });
export { redshiftClient };
```

建立 叢集

```
// Import required AWS SDK clients and commands for Node.js
import { CreateClusterCommand } from "@aws-sdk/client-redshift";
import { redshiftClient } from "./libs/redshiftClient.js";

const params = {
  ClusterIdentifier: "CLUSTER_NAME", // Required
  NodeType: "NODE_TYPE", //Required
  MasterUsername: "MASTER_USER_NAME", // Required - must be lowercase
  MasterUserPassword: "MASTER_USER_PASSWORD", // Required - must contain at least
 one uppercase letter, and one number
  ClusterType: "CLUSTER_TYPE", // Required
  IAMRoleARN: "IAM_ROLE_ARN", // Optional - the ARN of an IAM role with permissions
 your cluster needs to access other AWS services on your behalf, such as Amazon S3.
  ClusterSubnetGroupName: "CLUSTER_SUBNET_GROUPNAME", //Optional - the name of a
 cluster subnet group to be associated with this cluster. Defaults to 'default' if
 not specified.
```

```
  DBName: "DATABASE_NAME", // Optional - defaults to 'dev' if not specified
  Port: "PORT_NUMBER", // Optional - defaults to '5439' if not specified
};

const run = async () => {
  try {
    const data = await redshiftClient.send(new CreateClusterCommand(params));
    console.log(
      "Cluster " + data.Cluster.ClusterIdentifier + " successfully created",
    );
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考CreateCluster中的。

`DeleteCluster`

下列程式碼範例會示範如何使用`DeleteCluster`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

建立用戶端。

```
import { RedshiftClient } from "@aws-sdk/client-redshift";
// Set the AWS Region.
const REGION = "REGION";
//Set the Redshift Service Object
const redshiftClient = new RedshiftClient({ region: REGION });
export { redshiftClient };
```

建立 叢集

```
// Import required AWS SDK clients and commands for Node.js
import { DeleteClusterCommand } from "@aws-sdk/client-redshift";
import { redshiftClient } from "./libs/redshiftClient.js";

const params = {
  ClusterIdentifier: "CLUSTER_NAME",
  SkipFinalClusterSnapshot: false,
  FinalClusterSnapshotIdentifier: "CLUSTER_SNAPSHOT_ID",
};

const run = async () => {
  try {
    const data = await redshiftClient.send(new DeleteClusterCommand(params));
    console.log("Success, cluster deleted. ", data);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考DeleteCluster中的。

`DescribeClusters`

下列程式碼範例會示範如何使用`DescribeClusters`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

建立用戶端。

```
import { RedshiftClient } from "@aws-sdk/client-redshift";
// Set the AWS Region.
```

```
const REGION = "REGION";
//Set the Redshift Service Object
const redshiftClient = new RedshiftClient({ region: REGION });
export { redshiftClient };
```

描述您的叢集。

```
// Import required AWS SDK clients and commands for Node.js
import { DescribeClustersCommand } from "@aws-sdk/client-redshift";
import { redshiftClient } from "./libs/redshiftClient.js";

const params = {
  ClusterIdentifier: "CLUSTER_NAME",
};

const run = async () => {
  try {
    const data = await redshiftClient.send(new DescribeClustersCommand(params));
    console.log("Success", data);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考DescribeClusters中的。

`ModifyCluster`

下列程式碼範例會示範如何使用`ModifyCluster`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

建立用戶端。

```
import { RedshiftClient } from "@aws-sdk/client-redshift";
// Set the AWS Region.
const REGION = "REGION";
//Set the Redshift Service Object
const redshiftClient = new RedshiftClient({ region: REGION });
export { redshiftClient };
```

修改叢集。

```
// Import required AWS SDK clients and commands for Node.js
import { ModifyClusterCommand } from "@aws-sdk/client-redshift";
import { redshiftClient } from "./libs/redshiftClient.js";

// Set the parameters
const params = {
  ClusterIdentifier: "CLUSTER_NAME",
  MasterUserPassword: "NEW_MASTER_USER_PASSWORD",
};

const run = async () => {
  try {
    const data = await redshiftClient.send(new ModifyClusterCommand(params));
    console.log("Success was modified.", data);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考ModifyCluster中的。

Amazon Rekognition 示例使用 SDK for JavaScript (v3)

下列程式碼範例說明如何透過使用 AWS SDK for JavaScript (v3) 搭配 Amazon Rekognition 來執行動作和實作常見案例。

案例是程式碼範例，向您展示如何透過呼叫服務中的多個函式或與其他函式結合來完成特定工作 AWS 服務。

每個範例都包含完整原始程式碼的連結，您可以在其中找到如何在內容中設定和執行程式碼的指示。

主題

案例

建立無伺服器應用程式來管理相片

下列程式碼範例示範如何建立無伺服器應用程式，讓使用者以標籤管理相片。

SDK對於 JavaScript（3）

顯示如何開發照片資產管理應用程式，以便使用 Amazon Rekognition 偵測圖片中的標籤，並將其儲存以供日後擷取。

有關如何設置和運行的完整源代碼和說明，請參閱中的完整示例 GitHub。

如要深入探索此範例的來源，請參閱 AWS 社群上的文章。

此範例中使用的服務

- API閘道
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

PPE在影像中偵測

下列程式碼範例示範如何建立使用 Amazon Rekognition 偵測影像中個人防護設備 (PPE) 的應用程式。

SDK對於 JavaScript（3）

示範如何搭配使用 Amazon Rekognition AWS SDK for JavaScript 來建立應用程式，以偵測位於亞馬遜簡單儲存服務 (Amazon S3PPE) 儲存貯體中的映像中的個人防護設備 ()。該應用程式會將結果

儲存到 Amazon DynamoDB 表，並使用 Amazon 簡單電子郵件服務 (AmazonSES) 向管理員傳送包含結果的電子郵件通知。

了解如何：

- 使用 Amazon Cognito 建立未經身分驗證的使用者。
- 分析圖像以PPE使用 Amazon Rekognition。
- 驗證 Amazon 的電子郵件地址SES。
- 以結果更新 DynamoDB 資料表。
- 使用 Amazon 發送電子郵件通知SES。

有關如何設置和運行的完整源代碼和說明，請參閱中的完整示例GitHub。

此範例中使用的服務

- DynamoDB
- Amazon Rekognition
- Amazon S3
- Amazon SES

偵測映像中的物件

下列程式碼範例示範如何建立使用 Amazon Rekognition 依照類別偵測影像中物件的應用程式。

SDK對於 JavaScript（3）

示範如何搭配使用 Amazon Rekognition AWS SDK for JavaScript 來建立使用 Amazon Rekognition 的應用程式，在位於亞馬遜簡單儲存服務 (Amazon S3) 儲存貯體中的映像中依類別識別物件。該應用程序向管理員發送電子郵件通知，其中包含使用 Amazon 簡單電子郵件服務（AmazonSES）的結果

了解如何：

- 使用 Amazon Cognito 建立未經身分驗證的使用者。
- 使用 Amazon Rekognition 分析映像中的物件。
- 驗證 Amazon 的電子郵件地址SES。
- 使用 Amazon 發送電子郵件通知SES。

有關如何設置和運行的完整源代碼和說明，請參閱中的完整示例GitHub。

此範例中使用的服務

- Amazon Rekognition
- Amazon S3
- Amazon SES

偵測映像中的人物和物件

下列程式碼範例示範如何使用 Amazon Rekognition 偵測影片中的人物和物件。

SDK對於 JavaScript（3）

示範如何搭配使用 Amazon Rekognition AWS SDK for JavaScript 來建立應用程式，以偵測位於亞馬遜簡單儲存服務 (Amazon S3) 儲存貯體中影片中的臉部和物件。該應用程序向管理員發送電子郵件通知，其中包含使用 Amazon 簡單電子郵件服務（AmazonSES）的結果

了解如何：

- 使用 Amazon Cognito 建立未經身分驗證的使用者。
- 分析圖像以PPE使用 Amazon Rekognition。
- 驗證 Amazon 的電子郵件地址SES。
- 使用 Amazon 發送電子郵件通知SES。

有關如何設置和運行的完整源代碼和說明，請參閱中的完整示例GitHub。

此範例中使用的服務

- Amazon Rekognition
- Amazon S3
- Amazon SES

Amazon S3 示例使SDK用 JavaScript（v3）

下列程式碼範例說明如何透過使用 AWS SDK for JavaScript (v3) 搭配 Amazon S3 來執行動作和實作常見案例。

基本概念是程式碼範例，會示範如何在服務中執行基本作業。

Actions 是大型程式的程式碼摘錄，必須在內容中執行。雖然動作會顯示如何呼叫個別服務函數，但您可以在其相關案例中查看內容中的動作。

案例是程式碼範例，向您展示如何透過呼叫服務中的多個函式或與其他函式結合來完成特定工作 AWS 服務。

每個範例都包含完整原始程式碼的連結，您可以在其中找到如何在內容中設定和執行程式碼的指示。

開始使用

您好 Amazon S3

下列程式碼範例示範如何開始使用 Amazon S3。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import { ListBucketsCommand, S3Client } from "@aws-sdk/client-s3";

// When no region or credentials are provided, the SDK will use the
// region and credentials from the local AWS config.
const client = new S3Client({});

export const helloS3 = async () => {
  const command = new ListBucketsCommand({});

  const { Buckets } = await client.send(command);
  console.log("Buckets: ");
  console.log(Buckets.map((bucket) => bucket.Name).join("\n"));
  return Buckets;
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考ListBuckets中的。

主題

- 基本概念
- 動作

- 案例
- 無伺服器範例

基本概念

學習基礎知識

以下程式碼範例顯示做法：

- 建立儲存貯體並上傳檔案到該儲存貯體。
- 從儲存貯體下載物件。
- 將物件複製至儲存貯體中的子文件夾。
- 列出儲存貯體中的物件。
- 刪除儲存貯體物件和該儲存貯體。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

首先，匯入所有必要模組。

```
// Used to check if currently running file is this file.
import { fileURLToPath } from "url";
import { readdirSync, readFileSync, writeFileSync } from "fs";

// Local helper utils.
import { dirnameFromMetaUrl } from "@aws-doc-sdk-examples/lib/utils/util-fs.js";
import { Prompter } from "@aws-doc-sdk-examples/lib/prompter.js";
import { wrapText } from "@aws-doc-sdk-examples/lib/utils/util-string.js";

import {
  S3Client,
  CreateBucketCommand,
  PutObjectCommand,
```

```
  ListObjectsCommand,
  CopyObjectCommand,
  GetObjectCommand,
  DeleteObjectsCommand,
  DeleteBucketCommand,
} from "@aws-sdk/client-s3";
```

前面的匯入動作參考了一些協助公用程式。這些公用程式位於本節開頭所連結之 GitHub 儲存庫的本機公用程式。如需相關內容，請參閱下列公用程式的實作方式。

```
export const dirnameFromMetaUrl = (metaUrl) =>
  fileURLToPath(new URL(".", metaUrl));

import { select, input, confirm, checkbox } from "@inquirer/prompts";

export class Prompter {
  /**
   * @param {{ message: string, choices: { name: string, value: string }[]}} options
   */
  select(options) {
    return select(options);
  }

  /**
   * @param {{ message: string }} options
   */
  input(options) {
    return input(options);
  }

  /**
   * @param {string} prompt
   */
  checkContinue = async (prompt = "") => {
    const prefix = prompt && prompt + " ";
    let ok = await this.confirm({
      message: `${prefix}Continue?`,
    });
    if (!ok) throw new Error("Exiting...");
  };

  /**
```

```
   * @param {{ message: string }} options
   */
  confirm(options) {
    return confirm(options);
  }

  /**
   * @param {{ message: string, choices: { name: string, value: string }[]}} options
   */
  checkbox(options) {
    return checkbox(options);
  }
}

export const wrapText = (text, char = "=") => {
  const rule = char.repeat(80);
  return `${rule}\n    ${text}\n${rule}\n`;
};
```

S3 的物件儲存在「儲存貯體」。接下來，來定義一個建立新儲存貯體的函數。

```
export const createBucket = async () => {
  const bucketName = await prompter.input({
    message: "Enter a bucket name. Bucket names must be globally unique:",
  });
  const command = new CreateBucketCommand({ Bucket: bucketName });
  await s3Client.send(command);
  console.log("Bucket created successfully.\n");
  return bucketName;
};
```

儲存貯體包含「物件」。此功能會將儲存庫的內容做為物件上傳至您的儲存貯體。

```
export const uploadFilesToBucket = async ({ bucketName, folderPath }) => {
  console.log(`Uploading files from ${folderPath}\n`);
  const keys = readdirSync(folderPath);
  const files = keys.map((key) => {
    const filePath = `${folderPath}/${key}`;
    const fileContent = readFileSync(filePath);
    return {
      Key: key,
```

```
      Body: fileContent,
    };
  });

  for (let file of files) {
    await s3Client.send(
      new PutObjectCommand({
        Bucket: bucketName,
        Body: file.Body,
        Key: file.Key,
      }),
    );
    console.log(`${file.Key} uploaded successfully.`);
  }
};
```

上傳物件之後，請檢查確認已正確上傳物件。您可以使 ListObjects 用的。您將使用「金鑰」屬性，但在回應中也有其他有用的屬性。

```
export const listFilesInBucket = async ({ bucketName }) => {
  const command = new ListObjectsCommand({ Bucket: bucketName });
  const { Contents } = await s3Client.send(command);
  const contentsList = Contents.map((c) => ` • ${c.Key}`).join("\n");
  console.log("\nHere's a list of files in the bucket:");
  console.log(contentsList + "\n");
};
```

有時您可能想從儲存貯體複製物件到另一個儲存貯體。為此使用 CopyObject 命令。

```
export const copyFileFromBucket = async ({ destinationBucket }) => {
  const proceed = await prompter.confirm({
    message: "Would you like to copy an object from another bucket?",
  });

  if (!proceed) {
    return;
  } else {
    const copy = async () => {
      try {
        const sourceBucket = await prompter.input({
          message: "Enter source bucket name:",
```

```
        });
        const sourceKey = await prompter.input({
          message: "Enter source key:",
        });
        const destinationKey = await prompter.input({
          message: "Enter destination key:",
        });

        const command = new CopyObjectCommand({
          Bucket: destinationBucket,
          CopySource: `${sourceBucket}/${sourceKey}`,
          Key: destinationKey,
        });
        await s3Client.send(command);
        await copyFileFromBucket({ destinationBucket });
      } catch (err) {
        console.error(`Copy error.`);
        console.error(err);
        const retryAnswer = await prompter.confirm({ message: "Try again?" });
        if (retryAnswer) {
          await copy();
        }
      }
    };
    await copy();
  }
};
```

沒有方SDK法可以從存儲桶中獲取多個對象。反之，您會建立一份待下載的物件清單，並重複執行這些物件。

```
export const downloadFilesFromBucket = async ({ bucketName }) => {
  const { Contents } = await s3Client.send(
    new ListObjectsCommand({ Bucket: bucketName }),
  );
  const path = await prompter.input({
    message: "Enter destination path for files:",
  });

  for (let content of Contents) {
    const obj = await s3Client.send(
      new GetObjectCommand({ Bucket: bucketName, Key: content.Key }),
```

```
    );
    writeFileSync(
      `${path}/${content.Key}`,
      await obj.Body.transformToByteArray(),
    );
  }
  console.log("Files downloaded successfully.\n");
};
```

該清除資源了。儲存貯體在刪除之前必須先清空。這兩個函數會清空並刪除儲存貯體。

```
export const emptyBucket = async ({ bucketName }) => {
  const listObjectsCommand = new ListObjectsCommand({ Bucket: bucketName });
  const { Contents } = await s3Client.send(listObjectsCommand);
  const keys = Contents.map((c) => c.Key);

  const deleteObjectsCommand = new DeleteObjectsCommand({
    Bucket: bucketName,
    Delete: { Objects: keys.map((key) => ({ Key: key })) },
  });
  await s3Client.send(deleteObjectsCommand);
  console.log(`${bucketName} emptied successfully.\n`);
};

export const deleteBucket = async ({ bucketName }) => {
  const command = new DeleteBucketCommand({ Bucket: bucketName });
  await s3Client.send(command);
  console.log(`${bucketName} deleted successfully.\n`);
};
```

「主要」函數會將所有內容放在一起。若你直接執行這個檔案，主要函數將受到叫用。

```
const main = async () => {
  const OBJECT_DIRECTORY = `${dirnameFromMetaUrl(
    import.meta.url,
  )}../../../../resources/sample_files/.sample_media`;

  try {
    console.log(wrapText("Welcome to the Amazon S3 getting started example."));
    console.log("Let's create a bucket.");
    const bucketName = await createBucket();
```

```
    await prompter.confirm({ message: continueMessage });

    console.log(wrapText("File upload."));
    console.log(
      "I have some default files ready to go. You can edit the source code to
 provide your own.",
    );
    await uploadFilesToBucket({
      bucketName,
      folderPath: OBJECT_DIRECTORY,
    });

    await listFilesInBucket({ bucketName });
    await prompter.confirm({ message: continueMessage });

    console.log(wrapText("Copy files."));
    await copyFileFromBucket({ destinationBucket: bucketName });
    await listFilesInBucket({ bucketName });
    await prompter.confirm({ message: continueMessage });

    console.log(wrapText("Download files."));
    await downloadFilesFromBucket({ bucketName });

    console.log(wrapText("Clean up."));
    await emptyBucket({ bucketName });
    await deleteBucket({ bucketName });
  } catch (err) {
    console.error(err);
  }
};
```

- 如需詳API細資訊，請參閱「AWS SDK for JavaScript API參考」中的下列主題。
  - CopyObject
  - CreateBucket
  - DeleteBucket
  - DeleteObjects
  - GetObject
  - ListObjectsV2
  - PutObject

動作

`CopyObject`

下列程式碼範例會示範如何使用`CopyObject`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

複製物件

```
import { S3Client, CopyObjectCommand } from "@aws-sdk/client-s3";

const client = new S3Client({});

export const main = async () => {
  const command = new CopyObjectCommand({
    CopySource: "SOURCE_BUCKET/SOURCE_OBJECT_KEY",
    Bucket: "DESTINATION_BUCKET",
    Key: "NEW_OBJECT_KEY",
  });

  try {
    const response = await client.send(command);
    console.log(response);
  } catch (err) {
    console.error(err);
  }
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考CopyObject中的。

`CreateBucket`

下列程式碼範例會示範如何使用`CreateBucket`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

建立儲存貯體。

```
import { CreateBucketCommand, S3Client } from "@aws-sdk/client-s3";

const client = new S3Client({});

export const main = async () => {
  const command = new CreateBucketCommand({
    // The name of the bucket. Bucket names are unique and have several other
 constraints.
    // See https://docs.aws.amazon.com/AmazonS3/latest/userguide/
bucketnamingrules.html
    Bucket: "bucket-name",
  });

  try {
    const { Location } = await client.send(command);
    console.log(`Bucket created with location ${Location}`);
  } catch (err) {
    console.error(err);
  }
};
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考CreateBucket中的。

`DeleteBucket`

下列程式碼範例會示範如何使用`DeleteBucket`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

刪除儲存貯體。

```
import { DeleteBucketCommand, S3Client } from "@aws-sdk/client-s3";

const client = new S3Client({});

// Delete a bucket.
export const main = async () => {
  const command = new DeleteBucketCommand({
    Bucket: "test-bucket",
  });

  try {
    const response = await client.send(command);
    console.log(response);
  } catch (err) {
    console.error(err);
  }
};
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考DeleteBucket中的。

`DeleteBucketPolicy`

下列程式碼範例會示範如何使用`DeleteBucketPolicy`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

刪除儲存貯體政策。

```
import { DeleteBucketPolicyCommand, S3Client } from "@aws-sdk/client-s3";

const client = new S3Client({});

// This will remove the policy from the bucket.
export const main = async () => {
  const command = new DeleteBucketPolicyCommand({
    Bucket: "test-bucket",
  });

  try {
    const response = await client.send(command);
    console.log(response);
  } catch (err) {
    console.error(err);
  }
};
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考DeleteBucketPolicy中的。

`DeleteBucketWebsite`

下列程式碼範例會示範如何使用`DeleteBucketWebsite`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

從儲存貯體刪除網站組態

```
import { DeleteBucketWebsiteCommand, S3Client } from "@aws-sdk/client-s3";

const client = new S3Client({});

// Disable static website hosting on the bucket.
export const main = async () => {
  const command = new DeleteBucketWebsiteCommand({
    Bucket: "test-bucket",
  });

  try {
    const response = await client.send(command);
    console.log(response);
  } catch (err) {
    console.error(err);
  }
};
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考DeleteBucketWebsite中的。

`DeleteObject`

下列程式碼範例會示範如何使用`DeleteObject`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

刪除物件。

```
import { DeleteObjectCommand, S3Client } from "@aws-sdk/client-s3";

const client = new S3Client({});

export const main = async () => {
  const command = new DeleteObjectCommand({
    Bucket: "test-bucket",
    Key: "test-key.txt",
  });

  try {
    const response = await client.send(command);
    console.log(response);
  } catch (err) {
    console.error(err);
  }
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考DeleteObject中的。

`DeleteObjects`

下列程式碼範例會示範如何使用`DeleteObjects`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

刪除多個物件。

```
import { DeleteObjectsCommand, S3Client } from "@aws-sdk/client-s3";

const client = new S3Client({});

export const main = async () => {
  const command = new DeleteObjectsCommand({
    Bucket: "test-bucket",
    Delete: {
      Objects: [{ Key: "object1.txt" }, { Key: "object2.txt" }],
    },
  });

  try {
    const { Deleted } = await client.send(command);
    console.log(
      `Successfully deleted ${Deleted.length} objects from S3 bucket. Deleted
 objects:`,
    );
    console.log(Deleted.map((d) => ` • ${d.Key}`).join("\n"));
  } catch (err) {
    console.error(err);
  }
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考DeleteObjects中的。

`GetBucketAcl`

下列程式碼範例會示範如何使用`GetBucketAcl`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

取得ACL權限。

```
import { GetBucketAclCommand, S3Client } from "@aws-sdk/client-s3";

const client = new S3Client({});

export const main = async () => {
  const command = new GetBucketAclCommand({
    Bucket: "test-bucket",
  });

  try {
    const response = await client.send(command);
    console.log(response);
  } catch (err) {
    console.error(err);
  }
};
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考GetBucketAcl中的。

`GetBucketCors`

下列程式碼範例會示範如何使用`GetBucketCors`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

取得值區的CORS政策。

```
import { GetBucketCorsCommand, S3Client } from "@aws-sdk/client-s3";

const client = new S3Client({});

export const main = async () => {
```

```
  const command = new GetBucketCorsCommand({
    Bucket: "test-bucket",
  });

  try {
    const { CORSRules } = await client.send(command);
    CORSRules.forEach((cr, i) => {
      console.log(
        `\nCORSRule ${i + 1}`,
        `\n${"-".repeat(10)}`,
        `\nAllowedHeaders: ${cr.AllowedHeaders.join(" ")}`,
        `\nAllowedMethods: ${cr.AllowedMethods.join(" ")}`,
        `\nAllowedOrigins: ${cr.AllowedOrigins.join(" ")}`,
        `\nExposeHeaders: ${cr.ExposeHeaders.join(" ")}`,
        `\nMaxAgeSeconds: ${cr.MaxAgeSeconds}`,
      );
    });
  } catch (err) {
    console.error(err);
  }
};
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考GetBucketCors中的。

`GetBucketPolicy`

下列程式碼範例會示範如何使用`GetBucketPolicy`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

取得儲存貯體政策。

```
import { GetBucketPolicyCommand, S3Client } from "@aws-sdk/client-s3";
```

```
const client = new S3Client({});

export const main = async () => {
  const command = new GetBucketPolicyCommand({
    Bucket: "test-bucket",
  });

  try {
    const { Policy } = await client.send(command);
    console.log(JSON.parse(Policy));
  } catch (err) {
    console.error(err);
  }
};
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考GetBucketPolicy中的。

`GetBucketWebsite`

下列程式碼範例會示範如何使用`GetBucketWebsite`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

取得網站組態。

```
import { GetBucketWebsiteCommand, S3Client } from "@aws-sdk/client-s3";

const client = new S3Client({});

export const main = async () => {
  const command = new GetBucketWebsiteCommand({
    Bucket: "test-bucket",
```

```
  });

  try {
    const { ErrorDocument, IndexDocument } = await client.send(command);
    console.log(
      `Your bucket is set up to host a website. It has an error document:`,
      `${ErrorDocument.Key}, and an index document: ${IndexDocument.Suffix}.`,
    );
  } catch (err) {
    console.error(err);
  }
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考GetBucketWebsite中的。

`GetObject`

下列程式碼範例會示範如何使用`GetObject`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

下載物件。

```
import { GetObjectCommand, S3Client } from "@aws-sdk/client-s3";

const client = new S3Client({});

export const main = async () => {
  const command = new GetObjectCommand({
    Bucket: "test-bucket",
    Key: "hello-s3.txt",
  });

  try {
    const response = await client.send(command);
```

```
    // The Body object also has 'transformToByteArray' and 'transformToWebStream'
 methods.
    const str = await response.Body.transformToString();
    console.log(str);
  } catch (err) {
    console.error(err);
  }
};
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考GetObject中的。

`GetObjectLockConfiguration`

下列程式碼範例會示範如何使用`GetObjectLockConfiguration`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import { fileURLToPath } from "url";
import {
  GetObjectLockConfigurationCommand,
  S3Client,
} from "@aws-sdk/client-s3";

/**
 * @param {S3Client} client
 * @param {string} bucketName
 */
export const main = async (client, bucketName) => {
  const command = new GetObjectLockConfigurationCommand({
    Bucket: bucketName,
    // Optionally, you can provide additional parameters
    // ExpectedBucketOwner: "ACCOUNT_ID",
  });
```

```
  try {
    const { ObjectLockConfiguration } = await client.send(command);
    console.log(`Object Lock Configuration: ${ObjectLockConfiguration}`);
  } catch (err) {
    console.error(err);
  }
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  main(new S3Client(), "BUCKET_NAME");
}
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考GetObjectLockConfiguration中的。

`GetObjectRetention`

下列程式碼範例會示範如何使用`GetObjectRetention`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import { fileURLToPath } from "url";
import { GetObjectRetentionCommand, S3Client } from "@aws-sdk/client-s3";

/**
 * @param {S3Client} client
 * @param {string} bucketName
 * @param {string} objectKey
 */
export const main = async (client, bucketName, objectKey) => {
  const command = new GetObjectRetentionCommand({
    Bucket: bucketName,
    Key: objectKey,
```

```
    // Optionally, you can provide additional parameters
    // ExpectedBucketOwner: "ACCOUNT_ID",
    // RequestPayer: "requester",
    // VersionId: "OBJECT_VERSION_ID",
  });

  try {
    const { Retention } = await client.send(command);
    console.log(`Object Retention Settings: ${Retention.Status}`);
  } catch (err) {
    console.error(err);
  }
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  main(new S3Client(), "BUCKET_NAME", "OBJECT_KEY");
}
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考GetObjectRetention中的。

`ListBuckets`

下列程式碼範例會示範如何使用`ListBuckets`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

列出儲存貯體。

```
import { ListBucketsCommand, S3Client } from "@aws-sdk/client-s3";

const client = new S3Client({});

export const main = async () => {
  const command = new ListBucketsCommand({});
```

```
  try {
    const { Owner, Buckets } = await client.send(command);
    console.log(
      `${Owner.DisplayName} owns ${Buckets.length} bucket${
        Buckets.length === 1 ? "" : "s"
      }:`,
    );
    console.log(`${Buckets.map((b) => ` • ${b.Name}`).join("\n")}`);
  } catch (err) {
    console.error(err);
  }
};
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考ListBuckets中的。

`ListObjectsV2`

下列程式碼範例會示範如何使用`ListObjectsV2`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

條列儲存貯體中的所有物件。如果有多個對象， IsTruncated 並且 NextContinuationToken 將被用於遍歷完整列表。

```
import {
  S3Client,
  // This command supersedes the ListObjectsCommand and is the recommended way to
 list objects.
  ListObjectsV2Command,
} from "@aws-sdk/client-s3";

const client = new S3Client({});
```

```
export const main = async () => {
  const command = new ListObjectsV2Command({
    Bucket: "my-bucket",
    // The default and maximum number of keys returned is 1000. This limits it to
    // one for demonstration purposes.
    MaxKeys: 1,
  });

  try {
    let isTruncated = true;

    console.log("Your bucket contains the following objects:\n");
    let contents = "";

    while (isTruncated) {
      const { Contents, IsTruncated, NextContinuationToken } =
        await client.send(command);
      const contentsList = Contents.map((c) => ` • ${c.Key}`).join("\n");
      contents += contentsList + "\n";
      isTruncated = IsTruncated;
      command.input.ContinuationToken = NextContinuationToken;
    }
    console.log(contents);
  } catch (err) {
    console.error(err);
  }
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考中的 ListObjectsV2。

`PutBucketAcl`

下列程式碼範例會示範如何使用`PutBucketAcl`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

把桶ACL。

```
import { PutBucketAclCommand, S3Client } from "@aws-sdk/client-s3";

const client = new S3Client({});

// Most Amazon S3 use cases don't require the use of access control lists (ACLs).
// We recommend that you disable ACLs, except in unusual circumstances where
// you need to control access for each object individually.
// Consider a policy instead. For more information see https://docs.aws.amazon.com/
AmazonS3/latest/userguide/bucket-policies.html.
export const main = async () => {
  // Grant a user READ access to a bucket.
  const command = new PutBucketAclCommand({
    Bucket: "test-bucket",
    AccessControlPolicy: {
      Grants: [
        {
          Grantee: {
            // The canonical ID of the user. This ID is an obfuscated form of your
 AWS account number.
            // It's unique to Amazon S3 and can't be found elsewhere.
            // For more information, see https://docs.aws.amazon.com/AmazonS3/
latest/userguide/finding-canonical-user-id.html.
            ID: "canonical-id-1",
            Type: "CanonicalUser",
          },
          // One of FULL_CONTROL | READ | WRITE | READ_ACP | WRITE_ACP
          // https://docs.aws.amazon.com/AmazonS3/latest/API/
API_Grant.html#AmazonS3-Type-Grant-Permission
          Permission: "FULL_CONTROL",
        },
      ],
      Owner: {
        ID: "canonical-id-2",
      },
    },
  });

  try {
    const response = await client.send(command);
    console.log(response);
  } catch (err) {
    console.error(err);
```

```
  }
};
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考PutBucketAcl中的。

`PutBucketCors`

下列程式碼範例會示範如何使用`PutBucketCors`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

新增CORS規則。

```
import { PutBucketCorsCommand, S3Client } from "@aws-sdk/client-s3";

const client = new S3Client({});

// By default, Amazon S3 doesn't allow cross-origin requests. Use this command
// to explicitly allow cross-origin requests.
export const main = async () => {
  const command = new PutBucketCorsCommand({
    Bucket: "test-bucket",
    CORSConfiguration: {
      CORSRules: [
        {
          // Allow all headers to be sent to this bucket.
          AllowedHeaders: ["*"],
          // Allow only GET and PUT methods to be sent to this bucket.
          AllowedMethods: ["GET", "PUT"],
          // Allow only requests from the specified origin.
          AllowedOrigins: ["https://www.example.com"],
          // Allow the entity tag (ETag) header to be returned in the response. The
 ETag header
```

```
          // The entity tag represents a specific version of the object. The ETag
 reflects
          // changes only to the contents of an object, not its metadata.
          ExposeHeaders: ["ETag"],
          // How long the requesting browser should cache the preflight response.
 After
          // this time, the preflight request will have to be made again.
          MaxAgeSeconds: 3600,
        },
      ],
    },
  });

  try {
    const response = await client.send(command);
    console.log(response);
  } catch (err) {
    console.error(err);
  }
};
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考PutBucketCors中的。

`PutBucketPolicy`

下列程式碼範例會示範如何使用`PutBucketPolicy`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

新增政策。

```
import { PutBucketPolicyCommand, S3Client } from "@aws-sdk/client-s3";
```

```
const client = new S3Client({});

export const main = async () => {
  const command = new PutBucketPolicyCommand({
    Policy: JSON.stringify({
      Version: "2012-10-17",
      Statement: [
        {
          Sid: "AllowGetObject",
          // Allow this particular user to call GetObject on any object in this
 bucket.
          Effect: "Allow",
          Principal: {
            AWS: "arn:aws:iam::ACCOUNT-ID:user/USERNAME",
          },
          Action: "s3:GetObject",
          Resource: "arn:aws:s3:::BUCKET-NAME/*",
        },
      ],
    }),
    // Apply the preceding policy to this bucket.
    Bucket: "BUCKET-NAME",
  });

  try {
    const response = await client.send(command);
    console.log(response);
  } catch (err) {
    console.error(err);
  }
};
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考PutBucketPolicy中的。

PutBucketWebsite

下列程式碼範例會示範如何使用PutBucketWebsite。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

設定儲存貯體網站組態。

```
import { PutBucketWebsiteCommand, S3Client } from "@aws-sdk/client-s3";

const client = new S3Client({});

// Set up a bucket as a static website.
// The bucket needs to be publicly accessible.
export const main = async () => {
  const command = new PutBucketWebsiteCommand({
    Bucket: "test-bucket",
    WebsiteConfiguration: {
      ErrorDocument: {
        // The object key name to use when a 4XX class error occurs.
        Key: "error.html",
      },
      IndexDocument: {
        // A suffix that is appended to a request that is for a directory.
        Suffix: "index.html",
      },
    },
  });

  try {
    const response = await client.send(command);
    console.log(response);
  } catch (err) {
    console.error(err);
  }
};
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考PutBucketWebsite中的。

`PutObject`

下列程式碼範例會示範如何使用`PutObject`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

上傳物件。

```
import { PutObjectCommand, S3Client } from "@aws-sdk/client-s3";

const client = new S3Client({});

export const main = async () => {
  const command = new PutObjectCommand({
    Bucket: "test-bucket",
    Key: "hello-s3.txt",
    Body: "Hello S3!",
  });

  try {
    const response = await client.send(command);
    console.log(response);
  } catch (err) {
    console.error(err);
  }
};
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考PutObject中的。

`PutObjectLegalHold`

下列程式碼範例會示範如何使用`PutObjectLegalHold`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import { fileURLToPath } from "url";
import { PutObjectLegalHoldCommand, S3Client } from "@aws-sdk/client-s3";

/**
 * @param {S3Client} client
 * @param {string} bucketName
 * @param {string} objectKey
 */
export const main = async (client, bucketName, objectKey) => {
  const command = new PutObjectLegalHoldCommand({
    Bucket: bucketName,
    Key: objectKey,
    LegalHold: {
      // Set the status to 'ON' to place a legal hold on the object.
      // Set the status to 'OFF' to remove the legal hold.
      Status: "ON",
    },
    // Optionally, you can provide additional parameters
    // ChecksumAlgorithm: "ALGORITHM",
    // ContentMD5: "MD5_HASH",
    // ExpectedBucketOwner: "ACCOUNT_ID",
    // RequestPayer: "requester",
    // VersionId: "OBJECT_VERSION_ID",
  });

  try {
    const response = await client.send(command);
    console.log(
      `Object legal hold status: ${response.$metadata.httpStatusCode}`,
    );
  } catch (err) {
    console.error(err);
  }
};
```

```
// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  main(new S3Client(), "BUCKET_NAME", "OBJECT_KEY");
}
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考PutObjectLegalHold中的。

`PutObjectLockConfiguration`

下列程式碼範例會示範如何使用`PutObjectLockConfiguration`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

設定值區的物件鎖定組態。

```
import { fileURLToPath } from "url";
import {
  PutObjectLockConfigurationCommand,
  S3Client,
} from "@aws-sdk/client-s3";

/**
 * @param {S3Client} client
 * @param {string} bucketName
 */
export const main = async (client, bucketName) => {
  const command = new PutObjectLockConfigurationCommand({
    Bucket: bucketName,
    // The Object Lock configuration that you want to apply to the specified bucket.
    ObjectLockConfiguration: {
      ObjectLockEnabled: "Enabled",
    },
    // Optionally, you can provide additional parameters
    // ExpectedBucketOwner: "ACCOUNT_ID",
```

```
    // RequestPayer: "requester",
    // Token: "OPTIONAL_TOKEN",
  });

  try {
    const response = await client.send(command);
    console.log(
      `Object Lock Configuration updated: ${response.$metadata.httpStatusCode}`,
    );
  } catch (err) {
    console.error(err);
  }
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  main(new S3Client(), "BUCKET_NAME");
}
```

設定值區的預設保留期間。

```
import { fileURLToPath } from "url";
import {
  PutObjectLockConfigurationCommand,
  S3Client,
} from "@aws-sdk/client-s3";

/**
 * @param {S3Client} client
 * @param {string} bucketName
 */
export const main = async (client, bucketName) => {
  const command = new PutObjectLockConfigurationCommand({
    Bucket: bucketName,
    // The Object Lock configuration that you want to apply to the specified bucket.
    ObjectLockConfiguration: {
      ObjectLockEnabled: "Enabled",
      Rule: {
        DefaultRetention: {
          Mode: "GOVERNANCE",
          Years: 3,
        },
```

```
      },
    },
    // Optionally, you can provide additional parameters
    // ExpectedBucketOwner: "ACCOUNT_ID",
    // RequestPayer: "requester",
    // Token: "OPTIONAL_TOKEN",
  });

  try {
    const response = await client.send(command);
    console.log(
      `Default Object Lock Configuration updated: ${response.
$metadata.httpStatusCode}`,
    );
  } catch (err) {
    console.error(err);
  }
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  main(new S3Client(), "BUCKET_NAME");
}
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考PutObjectLockConfiguration中的。

`PutObjectRetention`

下列程式碼範例會示範如何使用`PutObjectRetention`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import { fileURLToPath } from "url";
import { PutObjectRetentionCommand, S3Client } from "@aws-sdk/client-s3";
```

```
/**
 * @param {S3Client} client
 * @param {string} bucketName
 * @param {string} objectKey
 */
export const main = async (client, bucketName, objectKey) => {
  const command = new PutObjectRetentionCommand({
    Bucket: bucketName,
    Key: objectKey,
    BypassGovernanceRetention: false,
    // ChecksumAlgorithm: "ALGORITHM",
    // ContentMD5: "MD5_HASH",
    // ExpectedBucketOwner: "ACCOUNT_ID",
    // RequestPayer: "requester",
    Retention: {
      Mode: "GOVERNANCE", // or "COMPLIANCE"
      RetainUntilDate: new Date(new Date().getTime() + 24 * 60 * 60 * 1000),
    },
    // VersionId: "OBJECT_VERSION_ID",
  });

  try {
    const response = await client.send(command);
    console.log(
      `Object Retention settings updated: ${response.$metadata.httpStatusCode}`,
    );
  } catch (err) {
    console.error(err);
  }
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  main(new S3Client(), "BUCKET_NAME", "OBJECT_KEY");
}
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考PutObjectRetention中的。

案例

建立預先簽署 URL

下列程式碼範例示範如何URL為 Amazon S3 建立預先簽署並上傳物件。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

建立預先簽署URL以將物件上傳至值區。

```
import https from "https";
import { PutObjectCommand, S3Client } from "@aws-sdk/client-s3";
import { fromIni } from "@aws-sdk/credential-providers";
import { HttpRequest } from "@smithy/protocol-http";
import {
  getSignedUrl,
  S3RequestPresigner,
} from "@aws-sdk/s3-request-presigner";
import { parseUrl } from "@smithy/url-parser";
import { formatUrl } from "@aws-sdk/util-format-url";
import { Hash } from "@smithy/hash-node";

const createPresignedUrlWithoutClient = async ({ region, bucket, key }) => {
  const url = parseUrl(`https://${bucket}.s3.${region}.amazonaws.com/${key}`);
  const presigner = new S3RequestPresigner({
    credentials: fromIni(),
    region,
    sha256: Hash.bind(null, "sha256"),
  });

  const signedUrlObject = await presigner.presign(
    new HttpRequest({ ...url, method: "PUT" }),
  );
  return formatUrl(signedUrlObject);
};

const createPresignedUrlWithClient = ({ region, bucket, key }) => {
```

```
  const client = new S3Client({ region });
  const command = new PutObjectCommand({ Bucket: bucket, Key: key });
  return getSignedUrl(client, command, { expiresIn: 3600 });
};

function put(url, data) {
  return new Promise((resolve, reject) => {
    const req = https.request(
      url,
      { method: "PUT", headers: { "Content-Length": new Blob([data]).size } },
      (res) => {
        let responseBody = "";
        res.on("data", (chunk) => {
          responseBody += chunk;
        });
        res.on("end", () => {
          resolve(responseBody);
        });
      },
    );
    req.on("error", (err) => {
      reject(err);
    });
    req.write(data);
    req.end();
  });
}

export const main = async () => {
  const REGION = "us-east-1";
  const BUCKET = "example_bucket";
  const KEY = "example_file.txt";

  // There are two ways to generate a presigned URL.
  // 1. Use createPresignedUrl without the S3 client.
  // 2. Use getSignedUrl in conjunction with the S3 client and GetObjectCommand.
  try {
    const noClientUrl = await createPresignedUrlWithoutClient({
      region: REGION,
      bucket: BUCKET,
      key: KEY,
    });

    const clientUrl = await createPresignedUrlWithClient({
```

```
      region: REGION,
      bucket: BUCKET,
      key: KEY,
    });

    // After you get the presigned URL, you can provide your own file
    // data. Refer to put() above.
    console.log("Calling PUT using presigned URL without client");
    await put(noClientUrl, "Hello World");

    console.log("Calling PUT using presigned URL with client");
    await put(clientUrl, "Hello World");

    console.log("\nDone. Check your S3 console.");
  } catch (err) {
    console.error(err);
  }
};
```

建立預先簽署URL以從值區下載物件。

```
import { GetObjectCommand, S3Client } from "@aws-sdk/client-s3";
import { fromIni } from "@aws-sdk/credential-providers";
import { HttpRequest } from "@smithy/protocol-http";
import {
  getSignedUrl,
  S3RequestPresigner,
} from "@aws-sdk/s3-request-presigner";
import { parseUrl } from "@smithy/url-parser";
import { formatUrl } from "@aws-sdk/util-format-url";
import { Hash } from "@smithy/hash-node";

const createPresignedUrlWithoutClient = async ({ region, bucket, key }) => {
  const url = parseUrl(`https://${bucket}.s3.${region}.amazonaws.com/${key}`);
  const presigner = new S3RequestPresigner({
    credentials: fromIni(),
    region,
    sha256: Hash.bind(null, "sha256"),
  });

  const signedUrlObject = await presigner.presign(new HttpRequest(url));
  return formatUrl(signedUrlObject);
```

```
};

const createPresignedUrlWithClient = ({ region, bucket, key }) => {
  const client = new S3Client({ region });
  const command = new GetObjectCommand({ Bucket: bucket, Key: key });
  return getSignedUrl(client, command, { expiresIn: 3600 });
};

export const main = async () => {
  const REGION = "us-east-1";
  const BUCKET = "example_bucket";
  const KEY = "example_file.jpg";

  try {
    const noClientUrl = await createPresignedUrlWithoutClient({
      region: REGION,
      bucket: BUCKET,
      key: KEY,
    });

    const clientUrl = await createPresignedUrlWithClient({
      region: REGION,
      bucket: BUCKET,
      key: KEY,
    });

    console.log("Presigned URL without client");
    console.log(noClientUrl);
    console.log("\n");

    console.log("Presigned URL with client");
    console.log(clientUrl);
  } catch (err) {
    console.error(err);
  }
};
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。

建立無伺服器應用程式來管理相片

下列程式碼範例示範如何建立無伺服器應用程式，讓使用者以標籤管理相片。

SDK對於 JavaScript（3）

顯示如何開發照片資產管理應用程式，以便使用 Amazon Rekognition 偵測圖片中的標籤，並將其儲存以供日後擷取。

有關如何設置和運行的完整源代碼和說明，請參閱中的完整示例 GitHub。

如要深入探索此範例的來源，請參閱 AWS 社群上的文章。

此範例中使用的服務

- API閘道
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

建立列出 Amazon S3 物件的網頁

下列程式碼範例顯示如何在網頁中列出 Amazon S3 物件。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

下列程式碼是相關的 React 元件，會呼叫 AWS SDK. 可以在前面 GitHub 的鏈接中找到包含此組件的應用程序的可運行版本。

```
import { useEffect, useState } from "react";
import {
  ListObjectsCommand,
  ListObjectsCommandOutput,
  S3Client,
} from "@aws-sdk/client-s3";
import { fromCognitoIdentityPool } from "@aws-sdk/credential-providers";
import "./App.css";
```

```
function App() {
  const [objects, setObjects] = useState<
    Required<ListObjectsCommandOutput>["Contents"]
  >([]);

  useEffect(() => {
    const client = new S3Client({
      region: "us-east-1",
      // Unless you have a public bucket, you'll need access to a private bucket.
      // One way to do this is to create an Amazon Cognito identity pool, attach a
 role to the pool,
      // and grant the role access to the 's3:GetObject' action.
      //
      // You'll also need to configure the CORS settings on the bucket to allow
 traffic from
      // this example site. Here's an example configuration that allows all origins.
 Don't
      // do this in production.
      //[
      //  {
      //    "AllowedHeaders": ["*"],
      //    "AllowedMethods": ["GET"],
      //    "AllowedOrigins": ["*"],
      //    "ExposeHeaders": [],
      //  },
      //]
      //
      credentials: fromCognitoIdentityPool({
        clientConfig: { region: "us-east-1" },
        identityPoolId: "<YOUR_IDENTITY_POOL_ID>",
      }),
    });
    const command = new ListObjectsCommand({ Bucket: "bucket-name" });
    client.send(command).then(({ Contents }) => setObjects(Contents || []));
  }, []);

  return (
    <div className="App">
      {objects.map((o) => (
        <div key={o.ETag}>{o.Key}</div>
      ))}
    </div>
  );
```

```
}

export default App;
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考ListObjects中的。

建立 Amazon Textract Explorer 應用程式

下列程式碼範例顯示如何透過互動式應用程式探索 Amazon Textract 輸出。

SDK對於 JavaScript（3）

示範如何使用建置 React 應用程式，該應用程式使用 Amazon Textract 擷取文件影像中的資料，並將其顯示在互動式網頁中。 AWS SDK for JavaScript 此範例會在 Web 瀏覽器中執行，且登入資料需要經過驗證的 Amazon Cognito 身分。它使用 Amazon 簡單存儲服務（Amazon S3）進行存儲，並對於通知，它會輪詢訂閱 Amazon 簡單通知服務（AmazonSQS）主題的 Amazon 簡單隊列（亞馬遜SNS）隊列。

有關如何設置和運行的完整源代碼和說明，請參閱中的完整示例GitHub。

此範例中使用的服務

- Amazon Cognito Identity
- Amazon S3
- Amazon SNS
- Amazon SQS
- Amazon Textract

PPE在影像中偵測

下列程式碼範例示範如何建立使用 Amazon Rekognition 偵測影像中個人防護設備 (PPE) 的應用程式。

SDK對於 JavaScript（3）

示範如何搭配使用 Amazon Rekognition AWS SDK for JavaScript 來建立應用程式，以偵測位於亞馬遜簡單儲存服務 (Amazon S3PPE) 儲存貯體中的映像中的個人防護設備 ()。該應用程式會將結果儲存到 Amazon DynamoDB 表，並使用 Amazon 簡單電子郵件服務 (AmazonSES) 向管理員傳送包含結果的電子郵件通知。

了解如何：

- 使用 Amazon Cognito 建立未經身分驗證的使用者。
- 分析圖像以PPE使用 Amazon Rekognition。
- 驗證 Amazon 的電子郵件地址SES。
- 以結果更新 DynamoDB 資料表。
- 使用 Amazon 發送電子郵件通知SES。

有關如何設置和運行的完整源代碼和說明，請參閱中的完整示例GitHub。

此範例中使用的服務

- DynamoDB
- Amazon Rekognition
- Amazon S3
- Amazon SES

偵測映像中的物件

下列程式碼範例示範如何建立使用 Amazon Rekognition 依照類別偵測影像中物件的應用程式。

SDK對於 JavaScript（3）

示範如何搭配使用 Amazon Rekognition AWS SDK for JavaScript 來建立使用 Amazon Rekognition 的應用程式，在位於亞馬遜簡單儲存服務 (Amazon S3) 儲存貯體中的映像中依類別識別物件。該應用程序向管理員發送電子郵件通知，其中包含使用 Amazon 簡單電子郵件服務（AmazonSES）的結果

了解如何：

- 使用 Amazon Cognito 建立未經身分驗證的使用者。
- 使用 Amazon Rekognition 分析映像中的物件。
- 驗證 Amazon 的電子郵件地址SES。
- 使用 Amazon 發送電子郵件通知SES。

有關如何設置和運行的完整源代碼和說明，請參閱中的完整示例GitHub。

此範例中使用的服務

- Amazon Rekognition
- Amazon S3

- Amazon SES

取得物件的合法訴訟保留組態

下列程式碼範例顯示如何取得 S3 儲存貯體的合法保留組態。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import { fileURLToPath } from "url";
import { GetObjectLegalHoldCommand, S3Client } from "@aws-sdk/client-s3";

/**
 * @param {S3Client} client
 * @param {string} bucketName
 * @param {string} objectKey
 */
export const main = async (client, bucketName, objectKey) => {
  const command = new GetObjectLegalHoldCommand({
    Bucket: bucketName,
    Key: objectKey,
    // Optionally, you can provide additional parameters
    // ExpectedBucketOwner: "ACCOUNT_ID",
    // RequestPayer: "requester",
    // VersionId: "OBJECT_VERSION_ID",
  });

  try {
    const response = await client.send(command);
    console.log(`Legal Hold Status: ${response.LegalHold.Status}`);
  } catch (err) {
    console.error(err);
  }
};

// Invoke main function if this file was run directly.
```

```
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  main(new S3Client(), "DOC-EXAMPLE-BUCKET", "OBJECT_KEY");
}
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考GetObjectLegalHold中的。

鎖定 Amazon S3 對象

下列程式碼範例顯示如何使用 S3 物件鎖定功能。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

index.js-工作流程的入口點。這會協調所有步驟。請造訪 GitHub 以查看 Scenario、ScenarioInput、 ScenarioOutput和的實作詳細資訊 ScenarioAction。

```
import * as Scenarios from "@aws-doc-sdk-examples/lib/scenario/index.js";
import {
  exitOnFalse,
  loadState,
  saveState,
} from "@aws-doc-sdk-examples/lib/scenario/steps-common.js";

import { welcome, welcomeContinue } from "./welcome.steps.js";
import {
  confirmCreateBuckets,
  confirmPopulateBuckets,
  confirmSetLegalHoldFileEnabled,
  confirmSetLegalHoldFileRetention,
  confirmSetRetentionPeriodFileEnabled,
  confirmSetRetentionPeriodFileRetention,
  confirmUpdateLockPolicy,
  confirmUpdateRetention,
  createBuckets,
  createBucketsAction,
  populateBuckets,
```

```
  populateBucketsAction,
  setLegalHoldFileEnabledAction,
  setLegalHoldFileRetentionAction,
  setRetentionPeriodFileEnabledAction,
  setRetentionPeriodFileRetentionAction,
  updateLockPolicy,
  updateLockPolicyAction,
  updateRetention,
  updateRetentionAction,
} from "./setup.steps.js";

/**
 * @param {Scenarios} scenarios
 * @param {Record<string, any>} initialState
 */
export const getWorkflowStages = (scenarios, initialState = {}) => {
  const client = new S3Client({});

  return {
    deploy: new scenarios.Scenario(
      "S3 Object Locking - Deploy",
      [
        welcome(scenarios),
        welcomeContinue(scenarios),
        exitOnFalse(scenarios, "welcomeContinue"),
        createBuckets(scenarios),
        confirmCreateBuckets(scenarios),
        exitOnFalse(scenarios, "confirmCreateBuckets"),
        createBucketsAction(scenarios, client),
        updateRetention(scenarios),
        confirmUpdateRetention(scenarios),
        exitOnFalse(scenarios, "confirmUpdateRetention"),
        updateRetentionAction(scenarios, client),
        populateBuckets(scenarios),
        confirmPopulateBuckets(scenarios),
        exitOnFalse(scenarios, "confirmPopulateBuckets"),
        populateBucketsAction(scenarios, client),
        updateLockPolicy(scenarios),
        confirmUpdateLockPolicy(scenarios),
        exitOnFalse(scenarios, "confirmUpdateLockPolicy"),
        updateLockPolicyAction(scenarios, client),
        confirmSetLegalHoldFileEnabled(scenarios),
        setLegalHoldFileEnabledAction(scenarios, client),
        confirmSetRetentionPeriodFileEnabled(scenarios),
```

```
        setRetentionPeriodFileEnabledAction(scenarios, client),
        confirmSetLegalHoldFileRetention(scenarios),
        setLegalHoldFileRetentionAction(scenarios, client),
        confirmSetRetentionPeriodFileRetention(scenarios),
        setRetentionPeriodFileRetentionAction(scenarios, client),
        saveState,
      ],
      initialState,
    ),
    demo: new scenarios.Scenario(
      "S3 Object Locking - Demo",
      [loadState, replAction(scenarios, client)],
      initialState,
    ),
    clean: new scenarios.Scenario(
      "S3 Object Locking - Destroy",
      [
        loadState,
        confirmCleanup(scenarios),
        exitOnFalse(scenarios, "confirmCleanup"),
        cleanupAction(scenarios, client),
      ],
      initialState,
    ),
  };
};

// Call function if run directly
import { fileURLToPath } from "url";
import { S3Client } from "@aws-sdk/client-s3";
import { cleanupAction, confirmCleanup } from "./clean.steps.js";
import { replAction } from "./repl.steps.js";

if (process.argv[1] === fileURLToPath(import.meta.url)) {
  const objectLockingScenarios = getWorkflowStages(Scenarios);
  Scenarios.parseScenarioArgs(objectLockingScenarios);
}
```

welcome.steps.js-將歡迎消息輸出到控制台。

```
/**
 * @typedef {import("@aws-doc-sdk-examples/lib/scenario/index.js")} Scenarios
```

```
 */

/**
 * @param {Scenarios} scenarios
 */
const welcome = (scenarios) =>
  new scenarios.ScenarioOutput(
    "welcome",
    `Welcome to the Amazon Simple Storage Service (S3) Object Locking Workflow
 Scenario. For this workflow, we will use the AWS SDK for JavaScript to create
 several S3 buckets and files to demonstrate working with S3 locking features.`,
    { header: true },
  );

/**
 * @param {Scenarios} scenarios
 */
const welcomeContinue = (scenarios) =>
  new scenarios.ScenarioInput(
    "welcomeContinue",
    "Press Enter when you are ready to start.",
    { type: "confirm" },
  );

export { welcome, welcomeContinue };
```

setup.steps.js-部署存儲桶，對象和文件設置。

```
import {
  BucketVersioningStatus,
  ChecksumAlgorithm,
  CreateBucketCommand,
  MFADeleteStatus,
  PutBucketVersioningCommand,
  PutObjectCommand,
  PutObjectLockConfigurationCommand,
  PutObjectLegalHoldCommand,
  PutObjectRetentionCommand,
  ObjectLockLegalHoldStatus,
  ObjectLockRetentionMode,
} from "@aws-sdk/client-s3";
```

```
/**
 * @typedef {import("@aws-doc-sdk-examples/lib/scenario/index.js")} Scenarios
 */

/**
 * @typedef {import("@aws-sdk/client-s3").S3Client} S3Client
 */

const bucketPrefix = "js-object-locking";

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const createBuckets = (scenarios) =>
  new scenarios.ScenarioOutput(
    "createBuckets",
    `The following buckets will be created:
         ${bucketPrefix}-no-lock with object lock False.
         ${bucketPrefix}-lock-enabled with object lock True.
         ${bucketPrefix}-retention-after-creation with object lock False.`,
    { preformatted: true },
  );

/**
 * @param {Scenarios} scenarios
 */
const confirmCreateBuckets = (scenarios) =>
  new scenarios.ScenarioInput("confirmCreateBuckets", "Create the buckets?", {
    type: "confirm",
  });

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const createBucketsAction = (scenarios, client) =>
  new scenarios.ScenarioAction("createBucketsAction", async (state) => {
    const noLockBucketName = `${bucketPrefix}-no-lock`;
    const lockEnabledBucketName = `${bucketPrefix}-lock-enabled`;
    const retentionBucketName = `${bucketPrefix}-retention-after-creation`;

    await client.send(new CreateBucketCommand({ Bucket: noLockBucketName }));
    await client.send(
```

```
      new CreateBucketCommand({
        Bucket: lockEnabledBucketName,
        ObjectLockEnabledForBucket: true,
      }),
    );
    await client.send(new CreateBucketCommand({ Bucket: retentionBucketName }));

    state.noLockBucketName = noLockBucketName;
    state.lockEnabledBucketName = lockEnabledBucketName;
    state.retentionBucketName = retentionBucketName;
  });

/**
 * @param {Scenarios} scenarios
 */
const populateBuckets = (scenarios) =>
  new scenarios.ScenarioOutput(
    "populateBuckets",
    `The following test files will be created:
         file0.txt in ${bucketPrefix}-no-lock.
         file1.txt in ${bucketPrefix}-no-lock.
         file0.txt in ${bucketPrefix}-lock-enabled.
         file1.txt in ${bucketPrefix}-lock-enabled.
         file0.txt in ${bucketPrefix}-retention-after-creation.
         file1.txt in ${bucketPrefix}-retention-after-creation.`,
    { preformatted: true },
  );

/**
 * @param {Scenarios} scenarios
 */
const confirmPopulateBuckets = (scenarios) =>
  new scenarios.ScenarioInput(
    "confirmPopulateBuckets",
    "Populate the buckets?",
    { type: "confirm" },
  );

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const populateBucketsAction = (scenarios, client) =>
  new scenarios.ScenarioAction("populateBucketsAction", async (state) => {
```

```
    await client.send(
      new PutObjectCommand({
        Bucket: state.noLockBucketName,
        Key: "file0.txt",
        Body: "Content",
        ChecksumAlgorithm: ChecksumAlgorithm.SHA256,
      }),
    );
    await client.send(
      new PutObjectCommand({
        Bucket: state.noLockBucketName,
        Key: "file1.txt",
        Body: "Content",
        ChecksumAlgorithm: ChecksumAlgorithm.SHA256,
      }),
    );
    await client.send(
      new PutObjectCommand({
        Bucket: state.lockEnabledBucketName,
        Key: "file0.txt",
        Body: "Content",
        ChecksumAlgorithm: ChecksumAlgorithm.SHA256,
      }),
    );
    await client.send(
      new PutObjectCommand({
        Bucket: state.lockEnabledBucketName,
        Key: "file1.txt",
        Body: "Content",
        ChecksumAlgorithm: ChecksumAlgorithm.SHA256,
      }),
    );
    await client.send(
      new PutObjectCommand({
        Bucket: state.retentionBucketName,
        Key: "file0.txt",
        Body: "Content",
        ChecksumAlgorithm: ChecksumAlgorithm.SHA256,
      }),
    );
    await client.send(
      new PutObjectCommand({
        Bucket: state.retentionBucketName,
        Key: "file1.txt",
```

```
        Body: "Content",
        ChecksumAlgorithm: ChecksumAlgorithm.SHA256,
      }),
    );
  });

/**
 * @param {Scenarios} scenarios
 */
const updateRetention = (scenarios) =>
  new scenarios.ScenarioOutput(
    "updateRetention",
    `A bucket can be configured to use object locking with a default retention
 period.
   A default retention period will be configured for ${bucketPrefix}-retention-
after-creation.`,
    { preformatted: true },
  );

/**
 * @param {Scenarios} scenarios
 */
const confirmUpdateRetention = (scenarios) =>
  new scenarios.ScenarioInput(
    "confirmUpdateRetention",
    "Configure default retention period?",
    { type: "confirm" },
  );

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const updateRetentionAction = (scenarios, client) =>
  new scenarios.ScenarioAction("updateRetentionAction", async (state) => {
    await client.send(
      new PutBucketVersioningCommand({
        Bucket: state.retentionBucketName,
        VersioningConfiguration: {
          MFADelete: MFADeleteStatus.Disabled,
          Status: BucketVersioningStatus.Enabled,
        },
      }),
    );
```

```
    await client.send(
      new PutObjectLockConfigurationCommand({
        Bucket: state.retentionBucketName,
        ObjectLockConfiguration: {
          ObjectLockEnabled: "Enabled",
          Rule: {
            DefaultRetention: {
              Mode: "GOVERNANCE",
              Years: 1,
            },
          },
        },
      }),
    );
  });

/**
 * @param {Scenarios} scenarios
 */
const updateLockPolicy = (scenarios) =>
  new scenarios.ScenarioOutput(
    "updateLockPolicy",
    `Object lock policies can also be added to existing buckets.
   An object lock policy will be added to ${bucketPrefix}-lock-enabled.`,
    { preformatted: true },
  );

/**
 * @param {Scenarios} scenarios
 */
const confirmUpdateLockPolicy = (scenarios) =>
  new scenarios.ScenarioInput(
    "confirmUpdateLockPolicy",
    "Add object lock policy?",
    { type: "confirm" },
  );

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const updateLockPolicyAction = (scenarios, client) =>
  new scenarios.ScenarioAction("updateLockPolicyAction", async (state) => {
```

```
    await client.send(
      new PutObjectLockConfigurationCommand({
        Bucket: state.lockEnabledBucketName,
        ObjectLockConfiguration: {
          ObjectLockEnabled: "Enabled",
        },
      }),
    );
  });

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const confirmSetLegalHoldFileEnabled = (scenarios) =>
  new scenarios.ScenarioInput(
    "confirmSetLegalHoldFileEnabled",
    (state) =>
      `Would you like to add a legal hold to file0.txt in
 ${state.lockEnabledBucketName}?`,
    {
      type: "confirm",
    },
  );

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const setLegalHoldFileEnabledAction = (scenarios, client) =>
  new scenarios.ScenarioAction(
    "setLegalHoldFileEnabledAction",
    async (state) => {
      await client.send(
        new PutObjectLegalHoldCommand({
          Bucket: state.lockEnabledBucketName,
          Key: "file0.txt",
          LegalHold: {
            Status: ObjectLockLegalHoldStatus.ON,
          },
        }),
      );
      console.log(
        `Modified legal hold for file0.txt in ${state.lockEnabledBucketName}.`,
```

```
      );
    },
    { skipWhen: (state) => !state.confirmSetLegalHoldFileEnabled },
  );

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const confirmSetRetentionPeriodFileEnabled = (scenarios) =>
  new scenarios.ScenarioInput(
    "confirmSetRetentionPeriodFileEnabled",
    (state) =>
      `Would you like to add a 1 day Governance retention period to file1.txt in
 ${state.lockEnabledBucketName}?
Reminder: Only a user with the s3:BypassGovernanceRetention permission will be able
 to delete this file or its bucket until the retention period has expired.`,
    {
      type: "confirm",
    },
  );

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const setRetentionPeriodFileEnabledAction = (scenarios, client) =>
  new scenarios.ScenarioAction(
    "setRetentionPeriodFileEnabledAction",
    async (state) => {
      const retentionDate = new Date();
      retentionDate.setDate(retentionDate.getDate() + 1);
      await client.send(
        new PutObjectRetentionCommand({
          Bucket: state.lockEnabledBucketName,
          Key: "file1.txt",
          Retention: {
            Mode: ObjectLockRetentionMode.GOVERNANCE,
            RetainUntilDate: retentionDate,
          },
        }),
      );
      console.log(
```

```
        `Set retention for file1.txt in ${state.lockEnabledBucketName} until
 ${retentionDate.toISOString().split("T")[0]}.`,
      );
    },
    { skipWhen: (state) => !state.confirmSetRetentionPeriodFileEnabled },
  );

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const confirmSetLegalHoldFileRetention = (scenarios) =>
  new scenarios.ScenarioInput(
    "confirmSetLegalHoldFileRetention",
    (state) =>
      `Would you like to add a legal hold to file0.txt in
 ${state.retentionBucketName}?`,
    {
      type: "confirm",
    },
  );

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const setLegalHoldFileRetentionAction = (scenarios, client) =>
  new scenarios.ScenarioAction(
    "setLegalHoldFileRetentionAction",
    async (state) => {
      await client.send(
        new PutObjectLegalHoldCommand({
          Bucket: state.retentionBucketName,
          Key: "file0.txt",
          LegalHold: {
            Status: ObjectLockLegalHoldStatus.ON,
          },
        }),
      );
      console.log(
        `Modified legal hold for file0.txt in ${state.retentionBucketName}.`,
      );
    },
    { skipWhen: (state) => !state.confirmSetLegalHoldFileRetention },
```

```
  );

/**
 * @param {Scenarios} scenarios
 */
const confirmSetRetentionPeriodFileRetention = (scenarios) =>
  new scenarios.ScenarioInput(
    "confirmSetRetentionPeriodFileRetention",
    (state) =>
      `Would you like to add a 1 day Governance retention period to file1.txt in
 ${state.retentionBucketName}?
Reminder: Only a user with the s3:BypassGovernanceRetention permission will be able
 to delete this file or its bucket until the retention period has expired.`,
    {
      type: "confirm",
    },
  );

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const setRetentionPeriodFileRetentionAction = (scenarios, client) =>
  new scenarios.ScenarioAction(
    "setRetentionPeriodFileRetentionAction",
    async (state) => {
      const retentionDate = new Date();
      retentionDate.setDate(retentionDate.getDate() + 1);
      await client.send(
        new PutObjectRetentionCommand({
          Bucket: state.retentionBucketName,
          Key: "file1.txt",
          Retention: {
            Mode: ObjectLockRetentionMode.GOVERNANCE,
            RetainUntilDate: retentionDate,
          },
          BypassGovernanceRetention: true,
        }),
      );
      console.log(
        `Set retention for file1.txt in ${state.retentionBucketName} until
 ${retentionDate.toISOString().split("T")[0]}.`,
      );
    },
```

```
    { skipWhen: (state) => !state.confirmSetRetentionPeriodFileRetention },
  );

export {
  createBuckets,
  confirmCreateBuckets,
  createBucketsAction,
  populateBuckets,
  confirmPopulateBuckets,
  populateBucketsAction,
  updateRetention,
  confirmUpdateRetention,
  updateRetentionAction,
  updateLockPolicy,
  confirmUpdateLockPolicy,
  updateLockPolicyAction,
  confirmSetLegalHoldFileEnabled,
  setLegalHoldFileEnabledAction,
  confirmSetRetentionPeriodFileEnabled,
  setRetentionPeriodFileEnabledAction,
  confirmSetLegalHoldFileRetention,
  setLegalHoldFileRetentionAction,
  confirmSetRetentionPeriodFileRetention,
  setRetentionPeriodFileRetentionAction,
};
```

repl.steps.js-查看和刪除存儲桶中的文件。

```
import {
  ChecksumAlgorithm,
  DeleteObjectCommand,
  GetObjectLegalHoldCommand,
  GetObjectLockConfigurationCommand,
  GetObjectRetentionCommand,
  ListObjectVersionsCommand,
  PutObjectCommand,
} from "@aws-sdk/client-s3";

/**
 * @typedef {import("@aws-doc-sdk-examples/lib/scenario/index.js")} Scenarios
 */
```

```
/**
 * @typedef {import("@aws-sdk/client-s3").S3Client} S3Client
 */

const choices = {
  EXIT: 0,
  LIST_ALL_FILES: 1,
  DELETE_FILE: 2,
  DELETE_FILE_WITH_RETENTION: 3,
  OVERWRITE_FILE: 4,
  VIEW_RETENTION_SETTINGS: 5,
  VIEW_LEGAL_HOLD_SETTINGS: 6,
};

/**
 * @param {Scenarios} scenarios
 */
const replInput = (scenarios) =>
  new scenarios.ScenarioInput(
    "replChoice",
    `Explore the S3 locking features by selecting one of the following choices`,
    {
      type: "select",
      choices: [
        { name: "List all files in buckets", value: choices.LIST_ALL_FILES },
        { name: "Attempt to delete a file.", value: choices.DELETE_FILE },
        {
          name: "Attempt to delete a file with retention period bypass.",
          value: choices.DELETE_FILE_WITH_RETENTION,
        },
        { name: "Attempt to overwrite a file.", value: choices.OVERWRITE_FILE },
        {
          name: "View the object and bucket retention settings for a file.",
          value: choices.VIEW_RETENTION_SETTINGS,
        },
        {
          name: "View the legal hold settings for a file.",
          value: choices.VIEW_LEGAL_HOLD_SETTINGS,
        },
        { name: "Finish the workflow.", value: choices.EXIT },
      ],
    },
  );
```

```
/**
 * @param {S3Client} client
 * @param {string[]} buckets
 */
const getAllFiles = async (client, buckets) => {
  /** @type {{bucket: string, key: string, version: string}[]} */
  const files = [];
  for (const bucket of buckets) {
    const objectsResponse = await client.send(
      new ListObjectVersionsCommand({ Bucket: bucket }),
    );
    for (const version of objectsResponse.Versions || []) {
      const { Key, VersionId } = version;
      files.push({ bucket, key: Key, version: VersionId });
    }
  }

  return files;
};

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const replAction = (scenarios, client) =>
  new scenarios.ScenarioAction(
    "replAction",
    async (state) => {
      const files = await getAllFiles(client, [
        state.noLockBucketName,
        state.lockEnabledBucketName,
        state.retentionBucketName,
      ]);

      const fileInput = new scenarios.ScenarioInput(
        "selectedFile",
        "Select a file:",
        {
          type: "select",
          choices: files.map((file, index) => ({
            name: `${index + 1}: ${file.bucket}: ${file.key} (version: ${
              file.version
            })`,
            value: index,
```

```
          })),
        },
      );

      const { replChoice } = state;

      switch (replChoice) {
        case choices.LIST_ALL_FILES: {
          const files = await getAllFiles(client, [
            state.noLockBucketName,
            state.lockEnabledBucketName,
            state.retentionBucketName,
          ]);
          state.replOutput = files
            .map(
              (file) =>
                `${file.bucket}: ${file.key} (version: ${file.version})`,
            )
            .join("\n");
          break;
        }
        case choices.DELETE_FILE: {
          /** @type {number} */
          const fileToDelete = await fileInput.handle(state);
          const selectedFile = files[fileToDelete];
          try {
            await client.send(
              new DeleteObjectCommand({
                Bucket: selectedFile.bucket,
                Key: selectedFile.key,
                VersionId: selectedFile.version,
              }),
            );
            state.replOutput = `Deleted ${selectedFile.key} in
${selectedFile.bucket}.`;
          } catch (err) {
            state.replOutput = `Unable to delete object ${selectedFile.key} in
bucket ${selectedFile.bucket}: ${err.message}`;
          }
          break;
        }
        case choices.DELETE_FILE_WITH_RETENTION: {
          /** @type {number} */
          const fileToDelete = await fileInput.handle(state);
```

```
          const selectedFile = files[fileToDelete];
          try {
            await client.send(
              new DeleteObjectCommand({
                Bucket: selectedFile.bucket,
                Key: selectedFile.key,
                VersionId: selectedFile.version,
                BypassGovernanceRetention: true,
              }),
            );
            state.replOutput = `Deleted ${selectedFile.key} in
${selectedFile.bucket}.`;
          } catch (err) {
            state.replOutput = `Unable to delete object ${selectedFile.key} in
bucket ${selectedFile.bucket}: ${err.message}`;
          }
          break;
        }
        case choices.OVERWRITE_FILE: {
          /** @type {number} */
          const fileToOverwrite = await fileInput.handle(state);
          const selectedFile = files[fileToOverwrite];
          try {
            await client.send(
              new PutObjectCommand({
                Bucket: selectedFile.bucket,
                Key: selectedFile.key,
                Body: "New content",
                ChecksumAlgorithm: ChecksumAlgorithm.SHA256,
              }),
            );
            state.replOutput = `Overwrote ${selectedFile.key} in
${selectedFile.bucket}.`;
          } catch (err) {
            state.replOutput = `Unable to overwrite object ${selectedFile.key} in
bucket ${selectedFile.bucket}: ${err.message}`;
          }
          break;
        }
        case choices.VIEW_RETENTION_SETTINGS: {
          /** @type {number} */
          const fileToView = await fileInput.handle(state);
          const selectedFile = files[fileToView];
          try {
```

```
            const retention = await client.send(
              new GetObjectRetentionCommand({
                Bucket: selectedFile.bucket,
                Key: selectedFile.key,
                VersionId: selectedFile.version,
              }),
            );
            const bucketConfig = await client.send(
              new GetObjectLockConfigurationCommand({
                Bucket: selectedFile.bucket,
              }),
            );
            state.replOutput = `Object retention for ${selectedFile.key}
 in ${selectedFile.bucket}: ${retention.Retention?.Mode} until
 ${retention.Retention?.RetainUntilDate?.toISOString()}.
Bucket object lock config for ${selectedFile.bucket} in ${selectedFile.bucket}:
Enabled: ${bucketConfig.ObjectLockConfiguration?.ObjectLockEnabled}
Rule:
 ${JSON.stringify(bucketConfig.ObjectLockConfiguration?.Rule?.DefaultRetention)}`;
          } catch (err) {
            state.replOutput = `Unable to fetch object lock retention:
 '${err.message}'`;
          }
          break;
        }
        case choices.VIEW_LEGAL_HOLD_SETTINGS: {
          /** @type {number} */
          const fileToView = await fileInput.handle(state);
          const selectedFile = files[fileToView];
          try {
            const legalHold = await client.send(
              new GetObjectLegalHoldCommand({
                Bucket: selectedFile.bucket,
                Key: selectedFile.key,
                VersionId: selectedFile.version,
              }),
            );
            state.replOutput = `Object legal hold for ${selectedFile.key} in
 ${selectedFile.bucket}: Status: ${legalHold.LegalHold?.Status}`;
          } catch (err) {
            state.replOutput = `Unable to fetch legal hold: '${err.message}'`;
          }
          break;
        }
```

```
        default:
          throw new Error(`Invalid replChoice: ${replChoice}`);
      }
    },
    {
      whileConfig: {
        whileFn: ({ replChoice }) => replChoice !== choices.EXIT,
        input: replInput(scenarios),
        output: new scenarios.ScenarioOutput(
          "REPL output",
          (state) => state.replOutput,
          { preformatted: true },
        ),
      },
    },
  );

export { replInput, replAction, choices };
```

clean.steps.js-銷毀所有創建的資源。

```
import {
  DeleteObjectCommand,
  DeleteBucketCommand,
  ListObjectVersionsCommand,
  GetObjectLegalHoldCommand,
  GetObjectRetentionCommand,
  PutObjectLegalHoldCommand,
} from "@aws-sdk/client-s3";

/**
 * @typedef {import("@aws-doc-sdk-examples/lib/scenario/index.js")} Scenarios
 */

/**
 * @typedef {import("@aws-sdk/client-s3").S3Client} S3Client
 */

/**
 * @param {Scenarios} scenarios
 */
const confirmCleanup = (scenarios) =>
```

```
  new scenarios.ScenarioInput("confirmCleanup", "Clean up resources?", {
    type: "confirm",
  });

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const cleanupAction = (scenarios, client) =>
  new scenarios.ScenarioAction("cleanupAction", async (state) => {
    const { noLockBucketName, lockEnabledBucketName, retentionBucketName } =
      state;

    const buckets = [
      noLockBucketName,
      lockEnabledBucketName,
      retentionBucketName,
    ];

    for (const bucket of buckets) {
      /** @type {import("@aws-sdk/client-s3").ListObjectVersionsCommandOutput} */
      let objectsResponse;

      try {
        objectsResponse = await client.send(
          new ListObjectVersionsCommand({
            Bucket: bucket,
          }),
        );
      } catch (e) {
        if (e instanceof Error && e.name === "NoSuchBucket") {
          console.log("Object's bucket has already been deleted.");
          continue;
        } else {
          throw e;
        }
      }

      for (const version of objectsResponse.Versions || []) {
        const { Key, VersionId } = version;

        try {
          const legalHold = await client.send(
            new GetObjectLegalHoldCommand({
```

```
        Bucket: bucket,
        Key,
        VersionId,
      }),
    );

    if (legalHold.LegalHold?.Status === "ON") {
      await client.send(
        new PutObjectLegalHoldCommand({
          Bucket: bucket,
          Key,
          VersionId,
          LegalHold: {
            Status: "OFF",
          },
        }),
      );
    }
  } catch (err) {
    console.log(
      `Unable to fetch legal hold for ${Key} in ${bucket}: '${err.message}'`,
    );
  }

  try {
    const retention = await client.send(
      new GetObjectRetentionCommand({
        Bucket: bucket,
        Key,
        VersionId,
      }),
    );

    if (retention.Retention?.Mode === "GOVERNANCE") {
      await client.send(
        new DeleteObjectCommand({
          Bucket: bucket,
          Key,
          VersionId,
          BypassGovernanceRetention: true,
        }),
      );
    }
  } catch (err) {
```

```
          console.log(
            `Unable to fetch object lock retention for ${Key} in ${bucket}:
'${err.message}'`,
          );
        }

        await client.send(
          new DeleteObjectCommand({
            Bucket: bucket,
            Key,
            VersionId,
          }),
        );
      }

      await client.send(new DeleteBucketCommand({ Bucket: bucket }));
      console.log(`Delete for ${bucket} complete.`);
    }
  });

export { confirmCleanup, cleanupAction };
```

- 如需詳API細資訊，請參閱「AWS SDK for JavaScript API參考」中的下列主題。
  - GetObjectLegalHold
  - GetObjectLockConfiguration
  - GetObjectRetention
  - PutObjectLegalHold
  - PutObjectLockConfiguration
  - PutObjectRetention

上傳或下載大型檔案

下列程式碼範例示範如何將大型檔案上傳或下載到 Amazon S3，以及從 Amazon S3 上傳或下載。

如需詳細資訊，請參閱使用分段上傳以上傳物件。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

上傳大型檔案。

```
import {
  CreateMultipartUploadCommand,
  UploadPartCommand,
  CompleteMultipartUploadCommand,
  AbortMultipartUploadCommand,
  S3Client,
} from "@aws-sdk/client-s3";

const twentyFiveMB = 25 * 1024 * 1024;

export const createString = (size = twentyFiveMB) => {
  return "x".repeat(size);
};

export const main = async () => {
  const s3Client = new S3Client({});
  const bucketName = "test-bucket";
  const key = "multipart.txt";
  const str = createString();
  const buffer = Buffer.from(str, "utf8");

  let uploadId;

  try {
    const multipartUpload = await s3Client.send(
      new CreateMultipartUploadCommand({
        Bucket: bucketName,
        Key: key,
      }),
    );

    uploadId = multipartUpload.UploadId;
```

```
  const uploadPromises = [];
  // Multipart uploads require a minimum size of 5 MB per part.
  const partSize = Math.ceil(buffer.length / 5);

  // Upload each part.
  for (let i = 0; i < 5; i++) {
    const start = i * partSize;
    const end = start + partSize;
    uploadPromises.push(
      s3Client
        .send(
          new UploadPartCommand({
            Bucket: bucketName,
            Key: key,
            UploadId: uploadId,
            Body: buffer.subarray(start, end),
            PartNumber: i + 1,
          }),
        )
        .then((d) => {
          console.log("Part", i + 1, "uploaded");
          return d;
        }),
    );
  }

  const uploadResults = await Promise.all(uploadPromises);

  return await s3Client.send(
    new CompleteMultipartUploadCommand({
      Bucket: bucketName,
      Key: key,
      UploadId: uploadId,
      MultipartUpload: {
        Parts: uploadResults.map(({ ETag }, i) => ({
          ETag,
          PartNumber: i + 1,
        })),
      },
    }),
  );

  // Verify the output by downloading the file from the Amazon Simple Storage
Service (Amazon S3) console.
```

```
    // Because the output is a 25 MB string, text editors might struggle to open the
 file.
  } catch (err) {
    console.error(err);

    if (uploadId) {
      const abortCommand = new AbortMultipartUploadCommand({
        Bucket: bucketName,
        Key: key,
        UploadId: uploadId,
      });

      await s3Client.send(abortCommand);
    }
  }
};
```

下載大型檔案。

```
import { GetObjectCommand, S3Client } from "@aws-sdk/client-s3";
import { createWriteStream } from "fs";

const s3Client = new S3Client({});
const oneMB = 1024 * 1024;

export const getObjectRange = ({ bucket, key, start, end }) => {
  const command = new GetObjectCommand({
    Bucket: bucket,
    Key: key,
    Range: `bytes=${start}-${end}`,
  });

  return s3Client.send(command);
};

/**
 * @param {string | undefined} contentRange
 */
export const getRangeAndLength = (contentRange) => {
  const [range, length] = contentRange.split("/");
  const [start, end] = range.split("-");
  return {
```

```
    start: parseInt(start),
    end: parseInt(end),
    length: parseInt(length),
  };
};

export const isComplete = ({ end, length }) => end === length - 1;

// When downloading a large file, you might want to break it down into
// smaller pieces. Amazon S3 accepts a Range header to specify the start
// and end of the byte range to be downloaded.
const downloadInChunks = async ({ bucket, key }) => {
  const writeStream = createWriteStream(
    fileURLToPath(new URL(`./${key}`, import.meta.url)),
  ).on("error", (err) => console.error(err));

  let rangeAndLength = { start: -1, end: -1, length: -1 };

  while (!isComplete(rangeAndLength)) {
    const { end } = rangeAndLength;
    const nextRange = { start: end + 1, end: end + oneMB };

    console.log(`Downloading bytes ${nextRange.start} to ${nextRange.end}`);

    const { ContentRange, Body } = await getObjectRange({
      bucket,
      key,
      ...nextRange,
    });

    writeStream.write(await Body.transformToByteArray());
    rangeAndLength = getRangeAndLength(ContentRange);
  }
};

export const main = async () => {
  await downloadInChunks({
    bucket: "my-cool-bucket",
    key: "my-cool-object.txt",
  });
};
```

無伺服器範例

使用 Amazon S3 觸發條件調用 Lambda 函數

下列程式碼範例示範如何實作 Lambda 函數，該函數會接收透過將物件上傳至 S3 儲存貯體而觸發的事件。函數會從事件參數擷取 S3 儲存貯體名稱和物件金鑰，並呼叫 Amazon S3 API 以擷取和記錄物件的內容類型。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在無伺服器範例儲存庫中設定和執行。

使用使 Lambda JavaScript.

```
import { S3Client, HeadObjectCommand } from "@aws-sdk/client-s3";

const client = new S3Client();

export const handler = async (event, context) => {

    // Get the object from the event and show its content type
    const bucket = event.Records[0].s3.bucket.name;
    const key = decodeURIComponent(event.Records[0].s3.object.key.replace(/\+/g, '
 '));

    try {
        const { ContentType } = await client.send(new HeadObjectCommand({
            Bucket: bucket,
            Key: key,
        }));

        console.log('CONTENT TYPE:', ContentType);
        return ContentType;

    } catch (err) {
        console.log(err);
        const message = `Error getting object ${key} from bucket ${bucket}. Make
 sure they exist and your bucket is in the same region as this function.`;
        console.log(message);
```

```
        throw new Error(message);
    }
};
```

使用使 Lambda TypeScript.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { S3Event } from 'aws-lambda';
import { S3Client, HeadObjectCommand } from '@aws-sdk/client-s3';

const s3 = new S3Client({ region: process.env.AWS_REGION });

export const handler = async (event: S3Event): Promise<string | undefined> => {
  // Get the object from the event and show its content type
  const bucket = event.Records[0].s3.bucket.name;
  const key = decodeURIComponent(event.Records[0].s3.object.key.replace(/\+/g, '
 '));
  const params = {
    Bucket: bucket,
    Key: key,
  };
  try {
    const { ContentType } = await s3.send(new HeadObjectCommand(params));
    console.log('CONTENT TYPE:', ContentType);
    return ContentType;
  } catch (err) {
    console.log(err);
    const message = `Error getting object ${key} from bucket ${bucket}. Make sure
 they exist and your bucket is in the same region as this function.`;
    console.log(message);
    throw new Error(message);
  }
};
```

S3 冰川範SDK例使用 JavaScript (v3)

下列程式碼範例說明如何使用 AWS SDK for JavaScript (v3) 搭配 S3 Glacier 來執行動作和實作常見案例。

Actions 是大型程式的程式碼摘錄，必須在內容中執行。雖然動作會顯示如何呼叫個別服務函數，但您可以在其相關案例中查看內容中的動作。

每個範例都包含完整原始程式碼的連結，您可以在其中找到如何在內容中設定和執行程式碼的指示。

主題

- 動作

動作

`CreateVault`

下列程式碼範例會示範如何使用`CreateVault`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

建立用戶端。

```
const { GlacierClient } = require("@aws-sdk/client-glacier");
// Set the AWS Region.
const REGION = "REGION";
//Set the Redshift Service Object
const glacierClient = new GlacierClient({ region: REGION });
export { glacierClient };
```

建立保存庫。

```
// Load the SDK for JavaScript
import { CreateVaultCommand } from "@aws-sdk/client-glacier";
import { glacierClient } from "./libs/glacierClient.js";

// Set the parameters
```

```
const vaultname = "VAULT_NAME"; // VAULT_NAME
const params = { vaultName: vaultname };

const run = async () => {
  try {
    const data = await glacierClient.send(new CreateVaultCommand(params));
    console.log("Success, vault created!");
    return data; // For unit tests.
  } catch (err) {
    console.log("Error");
  }
};
run();
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考CreateVault中的。

SDK對於 JavaScript（第 2 個）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
// Load the SDK for JavaScript
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create a new service object
var glacier = new AWS.Glacier({ apiVersion: "2012-06-01" });
// Call Glacier to create the vault
glacier.createVault({ vaultName: "YOUR_VAULT_NAME" }, function (err) {
  if (!err) {
    console.log("Created vault!");
  }
});
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考CreateVault中的。

`UploadArchive`

下列程式碼範例會示範如何使用UploadArchive。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

建立用戶端。

```
const { GlacierClient } = require("@aws-sdk/client-glacier");
// Set the AWS Region.
const REGION = "REGION";
//Set the Redshift Service Object
const glacierClient = new GlacierClient({ region: REGION });
export { glacierClient };
```

上傳封存。

```
// Load the SDK for JavaScript
import { UploadArchiveCommand } from "@aws-sdk/client-glacier";
import { glacierClient } from "./libs/glacierClient.js";

// Set the parameters
const vaultname = "VAULT_NAME"; // VAULT_NAME

// Create a new service object and buffer
const buffer = new Buffer.alloc(2.5 * 1024 * 1024); // 2.5MB buffer
const params = { vaultName: vaultname, body: buffer };

const run = async () => {
  try {
    const data = await glacierClient.send(new UploadArchiveCommand(params));
    console.log("Archive ID", data.archiveId);
```

```
    return data; // For unit tests.
  } catch (err) {
    console.log("Error uploading archive!", err);
  }
};
run();
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考UploadArchive中的。

SDK對於 JavaScript（第 2 個）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
// Load the SDK for JavaScript
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create a new service object and buffer
var glacier = new AWS.Glacier({ apiVersion: "2012-06-01" });
buffer = Buffer.alloc(2.5 * 1024 * 1024); // 2.5MB buffer

var params = { vaultName: "YOUR_VAULT_NAME", body: buffer };
// Call Glacier to upload the archive.
glacier.uploadArchive(params, function (err, data) {
  if (err) {
    console.log("Error uploading archive!", err);
  } else {
    console.log("Archive ID", data.archiveId);
  }
});
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考UploadArchive中的。

SageMaker 使用 JavaScript （ v3 ）SDK的示例

下列程式碼範例會示範如何使用 AWS SDK for JavaScript (v3) 搭配使用來執行動作及實作常見案例 SageMaker。

Actions 是大型程式的程式碼摘錄，必須在內容中執行。雖然動作會顯示如何呼叫個別服務函數，但您可以在其相關案例中查看內容中的動作。

案例是程式碼範例，向您展示如何透過呼叫服務中的多個函式或與其他函式結合來完成特定工作 AWS 服務。

每個範例都包含完整原始程式碼的連結，您可以在其中找到如何在內容中設定和執行程式碼的指示。

開始使用

你好 SageMaker

下列程式碼範例會示範如何開始使用 SageMaker。

SDK對於 JavaScript （ 3 ）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import {
  SageMakerClient,
  ListNotebookInstancesCommand,
} from "@aws-sdk/client-sagemaker";

const client = new SageMakerClient({
  region: "us-west-2",
});

export const helloSagemaker = async () => {
  const command = new ListNotebookInstancesCommand({ MaxResults: 5 });

  const response = await client.send(command);
```

```
  console.log(
    "Hello Amazon SageMaker! Let's list some of your notebook instances:",
  );

  const instances = response.NotebookInstances || [];

  if (instances.length === 0) {
    console.log(
      "• No notebook instances found. Try creating one in the AWS Management Console
 or with the CreateNotebookInstanceCommand.",
    );
  } else {
    console.log(
      instances
        .map(
          (i) =>
            `• Instance: ${i.NotebookInstanceName}\n  Arn:${
              i.NotebookInstanceArn
            } \n  Creation Date: ${i.CreationTime.toISOString()}`,
        )
        .join("\n"),
    );
  }

  return response;
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考ListNotebookInstances中的。

主題

- 動作
- 案例

動作

CreatePipeline

下列程式碼範例會示範如何使用CreatePipeline。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

使用本機提供的JSON定義建立 SageMaker 管線的函數。

```
/**
 * Create the Amazon SageMaker pipeline using a JSON pipeline definition. The
 definition
 * can also be provided as an Amazon S3 object using PipelineDefinitionS3Location.
 * @param {{roleArn: string, name: string, sagemakerClient: import('@aws-sdk/client-
sagemaker').SageMakerClient}} props
 */
export async function createSagemakerPipeline({
  // Assumes an AWS IAM role has been created for this pipeline.
  roleArn,
  name,
  // Assumes an AWS Lambda function has been created for this pipeline.
  functionArn,
  sagemakerClient,
}) {
  const pipelineDefinition = readFileSync(
    // dirnameFromMetaUrl is a local utility function. You can find its
 implementation
    // on GitHub.
    `${dirnameFromMetaUrl(
      import.meta.url,
    )}../../../../../workflows/sagemaker_pipelines/resources/
GeoSpatialPipeline.json`,
  )
    .toString()
    .replace(/\*FUNCTION_ARN\*/g, functionArn);

  let arn = null;

  const createPipeline = () =>
    sagemakerClient.send(
      new CreatePipelineCommand({
        PipelineName: name,
```

```
        PipelineDefinition: pipelineDefinition,
        RoleArn: roleArn,
      }),
    );

  try {
    const { PipelineArn } = await createPipeline();
    arn = PipelineArn;
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "ValidationException" &&
      caught.message.includes(
        "Pipeline names must be unique within an AWS account and region",
      )
    ) {
      const { PipelineArn } = await sagemakerClient.send(
        new DescribePipelineCommand({ PipelineName: name }),
      );
      arn = PipelineArn;
    } else {
      throw caught;
    }
  }

  return {
    arn,
    cleanUp: async () => {
      await sagemakerClient.send(
        new DeletePipelineCommand({ PipelineName: name }),
      );
    },
  };
}
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考CreatePipeline中的。

DeletePipeline

下列程式碼範例會示範如何使用DeletePipeline。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

刪除 SageMaker 配管的語法。此代碼是更大函數的一部分。有關更多內容，請參閱「創建管線」或 GitHub 存儲庫。

```
    await sagemakerClient.send(
      new DeletePipelineCommand({ PipelineName: name }),
    );
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考DeletePipeline中的。

`DescribePipelineExecution`

下列程式碼範例會示範如何使用`DescribePipelineExecution`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

等待 SageMaker 管線執行成功、失敗或停止。

```
/**
 * Poll the executing pipeline until the status is 'SUCCEEDED', 'STOPPED', or
 'FAILED'.
 * @param {{ arn: string, sagemakerClient: import('@aws-sdk/client-
sagemaker').SageMakerClient, wait: (ms: number) => Promise<void>}} props
 */
export async function waitForPipelineComplete({ arn, sagemakerClient, wait }) {
```

```
  const command = new DescribePipelineExecutionCommand({
    PipelineExecutionArn: arn,
  });

  let complete = false;
  let intervalInSeconds = 15;
  const COMPLETION_STATUSES = [
    PipelineExecutionStatus.FAILED,
    PipelineExecutionStatus.STOPPED,
    PipelineExecutionStatus.SUCCEEDED,
  ];

  do {
    const { PipelineExecutionStatus: status, FailureReason } =
      await sagemakerClient.send(command);

    complete = COMPLETION_STATUSES.includes(status);

    if (!complete) {
      console.log(
        `Pipeline is ${status}. Waiting ${intervalInSeconds} seconds before checking
 again.`,
      );
      await wait(intervalInSeconds);
    } else if (status === PipelineExecutionStatus.FAILED) {
      throw new Error(`Pipeline failed because: ${FailureReason}`);
    } else if (status === PipelineExecutionStatus.STOPPED) {
      throw new Error(`Pipeline was forcefully stopped.`);
    } else {
      console.log(`Pipeline execution ${status}.`);
    }
  } while (!complete);
}
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考DescribePipelineExecution中的。

StartPipelineExecution

下列程式碼範例會示範如何使用StartPipelineExecution。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

開始 SageMaker 管線執行。

```
/**
 * Start the execution of the Amazon SageMaker pipeline. Parameters that are
 * passed in are used in the AWS Lambda function.
 * @param {{
 *   name: string,
 *   sagemakerClient: import('@aws-sdk/client-sagemaker').SageMakerClient,
 *   roleArn: string,
 *   queueUrl: string,
 *   s3InputBucketName: string,
 * }} props
 */
export async function startPipelineExecution({
  sagemakerClient,
  name,
  bucketName,
  roleArn,
  queueUrl,
}) {
  /**
   * The Vector Enrichment Job requests CSV data. This configuration points to a CSV
   * file in an Amazon S3 bucket.
   * @type {import("@aws-sdk/client-sagemaker-
geospatial").VectorEnrichmentJobInputConfig}
   */
  const inputConfig = {
    DataSourceConfig: {
      S3Data: {
        S3Uri: `s3://${bucketName}/input/sample_data.csv`,
      },
    },
    DocumentType: VectorEnrichmentJobDocumentType.CSV,
  };
```

```
  /**
   * The Vector Enrichment Job adds additional data to the source CSV. This
 configuration points
   * to an Amazon S3 prefix where the output will be stored.
   * @type {import("@aws-sdk/client-sagemaker-
geospatial").ExportVectorEnrichmentJobOutputConfig}
   */
  const outputConfig = {
    S3Data: {
      S3Uri: `s3://${bucketName}/output/`,
    },
  };

  /**
   * This job will be a Reverse Geocoding Vector Enrichment Job. Reverse Geocoding
 requires
   * latitude and longitude values.
   * @type {import("@aws-sdk/client-sagemaker-
geospatial").VectorEnrichmentJobConfig}
   */
  const jobConfig = {
    ReverseGeocodingConfig: {
      XAttributeName: "Longitude",
      YAttributeName: "Latitude",
    },
  };

  const { PipelineExecutionArn } = await sagemakerClient.send(
    new StartPipelineExecutionCommand({
      PipelineName: name,
      PipelineExecutionDisplayName: `${name}-example-execution`,
      PipelineParameters: [
        { Name: "parameter_execution_role", Value: roleArn },
        { Name: "parameter_queue_url", Value: queueUrl },
        {
          Name: "parameter_vej_input_config",
          Value: JSON.stringify(inputConfig),
        },
        {
          Name: "parameter_vej_export_config",
          Value: JSON.stringify(outputConfig),
        },
        {
          Name: "parameter_step_1_vej_config",
```

```
          Value: JSON.stringify(jobConfig),
        },
      ],
    }),
  );

  return {
    arn: PipelineExecutionArn,
  };
}
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考StartPipelineExecution中的。

案例

開始使用地理空間工作和管道

以下程式碼範例顯示做法：

- 設定管線的資源。
- 設置執行空間工作的管線。
- 啟動管道執行。
- 監視執行狀態。
- 檢視管線的輸出。
- 清理資源。

如需詳細資訊，請參閱AWS SDKs在 Community.AWS 上使用建立和執行 SageMaker 管道。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

下列檔案摘錄包含使用用 SageMaker 戶端管理管線的函數。

```
import { readFileSync } from "fs";

import {
  CreateRoleCommand,
  DeleteRoleCommand,
  CreatePolicyCommand,
  DeletePolicyCommand,
  AttachRolePolicyCommand,
  DetachRolePolicyCommand,
  GetRoleCommand,
  ListPoliciesCommand,
} from "@aws-sdk/client-iam";

import {
  PublishLayerVersionCommand,
  DeleteLayerVersionCommand,
  CreateFunctionCommand,
  Runtime,
  DeleteFunctionCommand,
  CreateEventSourceMappingCommand,
  DeleteEventSourceMappingCommand,
  GetFunctionCommand,
} from "@aws-sdk/client-lambda";

import {
  PutObjectCommand,
  CreateBucketCommand,
  DeleteBucketCommand,
  DeleteObjectCommand,
  GetObjectCommand,
  ListObjectsV2Command,
} from "@aws-sdk/client-s3";

import {
  CreatePipelineCommand,
  DeletePipelineCommand,
  DescribePipelineCommand,
  DescribePipelineExecutionCommand,
  PipelineExecutionStatus,
  StartPipelineExecutionCommand,
} from "@aws-sdk/client-sagemaker";
```

```
import { VectorEnrichmentJobDocumentType } from "@aws-sdk/client-sagemaker-
geospatial";

import {
  CreateQueueCommand,
  DeleteQueueCommand,
  GetQueueAttributesCommand,
  GetQueueUrlCommand,
} from "@aws-sdk/client-sqs";

import { dirnameFromMetaUrl } from "@aws-doc-sdk-examples/lib/utils/util-fs.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

/**
 * Create the AWS IAM role that will be assumed by AWS Lambda.
 * @param {{ name: string, iamClient: import('@aws-sdk/client-iam').IAMClient }}
 props
 */
export async function createLambdaExecutionRole({ name, iamClient }) {
  const createRole = () =>
    iamClient.send(
      new CreateRoleCommand({
        RoleName: name,
        AssumeRolePolicyDocument: JSON.stringify({
          Version: "2012-10-17",
          Statement: [
            {
              Effect: "Allow",
              Action: ["sts:AssumeRole"],
              Principal: { Service: ["lambda.amazonaws.com"] },
            },
          ],
        }),
      }),
    );

  let role = null;

  try {
    const { Role } = await createRole();
    role = Role;
  } catch (caught) {
    if (
      caught instanceof Error &&
```

```
      caught.name === "EntityAlreadyExistsException"
    ) {
      const { Role } = await iamClient.send(
        new GetRoleCommand({ RoleName: name }),
      );
      role = Role;
    } else {
      throw caught;
    }
  }

  return {
    arn: role.Arn,
    cleanUp: async () => {
      await iamClient.send(new DeleteRoleCommand({ RoleName: name }));
    },
  };
}

/**
 * Create an AWS IAM policy that will be attached to the AWS IAM role assumed by the
 AWS Lambda function.
 * The policy grants permission to work with Amazon SQS, Amazon CloudWatch, and
 Amazon SageMaker.
 * @param {{name: string, iamClient: import('@aws-sdk/client-iam').IAMClient,
 pipelineExecutionRoleArn: string}} props
 */
export async function createLambdaExecutionPolicy({
  name,
  iamClient,
  pipelineExecutionRoleArn,
}) {
  const policyConfig = {
    Version: "2012-10-17",
    Statement: [
      {
        Effect: "Allow",
        Action: [
          "sqs:ReceiveMessage",
          "sqs:DeleteMessage",
          "sqs:GetQueueAttributes",
          "logs:CreateLogGroup",
          "logs:CreateLogStream",
          "logs:PutLogEvents",
```

```
          "sagemaker-geospatial:StartVectorEnrichmentJob",
          "sagemaker-geospatial:GetVectorEnrichmentJob",
          "sagemaker:SendPipelineExecutionStepFailure",
          "sagemaker:SendPipelineExecutionStepSuccess",
          "sagemaker-geospatial:ExportVectorEnrichmentJob",
        ],
        Resource: "*",
      },
      {
        Effect: "Allow",
        // The AWS Lambda function needs permission to pass the pipeline execution
 role to
        // the StartVectorEnrichmentCommand. This restriction prevents an AWS Lambda
 function
        // from elevating privileges. For more information, see:
        // https://docs.aws.amazon.com/IAM/latest/UserGuide/
id_roles_use_passrole.html
        Action: ["iam:PassRole"],
        Resource: `${pipelineExecutionRoleArn}`,
        Condition: {
          StringEquals: {
            "iam:PassedToService": [
              "sagemaker.amazonaws.com",
              "sagemaker-geospatial.amazonaws.com",
            ],
          },
        },
      },
    ],
  };

  const createPolicy = () =>
    iamClient.send(
      new CreatePolicyCommand({
        PolicyDocument: JSON.stringify(policyConfig),
        PolicyName: name,
      }),
    );

  let policy = null;

  try {
    const { Policy } = await createPolicy();
    policy = Policy;
```

```
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "EntityAlreadyExistsException"
    ) {
      const { Policies } = await iamClient.send(new ListPoliciesCommand({}));
      if (Policies) {
        policy = Policies.find((p) => p.PolicyName === name);
      } else {
        throw new Error("No policies found.");
      }
    } else {
      throw caught;
    }
  }

  return {
    arn: policy?.Arn,
    policyConfig,
    cleanUp: async () => {
      await iamClient.send(new DeletePolicyCommand({ PolicyArn: policy?.Arn }));
    },
  };
}

/**
 * Attach an AWS IAM policy to an AWS IAM role.
 * @param {{roleName: string, policyArn: string, iamClient: import('@aws-sdk/client-
iam').IAMClient}} props
 */
export async function attachPolicy({ roleName, policyArn, iamClient }) {
  const attachPolicyCommand = new AttachRolePolicyCommand({
    RoleName: roleName,
    PolicyArn: policyArn,
  });

  await iamClient.send(attachPolicyCommand);
  return {
    cleanUp: async () => {
      await iamClient.send(
        new DetachRolePolicyCommand({
          RoleName: roleName,
          PolicyArn: policyArn,
        }),
```

```
      );
    },
  };
}

/**
 * Create an AWS Lambda layer that contains the Amazon SageMaker and Amazon
 SageMaker Geospatial clients
 * in the runtime. The default runtime supports v3.188.0 of the JavaScript SDK. The
 Amazon SageMaker
 * Geospatial client wasn't introduced until v3.221.0.
 * @param {{ name: string, lambdaClient: import('@aws-sdk/client-
lambda').LambdaClient }} props
 */
export async function createLambdaLayer({ name, lambdaClient }) {
  const layerPath = `${dirnameFromMetaUrl(import.meta.url)}lambda/nodejs.zip`;
  const { LayerVersionArn, Version } = await lambdaClient.send(
    new PublishLayerVersionCommand({
      LayerName: name,
      Content: {
        ZipFile: Uint8Array.from(readFileSync(layerPath)),
      },
    }),
  );

  return {
    versionArn: LayerVersionArn,
    version: Version,
    cleanUp: async () => {
      await lambdaClient.send(
        new DeleteLayerVersionCommand({
          LayerName: name,
          VersionNumber: Version,
        }),
      );
    },
  };
}

/**
 * Deploy the AWS Lambda function that will be used to respond to Amazon SageMaker
 pipeline
 * execution steps.
```

```
 * @param {{roleArn: string, name: string, lambdaClient: import('@aws-sdk/client-
lambda').LambdaClient, layerVersionArn: string}} props
 */
export async function createLambdaFunction({
  name,
  roleArn,
  lambdaClient,
  layerVersionArn,
}) {
  const lambdaPath = `${dirnameFromMetaUrl(
    import.meta.url,
  )}lambda/dist/index.mjs.zip`;

  // If a function of the same name already exists, return that
  // function's ARN instead. By default this is
  // "sagemaker-wkflw-lambda-function", so collisions are
  // unlikely.
  const createFunction = async () => {
    try {
      return await lambdaClient.send(
        new CreateFunctionCommand({
          Code: {
            ZipFile: Uint8Array.from(readFileSync(lambdaPath)),
          },
          Runtime: Runtime.nodejs18x,
          Handler: "index.handler",
          Layers: [layerVersionArn],
          FunctionName: name,
          Role: roleArn,
        }),
      );
    } catch (caught) {
      if (
        caught instanceof Error &&
        caught.name === "ResourceConflictException"
      ) {
        const { Configuration } = await lambdaClient.send(
          new GetFunctionCommand({ FunctionName: name }),
        );
        return Configuration;
      } else {
        throw caught;
      }
    }
```

```
  };

  // Function creation fails if the Role is not ready. This retries
  // function creation until it succeeds or it times out.
  const { FunctionArn } = await retry(
    { intervalInMs: 1000, maxRetries: 60 },
    createFunction,
  );

  return {
    arn: FunctionArn,
    cleanUp: async () => {
      await lambdaClient.send(
        new DeleteFunctionCommand({ FunctionName: name }),
      );
    },
  };
}

/**
 * This uploads some sample coordinate data to an Amazon S3 bucket.
 * The Amazon SageMaker Geospatial vector enrichment job will take the simple Lat/
Long
 * coordinates in this file and augment them with more detailed location data.
 * @param {{bucketName: string, s3Client: import('@aws-sdk/client-s3').S3Client}}
 props
 */
export async function uploadCSVDataToS3({ bucketName, s3Client }) {
  const s3Path = `${dirnameFromMetaUrl(
    import.meta.url,
  )}../../../../../workflows/sagemaker_pipelines/resources/latlongtest.csv`;

  await s3Client.send(
    new PutObjectCommand({
      Bucket: bucketName,
      Key: "input/sample_data.csv",
      Body: readFileSync(s3Path),
    }),
  );
}

/**
 * Create the AWS IAM role that will be assumed by the Amazon SageMaker pipeline.
```

```
 * @param {{name: string, iamClient: import('@aws-sdk/client-iam').IAMClient, wait:
 (ms: number) => Promise<void>}} props
 */
export async function createSagemakerRole({ name, iamClient, wait }) {
  let role = null;

  const createRole = () =>
    iamClient.send(
      new CreateRoleCommand({
        RoleName: name,
        AssumeRolePolicyDocument: JSON.stringify({
          Version: "2012-10-17",
          Statement: [
            {
              Effect: "Allow",
              Action: ["sts:AssumeRole"],
              Principal: {
                Service: [
                  "sagemaker.amazonaws.com",
                  "sagemaker-geospatial.amazonaws.com",
                ],
              },
            },
          ],
        }),
      }),
    );

  try {
    const { Role } = await createRole();
    role = Role;
    // Wait for the role to be ready.
    await wait(10);
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "EntityAlreadyExistsException"
    ) {
      const { Role } = await iamClient.send(
        new GetRoleCommand({ RoleName: name }),
      );
      role = Role;
    } else {
      throw caught;
```

```
    }
  }

  return {
    arn: role.Arn,
    cleanUp: async () => {
      await iamClient.send(new DeleteRoleCommand({ RoleName: name }));
    },
  };
}

/**
 * Create the Amazon SageMaker execution policy. This policy grants permission to
 * invoke the AWS Lambda function, read/write to the Amazon S3 bucket, and send
 messages to
 * the Amazon SQS queue.
 * @param {{ name: string, sqsQueueArn: string, lambdaArn: string, iamClient:
 import('@aws-sdk/client-iam').IAMClient, s3BucketName: string}} props
 */
export async function createSagemakerExecutionPolicy({
  sqsQueueArn,
  lambdaArn,
  iamClient,
  name,
  s3BucketName,
}) {
  const policyConfig = {
    Version: "2012-10-17",
    Statement: [
      {
        Effect: "Allow",
        Action: ["lambda:InvokeFunction"],
        Resource: lambdaArn,
      },
      {
        Effect: "Allow",
        Action: ["s3:*"],
        Resource: [
          `arn:aws:s3:::${s3BucketName}`,
          `arn:aws:s3:::${s3BucketName}/*`,
        ],
      },
      {
        Effect: "Allow",
```

```
        Action: ["sqs:SendMessage"],
        Resource: sqsQueueArn,
      },
    ],
  };

  const createPolicy = () =>
    iamClient.send(
      new CreatePolicyCommand({
        PolicyDocument: JSON.stringify(policyConfig),
        PolicyName: name,
      }),
    );

  let policy = null;

  try {
    const { Policy } = await createPolicy();
    policy = Policy;
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "EntityAlreadyExistsException"
    ) {
      const { Policies } = await iamClient.send(new ListPoliciesCommand({}));
      if (Policies) {
        policy = Policies.find((p) => p.PolicyName === name);
      } else {
        throw new Error("No policies found.");
      }
    } else {
      throw caught;
    }
  }

  return {
    arn: policy?.Arn,
    policyConfig,
    cleanUp: async () => {
      await iamClient.send(new DeletePolicyCommand({ PolicyArn: policy?.Arn }));
    },
  };
}
```

```
/**
 * Create the Amazon SageMaker pipeline using a JSON pipeline definition. The
 definition
 * can also be provided as an Amazon S3 object using PipelineDefinitionS3Location.
 * @param {{roleArn: string, name: string, sagemakerClient: import('@aws-sdk/client-
sagemaker').SageMakerClient}} props
 */
export async function createSagemakerPipeline({
  // Assumes an AWS IAM role has been created for this pipeline.
  roleArn,
  name,
  // Assumes an AWS Lambda function has been created for this pipeline.
  functionArn,
  sagemakerClient,
}) {
  const pipelineDefinition = readFileSync(
    // dirnameFromMetaUrl is a local utility function. You can find its
 implementation
    // on GitHub.
    `${dirnameFromMetaUrl(
      import.meta.url,
    )}../../../../../workflows/sagemaker_pipelines/resources/
GeoSpatialPipeline.json`,
  )
    .toString()
    .replace(/\*FUNCTION_ARN\*/g, functionArn);

  let arn = null;

  const createPipeline = () =>
    sagemakerClient.send(
      new CreatePipelineCommand({
        PipelineName: name,
        PipelineDefinition: pipelineDefinition,
        RoleArn: roleArn,
      }),
    );

  try {
    const { PipelineArn } = await createPipeline();
    arn = PipelineArn;
  } catch (caught) {
    if (
      caught instanceof Error &&
```

```
      caught.name === "ValidationException" &&
      caught.message.includes(
        "Pipeline names must be unique within an AWS account and region",
      )
    ) {
      const { PipelineArn } = await sagemakerClient.send(
        new DescribePipelineCommand({ PipelineName: name }),
      );
      arn = PipelineArn;
    } else {
      throw caught;
    }
  }

  return {
    arn,
    cleanUp: async () => {
      await sagemakerClient.send(
        new DeletePipelineCommand({ PipelineName: name }),
      );
    },
  };
}

/**
 * Create an Amazon SQS queue. The Amazon SageMaker pipeline will send messages
 * to this queue that are then processed by the AWS Lambda function.
 * @param {{name: string, sqsClient: import('@aws-sdk/client-sqs').SQSClient}} props
 */
export async function createSQSQueue({ name, sqsClient }) {
  const createSqsQueue = () =>
    sqsClient.send(
      new CreateQueueCommand({
        QueueName: name,
        Attributes: {
          DelaySeconds: "5",
          ReceiveMessageWaitTimeSeconds: "5",
          VisibilityTimeout: "300",
        },
      }),
    );

  let queueUrl = null;
  try {
```

```
    const { QueueUrl } = await createSqsQueue();
    queueUrl = QueueUrl;
  } catch (caught) {
    if (caught instanceof Error && caught.name === "QueueNameExists") {
      const { QueueUrl } = await sqsClient.send(
        new GetQueueUrlCommand({ QueueName: name }),
      );
      queueUrl = QueueUrl;
    } else {
      throw caught;
    }
  }

  const { Attributes } = await retry(
    { intervalInMs: 1000, maxRetries: 60 },
    () =>
      sqsClient.send(
        new GetQueueAttributesCommand({
          QueueUrl: queueUrl,
          AttributeNames: ["QueueArn"],
        }),
      ),
  );

  return {
    queueUrl,
    queueArn: Attributes.QueueArn,
    cleanUp: async () => {
      await sqsClient.send(new DeleteQueueCommand({ QueueUrl: queueUrl }));
    },
  };
}

/**
 * Configure the AWS Lambda function to long poll for messages from the Amazon SQS
 * queue.
 * @param {{
 *   paginateListEventSourceMappings: () => Generator<import('@aws-sdk/client-
lambda').ListEventSourceMappingsCommandOutput>,
 *   lambdaName: string,
 *   queueArn: string,
 *   lambdaClient: import('@aws-sdk/client-lambda').LambdaClient}} props
 */
export async function configureLambdaSQSEventSource({
```

```
  lambdaName,
  queueArn,
  lambdaClient,
  paginateListEventSourceMappings,
}) {
  let uuid = null;
  const createEvenSourceMapping = () =>
    lambdaClient.send(
      new CreateEventSourceMappingCommand({
        EventSourceArn: queueArn,
        FunctionName: lambdaName,
      }),
    );

  try {
    const { UUID } = await createEvenSourceMapping();
    uuid = UUID;
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "ResourceConflictException"
    ) {
      const paginator = paginateListEventSourceMappings(
        { client: lambdaClient },
        {},
      );
      /**
       * @type {import('@aws-sdk/client-lambda').EventSourceMappingConfiguration[]}
       */
      const eventSourceMappings = [];
      for await (const page of paginator) {
        eventSourceMappings.concat(page.EventSourceMappings || []);
      }

      const { Configuration } = await lambdaClient.send(
        new GetFunctionCommand({ FunctionName: lambdaName }),
      );

      uuid = eventSourceMappings.find(
        (mapping) =>
          mapping.EventSourceArn === queueArn &&
          mapping.FunctionArn === Configuration.FunctionArn,
      ).UUID;
    } else {
```

```
      throw caught;
    }
  }

  return {
    cleanUp: async () => {
      await lambdaClient.send(
        new DeleteEventSourceMappingCommand({
          UUID: uuid,
        }),
      );
    },
  };
}

/**
 * Create an Amazon S3 bucket that will store the simple coordinate file as input
 * and the output of the Amazon SageMaker Geospatial vector enrichment job.
 * @param {{
 *   s3Client: import('@aws-sdk/client-s3').S3Client,
 *   name: string,
 *   paginateListObjectsV2: () => Generator<import('@aws-sdk/client-
s3').ListObjectsCommandOutput>
 * }} props
 */
export async function createS3Bucket({
  name,
  s3Client,
  paginateListObjectsV2,
}) {
  await s3Client.send(new CreateBucketCommand({ Bucket: name }));

  return {
    cleanUp: async () => {
      const paginator = paginateListObjectsV2(
        { client: s3Client },
        { Bucket: name },
      );
      for await (const page of paginator) {
        const objects = page.Contents;
        if (objects) {
          for (const object of objects) {
            await s3Client.send(
              new DeleteObjectCommand({ Bucket: name, Key: object.Key }),
```

```
            );
          }
        }
      }
      await s3Client.send(new DeleteBucketCommand({ Bucket: name }));
    },
  };
}

/**
 * Start the execution of the Amazon SageMaker pipeline. Parameters that are
 * passed in are used in the AWS Lambda function.
 * @param {{
 *   name: string,
 *   sagemakerClient: import('@aws-sdk/client-sagemaker').SageMakerClient,
 *   roleArn: string,
 *   queueUrl: string,
 *   s3InputBucketName: string,
 * }} props
 */
export async function startPipelineExecution({
  sagemakerClient,
  name,
  bucketName,
  roleArn,
  queueUrl,
}) {
  /**
   * The Vector Enrichment Job requests CSV data. This configuration points to a CSV
   * file in an Amazon S3 bucket.
   * @type {import("@aws-sdk/client-sagemaker-
geospatial").VectorEnrichmentJobInputConfig}
   */
  const inputConfig = {
    DataSourceConfig: {
      S3Data: {
        S3Uri: `s3://${bucketName}/input/sample_data.csv`,
      },
    },
    DocumentType: VectorEnrichmentJobDocumentType.CSV,
  };

  /**
```

```
   * The Vector Enrichment Job adds additional data to the source CSV. This
 configuration points
   * to an Amazon S3 prefix where the output will be stored.
   * @type {import("@aws-sdk/client-sagemaker-
geospatial").ExportVectorEnrichmentJobOutputConfig}
   */
  const outputConfig = {
    S3Data: {
      S3Uri: `s3://${bucketName}/output/`,
    },
  };

  /**
   * This job will be a Reverse Geocoding Vector Enrichment Job. Reverse Geocoding
 requires
   * latitude and longitude values.
   * @type {import("@aws-sdk/client-sagemaker-
geospatial").VectorEnrichmentJobConfig}
   */
  const jobConfig = {
    ReverseGeocodingConfig: {
      XAttributeName: "Longitude",
      YAttributeName: "Latitude",
    },
  };

  const { PipelineExecutionArn } = await sagemakerClient.send(
    new StartPipelineExecutionCommand({
      PipelineName: name,
      PipelineExecutionDisplayName: `${name}-example-execution`,
      PipelineParameters: [
        { Name: "parameter_execution_role", Value: roleArn },
        { Name: "parameter_queue_url", Value: queueUrl },
        {
          Name: "parameter_vej_input_config",
          Value: JSON.stringify(inputConfig),
        },
        {
          Name: "parameter_vej_export_config",
          Value: JSON.stringify(outputConfig),
        },
        {
          Name: "parameter_step_1_vej_config",
          Value: JSON.stringify(jobConfig),
```

```
        },
      ],
    }),
  );

  return {
    arn: PipelineExecutionArn,
  };
}

/**
 * Poll the executing pipeline until the status is 'SUCCEEDED', 'STOPPED', or
 'FAILED'.
 * @param {{ arn: string, sagemakerClient: import('@aws-sdk/client-
sagemaker').SageMakerClient, wait: (ms: number) => Promise<void>}} props
 */
export async function waitForPipelineComplete({ arn, sagemakerClient, wait }) {
  const command = new DescribePipelineExecutionCommand({
    PipelineExecutionArn: arn,
  });

  let complete = false;
  let intervalInSeconds = 15;
  const COMPLETION_STATUSES = [
    PipelineExecutionStatus.FAILED,
    PipelineExecutionStatus.STOPPED,
    PipelineExecutionStatus.SUCCEEDED,
  ];

  do {
    const { PipelineExecutionStatus: status, FailureReason } =
      await sagemakerClient.send(command);

    complete = COMPLETION_STATUSES.includes(status);

    if (!complete) {
      console.log(
        `Pipeline is ${status}. Waiting ${intervalInSeconds} seconds before checking
 again.`,
      );
      await wait(intervalInSeconds);
    } else if (status === PipelineExecutionStatus.FAILED) {
      throw new Error(`Pipeline failed because: ${FailureReason}`);
    } else if (status === PipelineExecutionStatus.STOPPED) {
```

```
      throw new Error(`Pipeline was forcefully stopped.`);
    } else {
      console.log(`Pipeline execution ${status}.`);
    }
  } while (!complete);
}

/**
 * Return the string value of an Amazon S3 object.
 * @param {{ bucket: string, key: string, s3Client: import('@aws-sdk/client-
s3').S3Client}} param0
 */
export async function getObject({ bucket, s3Client }) {
  const prefix = "output/";
  const { Contents } = await s3Client.send(
    new ListObjectsV2Command({ MaxKeys: 1, Bucket: bucket, Prefix: prefix }),
  );

  if (!Contents.length) {
    throw new Error("No objects found in bucket.");
  }

  // Find the CSV file.
  const outputObject = Contents.find((obj) => obj.Key.endsWith(".csv"));

  if (!outputObject) {
    throw new Error(`No CSV file found in bucket with the prefix "${prefix}".`);
  }

  const { Body } = await s3Client.send(
    new GetObjectCommand({
      Bucket: bucket,
      Key: outputObject.Key,
    }),
  );

  return Body.transformToString();
}
```

此函數摘錄自檔案，該檔案使用先前的程式庫函數來設定 SageMaker 管線、執行並刪除所有建立的資源。

```
import { retry, wait } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";
import {
  attachPolicy,
  configureLambdaSQSEventSource,
  createLambdaExecutionPolicy,
  createLambdaExecutionRole,
  createLambdaFunction,
  createLambdaLayer,
  createS3Bucket,
  createSQSQueue,
  createSagemakerExecutionPolicy,
  createSagemakerPipeline,
  createSagemakerRole,
  getObject,
  startPipelineExecution,
  uploadCSVDataToS3,
  waitForPipelineComplete,
} from "./lib.js";
import { MESSAGES } from "./messages.js";

export class SageMakerPipelinesWkflw {
  names = {
    LAMBDA_EXECUTION_ROLE: "sagemaker-wkflw-lambda-execution-role",
    LAMBDA_EXECUTION_ROLE_POLICY:
      "sagemaker-wkflw-lambda-execution-role-policy",
    LAMBDA_FUNCTION: "sagemaker-wkflw-lambda-function",
    LAMBDA_LAYER: "sagemaker-wkflw-lambda-layer",
    SAGE_MAKER_EXECUTION_ROLE: "sagemaker-wkflw-pipeline-execution-role",
    SAGE_MAKER_EXECUTION_ROLE_POLICY:
      "sagemaker-wkflw-pipeline-execution-role-policy",
    SAGE_MAKER_PIPELINE: "sagemaker-wkflw-pipeline",
    SQS_QUEUE: "sagemaker-wkflw-sqs-queue",
    S3_BUCKET: `sagemaker-wkflw-s3-bucket-${Date.now()}`,
  };

  cleanUpFunctions = [];

  /**
   * @param {import("@aws-doc-sdk-examples/lib/prompter.js").Prompter} prompter
   * @param {import("@aws-doc-sdk-examples/lib/logger.js").Logger} logger
   * @param {{ IAM: import("@aws-sdk/client-iam").IAMClient, Lambda: import("@aws-
sdk/client-lambda").LambdaClient, SageMaker: import("@aws-sdk/client-
```

```
sagemaker").SageMakerClient, S3: import("@aws-sdk/client-s3").S3Client, SQS:
 import("@aws-sdk/client-sqs").SQSClient }} clients
   */
  constructor(prompter, logger, clients) {
    this.prompter = prompter;
    this.logger = logger;
    this.clients = clients;
  }

  async run() {
    try {
      await this.startWorkflow();
    } catch (err) {
      console.error(err);
      throw err;
    } finally {
      this.logger.logSeparator();
      const doCleanUp = await this.prompter.confirm({
        message: "Clean up resources?",
      });
      if (doCleanUp) {
        await this.cleanUp();
      }
    }
  }

  async cleanUp() {
    // Run all of the clean up functions. If any fail, we log the error and
continue.
    // This ensures all clean up functions are run.
    for (let i = this.cleanUpFunctions.length - 1; i >= 0; i--) {
      await retry(
        { intervalInMs: 1000, maxRetries: 60, swallowError: true },
        this.cleanUpFunctions[i],
      );
    }
  }

  async startWorkflow() {
    this.logger.logSeparator(MESSAGES.greetingHeader);
    await this.logger.log(MESSAGES.greeting);

    this.logger.logSeparator();
    await this.logger.log(
```

```
      MESSAGES.creatingRole.replace(
        "${ROLE_NAME}",
        this.names.LAMBDA_EXECUTION_ROLE,
      ),
    );

    // Create an IAM role that will be assumed by the AWS Lambda function. This
function
    // is triggered by Amazon SQS messages and calls SageMaker and SageMaker
GeoSpatial actions.
    const { arn: lambdaExecutionRoleArn, cleanUp: lambdaExecutionRoleCleanUp } =
      await createLambdaExecutionRole({
        name: this.names.LAMBDA_EXECUTION_ROLE,
        iamClient: this.clients.IAM,
      });
    // Add a clean up step to a stack for every resource created.
    this.cleanUpFunctions.push(lambdaExecutionRoleCleanUp);

    await this.logger.log(
      MESSAGES.roleCreated.replace(
        "${ROLE_NAME}",
        this.names.LAMBDA_EXECUTION_ROLE,
      ),
    );

    this.logger.logSeparator();

    await this.logger.log(
      MESSAGES.creatingRole.replace(
        "${ROLE_NAME}",
        this.names.SAGE_MAKER_EXECUTION_ROLE,
      ),
    );

    // Create an IAM role that will be assumed by the SageMaker pipeline. The
pipeline
    // sends messages to an Amazon SQS queue and puts/retrieves Amazon S3 objects.
    const {
      arn: pipelineExecutionRoleArn,
      cleanUp: pipelineExecutionRoleCleanUp,
    } = await createSagemakerRole({
      iamClient: this.clients.IAM,
      name: this.names.SAGE_MAKER_EXECUTION_ROLE,
      wait,
```

```
    });
    this.cleanUpFunctions.push(pipelineExecutionRoleCleanUp);

    await this.logger.log(
      MESSAGES.roleCreated.replace(
        "${ROLE_NAME}",
        this.names.SAGE_MAKER_EXECUTION_ROLE,
      ),
    );

    this.logger.logSeparator();

    // Create an IAM policy that allows the AWS Lambda function to invoke SageMaker
APIs.
    const {
      arn: lambdaExecutionPolicyArn,
      policy: lambdaPolicy,
      cleanUp: lambdaExecutionPolicyCleanUp,
    } = await createLambdaExecutionPolicy({
      name: this.names.LAMBDA_EXECUTION_ROLE_POLICY,
      s3BucketName: this.names.S3_BUCKET,
      iamClient: this.clients.IAM,
      pipelineExecutionRoleArn,
    });
    this.cleanUpFunctions.push(lambdaExecutionPolicyCleanUp);

    console.log(JSON.stringify(lambdaPolicy, null, 2), "\n");

    await this.logger.log(
      MESSAGES.attachPolicy
        .replace("${POLICY_NAME}", this.names.LAMBDA_EXECUTION_ROLE_POLICY)
        .replace("${ROLE_NAME}", this.names.LAMBDA_EXECUTION_ROLE),
    );

    await this.prompter.checkContinue();

    // Attach the Lambda execution policy to the execution role.
    const { cleanUp: lambdaExecutionRolePolicyCleanUp } = await attachPolicy({
      roleName: this.names.LAMBDA_EXECUTION_ROLE,
      policyArn: lambdaExecutionPolicyArn,
      iamClient: this.clients.IAM,
    });
    this.cleanUpFunctions.push(lambdaExecutionRolePolicyCleanUp);
```

```
    await this.logger.log(MESSAGES.policyAttached);

    this.logger.logSeparator();

    // Create Lambda layer for SageMaker packages.
    const { versionArn: layerVersionArn, cleanUp: lambdaLayerCleanUp } =
      await createLambdaLayer({
        name: this.names.LAMBDA_LAYER,
        lambdaClient: this.clients.Lambda,
      });
    this.cleanUpFunctions.push(lambdaLayerCleanUp);

    await this.logger.log(
      MESSAGES.creatingFunction.replace(
        "${FUNCTION_NAME}",
        this.names.LAMBDA_FUNCTION,
      ),
    );

    // Create the Lambda function with the execution role.
    const { arn: lambdaArn, cleanUp: lambdaCleanUp } =
      await createLambdaFunction({
        roleArn: lambdaExecutionRoleArn,
        lambdaClient: this.clients.Lambda,
        name: this.names.LAMBDA_FUNCTION,
        layerVersionArn,
      });
    this.cleanUpFunctions.push(lambdaCleanUp);

    await this.logger.log(
      MESSAGES.functionCreated.replace(
        "${FUNCTION_NAME}",
        this.names.LAMBDA_FUNCTION,
      ),
    );

    this.logger.logSeparator();

    await this.logger.log(
      MESSAGES.creatingSQSQueue.replace("${QUEUE_NAME}", this.names.SQS_QUEUE),
    );

    // Create an SQS queue for the SageMaker pipeline.
    const {
```

```
    queueUrl,
    queueArn,
    cleanUp: queueCleanUp,
  } = await createSQSQueue({
    name: this.names.SQS_QUEUE,
    sqsClient: this.clients.SQS,
  });
  this.cleanUpFunctions.push(queueCleanUp);

  await this.logger.log(
    MESSAGES.sqsQueueCreated.replace("${QUEUE_NAME}", this.names.SQS_QUEUE),
  );

  this.logger.logSeparator();

  await this.logger.log(
    MESSAGES.configuringLambdaSQSEventSource
      .replace("${LAMBDA_NAME}", this.names.LAMBDA_FUNCTION)
      .replace("${QUEUE_NAME}", this.names.SQS_QUEUE),
  );

  // Configure the SQS queue as an event source for the Lambda.
  const { cleanUp: lambdaSQSEventSourceCleanUp } =
    await configureLambdaSQSEventSource({
      lambdaArn,
      lambdaName: this.names.LAMBDA_FUNCTION,
      queueArn,
      sqsClient: this.clients.SQS,
      lambdaClient: this.clients.Lambda,
    });
  this.cleanUpFunctions.push(lambdaSQSEventSourceCleanUp);

  await this.logger.log(
    MESSAGES.lambdaSQSEventSourceConfigured
      .replace("${LAMBDA_NAME}", this.names.LAMBDA_FUNCTION)
      .replace("${QUEUE_NAME}", this.names.SQS_QUEUE),
  );

  this.logger.logSeparator();

  // Create an IAM policy that allows the SageMaker pipeline to invoke AWS Lambda
  // and send messages to the Amazon SQS queue.
  const {
    arn: pipelineExecutionPolicyArn,
```

```
      policy: sagemakerPolicy,
      cleanUp: pipelineExecutionPolicyCleanUp,
    } = await createSagemakerExecutionPolicy({
      sqsQueueArn: queueArn,
      lambdaArn,
      iamClient: this.clients.IAM,
      name: this.names.SAGE_MAKER_EXECUTION_ROLE_POLICY,
      s3BucketName: this.names.S3_BUCKET,
    });
    this.cleanUpFunctions.push(pipelineExecutionPolicyCleanUp);

    console.log(JSON.stringify(sagemakerPolicy, null, 2));

    await this.logger.log(
      MESSAGES.attachPolicy
        .replace("${POLICY_NAME}", this.names.SAGE_MAKER_EXECUTION_ROLE_POLICY)
        .replace("${ROLE_NAME}", this.names.SAGE_MAKER_EXECUTION_ROLE),
    );

    await this.prompter.checkContinue();

    // Attach the SageMaker execution policy to the execution role.
    const { cleanUp: pipelineExecutionRolePolicyCleanUp } = await attachPolicy({
      roleName: this.names.SAGE_MAKER_EXECUTION_ROLE,
      policyArn: pipelineExecutionPolicyArn,
      iamClient: this.clients.IAM,
    });
    this.cleanUpFunctions.push(pipelineExecutionRolePolicyCleanUp);
    // Wait for the role to be ready. If the role is used immediately,
    // the pipeline will fail.
    await wait(5);

    await this.logger.log(MESSAGES.policyAttached);

    this.logger.logSeparator();

    await this.logger.log(
      MESSAGES.creatingPipeline.replace(
        "${PIPELINE_NAME}",
        this.names.SAGE_MAKER_PIPELINE,
      ),
    );

    // Create the SageMaker pipeline.
```

```
    const { cleanUp: pipelineCleanUp } = await createSagemakerPipeline({
      roleArn: pipelineExecutionRoleArn,
      functionArn: lambdaArn,
      sagemakerClient: this.clients.SageMaker,
      name: this.names.SAGE_MAKER_PIPELINE,
    });
    this.cleanUpFunctions.push(pipelineCleanUp);

    await this.logger.log(
      MESSAGES.pipelineCreated.replace(
        "${PIPELINE_NAME}",
        this.names.SAGE_MAKER_PIPELINE,
      ),
    );

    this.logger.logSeparator();

    await this.logger.log(
      MESSAGES.creatingS3Bucket.replace("${BUCKET_NAME}", this.names.S3_BUCKET),
    );

    // Create an S3 bucket for storing inputs and outputs.
    const { cleanUp: s3BucketCleanUp } = await createS3Bucket({
      name: this.names.S3_BUCKET,
      s3Client: this.clients.S3,
    });
    this.cleanUpFunctions.push(s3BucketCleanUp);

    await this.logger.log(
      MESSAGES.s3BucketCreated.replace("${BUCKET_NAME}", this.names.S3_BUCKET),
    );

    this.logger.logSeparator();

    await this.logger.log(
      MESSAGES.uploadingInputData.replace(
        "${BUCKET_NAME}",
        this.names.S3_BUCKET,
      ),
    );

    // Upload CSV Lat/Long data to S3.
    await uploadCSVDataToS3({
      bucketName: this.names.S3_BUCKET,
```

```
      s3Client: this.clients.S3,
    });

    await this.logger.log(MESSAGES.inputDataUploaded);

    this.logger.logSeparator();

    await this.prompter.checkContinue(MESSAGES.executePipeline);

    // Execute the SageMaker pipeline.
    const { arn: pipelineExecutionArn } = await startPipelineExecution({
      name: this.names.SAGE_MAKER_PIPELINE,
      sagemakerClient: this.clients.SageMaker,
      roleArn: pipelineExecutionRoleArn,
      bucketName: this.names.S3_BUCKET,
      queueUrl,
    });

    // Wait for the pipeline execution to finish.
    await waitForPipelineComplete({
      arn: pipelineExecutionArn,
      sagemakerClient: this.clients.SageMaker,
      wait,
    });

    this.logger.logSeparator();

    await this.logger.log(MESSAGES.outputDelay);

    // The getOutput function will throw an error if the output is not
    // found. The retry function will retry a failed function call once
    // ever 10 seconds for 2 minutes.
    const output = await retry({ intervalInMs: 10000, maxRetries: 12 }, () =>
      getObject({
        bucket: this.names.S3_BUCKET,
        s3Client: this.clients.S3,
      }),
    );

    this.logger.logSeparator();
    await this.logger.log(MESSAGES.outputDataRetrieved);
    console.log(output.split("\n").slice(0, 6).join("\n"));
  }
}
```

- 如需詳API細資訊，請參閱「AWS SDK for JavaScript API參考」中的下列主題。
  - CreatePipeline
  - DeletePipeline
  - DescribePipelineExecution
  - StartPipelineExecution
  - UpdatePipeline

Secrets Manager 範例使用 SDK for JavaScript (v3)

下列程式碼範例說明如何透過搭配 Secrets Manager 使用 AWS SDK for JavaScript (v3) 來執行動作和實作常見案例。

Actions 是大型程式的程式碼摘錄，必須在內容中執行。雖然動作會顯示如何呼叫個別服務函數，但您可以在其相關案例中查看內容中的動作。

每個範例都包含完整原始程式碼的連結，您可以在其中找到如何在內容中設定和執行程式碼的指示。

主題

- 動作

動作

`GetSecretValue`

下列程式碼範例會示範如何使用`GetSecretValue`。

SDK對於 JavaScript （ 3 ）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import {
```

```
  GetSecretValueCommand,
  SecretsManagerClient,
} from "@aws-sdk/client-secrets-manager";

export const getSecretValue = async (secretName = "SECRET_NAME") => {
  const client = new SecretsManagerClient();
  const response = await client.send(
    new GetSecretValueCommand({
      SecretId: secretName,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '584eb612-f8b0-48c9-855e-6d246461b604',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   ARN: 'arn:aws:secretsmanager:us-east-1:xxxxxxxxxxxx:secret:binary-
secret-3873048-xxxxxx',
  //   CreatedDate: 2023-08-08T19:29:51.294Z,
  //   Name: 'binary-secret-3873048',
  //   SecretBinary: Uint8Array(11) [
  //      98, 105, 110, 97, 114,
  //     121,  32, 100, 97, 116,
  //      97
  //   ],
  //   VersionId: '712083f4-0d26-415e-8044-16735142cd6a',
  //   VersionStages: [ 'AWSCURRENT' ]
  // }

  if (response.SecretString) {
    return response.SecretString;
  }

  if (response.SecretBinary) {
    return response.SecretBinary;
  }
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考GetSecretValue中的。

Amazon SES 示例使用 SDK for JavaScript (v3)

下列程式碼範例說明如何透過使用 AWS SDK for JavaScript (v3) 搭配 Amazon 來執行動作和實作常見案例SES。

Actions 是大型程式的程式碼摘錄，必須在內容中執行。雖然動作會顯示如何呼叫個別服務函數，但您可以在其相關案例中查看內容中的動作。

案例是程式碼範例，向您展示如何透過呼叫服務中的多個函式或與其他函式結合來完成特定工作 AWS 服務。

每個範例都包含完整原始程式碼的連結，您可以在其中找到如何在內容中設定和執行程式碼的指示。

主題

動作

`CreateReceiptFilter`

下列程式碼範例會示範如何使用`CreateReceiptFilter`。

SDK對於 JavaScript （ 3 ）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import {
  CreateReceiptFilterCommand,
  ReceiptFilterPolicy,
} from "@aws-sdk/client-ses";
import { sesClient } from "./libs/sesClient.js";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";
```

```
const createCreateReceiptFilterCommand = ({ policy, ipOrRange, name }) => {
  return new CreateReceiptFilterCommand({
    Filter: {
      IpFilter: {
        Cidr: ipOrRange, // string, either a single IP address (10.0.0.1) or an IP
 address range in CIDR notation (10.0.0.1/24)).
        Policy: policy, // enum ReceiptFilterPolicy, email traffic from the filtered
 addressesOptions.
      },
      /*
        The name of the IP address filter. Only ASCII letters, numbers, underscores,
 or dashes.
        Must be less than 64 characters and start and end with a letter or number.
       */
      Name: name,
    },
  });
};

const FILTER_NAME = getUniqueName("ReceiptFilter");

const run = async () => {
  const createReceiptFilterCommand = createCreateReceiptFilterCommand({
    policy: ReceiptFilterPolicy.Allow,
    ipOrRange: "10.0.0.1",
    name: FILTER_NAME,
  });

  try {
    return await sesClient.send(createReceiptFilterCommand);
  } catch (caught) {
    if (caught instanceof Error && caught.name === "MessageRejected") {
      /** @type { import('@aws-sdk/client-ses').MessageRejected} */
      const messageRejectedError = caught;
      return messageRejectedError;
    }
    throw caught;
  }
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考CreateReceiptFilter中的。

`CreateReceiptRule`

下列程式碼範例會示範如何使用`CreateReceiptRule`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import { CreateReceiptRuleCommand, TlsPolicy } from "@aws-sdk/client-ses";
import { sesClient } from "./libs/sesClient.js";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";

const RULE_SET_NAME = getUniqueName("RuleSetName");
const RULE_NAME = getUniqueName("RuleName");
const S3_BUCKET_NAME = getUniqueName("S3BucketName");

const createS3ReceiptRuleCommand = ({
  bucketName,
  emailAddresses,
  name,
  ruleSet,
}) => {
  return new CreateReceiptRuleCommand({
    Rule: {
      Actions: [
        {
          S3Action: {
            BucketName: bucketName,
            ObjectKeyPrefix: "email",
          },
        },
      ],
      Recipients: emailAddresses,
      Enabled: true,
      Name: name,
      ScanEnabled: false,
      TlsPolicy: TlsPolicy.Optional,
    },
    RuleSetName: ruleSet, // Required
```

```
  });
};

const run = async () => {
  const s3ReceiptRuleCommand = createS3ReceiptRuleCommand({
    bucketName: S3_BUCKET_NAME,
    emailAddresses: ["email@example.com"],
    name: RULE_NAME,
    ruleSet: RULE_SET_NAME,
  });

  try {
    return await sesClient.send(s3ReceiptRuleCommand);
  } catch (err) {
    console.log("Failed to create S3 receipt rule.", err);
    throw err;
  }
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考CreateReceiptRule中的。

`CreateReceiptRuleSet`

下列程式碼範例會示範如何使用`CreateReceiptRuleSet`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import { CreateReceiptRuleSetCommand } from "@aws-sdk/client-ses";
import { sesClient } from "./libs/sesClient.js";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";

const RULE_SET_NAME = getUniqueName("RuleSetName");

const createCreateReceiptRuleSetCommand = (ruleSetName) => {
  return new CreateReceiptRuleSetCommand({ RuleSetName: ruleSetName });
```

```
};

const run = async () => {
  const createReceiptRuleSetCommand =
    createCreateReceiptRuleSetCommand(RULE_SET_NAME);

  try {
    return await sesClient.send(createReceiptRuleSetCommand);
  } catch (err) {
    console.log("Failed to create receipt rule set", err);
    return err;
  }
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考CreateReceiptRuleSet中的。

`CreateTemplate`

下列程式碼範例會示範如何使用`CreateTemplate`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import { CreateTemplateCommand } from "@aws-sdk/client-ses";
import { sesClient } from "./libs/sesClient.js";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";

const TEMPLATE_NAME = getUniqueName("TestTemplateName");

const createCreateTemplateCommand = () => {
  return new CreateTemplateCommand({
    /**
     * The template feature in Amazon SES is based on the Handlebars template
 system.
```

```
     */
    Template: {
      /**
       * The name of an existing template in Amazon SES.
       */
      TemplateName: TEMPLATE_NAME,
      HtmlPart: `
        <h1>Hello, {{contact.firstName}}!</h1>
        <p>
        Did you know Amazon has a mascot named Peccy?
        </p>
      `,
      SubjectPart: "Amazon Tip",
    },
  });
};

const run = async () => {
  const createTemplateCommand = createCreateTemplateCommand();

  try {
    return await sesClient.send(createTemplateCommand);
  } catch (err) {
    console.log("Failed to create template.", err);
    return err;
  }
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考CreateTemplate中的。

`DeleteIdentity`

下列程式碼範例會示範如何使用`DeleteIdentity`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import { DeleteIdentityCommand } from "@aws-sdk/client-ses";
import { sesClient } from "./libs/sesClient.js";

const IDENTITY_EMAIL = "fake@example.com";

const createDeleteIdentityCommand = (identityName) => {
  return new DeleteIdentityCommand({
    Identity: identityName,
  });
};

const run = async () => {
  const deleteIdentityCommand = createDeleteIdentityCommand(IDENTITY_EMAIL);

  try {
    return await sesClient.send(deleteIdentityCommand);
  } catch (err) {
    console.log("Failed to delete identity.", err);
    return err;
  }
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考DeleteIdentity中的。

`DeleteReceiptFilter`

下列程式碼範例會示範如何使用`DeleteReceiptFilter`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import { DeleteReceiptFilterCommand } from "@aws-sdk/client-ses";
import { sesClient } from "./libs/sesClient.js";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";
```

```
const RECEIPT_FILTER_NAME = getUniqueName("ReceiptFilterName");

const createDeleteReceiptFilterCommand = (filterName) => {
  return new DeleteReceiptFilterCommand({ FilterName: filterName });
};

const run = async () => {
  const deleteReceiptFilterCommand =
    createDeleteReceiptFilterCommand(RECEIPT_FILTER_NAME);

  try {
    return await sesClient.send(deleteReceiptFilterCommand);
  } catch (err) {
    console.log("Error deleting receipt filter.", err);
    return err;
  }
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考DeleteReceiptFilter中的。

`DeleteReceiptRule`

下列程式碼範例會示範如何使用`DeleteReceiptRule`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import { DeleteReceiptRuleCommand } from "@aws-sdk/client-ses";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { sesClient } from "./libs/sesClient.js";

const RULE_NAME = getUniqueName("RuleName");
const RULE_SET_NAME = getUniqueName("RuleSetName");

const createDeleteReceiptRuleCommand = () => {
```

```
  return new DeleteReceiptRuleCommand({
    RuleName: RULE_NAME,
    RuleSetName: RULE_SET_NAME,
  });
};

const run = async () => {
  const deleteReceiptRuleCommand = createDeleteReceiptRuleCommand();
  try {
    return await sesClient.send(deleteReceiptRuleCommand);
  } catch (err) {
    console.log("Failed to delete receipt rule.", err);
    return err;
  }
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考DeleteReceiptRule中的。

`DeleteReceiptRuleSet`

下列程式碼範例會示範如何使用`DeleteReceiptRuleSet`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import { DeleteReceiptRuleSetCommand } from "@aws-sdk/client-ses";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { sesClient } from "./libs/sesClient.js";

const RULE_SET_NAME = getUniqueName("RuleSetName");

const createDeleteReceiptRuleSetCommand = () => {
  return new DeleteReceiptRuleSetCommand({ RuleSetName: RULE_SET_NAME });
};
```

```
const run = async () => {
  const deleteReceiptRuleSetCommand = createDeleteReceiptRuleSetCommand();

  try {
    return await sesClient.send(deleteReceiptRuleSetCommand);
  } catch (err) {
    console.log("Failed to delete receipt rule set.", err);
    return err;
  }
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考DeleteReceiptRuleSet中的。

`DeleteTemplate`

下列程式碼範例會示範如何使用`DeleteTemplate`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import { DeleteTemplateCommand } from "@aws-sdk/client-ses";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { sesClient } from "./libs/sesClient.js";

const TEMPLATE_NAME = getUniqueName("TemplateName");

const createDeleteTemplateCommand = (templateName) =>
  new DeleteTemplateCommand({ TemplateName: templateName });

const run = async () => {
  const deleteTemplateCommand = createDeleteTemplateCommand(TEMPLATE_NAME);

  try {
    return await sesClient.send(deleteTemplateCommand);
  } catch (err) {
```

```
    console.log("Failed to delete template.", err);
    return err;
  }
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考DeleteTemplate中的。

`GetTemplate`

下列程式碼範例會示範如何使用`GetTemplate`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import { GetTemplateCommand } from "@aws-sdk/client-ses";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { sesClient } from "./libs/sesClient.js";

const TEMPLATE_NAME = getUniqueName("TemplateName");

const createGetTemplateCommand = (templateName) =>
  new GetTemplateCommand({ TemplateName: templateName });

const run = async () => {
  const getTemplateCommand = createGetTemplateCommand(TEMPLATE_NAME);

  try {
    return await sesClient.send(getTemplateCommand);
  } catch (caught) {
    if (caught instanceof Error && caught.name === "MessageRejected") {
      /** @type { import('@aws-sdk/client-ses').MessageRejected} */
      const messageRejectedError = caught;
      return messageRejectedError;
    }
    throw caught;
  }
```

```
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考GetTemplate中的。

`ListIdentities`

下列程式碼範例會示範如何使用`ListIdentities`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import { ListIdentitiesCommand } from "@aws-sdk/client-ses";
import { sesClient } from "./libs/sesClient.js";

const createListIdentitiesCommand = () =>
  new ListIdentitiesCommand({ IdentityType: "EmailAddress", MaxItems: 10 });

const run = async () => {
  const listIdentitiesCommand = createListIdentitiesCommand();

  try {
    return await sesClient.send(listIdentitiesCommand);
  } catch (err) {
    console.log("Failed to list identities.", err);
    return err;
  }
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考ListIdentities中的。

`ListReceiptFilters`

下列程式碼範例會示範如何使用`ListReceiptFilters`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import { ListReceiptFiltersCommand } from "@aws-sdk/client-ses";
import { sesClient } from "./libs/sesClient.js";

const createListReceiptFiltersCommand = () => new ListReceiptFiltersCommand({});

const run = async () => {
  const listReceiptFiltersCommand = createListReceiptFiltersCommand();

  return await sesClient.send(listReceiptFiltersCommand);
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考ListReceiptFilters中的。

`ListTemplates`

下列程式碼範例會示範如何使用`ListTemplates`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import { ListTemplatesCommand } from "@aws-sdk/client-ses";
import { sesClient } from "./libs/sesClient.js";

const createListTemplatesCommand = (maxItems) =>
  new ListTemplatesCommand({ MaxItems: maxItems });
```

```
const run = async () => {
  const listTemplatesCommand = createListTemplatesCommand(10);

  try {
    return await sesClient.send(listTemplatesCommand);
  } catch (err) {
    console.log("Failed to list templates.", err);
    return err;
  }
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考ListTemplates中的。

`SendBulkTemplatedEmail`

下列程式碼範例會示範如何使用`SendBulkTemplatedEmail`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import { SendBulkTemplatedEmailCommand } from "@aws-sdk/client-ses";
import {
  getUniqueName,
  postfix,
} from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { sesClient } from "./libs/sesClient.js";

/**
 * Replace this with the name of an existing template.
 */
const TEMPLATE_NAME = getUniqueName("ReminderTemplate");

/**
 * Replace these with existing verified emails.
 */
const VERIFIED_EMAIL_1 = postfix(getUniqueName("Bilbo"), "@example.com");
```

```
const VERIFIED_EMAIL_2 = postfix(getUniqueName("Frodo"), "@example.com");

const USERS = [
  { firstName: "Bilbo", emailAddress: VERIFIED_EMAIL_1 },
  { firstName: "Frodo", emailAddress: VERIFIED_EMAIL_2 },
];

/**
 *
 * @param { { emailAddress: string, firstName: string }[] } users
 * @param { string } templateName the name of an existing template in SES
 * @returns { SendBulkTemplatedEmailCommand }
 */
const createBulkReminderEmailCommand = (users, templateName) => {
  return new SendBulkTemplatedEmailCommand({
    /**
     * Each 'Destination' uses a corresponding set of replacement data. We can map
 each user
     * to a 'Destination' and provide user specific replacement data to create
 personalized emails.
     *
     * Here's an example of how a template would be replaced with user data:
     * Template: <h1>Hello {{name}},</h1><p>Don't forget about the party gifts!</p>
     * Destination 1: <h1>Hello Bilbo,</h1><p>Don't forget about the party gifts!</
p>
     * Destination 2: <h1>Hello Frodo,</h1><p>Don't forget about the party gifts!</
p>
     */
    Destinations: users.map((user) => ({
      Destination: { ToAddresses: [user.emailAddress] },
      ReplacementTemplateData: JSON.stringify({ name: user.firstName }),
    })),
    DefaultTemplateData: JSON.stringify({ name: "Shireling" }),
    Source: VERIFIED_EMAIL_1,
    Template: templateName,
  });
};

const run = async () => {
  const sendBulkTemplateEmailCommand = createBulkReminderEmailCommand(
    USERS,
    TEMPLATE_NAME,
  );
  try {
```

```
    return await sesClient.send(sendBulkTemplateEmailCommand);
  } catch (caught) {
    if (caught instanceof Error && caught.name === "MessageRejected") {
      /** @type { import('@aws-sdk/client-ses').MessageRejected} */
      const messageRejectedError = caught;
      return messageRejectedError;
    }
    throw caught;
  }
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考SendBulkTemplatedEmail中的。

`SendEmail`

下列程式碼範例會示範如何使用`SendEmail`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import { SendEmailCommand } from "@aws-sdk/client-ses";
import { sesClient } from "./libs/sesClient.js";

const createSendEmailCommand = (toAddress, fromAddress) => {
  return new SendEmailCommand({
    Destination: {
      /* required */
      CcAddresses: [
        /* more items */
      ],
      ToAddresses: [
        toAddress,
        /* more To-email addresses */
      ],
    },
    Message: {
```

```
      /* required */
      Body: {
        /* required */
        Html: {
          Charset: "UTF-8",
          Data: "HTML_FORMAT_BODY",
        },
        Text: {
          Charset: "UTF-8",
          Data: "TEXT_FORMAT_BODY",
        },
      },
      Subject: {
        Charset: "UTF-8",
        Data: "EMAIL_SUBJECT",
      },
    },
    Source: fromAddress,
    ReplyToAddresses: [
      /* more items */
    ],
  });
};

const run = async () => {
  const sendEmailCommand = createSendEmailCommand(
    "recipient@example.com",
    "sender@example.com",
  );

  try {
    return await sesClient.send(sendEmailCommand);
  } catch (caught) {
    if (caught instanceof Error && caught.name === "MessageRejected") {
      /** @type { import('@aws-sdk/client-ses').MessageRejected} */
      const messageRejectedError = caught;
      return messageRejectedError;
    }
    throw caught;
  }
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考SendEmail中的。

SendRawEmail

下列程式碼範例會示範如何使用SendRawEmail。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

使用 nodemailer 發送含附件的電子郵件。

```
import sesClientModule from "@aws-sdk/client-ses";
/**
 * nodemailer wraps the SES SDK and calls SendRawEmail. Use this for more advanced
 * functionality like adding attachments to your email.
 *
 * https://nodemailer.com/transports/ses/
 */
import nodemailer from "nodemailer";

/**
 * @param {string} from An Amazon SES verified email address.
 * @param {*} to An Amazon SES verified email address.
 */
export const sendEmailWithAttachments = (
  from = "from@example.com",
  to = "to@example.com",
) => {
  const ses = new sesClientModule.SESClient({});
  const transporter = nodemailer.createTransport({
    SES: { ses, aws: sesClientModule },
  });

  return new Promise((resolve, reject) => {
    transporter.sendMail(
      {
        from,
```

```
        to,
        subject: "Hello World",
        text: "Greetings from Amazon SES!",
        attachments: [{ content: "Hello World!", filename: "hello.txt" }],
      },
      (err, info) => {
        if (err) {
          reject(err);
        } else {
          resolve(info);
        }
      },
    );
  });
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考SendRawEmail中的。

`SendTemplatedEmail`

下列程式碼範例會示範如何使用`SendTemplatedEmail`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import { SendTemplatedEmailCommand } from "@aws-sdk/client-ses";
import {
  getUniqueName,
  postfix,
} from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { sesClient } from "./libs/sesClient.js";

/**
 * Replace this with the name of an existing template.
 */
const TEMPLATE_NAME = getUniqueName("ReminderTemplate");
```

```
/**
 * Replace these with existing verified emails.
 */
const VERIFIED_EMAIL = postfix(getUniqueName("Bilbo"), "@example.com");

const USER = { firstName: "Bilbo", emailAddress: VERIFIED_EMAIL };

/**
 *
 * @param { { emailAddress: string, firstName: string } } user
 * @param { string } templateName - The name of an existing template in Amazon SES.
 * @returns { SendTemplatedEmailCommand }
 */
const createReminderEmailCommand = (user, templateName) => {
  return new SendTemplatedEmailCommand({
    /**
     * Here's an example of how a template would be replaced with user data:
     * Template: <h1>Hello {{contact.firstName}},</h1><p>Don't forget about the
 party gifts!</p>
     * Destination: <h1>Hello Bilbo,</h1><p>Don't forget about the party gifts!</p>
     */
    Destination: { ToAddresses: [user.emailAddress] },
    TemplateData: JSON.stringify({ contact: { firstName: user.firstName } }),
    Source: VERIFIED_EMAIL,
    Template: templateName,
  });
};

const run = async () => {
  const sendReminderEmailCommand = createReminderEmailCommand(
    USER,
    TEMPLATE_NAME,
  );
  try {
    return await sesClient.send(sendReminderEmailCommand);
  } catch (caught) {
    if (caught instanceof Error && caught.name === "MessageRejected") {
      /** @type { import('@aws-sdk/client-ses').MessageRejected} */
      const messageRejectedError = caught;
      return messageRejectedError;
    }
    throw caught;
  }
```

```
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考SendTemplatedEmail中的。

`UpdateTemplate`

下列程式碼範例會示範如何使用`UpdateTemplate`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import { UpdateTemplateCommand } from "@aws-sdk/client-ses";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { sesClient } from "./libs/sesClient.js";

const TEMPLATE_NAME = getUniqueName("TemplateName");
const HTML_PART = "<h1>Hello, World!</h1>";

const createUpdateTemplateCommand = () => {
  return new UpdateTemplateCommand({
    Template: {
      TemplateName: TEMPLATE_NAME,
      HtmlPart: HTML_PART,
      SubjectPart: "Example",
      TextPart: "Updated template text.",
    },
  });
};

const run = async () => {
  const updateTemplateCommand = createUpdateTemplateCommand();

  try {
    return await sesClient.send(updateTemplateCommand);
  } catch (err) {
```

```
    console.log("Failed to update template.", err);
    return err;
  }
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考UpdateTemplate中的。

`VerifyDomainIdentity`

下列程式碼範例會示範如何使用`VerifyDomainIdentity`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import { VerifyDomainIdentityCommand } from "@aws-sdk/client-ses";
import {
  getUniqueName,
  postfix,
} from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { sesClient } from "./libs/sesClient.js";

/**
 * You must have access to the domain's DNS settings to complete the
 * domain verification process.
 */
const DOMAIN_NAME = postfix(getUniqueName("Domain"), ".example.com");

const createVerifyDomainIdentityCommand = () => {
  return new VerifyDomainIdentityCommand({ Domain: DOMAIN_NAME });
};

const run = async () => {
  const VerifyDomainIdentityCommand = createVerifyDomainIdentityCommand();

  try {
```

```
    return await sesClient.send(VerifyDomainIdentityCommand);
  } catch (err) {
    console.log("Failed to verify domain.", err);
    return err;
  }
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考VerifyDomainIdentity中的。

`VerifyEmailIdentity`

下列程式碼範例會示範如何使用`VerifyEmailIdentity`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
// Import required AWS SDK clients and commands for Node.js
import { VerifyEmailIdentityCommand } from "@aws-sdk/client-ses";
import { sesClient } from "./libs/sesClient.js";

const EMAIL_ADDRESS = "name@example.com";

const createVerifyEmailIdentityCommand = (emailAddress) => {
  return new VerifyEmailIdentityCommand({ EmailAddress: emailAddress });
};

const run = async () => {
  const verifyEmailIdentityCommand =
    createVerifyEmailIdentityCommand(EMAIL_ADDRESS);
  try {
    return await sesClient.send(verifyEmailIdentityCommand);
  } catch (err) {
    console.log("Failed to verify email identity.", err);
    return err;
  }
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考VerifyEmailIdentity中的。

案例

建置 Amazon Transcribe 串流應用程式

下面的程式碼範例說明如何建置可即時記錄、轉錄和翻譯直播音訊並透過電子郵件傳送結果的應用程式。

SDK對於 JavaScript（3）

示範如何使用 Amazon Transcribe 建置可即時記錄、轉錄和轉譯即時音訊的應用程式，並使用 Amazon 簡易電子郵件服務 (Amazon) 將結果以電子郵件傳送給結果。SES

有關如何設置和運行的完整源代碼和說明，請參閱中的完整示例GitHub。

此範例中使用的服務

- Amazon Comprehend
- Amazon SES
- Amazon Transcribe
- Amazon Translate

建立 Web 應用程式以追蹤 DynamoDB 資料

下列程式碼範例示範如何建立可追蹤 Amazon DynamoDB 表中工作項目的 Web 應用程式，並使用 Amazon 簡單電子郵件服務 (AmazonSES) 傳送報告。

SDK對於 JavaScript（3）

示範如何使用 Amazon DynamoDB API 建立可追蹤 DynamoDB 工作資料的動態 Web 應用程式。

有關如何設置和運行的完整源代碼和說明，請參閱中的完整示例GitHub。

此範例中使用的服務

- DynamoDB
- Amazon SES

建立 Aurora 無伺服器工作項目追蹤器

下列程式碼範例示範如何建立可追蹤 Amazon Aurora 無伺服器資料庫中工作項目的 Web 應用程式，並使用 Amazon 簡單電子郵件服務 (AmazonSES) 傳送報告。

SDK對於 JavaScript （ 3 ）

示範如何使用 AWS SDK for JavaScript (v3) 建立 Web 應用程式，以追蹤 Amazon Aurora 資料庫中的工作項目，以及使用 Amazon 簡單電子郵件服務 (AmazonSES) 傳送電子郵件報告的 Web 應用程式。這個範例使用以 React.js 建置的前端與 Express Node.js 後端互動。

- 將 React.js 網路應用程式與 AWS 服務.
- 列出、新增和更新 Aurora 資料表中的項目。
- 使用 Amazon 傳送篩選工作項目的電子郵件報告SES。
- 使用隨附的 AWS CloudFormation 指令碼部署和管理範例資源。

有關如何設置和運行的完整源代碼和說明，請參閱中的完整示例GitHub。

此範例中使用的服務

- Aurora
- Amazon RDS
- Amazon RDS 數據服務
- Amazon SES

PPE在影像中偵測

下列程式碼範例示範如何建立使用 Amazon Rekognition 偵測影像中個人防護設備 (PPE) 的應用程式。

SDK對於 JavaScript （ 3 ）

示範如何搭配使用 Amazon Rekognition AWS SDK for JavaScript 來建立應用程式，以偵測位於亞馬遜簡單儲存服務 (Amazon S3PPE) 儲存貯體中的映像中的個人防護設備 ()。該應用程式會將結果儲存到 Amazon DynamoDB 表，並使用 Amazon 簡單電子郵件服務 (AmazonSES) 向管理員傳送包含結果的電子郵件通知。

了解如何：

- 使用 Amazon Cognito 建立未經身分驗證的使用者。
- 分析圖像以PPE使用 Amazon Rekognition。

- 驗證 Amazon 的電子郵件地址SES。
- 以結果更新 DynamoDB 資料表。
- 使用 Amazon 發送電子郵件通知SES。

有關如何設置和運行的完整源代碼和說明，請參閱中的完整示例GitHub。

此範例中使用的服務

- DynamoDB
- Amazon Rekognition
- Amazon S3
- Amazon SES

偵測映像中的物件

下列程式碼範例示範如何建立使用 Amazon Rekognition 依照類別偵測影像中物件的應用程式。

SDK對於 JavaScript（3）

示範如何搭配使用 Amazon Rekognition AWS SDK for JavaScript 來建立使用 Amazon Rekognition 的應用程式，在位於亞馬遜簡單儲存服務 (Amazon S3) 儲存貯體中的映像中依類別識別物件。該應用程序向管理員發送電子郵件通知，其中包含使用 Amazon 簡單電子郵件服務（AmazonSES）的結果

了解如何：

- 使用 Amazon Cognito 建立未經身分驗證的使用者。
- 使用 Amazon Rekognition 分析映像中的物件。
- 驗證 Amazon 的電子郵件地址SES。
- 使用 Amazon 發送電子郵件通知SES。

有關如何設置和運行的完整源代碼和說明，請參閱中的完整示例GitHub。

此範例中使用的服務

- Amazon Rekognition
- Amazon S3
- Amazon SES

Amazon SNS 示例使用 SDK for JavaScript (v3)

下列程式碼範例說明如何透過使用 AWS SDK for JavaScript (v3) 搭配 Amazon 來執行動作和實作常見案例SNS。

Actions 是大型程式的程式碼摘錄，必須在內容中執行。雖然動作會顯示如何呼叫個別服務函數，但您可以在其相關案例中查看內容中的動作。

案例是程式碼範例，向您展示如何透過呼叫服務中的多個函式或與其他函式結合來完成特定工作 AWS 服務。

每個範例都包含完整原始程式碼的連結，您可以在其中找到如何在內容中設定和執行程式碼的指示。

開始使用

你好 Amazon SNS

下列程式碼範例說明如何開始使用 Amazon SNS。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

初始化SNS客戶端並列出您帳戶中的主題。

```
import { SNSClient, paginateListTopics } from "@aws-sdk/client-sns";

export const helloSns = async () => {
  // The configuration object (`{}`) is required. If the region and credentials
  // are omitted, the SDK uses your local configuration if it exists.
  const client = new SNSClient({});

  // You can also use `ListTopicsCommand`, but to use that command you must
  // handle the pagination yourself. You can do that by sending the
 `ListTopicsCommand`
  // with the `NextToken` parameter from the previous request.
  const paginatedTopics = paginateListTopics({ client }, {});
```

```
  const topics = [];

  for await (const page of paginatedTopics) {
    if (page.Topics?.length) {
      topics.push(...page.Topics);
    }
  }

  const suffix = topics.length === 1 ? "" : "s";

  console.log(
    `Hello, Amazon SNS! You have ${topics.length} topic${suffix} in your account.`,
  );
  console.log(topics.map((t) => `  * ${t.TopicArn}`).join("\n"));
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考ListTopics中的。

主題

- 動作
- 案例
- 無伺服器範例

動作

CheckIfPhoneNumberIsOptedOut

下列程式碼範例會示範如何使用CheckIfPhoneNumberIsOptedOut。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

在單獨的模組中建立用戶端並將其匯出。

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
 blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

匯入SDK和用戶端模組並呼叫API.

```
import { CheckIfPhoneNumberIsOptedOutCommand } from "@aws-sdk/client-sns";

import { snsClient } from "../libs/snsClient.js";

export const checkIfPhoneNumberIsOptedOut = async (
  phoneNumber = "5555555555",
) => {
  const command = new CheckIfPhoneNumberIsOptedOutCommand({
    phoneNumber,
  });

  const response = await snsClient.send(command);
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '3341c28a-cdc8-5b39-a3ee-9fb0ee125732',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   isOptedOut: false
  // }
  return response;
};
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考CheckIfPhoneNumberIsOptedOut中的。

`ConfirmSubscription`

下列程式碼範例會示範如何使用`ConfirmSubscription`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

在單獨的模組中建立用戶端並將其匯出。

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
 blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

匯入SDK和用戶端模組並呼叫API.

```
import { ConfirmSubscriptionCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} token - This token is sent the subscriber. Only subscribers
 *                         that are not AWS services (HTTP/S, email) need to be
 confirmed.
 * @param {string} topicArn - The ARN of the topic for which you wish to confirm a
 subscription.
 */
export const confirmSubscription = async (
  token = "TOKEN",
  topicArn = "TOPIC_ARN",
) => {
  const response = await snsClient.send(
    // A subscription only needs to be confirmed if the endpoint type is
    // HTTP/S, email, or in another AWS account.
    new ConfirmSubscriptionCommand({
```

```
      Token: token,
      TopicArn: topicArn,
      // If this is true, the subscriber cannot unsubscribe while unauthenticated.
      AuthenticateOnUnsubscribe: "false",
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '4bb5bce9-805a-5517-8333-e1d2cface90b',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   SubscriptionArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxxx:TOPIC_NAME:xxxxxxxx-
xxxx-xxxx-xxxx-xxxxxxxxxxxx'
  // }
  return response;
};
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考ConfirmSubscription中的。

`CreateTopic`

下列程式碼範例會示範如何使用`CreateTopic`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

在單獨的模組中建立用戶端並將其匯出。

```
import { SNSClient } from "@aws-sdk/client-sns";
```

```
// The AWS Region can be provided here using the `region` property. If you leave it
 blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

匯入SDK和用戶端模組並呼叫API.

```
import { CreateTopicCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} topicName - The name of the topic to create.
 */
export const createTopic = async (topicName = "TOPIC_NAME") => {
  const response = await snsClient.send(
    new CreateTopicCommand({ Name: topicName }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '087b8ad2-4593-50c4-a496-d7e90b82cf3e',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   TopicArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxxx:TOPIC_NAME'
  // }
  return response;
};
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考CreateTopic中的。

DeleteTopic

下列程式碼範例會示範如何使用DeleteTopic。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

在單獨的模組中建立用戶端並將其匯出。

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
 blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

匯入SDK和用戶端模組並呼叫API.

```
import { DeleteTopicCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} topicArn - The ARN of the topic to delete.
 */
export const deleteTopic = async (topicArn = "TOPIC_ARN") => {
  const response = await snsClient.send(
    new DeleteTopicCommand({ TopicArn: topicArn }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'a10e2886-5a8f-5114-af36-75bd39498332',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   }
  // }
};
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考DeleteTopic中的。

`GetSMSAttributes`

下列程式碼範例會示範如何使用`GetSMSAttributes`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

在單獨的模組中建立用戶端並將其匯出。

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
 blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

匯入SDK和用戶端模組並呼叫API.

```
import { GetSMSAttributesCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

export const getSmsAttributes = async () => {
  const response = await snsClient.send(
    // If you have not modified the account-level mobile settings of SNS,
    // the DefaultSMSType is undefined. For this example, it was set to
    // Transactional.
    new GetSMSAttributesCommand({ attributes: ["DefaultSMSType"] }),
  );
```

```
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '67ad8386-4169-58f1-bdb9-debd281d48d5',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   attributes: { DefaultSMSType: 'Transactional' }
  // }
  return response;
};
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考資料etSMSAttributes中的 G

`GetTopicAttributes`

下列程式碼範例會示範如何使用`GetTopicAttributes`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

在單獨的模組中建立用戶端並將其匯出。

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
 blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

匯入SDK和用戶端模組並呼叫API.

```
import { GetTopicAttributesCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} topicArn - The ARN of the topic to retrieve attributes for.
 */
export const getTopicAttributes = async (topicArn = "TOPIC_ARN") => {
  const response = await snsClient.send(
    new GetTopicAttributesCommand({
      TopicArn: topicArn,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '36b6a24e-5473-5d4e-ac32-ff72d9a73d94',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   Attributes: {
  //     Policy: '{...}',
  //     Owner: 'xxxxxxxxxxxx',
  //     SubscriptionsPending: '1',
  //     TopicArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxxx:mytopic',
  //     TracingConfig: 'PassThrough',
  //     EffectiveDeliveryPolicy: '{"http":{"defaultHealthyRetryPolicy":
{"minDelayTarget":20,"maxDelayTarget":20,"numRetries":3,"numMaxDelayRetries":0,"numNoDelayRe
{"headerContentType":"text/plain; charset=UTF-8"}}}',
  //     SubscriptionsConfirmed: '0',
  //     DisplayName: '',
  //     SubscriptionsDeleted: '1'
  //   }
  // }
  return response;
};
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考GetTopicAttributes中的。

SDK對於 JavaScript （ 第 2 個 ）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

匯入SDK和用戶端模組並呼叫API.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set region
AWS.config.update({ region: "REGION" });

// Create promise and SNS service object
var getTopicAttribsPromise = new AWS.SNS({ apiVersion: "2010-03-31" })
  .getTopicAttributes({ TopicArn: "TOPIC_ARN" })
  .promise();

// Handle promise's fulfilled/rejected states
getTopicAttribsPromise
  .then(function (data) {
    console.log(data);
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考GetTopicAttributes中的。

`ListSubscriptions`

下列程式碼範例會示範如何使用`ListSubscriptions`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

在單獨的模組中建立用戶端並將其匯出。

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
 blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

匯入SDK和用戶端模組並呼叫API.

```
import { ListSubscriptionsByTopicCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} topicArn - The ARN of the topic for which you wish to list
 subscriptions.
 */
export const listSubscriptionsByTopic = async (topicArn = "TOPIC_ARN") => {
  const response = await snsClient.send(
    new ListSubscriptionsByTopicCommand({ TopicArn: topicArn }),
  );

  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '0934fedf-0c4b-572e-9ed2-a3e38fadb0c8',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
```

```
  //   Subscriptions: [
  //     {
  //       SubscriptionArn: 'PendingConfirmation',
  //       Owner: '901487484989',
  //       Protocol: 'email',
  //       Endpoint: 'corepyle@amazon.com',
  //       TopicArn: 'arn:aws:sns:us-east-1:901487484989:mytopic'
  //     }
  //   ]
  // }
  return response;
};
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考ListSubscriptions中的。

`ListTopics`

下列程式碼範例會示範如何使用`ListTopics`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

在單獨的模組中建立用戶端並將其匯出。

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
 blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

匯入SDK和用戶端模組並呼叫API.

```
import { ListTopicsCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

export const listTopics = async () => {
  const response = await snsClient.send(new ListTopicsCommand({}));
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '936bc5ad-83ca-53c2-b0b7-9891167b909e',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   Topics: [ { TopicArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxxx:mytopic' } ]
  // }
  return response;
};
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考ListTopics中的。

Publish

下列程式碼範例會示範如何使用Publish。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

在單獨的模組中建立用戶端並將其匯出。

```
import { SNSClient } from "@aws-sdk/client-sns";
```

```
// The AWS Region can be provided here using the `region` property. If you leave it
 blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

匯入SDK和用戶端模組並呼叫API.

```
import { PublishCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string | Record<string, any>} message - The message to send. Can be a
 plain string or an object
 *                                                    if you are using the `json`
 `MessageStructure`.
 * @param {string} topicArn - The ARN of the topic to which you would like to
 publish.
 */
export const publish = async (
  message = "Hello from SNS!",
  topicArn = "TOPIC_ARN",
) => {
  const response = await snsClient.send(
    new PublishCommand({
      Message: message,
      TopicArn: topicArn,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'e7f77526-e295-5325-9ee4-281a43ad1f05',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   MessageId: 'xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx'
  // }
  return response;
};
```

將訊息發佈至具有群組、複寫和屬性選項的主題。

```
async publishMessages() {
  const message = await this.prompter.input({
    message: MESSAGES.publishMessagePrompt,
  });

  let groupId, deduplicationId, choices;

  if (this.isFifo) {
    await this.logger.log(MESSAGES.groupIdNotice);
    groupId = await this.prompter.input({
      message: MESSAGES.groupIdPrompt,
    });

    if (this.autoDedup === false) {
      await this.logger.log(MESSAGES.deduplicationIdNotice);
      deduplicationId = await this.prompter.input({
        message: MESSAGES.deduplicationIdPrompt,
      });
    }

    choices = await this.prompter.checkbox({
      message: MESSAGES.messageAttributesPrompt,
      choices: toneChoices,
    });
  }

  await this.snsClient.send(
    new PublishCommand({
      TopicArn: this.topicArn,
      Message: message,
      ...(groupId
        ? {
            MessageGroupId: groupId,
          }
        : {}),
      ...(deduplicationId
        ? {
            MessageDeduplicationId: deduplicationId,
          }
```

```
          : {}),
        ...(choices
          ? {
              MessageAttributes: {
                tone: {
                  DataType: "String.Array",
                  StringValue: JSON.stringify(choices),
                },
              },
            }
          : {}),
      }),
    );

    const publishAnother = await this.prompter.confirm({
      message: MESSAGES.publishAnother,
    });

    if (publishAnother) {
      await this.publishMessages();
    }
  }
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱在AWS SDK for JavaScript API參考中發佈。

`SetSMSAttributes`

下列程式碼範例會示範如何使用`SetSMSAttributes`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

在單獨的模組中建立用戶端並將其匯出。

```
import { SNSClient } from "@aws-sdk/client-sns";
```

```
// The AWS Region can be provided here using the `region` property. If you leave it
 blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

匯入SDK和用戶端模組並呼叫API.

```
import { SetSMSAttributesCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {"Transactional" | "Promotional"} defaultSmsType
 */
export const setSmsType = async (defaultSmsType = "Transactional") => {
  const response = await snsClient.send(
    new SetSMSAttributesCommand({
      attributes: {
        // Promotional – (Default) Noncritical messages, such as marketing messages.
        // Transactional – Critical messages that support customer transactions,
        // such as one-time passcodes for multi-factor authentication.
        DefaultSMSType: defaultSmsType,
      },
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '1885b977-2d7e-535e-8214-e44be727e265',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   }
  // }
  return response;
};
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考etSMSAttributes中的 S。

SetTopicAttributes

下列程式碼範例會示範如何使用SetTopicAttributes。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

在單獨的模組中建立用戶端並將其匯出。

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
 blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

匯入SDK和用戶端模組並呼叫API.

```
import { SetTopicAttributesCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

export const setTopicAttributes = async (
  topicArn = "TOPIC_ARN",
  attributeName = "DisplayName",
  attributeValue = "Test Topic",
) => {
  const response = await snsClient.send(
    new SetTopicAttributesCommand({
      AttributeName: attributeName,
      AttributeValue: attributeValue,
      TopicArn: topicArn,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
```

```
  //     httpStatusCode: 200,
  //     requestId: 'd1b08d0e-e9a4-54c3-b8b1-d03238d2b935',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   }
  // }
  return response;
};
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考SetTopicAttributes中的。

Subscribe

下列程式碼範例會示範如何使用Subscribe。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

在單獨的模組中建立用戶端並將其匯出。

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
 blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

匯入SDK和用戶端模組並呼叫API.

```
import { SubscribeCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} topicArn - The ARN of the topic for which you wish to confirm a
 subscription.
 * @param {string} emailAddress - The email address that is subscribed to the topic.
 */
export const subscribeEmail = async (
  topicArn = "TOPIC_ARN",
  emailAddress = "usern@me.com",
) => {
  const response = await snsClient.send(
    new SubscribeCommand({
      Protocol: "email",
      TopicArn: topicArn,
      Endpoint: emailAddress,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'c8e35bcd-b3c0-5940-9f66-06f6fcc108f0',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   SubscriptionArn: 'pending confirmation'
  // }
};
```

訂閱某個主題的行動應用程式。

```
import { SubscribeCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} topicArn - The ARN of the topic the subscriber is subscribing to.
 * @param {string} endpoint - The Endpoint ARN of an application. This endpoint is
 created
```

```
 *                            when an application registers for notifications.
 */
export const subscribeApp = async (
  topicArn = "TOPIC_ARN",
  endpoint = "ENDPOINT",
) => {
  const response = await snsClient.send(
    new SubscribeCommand({
      Protocol: "application",
      TopicArn: topicArn,
      Endpoint: endpoint,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'c8e35bcd-b3c0-5940-9f66-06f6fcc108f0',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   SubscriptionArn: 'pending confirmation'
  // }
  return response;
};
```

訂閱某個主題的 Lambda 函數。

```
import { SubscribeCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} topicArn - The ARN of the topic the subscriber is subscribing to.
 * @param {string} endpoint - The Endpoint ARN of and AWS Lambda function.
 */
export const subscribeLambda = async (
  topicArn = "TOPIC_ARN",
  endpoint = "ENDPOINT",
) => {
  const response = await snsClient.send(
```

```
    new SubscribeCommand({
      Protocol: "lambda",
      TopicArn: topicArn,
      Endpoint: endpoint,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'c8e35bcd-b3c0-5940-9f66-06f6fcc108f0',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   SubscriptionArn: 'pending confirmation'
  // }
  return response;
};
```

訂閱SQS佇列以取得主題。

```
import { SubscribeCommand, SNSClient } from "@aws-sdk/client-sns";

const client = new SNSClient({});

export const subscribeQueue = async (
  topicArn = "TOPIC_ARN",
  queueArn = "QUEUE_ARN",
) => {
  const command = new SubscribeCommand({
    TopicArn: topicArn,
    Protocol: "sqs",
    Endpoint: queueArn,
  });

  const response = await client.send(command);
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
```

```
  //     requestId: '931e13d9-5e2b-543f-8781-4e9e494c5ff2',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   SubscriptionArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxxx:subscribe-queue-
test-430895:xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx'
  // }
  return response;
};
```

使用篩選器訂閱主題。

```
import { SubscribeCommand, SNSClient } from "@aws-sdk/client-sns";

const client = new SNSClient({});

export const subscribeQueueFiltered = async (
  topicArn = "TOPIC_ARN",
  queueArn = "QUEUE_ARN",
) => {
  const command = new SubscribeCommand({
    TopicArn: topicArn,
    Protocol: "sqs",
    Endpoint: queueArn,
    Attributes: {
      // This subscription will only receive messages with the 'event' attribute set
 to 'order_placed'.
      FilterPolicyScope: "MessageAttributes",
      FilterPolicy: JSON.stringify({
        event: ["order_placed"],
      }),
    },
  });

  const response = await client.send(command);
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '931e13d9-5e2b-543f-8781-4e9e494c5ff2',
```

```
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   SubscriptionArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxxx:subscribe-queue-
test-430895:xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx'
  // }
  return response;
};
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScriptAPI參考中的訂閱。

`Unsubscribe`

下列程式碼範例會示範如何使用Unsubscribe。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

在單獨的模組中建立用戶端並將其匯出。

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
 blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

匯入SDK和用戶端模組並呼叫API.

```
import { UnsubscribeCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";
```

```
/**
 * @param {string} subscriptionArn - The ARN of the subscription to cancel.
 */
const unsubscribe = async (
  subscriptionArn = "arn:aws:sns:us-east-1:xxxxxxxxxxxx:mytopic:xxxxxxxx-xxxx-xxxx-
xxxx-xxxxxxxxxxxx",
) => {
  const response = await snsClient.send(
    new UnsubscribeCommand({
      SubscriptionArn: subscriptionArn,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '0178259a-9204-507c-b620-78a7570a44c6',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   }
  // }
  return response;
};
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScriptAPI參考資料中的取

案例

建置應用程式以將資料提交至 DynamoDB 資料表

下列程式碼範例顯示如何建立將資料提交至 Amazon DynamoDB 表的應用程式，並在使用者更新表格時通知您。

SDK對於 JavaScript（3）

此範例說明如何建立可讓使用者將資料提交至 Amazon DynamoDB 表格的應用程式，以及如何使用 Amazon 簡單通知服務 (AmazonSNS) 傳送文字訊息給管理員。

有關如何設置和運行的完整源代碼和說明，請參閱中的完整示例GitHub。

此範例也可在 AWS SDK for JavaScript v3 開發人員指南中取得。

此範例中使用的服務

- DynamoDB
- Amazon SNS

建立無伺服器應用程式來管理相片

下列程式碼範例示範如何建立無伺服器應用程式，讓使用者以標籤管理相片。

SDK對於 JavaScript（3）

顯示如何開發照片資產管理應用程式，以便使用 Amazon Rekognition 偵測圖片中的標籤，並將其儲存以供日後擷取。

有關如何設置和運行的完整源代碼和說明，請參閱中的完整示例 GitHub。

如要深入探索此範例的來源，請參閱 AWS 社群上的文章。

此範例中使用的服務

- API閘道
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

建立 Amazon Textract Explorer 應用程式

下列程式碼範例顯示如何透過互動式應用程式探索 Amazon Textract 輸出。

SDK對於 JavaScript（3）

示範如何使用建置 React 應用程式，該應用程式使用 Amazon Textract 擷取文件影像中的資料，並將其顯示在互動式網頁中。 AWS SDK for JavaScript 此範例會在 Web 瀏覽器中執行，且登入資料需要經過驗證的 Amazon Cognito 身分。它使用 Amazon 簡單存儲服務（Amazon S3）進行存儲，

並對於通知，它會輪詢訂閱 Amazon 簡單通知服務（AmazonSQS）主題的 Amazon 簡單隊列（亞馬遜SNS）隊列。

有關如何設置和運行的完整源代碼和說明，請參閱中的完整示例GitHub。

此範例中使用的服務

- Amazon Cognito Identity
- Amazon S3
- Amazon SNS
- Amazon SQS
- Amazon Textract

將訊息發佈至佇列

以下程式碼範例顯示做法：

- 創建主題（FIFO或非FIFO）。
- 為主題訂閱多個佇列，並提供套用篩選條件的選擇。
- 發佈訊息至主題。
- 輪詢佇列以獲取收到的訊息。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

這是此工作流程的進入點。

```
import { SNSClient } from "@aws-sdk/client-sns";
import { SQSClient } from "@aws-sdk/client-sqs";

import { TopicsQueuesWkflw } from "./TopicsQueuesWkflw.js";
import { Prompter } from "@aws-doc-sdk-examples/lib/prompter.js";
import { SlowLogger } from "@aws-doc-sdk-examples/lib/slow-logger.js";
```

```
export const startSnsWorkflow = () => {
  const noLoggerDelay = process.argv.find((arg) => arg === "--no-logger-delay");
  const snsClient = new SNSClient({});
  const sqsClient = new SQSClient({});
  const prompter = new Prompter();
  const logger = noLoggerDelay ? console : new SlowLogger(25);

  const wkflw = new TopicsQueuesWkflw(snsClient, sqsClient, prompter, logger);

  wkflw.start();
};
```

先前的程式碼提供了必要的相依性並啟動工作流程。下一節包含範例的大量內容。

```
const toneChoices = [
  { name: "cheerful", value: "cheerful" },
  { name: "funny", value: "funny" },
  { name: "serious", value: "serious" },
  { name: "sincere", value: "sincere" },
];

export class TopicsQueuesWkflw {
  // SNS topic is configured as First-In-First-Out
  isFifo = true;

  // Automatic content-based deduplication is enabled.
  autoDedup = false;

  snsClient;
  sqsClient;
  topicName;
  topicArn;
  subscriptionArns = [];
  /**
   * @type {{ queueName: string, queueArn: string, queueUrl: string, policy?:
 string }[]}
   */
  queues = [];
  prompter;
```

```
/**
 * @param {import('@aws-sdk/client-sns').SNSClient} snsClient
 * @param {import('@aws-sdk/client-sqs').SQSClient} sqsClient
 * @param {import('../../libs/prompter.js').Prompter} prompter
 * @param {import('../../libs/logger.js').Logger} logger
 */
constructor(snsClient, sqsClient, prompter, logger) {
  this.snsClient = snsClient;
  this.sqsClient = sqsClient;
  this.prompter = prompter;
  this.logger = logger;
}

async welcome() {
  await this.logger.log(MESSAGES.description);
}

async confirmFifo() {
  await this.logger.log(MESSAGES.snsFifoDescription);
  this.isFifo = await this.prompter.confirm({
    message: MESSAGES.snsFifoPrompt,
  });

  if (this.isFifo) {
    this.logger.logSeparator(MESSAGES.headerDedup);
    await this.logger.log(MESSAGES.deduplicationNotice);
    await this.logger.log(MESSAGES.deduplicationDescription);
    this.autoDedup = await this.prompter.confirm({
      message: MESSAGES.deduplicationPrompt,
    });
  }
}

async createTopic() {
  await this.logger.log(MESSAGES.creatingTopics);
  this.topicName = await this.prompter.input({
    message: MESSAGES.topicNamePrompt,
  });
  if (this.isFifo) {
    this.topicName += ".fifo";
    this.logger.logSeparator(MESSAGES.headerFifoNaming);
    await this.logger.log(MESSAGES.appendFifoNotice);
  }
```

```
    const response = await this.snsClient.send(
      new CreateTopicCommand({
        Name: this.topicName,
        Attributes: {
          FifoTopic: this.isFifo ? "true" : "false",
          ...(this.autoDedup ? { ContentBasedDeduplication: "true" } : {}),
        },
      }),
    );

    this.topicArn = response.TopicArn;

    await this.logger.log(
      MESSAGES.topicCreatedNotice
        .replace("${TOPIC_NAME}", this.topicName)
        .replace("${TOPIC_ARN}", this.topicArn),
    );
  }

  async createQueues() {
    await this.logger.log(MESSAGES.createQueuesNotice);
    // Increase this number to add more queues.
    let maxQueues = 2;

    for (let i = 0; i < maxQueues; i++) {
      await this.logger.log(MESSAGES.queueCount.replace("${COUNT}", i + 1));
      let queueName = await this.prompter.input({
        message: MESSAGES.queueNamePrompt.replace(
          "${EXAMPLE_NAME}",
          i === 0 ? "good-news" : "bad-news",
        ),
      });

      if (this.isFifo) {
        queueName += ".fifo";
        await this.logger.log(MESSAGES.appendFifoNotice);
      }

      const response = await this.sqsClient.send(
        new CreateQueueCommand({
          QueueName: queueName,
          Attributes: { ...(this.isFifo ? { FifoQueue: "true" } : {}) },
        }),
```

```
      );

      const { Attributes } = await this.sqsClient.send(
        new GetQueueAttributesCommand({
          QueueUrl: response.QueueUrl,
          AttributeNames: ["QueueArn"],
        }),
      );

      this.queues.push({
        queueName,
        queueArn: Attributes.QueueArn,
        queueUrl: response.QueueUrl,
      });

      await this.logger.log(
        MESSAGES.queueCreatedNotice
          .replace("${QUEUE_NAME}", queueName)
          .replace("${QUEUE_URL}", response.QueueUrl)
          .replace("${QUEUE_ARN}", Attributes.QueueArn),
      );
    }
  }

  async attachQueueIamPolicies() {
    for (const [index, queue] of this.queues.entries()) {
      const policy = JSON.stringify(
        {
          Statement: [
            {
              Effect: "Allow",
              Principal: {
                Service: "sns.amazonaws.com",
              },
              Action: "sqs:SendMessage",
              Resource: queue.queueArn,
              Condition: {
                ArnEquals: {
                  "aws:SourceArn": this.topicArn,
                },
              },
            },
          ],
        },
```

```
        null,
        2,
      );

      if (index !== 0) {
        this.logger.logSeparator();
      }

      await this.logger.log(MESSAGES.attachPolicyNotice);
      console.log(policy);
      const addPolicy = await this.prompter.confirm({
        message: MESSAGES.addPolicyConfirmation.replace(
          "${QUEUE_NAME}",
          queue.queueName,
        ),
      });

      if (addPolicy) {
        await this.sqsClient.send(
          new SetQueueAttributesCommand({
            QueueUrl: queue.queueUrl,
            Attributes: {
              Policy: policy,
            },
          }),
        );
        queue.policy = policy;
      } else {
        await this.logger.log(
          MESSAGES.policyNotAttachedNotice.replace(
            "${QUEUE_NAME}",
            queue.queueName,
          ),
        );
      }
    }
  }

  async subscribeQueuesToTopic() {
    for (const [index, queue] of this.queues.entries()) {
      /**
       * @type {import('@aws-sdk/client-sns').SubscribeCommandInput}
       */
      const subscribeParams = {
```

```
        TopicArn: this.topicArn,
        Protocol: "sqs",
        Endpoint: queue.queueArn,
      };
      let tones = [];

      if (this.isFifo) {
        if (index === 0) {
          await this.logger.log(MESSAGES.fifoFilterNotice);
        }
        tones = await this.prompter.checkbox({
          message: MESSAGES.fifoFilterSelect.replace(
            "${QUEUE_NAME}",
            queue.queueName,
          ),
          choices: toneChoices,
        });

        if (tones.length) {
          subscribeParams.Attributes = {
            FilterPolicyScope: "MessageAttributes",
            FilterPolicy: JSON.stringify({
              tone: tones,
            }),
          };
        }
      }

      const { SubscriptionArn } = await this.snsClient.send(
        new SubscribeCommand(subscribeParams),
      );

      this.subscriptionArns.push(SubscriptionArn);

      await this.logger.log(
        MESSAGES.queueSubscribedNotice
          .replace("${QUEUE_NAME}", queue.queueName)
          .replace("${TOPIC_NAME}", this.topicName)
          .replace("${TONES}", tones.length ? tones.join(", ") : "none"),
      );
    }
  }

  async publishMessages() {
```

```
    const message = await this.prompter.input({
      message: MESSAGES.publishMessagePrompt,
    });

    let groupId, deduplicationId, choices;

    if (this.isFifo) {
      await this.logger.log(MESSAGES.groupIdNotice);
      groupId = await this.prompter.input({
        message: MESSAGES.groupIdPrompt,
      });

      if (this.autoDedup === false) {
        await this.logger.log(MESSAGES.deduplicationIdNotice);
        deduplicationId = await this.prompter.input({
          message: MESSAGES.deduplicationIdPrompt,
        });
      }

      choices = await this.prompter.checkbox({
        message: MESSAGES.messageAttributesPrompt,
        choices: toneChoices,
      });
    }

    await this.snsClient.send(
      new PublishCommand({
        TopicArn: this.topicArn,
        Message: message,
        ...(groupId
          ? {
              MessageGroupId: groupId,
            }
          : {}),
        ...(deduplicationId
          ? {
              MessageDeduplicationId: deduplicationId,
            }
          : {}),
        ...(choices
          ? {
              MessageAttributes: {
                tone: {
                  DataType: "String.Array",
```

```
                StringValue: JSON.stringify(choices),
              },
            },
          }
        : {}),
    }),
  );

  const publishAnother = await this.prompter.confirm({
    message: MESSAGES.publishAnother,
  });

  if (publishAnother) {
    await this.publishMessages();
  }
}

async receiveAndDeleteMessages() {
  for (const queue of this.queues) {
    const { Messages } = await this.sqsClient.send(
      new ReceiveMessageCommand({
        QueueUrl: queue.queueUrl,
      }),
    );

    if (Messages) {
      await this.logger.log(
        MESSAGES.messagesReceivedNotice.replace(
          "${QUEUE_NAME}",
          queue.queueName,
        ),
      );
      console.log(Messages);

      await this.sqsClient.send(
        new DeleteMessageBatchCommand({
          QueueUrl: queue.queueUrl,
          Entries: Messages.map((message) => ({
            Id: message.MessageId,
            ReceiptHandle: message.ReceiptHandle,
          })),
        }),
      );
    } else {
```

```
        await this.logger.log(
          MESSAGES.noMessagesReceivedNotice.replace(
            "${QUEUE_NAME}",
            queue.queueName,
          ),
        );
      }
    }

    const deleteAndPoll = await this.prompter.confirm({
      message: MESSAGES.deleteAndPollConfirmation,
    });

    if (deleteAndPoll) {
      await this.receiveAndDeleteMessages();
    }
  }

  async destroyResources() {
    for (const subscriptionArn of this.subscriptionArns) {
      await this.snsClient.send(
        new UnsubscribeCommand({ SubscriptionArn: subscriptionArn }),
      );
    }

    for (const queue of this.queues) {
      await this.sqsClient.send(
        new DeleteQueueCommand({ QueueUrl: queue.queueUrl }),
      );
    }

    if (this.topicArn) {
      await this.snsClient.send(
        new DeleteTopicCommand({ TopicArn: this.topicArn }),
      );
    }
  }

  async start() {
    console.clear();

    try {
      this.logger.logSeparator(MESSAGES.headerWelcome);
      await this.welcome();
```

```
      this.logger.logSeparator(MESSAGES.headerFifo);
      await this.confirmFifo();
      this.logger.logSeparator(MESSAGES.headerCreateTopic);
      await this.createTopic();
      this.logger.logSeparator(MESSAGES.headerCreateQueues);
      await this.createQueues();
      this.logger.logSeparator(MESSAGES.headerAttachPolicy);
      await this.attachQueueIamPolicies();
      this.logger.logSeparator(MESSAGES.headerSubscribeQueues);
      await this.subscribeQueuesToTopic();
      this.logger.logSeparator(MESSAGES.headerPublishMessage);
      await this.publishMessages();
      this.logger.logSeparator(MESSAGES.headerReceiveMessages);
      await this.receiveAndDeleteMessages();
    } catch (err) {
      console.error(err);
    } finally {
      await this.destroyResources();
    }
  }
}
```

- 如需詳API細資訊，請參閱「AWS SDK for JavaScript API參考」中的下列主題。
  - CreateQueue
  - CreateTopic
  - DeleteMessageBatch
  - DeleteQueue
  - DeleteTopic
  - GetQueueAttributes
  - 發布
  - ReceiveMessage
  - SetQueueAttributes
  - Subscribe
  - Unsubscribe

無伺服器範例

從 Amazon SNS 觸發器調用 Lambda 函數

下列程式碼範例示範如何實作 Lambda 函數，該函數會接收由接收來自SNS主題的訊息而觸發的事件。函數會從事件參數擷取訊息，並記錄每一則訊息的內容。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在無伺服器範例儲存庫中設定和執行。

使 Lambda JavaScript. SNS

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
  for (const record of event.Records) {
    await processMessageAsync(record);
  }
  console.info("done");
};

async function processMessageAsync(record) {
  try {
    const message = JSON.stringify(record.Sns.Message);
    console.log(`Processed message ${message}`);
    await Promise.resolve(1); //Placeholder for actual async work
  } catch (err) {
    console.error("An error occurred");
    throw err;
  }
}
```

使 Lambda TypeScript. SNS

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
```

```
import { SNSEvent, Context, SNSHandler, SNSEventRecord } from "aws-lambda";

export const functionHandler: SNSHandler = async (
  event: SNSEvent,
  context: Context
): Promise<void> => {
  for (const record of event.Records) {
    await processMessageAsync(record);
  }
  console.info("done");
};

async function processMessageAsync(record: SNSEventRecord): Promise<any> {
  try {
    const message: string = JSON.stringify(record.Sns.Message);
    console.log(`Processed message ${message}`);
    await Promise.resolve(1); //Placeholder for actual async work
  } catch (err) {
    console.error("An error occurred");
    throw err;
  }
}
```

Amazon SQS 示例使用 SDK for JavaScript (v3)

下列程式碼範例說明如何透過使用 AWS SDK for JavaScript (v3) 搭配 Amazon 來執行動作和實作常見案例SQS。

Actions 是大型程式的程式碼摘錄，必須在內容中執行。雖然動作會顯示如何呼叫個別服務函數，但您可以在其相關案例中查看內容中的動作。

案例是程式碼範例，向您展示如何透過呼叫服務中的多個函式或與其他函式結合來完成特定工作 AWS 服務。

每個範例都包含完整原始程式碼的連結，您可以在其中找到如何在內容中設定和執行程式碼的指示。

開始使用

你好 Amazon SQS

下列程式碼範例說明如何開始使用 Amazon SQS。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

初始化 Amazon SQS 用戶端並列出佇列。

```
import { SQSClient, paginateListQueues } from "@aws-sdk/client-sqs";

export const helloSqs = async () => {
  // The configuration object (`{}`) is required. If the region and credentials
  // are omitted, the SDK uses your local configuration if it exists.
  const client = new SQSClient({});

  // You can also use `ListQueuesCommand`, but to use that command you must
  // handle the pagination yourself. You can do that by sending the
 `ListQueuesCommand`
  // with the `NextToken` parameter from the previous request.
  const paginatedQueues = paginateListQueues({ client }, {});
  const queues = [];

  for await (const page of paginatedQueues) {
    if (page.QueueUrls?.length) {
      queues.push(...page.QueueUrls);
    }
  }

  const suffix = queues.length === 1 ? "" : "s";

  console.log(
    `Hello, Amazon SQS! You have ${queues.length} queue${suffix} in your account.`,
  );
  console.log(queues.map((t) => `  * ${t}`).join("\n"));
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考ListQueues中的。

主題

- 動作
- 案例
- 無伺服器範例

動作

`ChangeMessageVisibility`

下列程式碼範例會示範如何使用ChangeMessageVisibility。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

接收 Amazon SQS 訊息並變更其逾時可見性。

```
import {
  ReceiveMessageCommand,
  ChangeMessageVisibilityCommand,
  SQSClient,
} from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "queue_url";

const receiveMessage = (queueUrl) =>
  client.send(
    new ReceiveMessageCommand({
      AttributeNames: ["SentTimestamp"],
      MaxNumberOfMessages: 1,
      MessageAttributeNames: ["All"],
      QueueUrl: queueUrl,
      WaitTimeSeconds: 1,
    }),
  );

export const main = async (queueUrl = SQS_QUEUE_URL) => {
```

```
  const { Messages } = await receiveMessage(queueUrl);

  const response = await client.send(
    new ChangeMessageVisibilityCommand({
      QueueUrl: queueUrl,
      ReceiptHandle: Messages[0].ReceiptHandle,
      VisibilityTimeout: 20,
    }),
  );
  console.log(response);
  return response;
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考ChangeMessageVisibility中的。

SDK對於 JavaScript（第 2 個）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

接收 Amazon SQS 訊息並變更其逾時可見性。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region to us-west-2
AWS.config.update({ region: "us-west-2" });

// Create the SQS service object
var sqs = new AWS.SQS({ apiVersion: "2012-11-05" });

var queueURL = "https://sqs.REGION.amazonaws.com/ACCOUNT-ID/QUEUE-NAME";

var params = {
  AttributeNames: ["SentTimestamp"],
  MaxNumberOfMessages: 1,
  MessageAttributeNames: ["All"],
  QueueUrl: queueURL,
};
```

```
sqs.receiveMessage(params, function (err, data) {
  if (err) {
    console.log("Receive Error", err);
  } else {
    // Make sure we have a message
    if (data.Messages != null) {
      var visibilityParams = {
        QueueUrl: queueURL,
        ReceiptHandle: data.Messages[0].ReceiptHandle,
        VisibilityTimeout: 20, // 20 second timeout
      };
      sqs.changeMessageVisibility(visibilityParams, function (err, data) {
        if (err) {
          console.log("Delete Error", err);
        } else {
          console.log("Timeout Changed", data);
        }
      });
    } else {
      console.log("No messages to change");
    }
  }
});
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考ChangeMessageVisibility中的。

`CreateQueue`

下列程式碼範例會示範如何使用`CreateQueue`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

創建一個 Amazon SQS 標準隊列。

```
import { CreateQueueCommand, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_NAME = "test-queue";

export const main = async (sqsQueueName = SQS_QUEUE_NAME) => {
  const command = new CreateQueueCommand({
    QueueName: sqsQueueName,
    Attributes: {
      DelaySeconds: "60",
      MessageRetentionPeriod: "86400",
    },
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

使用長輪詢創建一個 Amazon SQS 隊列。

```
import { CreateQueueCommand, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_NAME = "queue_name";

export const main = async (queueName = SQS_QUEUE_NAME) => {
  const response = await client.send(
    new CreateQueueCommand({
      QueueName: queueName,
      Attributes: {
        // When the wait time for the ReceiveMessage API action is greater than 0,
        // long polling is in effect. The maximum long polling wait time is 20
        // seconds. Long polling helps reduce the cost of using Amazon SQS by,
        // eliminating the number of empty responses and false empty responses.
        // https://docs.aws.amazon.com/AWSSimpleQueueService/latest/
SQSDeveloperGuide/sqs-short-and-long-polling.html
        ReceiveMessageWaitTimeSeconds: "20",
      },
    }),
  );
  console.log(response);
```

```
  return response;
};
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考CreateQueue中的。

SDK對於 JavaScript（第 2 個）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

創建一個 Amazon SQS 標準隊列。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create an SQS service object
var sqs = new AWS.SQS({ apiVersion: "2012-11-05" });

var params = {
  QueueName: "SQS_QUEUE_NAME",
  Attributes: {
    DelaySeconds: "60",
    MessageRetentionPeriod: "86400",
  },
};

sqs.createQueue(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.QueueUrl);
  }
});
```

創建一個等待消息到達的 Amazon SQS 隊列。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the SQS service object
var sqs = new AWS.SQS({ apiVersion: "2012-11-05" });

var params = {
  QueueName: "SQS_QUEUE_NAME",
  Attributes: {
    ReceiveMessageWaitTimeSeconds: "20",
  },
};

sqs.createQueue(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.QueueUrl);
  }
});
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考CreateQueue中的。

`DeleteMessage`

下列程式碼範例會示範如何使用`DeleteMessage`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

接收和刪除 Amazon SQS 消息。

```
import {
  ReceiveMessageCommand,
  DeleteMessageCommand,
  SQSClient,
  DeleteMessageBatchCommand,
} from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "queue_url";

const receiveMessage = (queueUrl) =>
  client.send(
    new ReceiveMessageCommand({
      AttributeNames: ["SentTimestamp"],
      MaxNumberOfMessages: 10,
      MessageAttributeNames: ["All"],
      QueueUrl: queueUrl,
      WaitTimeSeconds: 20,
      VisibilityTimeout: 20,
    }),
  );

export const main = async (queueUrl = SQS_QUEUE_URL) => {
  const { Messages } = await receiveMessage(queueUrl);

  if (!Messages) {
    return;
  }

  if (Messages.length === 1) {
    console.log(Messages[0].Body);
    await client.send(
      new DeleteMessageCommand({
        QueueUrl: queueUrl,
        ReceiptHandle: Messages[0].ReceiptHandle,
      }),
    );
  } else {
    await client.send(
      new DeleteMessageBatchCommand({
        QueueUrl: queueUrl,
        Entries: Messages.map((message) => ({
```

```
          Id: message.MessageId,
          ReceiptHandle: message.ReceiptHandle,
        })),
      }),
    );
  }
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考DeleteMessage中的。

SDK對於 JavaScript（第 2 個）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

接收和刪除 Amazon SQS 消息。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create an SQS service object
var sqs = new AWS.SQS({ apiVersion: "2012-11-05" });

var queueURL = "SQS_QUEUE_URL";

var params = {
  AttributeNames: ["SentTimestamp"],
  MaxNumberOfMessages: 10,
  MessageAttributeNames: ["All"],
  QueueUrl: queueURL,
  VisibilityTimeout: 20,
  WaitTimeSeconds: 0,
};

sqs.receiveMessage(params, function (err, data) {
  if (err) {
    console.log("Receive Error", err);
```

```
  } else if (data.Messages) {
    var deleteParams = {
      QueueUrl: queueURL,
      ReceiptHandle: data.Messages[0].ReceiptHandle,
    };
    sqs.deleteMessage(deleteParams, function (err, data) {
      if (err) {
        console.log("Delete Error", err);
      } else {
        console.log("Message Deleted", data);
      }
    });
  }
});
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考DeleteMessage中的。

`DeleteMessageBatch`

下列程式碼範例會示範如何使用`DeleteMessageBatch`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import {
  ReceiveMessageCommand,
  DeleteMessageCommand,
  SQSClient,
  DeleteMessageBatchCommand,
} from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "queue_url";
```

```
const receiveMessage = (queueUrl) =>
  client.send(
    new ReceiveMessageCommand({
      AttributeNames: ["SentTimestamp"],
      MaxNumberOfMessages: 10,
      MessageAttributeNames: ["All"],
      QueueUrl: queueUrl,
      WaitTimeSeconds: 20,
      VisibilityTimeout: 20,
    }),
  );

export const main = async (queueUrl = SQS_QUEUE_URL) => {
  const { Messages } = await receiveMessage(queueUrl);

  if (!Messages) {
    return;
  }

  if (Messages.length === 1) {
    console.log(Messages[0].Body);
    await client.send(
      new DeleteMessageCommand({
        QueueUrl: queueUrl,
        ReceiptHandle: Messages[0].ReceiptHandle,
      }),
    );
  } else {
    await client.send(
      new DeleteMessageBatchCommand({
        QueueUrl: queueUrl,
        Entries: Messages.map((message) => ({
          Id: message.MessageId,
          ReceiptHandle: message.ReceiptHandle,
        })),
      }),
    );
  }
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考DeleteMessageBatch中的。

`DeleteQueue`

下列程式碼範例會示範如何使用`DeleteQueue`。

SDK對於 JavaScript （ 3 ）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

刪除 Amazon SQS 隊列。

```
import { DeleteQueueCommand, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "test-queue-url";

export const main = async (queueUrl = SQS_QUEUE_URL) => {
  const command = new DeleteQueueCommand({ QueueUrl: queueUrl });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考DeleteQueue中的。

SDK對於 JavaScript （ 第 2 個 ）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

刪除 Amazon SQS 隊列。

```
// Load the AWS SDK for Node.js
```

```
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create an SQS service object
var sqs = new AWS.SQS({ apiVersion: "2012-11-05" });

var params = {
  QueueUrl: "SQS_QUEUE_URL",
};

sqs.deleteQueue(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考DeleteQueue中的。

`GetQueueAttributes`

下列程式碼範例會示範如何使用`GetQueueAttributes`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import { GetQueueAttributesCommand, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "queue-url";

export const getQueueAttributes = async (queueUrl = SQS_QUEUE_URL) => {
```

```
  const command = new GetQueueAttributesCommand({
    QueueUrl: queueUrl,
    AttributeNames: ["DelaySeconds"],
  });

  const response = await client.send(command);
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '747a1192-c334-5682-a508-4cd5e8dc4e79',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   Attributes: { DelaySeconds: '1' }
  // }
  return response;
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考GetQueueAttributes中的。

`GetQueueUrl`

下列程式碼範例會示範如何使用`GetQueueUrl`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

獲URL取 Amazon SQS 隊列。

```
import { GetQueueUrlCommand, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_NAME = "test-queue";
```

```
export const main = async (queueName = SQS_QUEUE_NAME) => {
  const command = new GetQueueUrlCommand({ QueueName: queueName });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考GetQueueUrl中的。

SDK對於 JavaScript（第 2 個）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

獲URL取 Amazon SQS 隊列。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create an SQS service object
var sqs = new AWS.SQS({ apiVersion: "2012-11-05" });

var params = {
  QueueName: "SQS_QUEUE_NAME",
};

sqs.getQueueUrl(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.QueueUrl);
  }
});
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考GetQueueUrl中的。

`ListQueues`

下列程式碼範例會示範如何使用`ListQueues`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

列出您的 Amazon SQS 隊列。

```
import { paginateListQueues, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});

export const main = async () => {
  const paginatedListQueues = paginateListQueues({ client }, {});

  /** @type {string[]} */
  const urls = [];
  for await (const page of paginatedListQueues) {
    const nextUrls = page.QueueUrls?.filter((qurl) => !!qurl) || [];
    urls.push(...nextUrls);
    urls.forEach((url) => console.log(url));
  }

  return urls;
};
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考ListQueues中的。

SDK對於 JavaScript （ 第 2 個 ）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

列出您的 Amazon SQS 隊列。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create an SQS service object
var sqs = new AWS.SQS({ apiVersion: "2012-11-05" });

var params = {};

sqs.listQueues(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.QueueUrls);
  }
});
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考ListQueues中的。

`ReceiveMessage`

下列程式碼範例會示範如何使用`ReceiveMessage`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

從 Amazon SQS 隊列接收消息。

```
import {
  ReceiveMessageCommand,
  DeleteMessageCommand,
  SQSClient,
  DeleteMessageBatchCommand,
} from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "queue_url";

const receiveMessage = (queueUrl) =>
  client.send(
    new ReceiveMessageCommand({
      AttributeNames: ["SentTimestamp"],
      MaxNumberOfMessages: 10,
      MessageAttributeNames: ["All"],
      QueueUrl: queueUrl,
      WaitTimeSeconds: 20,
      VisibilityTimeout: 20,
    }),
  );

export const main = async (queueUrl = SQS_QUEUE_URL) => {
  const { Messages } = await receiveMessage(queueUrl);

  if (!Messages) {
    return;
  }

  if (Messages.length === 1) {
    console.log(Messages[0].Body);
    await client.send(
      new DeleteMessageCommand({
```

```
        QueueUrl: queueUrl,
        ReceiptHandle: Messages[0].ReceiptHandle,
      }),
    );
  } else {
    await client.send(
      new DeleteMessageBatchCommand({
        QueueUrl: queueUrl,
        Entries: Messages.map((message) => ({
          Id: message.MessageId,
          ReceiptHandle: message.ReceiptHandle,
        })),
      }),
    );
  }
};
```

使用長輪詢支援從 Amazon SQS 佇列接收訊息。

```
import { ReceiveMessageCommand, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "queue-url";

export const main = async (queueUrl = SQS_QUEUE_URL) => {
  const command = new ReceiveMessageCommand({
    AttributeNames: ["SentTimestamp"],
    MaxNumberOfMessages: 1,
    MessageAttributeNames: ["All"],
    QueueUrl: queueUrl,
    // The duration (in seconds) for which the call waits for a message
    // to arrive in the queue before returning. If a message is available,
    // the call returns sooner than WaitTimeSeconds. If no messages are
    // available and the wait time expires, the call returns successfully
    // with an empty list of messages.
    // https://docs.aws.amazon.com/AWSSimpleQueueService/latest/APIReference/
API_ReceiveMessage.html#API_ReceiveMessage_RequestSyntax
    WaitTimeSeconds: 20,
  });

  const response = await client.send(command);
  console.log(response);
```

```
  return response;
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考ReceiveMessage中的。

SDK對於 JavaScript（第 2 個）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

使用長輪詢支援從 Amazon SQS 佇列接收訊息。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the SQS service object
var sqs = new AWS.SQS({ apiVersion: "2012-11-05" });

var queueURL = "SQS_QUEUE_URL";

var params = {
  AttributeNames: ["SentTimestamp"],
  MaxNumberOfMessages: 1,
  MessageAttributeNames: ["All"],
  QueueUrl: queueURL,
  WaitTimeSeconds: 20,
};

sqs.receiveMessage(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考ReceiveMessage中的。

`SendMessage`

下列程式碼範例會示範如何使用`SendMessage`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

將消息發送到 Amazon SQS 隊列。

```
import { SendMessageCommand, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "queue_url";

export const main = async (sqsQueueUrl = SQS_QUEUE_URL) => {
  const command = new SendMessageCommand({
    QueueUrl: sqsQueueUrl,
    DelaySeconds: 10,
    MessageAttributes: {
      Title: {
        DataType: "String",
        StringValue: "The Whistler",
      },
      Author: {
        DataType: "String",
        StringValue: "John Grisham",
      },
      WeeksOn: {
        DataType: "Number",
        StringValue: "6",
      },
    },
    MessageBody:
```

```
      "Information about current NY Times fiction bestseller for week of
 12/11/2016.",
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考SendMessage中的。

SDK對於 JavaScript （ 第 2 個 ）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

將消息發送到 Amazon SQS 隊列。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create an SQS service object
var sqs = new AWS.SQS({ apiVersion: "2012-11-05" });

var params = {
  // Remove DelaySeconds parameter and value for FIFO queues
  DelaySeconds: 10,
  MessageAttributes: {
    Title: {
      DataType: "String",
      StringValue: "The Whistler",
    },
    Author: {
      DataType: "String",
      StringValue: "John Grisham",
    },
```

```
    WeeksOn: {
      DataType: "Number",
      StringValue: "6",
    },
  },
  MessageBody:
    "Information about current NY Times fiction bestseller for week of 12/11/2016.",
  // MessageDeduplicationId: "TheWhistler",  // Required for FIFO queues
  // MessageGroupId: "Group1",  // Required for FIFO queues
  QueueUrl: "SQS_QUEUE_URL",
};

sqs.sendMessage(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.MessageId);
  }
});
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考SendMessage中的。

`SetQueueAttributes`

下列程式碼範例會示範如何使用`SetQueueAttributes`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import { SetQueueAttributesCommand, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "queue-url";
```

```
export const main = async (queueUrl = SQS_QUEUE_URL) => {
  const command = new SetQueueAttributesCommand({
    QueueUrl: queueUrl,
    Attributes: {
      DelaySeconds: "1",
    },
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

設定 Amazon SQS 佇列以使用長輪詢。

```
import { SetQueueAttributesCommand, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "queue_url";

export const main = async (queueUrl = SQS_QUEUE_URL) => {
  const command = new SetQueueAttributesCommand({
    Attributes: {
      ReceiveMessageWaitTimeSeconds: "20",
    },
    QueueUrl: queueUrl,
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

設定無效信件佇列。

```
import { SetQueueAttributesCommand, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "queue_url";
const DEAD_LETTER_QUEUE_ARN = "dead_letter_queue_arn";
```

```
export const main = async (
  queueUrl = SQS_QUEUE_URL,
  deadLetterQueueArn = DEAD_LETTER_QUEUE_ARN,
) => {
  const command = new SetQueueAttributesCommand({
    Attributes: {
      RedrivePolicy: JSON.stringify({
        // Amazon SQS supports dead-letter queues (DLQ), which other
        // queues (source queues) can target for messages that can't
        // be processed (consumed) successfully.
        // https://docs.aws.amazon.com/AWSSimpleQueueService/latest/
SQSDeveloperGuide/sqs-dead-letter-queues.html
        deadLetterTargetArn: deadLetterQueueArn,
        maxReceiveCount: "10",
      }),
    },
    QueueUrl: queueUrl,
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考SetQueueAttributes中的。

案例

建立 Amazon Textract Explorer 應用程式

下列程式碼範例顯示如何透過互動式應用程式探索 Amazon Textract 輸出。

SDK對於 JavaScript（3）

示範如何使用建置 React 應用程式，該應用程式使用 Amazon Textract 擷取文件影像中的資料，並將其顯示在互動式網頁中。 AWS SDK for JavaScript 此範例會在 Web 瀏覽器中執行，且登入資料需要經過驗證的 Amazon Cognito 身分。它使用 Amazon 簡單存儲服務（Amazon S3）進行存儲，並對於通知，它會輪詢訂閱 Amazon 簡單通知服務（AmazonSQS）主題的 Amazon 簡單隊列（亞馬遜SNS）隊列。

有關如何設置和運行的完整源代碼和說明，請參閱中的完整示例GitHub。

此範例中使用的服務

- Amazon Cognito Identity
- Amazon S3
- Amazon SNS
- Amazon SQS
- Amazon Textract

將訊息發佈至佇列

以下程式碼範例顯示做法：

- 創建主題（FIFO或非FIFO）。
- 為主題訂閱多個佇列，並提供套用篩選條件的選擇。
- 發佈訊息至主題。
- 輪詢佇列以獲取收到的訊息。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

這是此工作流程的進入點。

```
import { SNSClient } from "@aws-sdk/client-sns";
import { SQSClient } from "@aws-sdk/client-sqs";

import { TopicsQueuesWkflw } from "./TopicsQueuesWkflw.js";
import { Prompter } from "@aws-doc-sdk-examples/lib/prompter.js";
import { SlowLogger } from "@aws-doc-sdk-examples/lib/slow-logger.js";

export const startSnsWorkflow = () => {
  const noLoggerDelay = process.argv.find((arg) => arg === "--no-logger-delay");
  const snsClient = new SNSClient({});
  const sqsClient = new SQSClient({});
```

```
  const prompter = new Prompter();
  const logger = noLoggerDelay ? console : new SlowLogger(25);

  const wkflw = new TopicsQueuesWkflw(snsClient, sqsClient, prompter, logger);

  wkflw.start();
};
```

先前的程式碼提供了必要的相依性並啟動工作流程。下一節包含範例的大量內容。

```
const toneChoices = [
  { name: "cheerful", value: "cheerful" },
  { name: "funny", value: "funny" },
  { name: "serious", value: "serious" },
  { name: "sincere", value: "sincere" },
];

export class TopicsQueuesWkflw {
  // SNS topic is configured as First-In-First-Out
  isFifo = true;

  // Automatic content-based deduplication is enabled.
  autoDedup = false;

  snsClient;
  sqsClient;
  topicName;
  topicArn;
  subscriptionArns = [];
  /**
   * @type {{ queueName: string, queueArn: string, queueUrl: string, policy?:
 string }[]}
   */
  queues = [];
  prompter;

  /**
   * @param {import('@aws-sdk/client-sns').SNSClient} snsClient
   * @param {import('@aws-sdk/client-sqs').SQSClient} sqsClient
   * @param {import('../../libs/prompter.js').Prompter} prompter
```

```
   * @param {import('../../libs/logger.js').Logger} logger
   */
  constructor(snsClient, sqsClient, prompter, logger) {
    this.snsClient = snsClient;
    this.sqsClient = sqsClient;
    this.prompter = prompter;
    this.logger = logger;
  }

  async welcome() {
    await this.logger.log(MESSAGES.description);
  }

  async confirmFifo() {
    await this.logger.log(MESSAGES.snsFifoDescription);
    this.isFifo = await this.prompter.confirm({
      message: MESSAGES.snsFifoPrompt,
    });

    if (this.isFifo) {
      this.logger.logSeparator(MESSAGES.headerDedup);
      await this.logger.log(MESSAGES.deduplicationNotice);
      await this.logger.log(MESSAGES.deduplicationDescription);
      this.autoDedup = await this.prompter.confirm({
        message: MESSAGES.deduplicationPrompt,
      });
    }
  }

  async createTopic() {
    await this.logger.log(MESSAGES.creatingTopics);
    this.topicName = await this.prompter.input({
      message: MESSAGES.topicNamePrompt,
    });
    if (this.isFifo) {
      this.topicName += ".fifo";
      this.logger.logSeparator(MESSAGES.headerFifoNaming);
      await this.logger.log(MESSAGES.appendFifoNotice);
    }

    const response = await this.snsClient.send(
      new CreateTopicCommand({
        Name: this.topicName,
        Attributes: {
```

```
          FifoTopic: this.isFifo ? "true" : "false",
          ...(this.autoDedup ? { ContentBasedDeduplication: "true" } : {}),
        },
      }),
    );

    this.topicArn = response.TopicArn;

    await this.logger.log(
      MESSAGES.topicCreatedNotice
        .replace("${TOPIC_NAME}", this.topicName)
        .replace("${TOPIC_ARN}", this.topicArn),
    );
  }

  async createQueues() {
    await this.logger.log(MESSAGES.createQueuesNotice);
    // Increase this number to add more queues.
    let maxQueues = 2;

    for (let i = 0; i < maxQueues; i++) {
      await this.logger.log(MESSAGES.queueCount.replace("${COUNT}", i + 1));
      let queueName = await this.prompter.input({
        message: MESSAGES.queueNamePrompt.replace(
          "${EXAMPLE_NAME}",
          i === 0 ? "good-news" : "bad-news",
        ),
      });

      if (this.isFifo) {
        queueName += ".fifo";
        await this.logger.log(MESSAGES.appendFifoNotice);
      }

      const response = await this.sqsClient.send(
        new CreateQueueCommand({
          QueueName: queueName,
          Attributes: { ...(this.isFifo ? { FifoQueue: "true" } : {}) },
        }),
      );

      const { Attributes } = await this.sqsClient.send(
        new GetQueueAttributesCommand({
          QueueUrl: response.QueueUrl,
```

```
          AttributeNames: ["QueueArn"],
        }),
      );

      this.queues.push({
        queueName,
        queueArn: Attributes.QueueArn,
        queueUrl: response.QueueUrl,
      });

      await this.logger.log(
        MESSAGES.queueCreatedNotice
          .replace("${QUEUE_NAME}", queueName)
          .replace("${QUEUE_URL}", response.QueueUrl)
          .replace("${QUEUE_ARN}", Attributes.QueueArn),
      );
    }
  }

  async attachQueueIamPolicies() {
    for (const [index, queue] of this.queues.entries()) {
      const policy = JSON.stringify(
        {
          Statement: [
            {
              Effect: "Allow",
              Principal: {
                Service: "sns.amazonaws.com",
              },
              Action: "sqs:SendMessage",
              Resource: queue.queueArn,
              Condition: {
                ArnEquals: {
                  "aws:SourceArn": this.topicArn,
                },
              },
            },
          ],
        },
        null,
        2,
      );

      if (index !== 0) {
```

```
        this.logger.logSeparator();
      }

      await this.logger.log(MESSAGES.attachPolicyNotice);
      console.log(policy);
      const addPolicy = await this.prompter.confirm({
        message: MESSAGES.addPolicyConfirmation.replace(
          "${QUEUE_NAME}",
          queue.queueName,
        ),
      });

      if (addPolicy) {
        await this.sqsClient.send(
          new SetQueueAttributesCommand({
            QueueUrl: queue.queueUrl,
            Attributes: {
              Policy: policy,
            },
          }),
        );
        queue.policy = policy;
      } else {
        await this.logger.log(
          MESSAGES.policyNotAttachedNotice.replace(
            "${QUEUE_NAME}",
            queue.queueName,
          ),
        );
      }
    }
  }

  async subscribeQueuesToTopic() {
    for (const [index, queue] of this.queues.entries()) {
      /**
       * @type {import('@aws-sdk/client-sns').SubscribeCommandInput}
       */
      const subscribeParams = {
        TopicArn: this.topicArn,
        Protocol: "sqs",
        Endpoint: queue.queueArn,
      };
      let tones = [];
```

```
    if (this.isFifo) {
      if (index === 0) {
        await this.logger.log(MESSAGES.fifoFilterNotice);
      }
      tones = await this.prompter.checkbox({
        message: MESSAGES.fifoFilterSelect.replace(
          "${QUEUE_NAME}",
          queue.queueName,
        ),
        choices: toneChoices,
      });

      if (tones.length) {
        subscribeParams.Attributes = {
          FilterPolicyScope: "MessageAttributes",
          FilterPolicy: JSON.stringify({
            tone: tones,
          }),
        };
      }
    }

    const { SubscriptionArn } = await this.snsClient.send(
      new SubscribeCommand(subscribeParams),
    );

    this.subscriptionArns.push(SubscriptionArn);

    await this.logger.log(
      MESSAGES.queueSubscribedNotice
        .replace("${QUEUE_NAME}", queue.queueName)
        .replace("${TOPIC_NAME}", this.topicName)
        .replace("${TONES}", tones.length ? tones.join(", ") : "none"),
    );
  }
}

async publishMessages() {
  const message = await this.prompter.input({
    message: MESSAGES.publishMessagePrompt,
  });

  let groupId, deduplicationId, choices;
```

```
    if (this.isFifo) {
      await this.logger.log(MESSAGES.groupIdNotice);
      groupId = await this.prompter.input({
        message: MESSAGES.groupIdPrompt,
      });

      if (this.autoDedup === false) {
        await this.logger.log(MESSAGES.deduplicationIdNotice);
        deduplicationId = await this.prompter.input({
          message: MESSAGES.deduplicationIdPrompt,
        });
      }

      choices = await this.prompter.checkbox({
        message: MESSAGES.messageAttributesPrompt,
        choices: toneChoices,
      });
    }

    await this.snsClient.send(
      new PublishCommand({
        TopicArn: this.topicArn,
        Message: message,
        ...(groupId
          ? {
              MessageGroupId: groupId,
            }
          : {}),
        ...(deduplicationId
          ? {
              MessageDeduplicationId: deduplicationId,
            }
          : {}),
        ...(choices
          ? {
              MessageAttributes: {
                tone: {
                  DataType: "String.Array",
                  StringValue: JSON.stringify(choices),
                },
              },
            }
          : {}),
```

```
      }),
    );

    const publishAnother = await this.prompter.confirm({
      message: MESSAGES.publishAnother,
    });

    if (publishAnother) {
      await this.publishMessages();
    }
  }

  async receiveAndDeleteMessages() {
    for (const queue of this.queues) {
      const { Messages } = await this.sqsClient.send(
        new ReceiveMessageCommand({
          QueueUrl: queue.queueUrl,
        }),
      );

      if (Messages) {
        await this.logger.log(
          MESSAGES.messagesReceivedNotice.replace(
            "${QUEUE_NAME}",
            queue.queueName,
          ),
        );
        console.log(Messages);

        await this.sqsClient.send(
          new DeleteMessageBatchCommand({
            QueueUrl: queue.queueUrl,
            Entries: Messages.map((message) => ({
              Id: message.MessageId,
              ReceiptHandle: message.ReceiptHandle,
            })),
          }),
        );
      } else {
        await this.logger.log(
          MESSAGES.noMessagesReceivedNotice.replace(
            "${QUEUE_NAME}",
            queue.queueName,
          ),
```

```
        );
      }
    }

    const deleteAndPoll = await this.prompter.confirm({
      message: MESSAGES.deleteAndPollConfirmation,
    });

    if (deleteAndPoll) {
      await this.receiveAndDeleteMessages();
    }
  }

  async destroyResources() {
    for (const subscriptionArn of this.subscriptionArns) {
      await this.snsClient.send(
        new UnsubscribeCommand({ SubscriptionArn: subscriptionArn }),
      );
    }

    for (const queue of this.queues) {
      await this.sqsClient.send(
        new DeleteQueueCommand({ QueueUrl: queue.queueUrl }),
      );
    }

    if (this.topicArn) {
      await this.snsClient.send(
        new DeleteTopicCommand({ TopicArn: this.topicArn }),
      );
    }
  }

  async start() {
    console.clear();

    try {
      this.logger.logSeparator(MESSAGES.headerWelcome);
      await this.welcome();
      this.logger.logSeparator(MESSAGES.headerFifo);
      await this.confirmFifo();
      this.logger.logSeparator(MESSAGES.headerCreateTopic);
      await this.createTopic();
      this.logger.logSeparator(MESSAGES.headerCreateQueues);
```

```
      await this.createQueues();
      this.logger.logSeparator(MESSAGES.headerAttachPolicy);
      await this.attachQueueIamPolicies();
      this.logger.logSeparator(MESSAGES.headerSubscribeQueues);
      await this.subscribeQueuesToTopic();
      this.logger.logSeparator(MESSAGES.headerPublishMessage);
      await this.publishMessages();
      this.logger.logSeparator(MESSAGES.headerReceiveMessages);
      await this.receiveAndDeleteMessages();
    } catch (err) {
      console.error(err);
    } finally {
      await this.destroyResources();
    }
  }
}
```

- 如需詳API細資訊，請參閱「AWS SDK for JavaScript API參考」中的下列主題。
  - CreateQueue
  - CreateTopic
  - DeleteMessageBatch
  - DeleteQueue
  - DeleteTopic
  - GetQueueAttributes
  - 發布
  - ReceiveMessage
  - SetQueueAttributes
  - Subscribe
  - Unsubscribe

無伺服器範例

從 Amazon SQS 觸發器調用 Lambda 函數

下列程式碼範例示範如何實作 Lambda 函數，該函數會接收透過從SQS佇列接收訊息觸發的事件。函數會從事件參數擷取訊息，並記錄每一則訊息的內容。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在無伺服器範例儲存庫中設定和執行。

使 Lambda JavaScript. SQS

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
  for (const message of event.Records) {
    await processMessageAsync(message);
  }
  console.info("done");
};

async function processMessageAsync(message) {
  try {
    console.log(`Processed message ${message.body}`);
    // TODO: Do interesting work based on the new message
    await Promise.resolve(1); //Placeholder for actual async work
  } catch (err) {
    console.error("An error occurred");
    throw err;
  }
}
```

使 Lambda TypeScript. SQS

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { SQSEvent, Context, SQSHandler, SQSRecord } from "aws-lambda";

export const functionHandler: SQSHandler = async (
  event: SQSEvent,
  context: Context
): Promise<void> => {
  for (const message of event.Records) {
    await processMessageAsync(message);
```

```
  }
  console.info("done");
};

async function processMessageAsync(message: SQSRecord): Promise<any> {
  try {
    console.log(`Processed message ${message.body}`);
    // TODO: Do interesting work based on the new message
    await Promise.resolve(1); //Placeholder for actual async work
  } catch (err) {
    console.error("An error occurred");
    throw err;
  }
}
```

使用 Amazon SQS 觸發器報告 Lambda 函數的批次項目失敗

下列程式碼範例示範如何針對接收SQS佇列事件的 Lambda 函數實作部分批次回應。此函數會在回應中報告批次項目失敗，指示 Lambda 稍後重試這些訊息。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在無伺服器範例儲存庫中設定和執行。

使用 Lambda 使用 JavaScript. SQS

```
// Node.js 20.x Lambda runtime, AWS SDK for Javascript V3
export const handler = async (event, context) => {
    const batchItemFailures = [];
    for (const record of event.Records) {
        try {
            await processMessageAsync(record, context);
        } catch (error) {
            batchItemFailures.push({ itemIdentifier: record.messageId });
        }
    }
    return { batchItemFailures };
```

```
};

async function processMessageAsync(record, context) {
    if (record.body && record.body.includes("error")) {
        throw new Error("There is an error in the SQS Message.");
    }
    console.log(`Processed message: ${record.body}`);
}
```

使用 Lambda 使用 TypeScript. SQS

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { SQSEvent, SQSBatchResponse, Context, SQSBatchItemFailure, SQSRecord } from
 'aws-lambda';

export const handler = async (event: SQSEvent, context: Context):
 Promise<SQSBatchResponse> => {
    const batchItemFailures: SQSBatchItemFailure[] = [];

    for (const record of event.Records) {
        try {
            await processMessageAsync(record);
        } catch (error) {
            batchItemFailures.push({ itemIdentifier: record.messageId });
        }
    }

    return {batchItemFailures: batchItemFailures};
};

async function processMessageAsync(record: SQSRecord): Promise<void> {
    if (record.body && record.body.includes("error")) {
        throw new Error('There is an error in the SQS Message.');
    }
    console.log(`Processed message ${record.body}`);
}
```

Step Functions 示例使用 SDK for JavaScript (v3)

下列程式碼範例會示範如何使用 AWS SDK for JavaScript (v3) 搭配 Step Functions 來執行動作及實作常見案例。

Actions 是大型程式的程式碼摘錄，必須在內容中執行。雖然動作會顯示如何呼叫個別服務函數，但您可以在其相關案例中查看內容中的動作。

每個範例都包含完整原始程式碼的連結，您可以在其中找到如何在內容中設定和執行程式碼的指示。

主題

動作

`StartExecution`

下列程式碼範例會示範如何使用`StartExecution`。

SDK對於 JavaScript （ 3 ）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import { SFNClient, StartExecutionCommand } from "@aws-sdk/client-sfn";

/**
 * @param {{ sfnClient: SFNClient, stateMachineArn: string }} config
 */
export async function startExecution({ sfnClient, stateMachineArn }) {
  const response = await sfnClient.send(
    new StartExecutionCommand({
      stateMachineArn,
    }),
  );
```

```
  console.log(response);
  // Example response:
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '202a9309-c16a-454b-adeb-c4d19afe3bf2',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   executionArn: 'arn:aws:states:us-
east-1:000000000000:execution:MyStateMachine:aaaaaaaa-f787-49fb-a20c-1b61c64eafe6',
  //   startDate: 2024-01-04T15:54:08.362Z
  // }
  return response;
}

// Call function if run directly
import { fileURLToPath } from "url";
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  startExecution({ sfnClient: new SFNClient({}), stateMachineArn: "ARN" });
}
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考StartExecution中的。

AWS STS 使用 JavaScript （ v3 ）SDK的示例

下列程式碼範例會示範如何使用 AWS SDK for JavaScript (v3) 搭配使用來執行動作及實作常見案例 AWS STS。

Actions 是大型程式的程式碼摘錄，必須在內容中執行。雖然動作會顯示如何呼叫個別服務函數，但您可以在其相關案例中查看內容中的動作。

每個範例都包含完整原始程式碼的連結，您可以在其中找到如何在內容中設定和執行程式碼的指示。

主題

- 動作

動作

`AssumeRole`

下列程式碼範例會示範如何使用AssumeRole。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

建立用戶端。

```
import { STSClient } from "@aws-sdk/client-sts";
// Set the AWS Region.
const REGION = "us-east-1";
// Create an AWS STS service client object.
export const client = new STSClient({ region: REGION });
```

假設IAM角色。

```
import { AssumeRoleCommand } from "@aws-sdk/client-sts";

import { client } from "../libs/client.js";

export const main = async () => {
  try {
    // Returns a set of temporary security credentials that you can use to
    // access Amazon Web Services resources that you might not normally
    // have access to.
    const command = new AssumeRoleCommand({
      // The Amazon Resource Name (ARN) of the role to assume.
      RoleArn: "ROLE_ARN",
      // An identifier for the assumed role session.
      RoleSessionName: "session1",
      // The duration, in seconds, of the role session. The value specified
      // can range from 900 seconds (15 minutes) up to the maximum session
```

```
      // duration set for the role.
      DurationSeconds: 900,
    });
    const response = await client.send(command);
    console.log(response);
  } catch (err) {
    console.error(err);
  }
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考AssumeRole中的。

SDK對於 JavaScript （ 第 2 個 ）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
// Load the AWS SDK for Node.js
const AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

var roleToAssume = {
  RoleArn: "arn:aws:iam::123456789012:role/RoleName",
  RoleSessionName: "session1",
  DurationSeconds: 900,
};
var roleCreds;

// Create the STS service object
var sts = new AWS.STS({ apiVersion: "2011-06-15" });

//Assume Role
sts.assumeRole(roleToAssume, function (err, data) {
  if (err) console.log(err, err.stack);
  else {
    roleCreds = {
      accessKeyId: data.Credentials.AccessKeyId,
      secretAccessKey: data.Credentials.SecretAccessKey,
```

```
      sessionToken: data.Credentials.SessionToken,
    };
    stsGetCallerIdentity(roleCreds);
  }
});

//Get Arn of current identity
function stsGetCallerIdentity(creds) {
  var stsParams = { credentials: creds };
  // Create STS service object
  var sts = new AWS.STS(stsParams);

  sts.getCallerIdentity({}, function (err, data) {
    if (err) {
      console.log(err, err.stack);
    } else {
      console.log(data.Arn);
    }
  });
}
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考AssumeRole中的。

AWS Support 使用 JavaScript （ v3 ）SDK的示例

下列程式碼範例會示範如何使用 AWS SDK for JavaScript (v3) 搭配使用來執行動作及實作常見案例 AWS Support。

基本概念是程式碼範例，會示範如何在服務中執行基本作業。

Actions 是大型程式的程式碼摘錄，必須在內容中執行。雖然動作會顯示如何呼叫個別服務函數，但您可以在其相關案例中查看內容中的動作。

每個範例都包含完整原始程式碼的連結，您可以在其中找到如何在內容中設定和執行程式碼的指示。

開始使用

你好 AWS Support

下列程式碼範例示範如何開始使用 AWS Support。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

呼叫 `main()` 來執行這個範例。

```
import {
  DescribeServicesCommand,
  SupportClient,
} from "@aws-sdk/client-support";

// Change the value of 'region' to your preferred AWS Region.
const client = new SupportClient({ region: "us-east-1" });

const getServiceCount = async () => {
  try {
    const { services } = await client.send(new DescribeServicesCommand({}));
    return services.length;
  } catch (err) {
    if (err.name === "SubscriptionRequiredException") {
      throw new Error(
        "You must be subscribed to the AWS Support plan to use this feature.",
      );
    } else {
      throw err;
    }
  }
};

export const main = async () => {
  try {
    const count = await getServiceCount();
    console.log(`Hello, AWS Support! There are ${count} services available.`);
  } catch (err) {
    console.error("Failed to get service count: ", err.message);
  }
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考DescribeServices中的。

主題

基本概念

學習基礎知識

以下程式碼範例顯示做法：

- 取得並顯示案例可用的服務和嚴重性層級。
- 根據選取的服務、類別和嚴重性層級建立支援案例。
- 取得並顯示當天開啟的案例清單。
- 將附件集和通訊新增至新案例。
- 描述案例的新附件和通訊。
- 解決案例。
- 取得並顯示當天已解決的案例清單。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

在終端中執行互動式案例。

```
import {
  AddAttachmentsToSetCommand,
  AddCommunicationToCaseCommand,
  CreateCaseCommand,
  DescribeAttachmentCommand,
  DescribeCasesCommand,
  DescribeCommunicationsCommand,
```

```
  DescribeServicesCommand,
  DescribeSeverityLevelsCommand,
  ResolveCaseCommand,
  SupportClient,
} from "@aws-sdk/client-support";
import * as inquirer from "@inquirer/prompts";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

const wrapText = (text, char = "=") => {
  const rule = char.repeat(80);
  return `${rule}\n    ${text}\n${rule}\n`;
};

const client = new SupportClient({ region: "us-east-1" });

// Verify that the account has a Support plan.
export const verifyAccount = async () => {
  const command = new DescribeServicesCommand({});

  try {
    await client.send(command);
  } catch (err) {
    if (err.name === "SubscriptionRequiredException") {
      throw new Error(
        "You must be subscribed to the AWS Support plan to use this feature.",
      );
    } else {
      throw err;
    }
  }
};

/**
 * Select a service from the list returned from DescribeServices.
 */
export const getService = async () => {
  const { services } = await client.send(new DescribeServicesCommand({}));
  const selectedService = await inquirer.select({
    message:
      "Select a service. Your support case will be created for this service. The
 list of services is truncated for readability.",
    choices: services.slice(0, 10).map((s) => ({ name: s.name, value: s })),
  });
  return selectedService;
```

```
};

/**
 * @param {{ categories: import('@aws-sdk/client-support').Category[]}} service
 */
export const getCategory = async (service) => {
  const selectedCategory = await inquirer.select({
    message: "Select a category.",
    choices: service.categories.map((c) => ({ name: c.name, value: c })),
  });
  return selectedCategory;
};

// Get the available severity levels for the account.
export const getSeverityLevel = async () => {
  const command = new DescribeSeverityLevelsCommand({});
  const { severityLevels } = await client.send(command);
  const selectedSeverityLevel = await inquirer.select({
    message: "Select a severity level.",
    choices: severityLevels.map((s) => ({ name: s.name, value: s })),
  });
  return selectedSeverityLevel;
};

/**
 * Create a new support case
 * @param {{
 *  selectedService: import('@aws-sdk/client-support').Service
 *  selectedCategory: import('@aws-sdk/client-support').Category
 *  selectedSeverityLevel: import('@aws-sdk/client-support').SeverityLevel
 * }} selections
 * @returns
 */
export const createCase = async ({
  selectedService,
  selectedCategory,
  selectedSeverityLevel,
}) => {
  const command = new CreateCaseCommand({
    subject: "IGNORE: Test case",
    communicationBody: "This is a test. Please ignore.",
    serviceCode: selectedService.code,
    categoryCode: selectedCategory.code,
    severityCode: selectedSeverityLevel.code,
```

```
  });
  const { caseId } = await client.send(command);
  return caseId;
};

// Get a list of open support cases created today.
export const getTodaysOpenCases = async () => {
  const d = new Date();
  const startOfToday = new Date(d.getFullYear(), d.getMonth(), d.getDate());
  const command = new DescribeCasesCommand({
    includeCommunications: false,
    afterTime: startOfToday.toISOString(),
  });

  const { cases } = await client.send(command);

  if (cases.length === 0) {
    throw new Error(
      "Unexpected number of cases. Expected more than 0 open cases.",
    );
  }
  return cases;
};

// Create an attachment set.
export const createAttachmentSet = async () => {
  const command = new AddAttachmentsToSetCommand({
    attachments: [
      {
        fileName: "example.txt",
        data: new TextEncoder().encode("some example text"),
      },
    ],
  });
  const { attachmentSetId } = await client.send(command);
  return attachmentSetId;
};

export const linkAttachmentSetToCase = async (attachmentSetId, caseId) => {
  const command = new AddCommunicationToCaseCommand({
    attachmentSetId,
    caseId,
    communicationBody: "Adding attachment set to case.",
  });
```

```
  await client.send(command);
};

// Get all communications for a support case.
export const getCommunications = async (caseId) => {
  const command = new DescribeCommunicationsCommand({
    caseId,
  });
  const { communications } = await client.send(command);
  return communications;
};

/**
 * @param {import('@aws-sdk/client-support').Communication[]} communications
 */
export const getFirstAttachment = (communications) => {
  const firstCommWithAttachment = communications.find(
    (c) => c.attachmentSet.length > 0,
  );
  return firstCommWithAttachment?.attachmentSet[0].attachmentId;
};

// Get an attachment.
export const getAttachment = async (attachmentId) => {
  const command = new DescribeAttachmentCommand({
    attachmentId,
  });
  const { attachment } = await client.send(command);
  return attachment;
};

// Resolve the case matching the given case ID.
export const resolveCase = async (caseId) => {
  const shouldResolve = await inquirer.confirm({
    message: `Do you want to resolve ${caseId}?`,
  });

  if (shouldResolve) {
    const command = new ResolveCaseCommand({
      caseId: caseId,
    });

    await client.send(command);
    return true;
```

```
  }
  return false;
};

/**
 * Find a specific case in the list of provided cases by case ID.
 * If the case is not found, and the results are paginated, continue
 * paging through the results.
 * @param {{
 *   caseId: string,
 *   cases: import('@aws-sdk/client-support').CaseDetails[]
 *   nextToken: string
 * }} options
 * @returns
 */
export const findCase = async ({ caseId, cases, nextToken }) => {
  const foundCase = cases.find((c) => c.caseId === caseId);

  if (foundCase) {
    return foundCase;
  }

  if (nextToken) {
    const response = await client.send(
      new DescribeCasesCommand({
        nextToken,
        includeResolvedCases: true,
      }),
    );
    return findCase({
      caseId,
      cases: response.cases,
      nextToken: response.nextToken,
    });
  }

  throw new Error(`${caseId} not found.`);
};

// Get all cases created today.
export const getTodaysResolvedCases = async (caseIdToWaitFor) => {
  const d = new Date("2023-01-18");
  const startOfToday = new Date(d.getFullYear(), d.getMonth(), d.getDate());
  const command = new DescribeCasesCommand({
```

```
    includeCommunications: false,
    afterTime: startOfToday.toISOString(),
    includeResolvedCases: true,
  });
  const { cases, nextToken } = await client.send(command);
  await findCase({ cases, caseId: caseIdToWaitFor, nextToken });
  return cases.filter((c) => c.status === "resolved");
};

const main = async () => {
  let caseId;
  try {
    console.log(wrapText("Welcome to the AWS Support basic usage scenario."));

    // Verify that the account is subscribed to support.
    await verifyAccount();

    // Provided a truncated list of services and prompt the user to select one.
    const selectedService = await getService();

    // Provided the categories for the selected service and prompt the user to
 select one.
    const selectedCategory = await getCategory(selectedService);

    // Provide the severity available severity levels for the account and prompt the
 user to select one.
    const selectedSeverityLevel = await getSeverityLevel();

    // Create a support case.
    console.log("\nCreating a support case.");
    caseId = await createCase({
      selectedService,
      selectedCategory,
      selectedSeverityLevel,
    });
    console.log(`Support case created: ${caseId}`);

    // Display a list of open support cases created today.
    const todaysOpenCases = await retry(
      { intervalInMs: 1000, maxRetries: 15 },
      getTodaysOpenCases,
    );
    console.log(
      `\nOpen support cases created today: ${todaysOpenCases.length}`,
```

```
    );
    console.log(todaysOpenCases.map((c) => `${c.caseId}`).join("\n"));

    // Create an attachment set.
    console.log("\nCreating an attachment set.");
    const attachmentSetId = await createAttachmentSet();
    console.log(`Attachment set created: ${attachmentSetId}`);

    // Add the attachment set to the support case.
    console.log(`\nAdding attachment set to ${caseId}`);
    await linkAttachmentSetToCase(attachmentSetId, caseId);
    console.log(`Attachment set added to ${caseId}`);

    // List the communications for a support case.
    console.log(`\nListing communications for ${caseId}`);
    const communications = await getCommunications(caseId);
    console.log(
      communications
        .map(
          (c) =>
            `Communication created on ${c.timeCreated}. Has
${c.attachmentSet.length} attachments.`,
        )
        .join("\n"),
    );

    // Describe the first attachment.
    console.log(`\nDescribing attachment ${attachmentSetId}`);
    const attachmentId = getFirstAttachment(communications);
    const attachment = await getAttachment(attachmentId);
    console.log(
      `Attachment is the file '${
        attachment.fileName
      }' with data: \n${new TextDecoder().decode(attachment.data)}`,
    );

    // Confirm that the support case should be resolved.
    const isResolved = await resolveCase(caseId);
    if (isResolved) {
      // List the resolved cases and include the one previously created.
      // Resolved cases can take a while to appear.
      console.log(
        "\nWaiting for case status to be marked as resolved. This can take some
time.",
```

```
      );
      const resolvedCases = await retry(
        { intervalInMs: 20000, maxRetries: 15 },
        () => getTodaysResolvedCases(caseId),
      );
      console.log("Resolved cases:");
      console.log(resolvedCases.map((c) => c.caseId).join("\n"));
    }
  } catch (err) {
    console.error(err);
  }
};
```

- 如需詳API細資訊，請參閱「AWS SDK for JavaScript API參考」中的下列主題。
  - AddAttachmentsToSet
  - AddCommunicationToCase
  - CreateCase
  - DescribeAttachment
  - DescribeCases
  - DescribeCommunications
  - DescribeServices
  - DescribeSeverityLevels
  - ResolveCase

動作

`AddAttachmentsToSet`

下列程式碼範例會示範如何使用`AddAttachmentsToSet`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import { AddAttachmentsToSetCommand } from "@aws-sdk/client-support";

import { client } from "../libs/client.js";

export const main = async () => {
  try {
    // Create a new attachment set or add attachments to an existing set.
    // Provide an 'attachmentSetId' value to add attachments to an existing set.
    // Use AddCommunicationToCase or CreateCase to associate an attachment set with
 a support case.
    const response = await client.send(
      new AddAttachmentsToSetCommand({
        // You can add up to three attachments per set. The size limit is 5 MB per
 attachment.
        attachments: [
          {
            fileName: "example.txt",
            data: new TextEncoder().encode("some example text"),
          },
        ],
      }),
    );
    // Use this ID in AddCommunicationToCase or CreateCase.
    console.log(response.attachmentSetId);
    return response;
  } catch (err) {
    console.error(err);
  }
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考AddAttachmentsToSet中的。

AddCommunicationToCase

下列程式碼範例會示範如何使用AddCommunicationToCase。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import { AddCommunicationToCaseCommand } from "@aws-sdk/client-support";

import { client } from "../libs/client.js";

export const main = async () => {
  let attachmentSetId;

  try {
    // Add a communication to a case.
    const response = await client.send(
      new AddCommunicationToCaseCommand({
        communicationBody: "Adding an attachment.",
        // Set value to an existing support case id.
        caseId: "CASE_ID",
        // Optional. Set value to an existing attachment set id to add attachments
 to the case.
        attachmentSetId,
      }),
    );
    console.log(response);
    return response;
  } catch (err) {
    console.error(err);
  }
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考AddCommunicationToCase中的。

`CreateCase`

下列程式碼範例會示範如何使用`CreateCase`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import { CreateCaseCommand } from "@aws-sdk/client-support";

import { client } from "../libs/client.js";

export const main = async () => {
  try {
    // Create a new case and log the case id.
    // Important: This creates a real support case in your account.
    const response = await client.send(
      new CreateCaseCommand({
        // The subject line of the case.
        subject: "IGNORE: Test case",
        // Use DescribeServices to find available service codes for each service.
        serviceCode: "service-quicksight-end-user",
        // Use DescribeSecurityLevels to find available severity codes for your
 support plan.
        severityCode: "low",
        // Use DescribeServices to find available category codes for each service.
        categoryCode: "end-user-support",
        // The main description of the support case.
        communicationBody: "This is a test. Please ignore.",
      }),
    );
    console.log(response.caseId);
    return response;
  } catch (err) {
    console.error(err);
  }
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考CreateCase中的。

`DescribeAttachment`

下列程式碼範例會示範如何使用`DescribeAttachment`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import { DescribeAttachmentCommand } from "@aws-sdk/client-support";

import { client } from "../libs/client.js";

export const main = async () => {
  try {
    // Get the metadata and content of an attachment.
    const response = await client.send(
      new DescribeAttachmentCommand({
        // Set value to an existing attachment id.
        // Use DescribeCommunications or DescribeCases to find an attachment id.
        attachmentId: "ATTACHMENT_ID",
      }),
    );
    console.log(response.attachment?.fileName);
    return response;
  } catch (err) {
    console.error(err);
  }
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考DescribeAttachment中的。

`DescribeCases`

下列程式碼範例會示範如何使用`DescribeCases`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import { DescribeCasesCommand } from "@aws-sdk/client-support";

import { client } from "../libs/client.js";

export const main = async () => {
  try {
    // Get all of the unresolved cases in your account.
    // Filter or expand results by providing parameters to the DescribeCasesCommand.
 Refer
    // to the TypeScript definition and the API doc for more information on possible
 parameters.
    // https://docs.aws.amazon.com/AWSJavaScriptSDK/v3/latest/clients/client-
support/interfaces/describecasescommandinput.html
    const response = await client.send(new DescribeCasesCommand({}));
    const caseIds = response.cases.map((supportCase) => supportCase.caseId);
    console.log(caseIds);
    return response;
  } catch (err) {
    console.error(err);
  }
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考DescribeCases中的。

`DescribeCommunications`

下列程式碼範例會示範如何使用DescribeCommunications。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import { DescribeCommunicationsCommand } from "@aws-sdk/client-support";

import { client } from "../libs/client.js";

export const main = async () => {
  try {
    // Get all communications for the support case.
    // Filter results by providing parameters to the DescribeCommunicationsCommand.
 Refer
    // to the TypeScript definition and the API doc for more information on possible
 parameters.
    // https://docs.aws.amazon.com/AWSJavaScriptSDK/v3/latest/clients/client-
support/interfaces/describecommunicationscommandinput.html
    const response = await client.send(
      new DescribeCommunicationsCommand({
        // Set value to an existing case id.
        caseId: "CASE_ID",
      }),
    );
    const text = response.communications.map((item) => item.body).join("\n");
    console.log(text);
    return response;
  } catch (err) {
    console.error(err);
  }
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考DescribeCommunications中的。

`DescribeSeverityLevels`

下列程式碼範例會示範如何使用`DescribeSeverityLevels`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import { DescribeSeverityLevelsCommand } from "@aws-sdk/client-support";

import { client } from "../libs/client.js";

export const main = async () => {
  try {
    // Get the list of severity levels.
    // The available values depend on the support plan for the account.
    const response = await client.send(new DescribeSeverityLevelsCommand({}));
    console.log(response.severityLevels);
    return response;
  } catch (err) {
    console.error(err);
  }
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考DescribeSeverityLevels中的。

`ResolveCase`

下列程式碼範例會示範如何使用`ResolveCase`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import { ResolveCaseCommand } from "@aws-sdk/client-support";
```

```
import { client } from "../libs/client.js";

const main = async () => {
  try {
    const response = await client.send(
      new ResolveCaseCommand({
        caseId: "CASE_ID",
      }),
    );

    console.log(response.finalCaseStatus);
    return response;
  } catch (err) {
    console.error(err);
  }
};
```

- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考ResolveCase中的。

Amazon Textract 範例使用 SDK for JavaScript (v3)

下列程式碼範例說明如何透過使用 AWS SDK for JavaScript (v3) 搭配 Amazon Textract 來執行動作和實作常見案例。

案例是程式碼範例，向您展示如何透過呼叫服務中的多個函式或與其他函式結合來完成特定工作 AWS 服務。

每個範例都包含完整原始程式碼的連結，您可以在其中找到如何在內容中設定和執行程式碼的指示。

主題

- 案例

案例

建立 Amazon Textract Explorer 應用程式

下列程式碼範例顯示如何透過互動式應用程式探索 Amazon Textract 輸出。

SDK對於 JavaScript （3）

示範如何使用建置 React 應用程式，該應用程式使用 Amazon Textract 擷取文件影像中的資料，並將其顯示在互動式網頁中。 AWS SDK for JavaScript 此範例會在 Web 瀏覽器中執行，且登入資料需要經過驗證的 Amazon Cognito 身分。它使用 Amazon 簡單存儲服務（Amazon S3）進行存儲，並對於通知，它會輪詢訂閱 Amazon 簡單通知服務（AmazonSQS）主題的 Amazon 簡單隊列（亞馬遜SNS）隊列。

有關如何設置和運行的完整源代碼和說明，請參閱中的完整示例GitHub。

此範例中使用的服務

- Amazon Cognito Identity
- Amazon S3
- Amazon SNS
- Amazon SQS
- Amazon Textract

建立應用程式以分析客戶意見回饋

下列程式碼範例會示範如何建立可分析客戶評論卡、從其原始語言進行翻譯、判斷對方情緒，以及透過翻譯後的文字產生音訊檔案的應用程式。

SDK對於 JavaScript （3）

此範例應用程式會分析和存儲客戶的意見回饋卡。具體來說，它滿足了紐約市一家虛構飯店的需求。飯店以實體評論卡的形式收到賓客以各種語言撰寫的意見回饋。這些意見回饋透過 Web 用戶端上傳至應用程式。評論卡的影像上傳後，系統會執行下列步驟：

- 文字內容是使用 Amazon Textract 從影像中擷取。
- Amazon Comprehend 會決定擷取文字及其用語的情感。
- 擷取的文字內容會使用 Amazon Translate 翻譯成英文。
- Amazon Polly 會使用擷取的文字內容合成音訊檔案。

完整的應用程式可透過 AWS CDK 部署。如需原始程式碼和部署指示，請參閱中的專案 GitHub。下列摘錄顯示如何在 AWS SDK for JavaScript Lambda 函數內部使用。

```
import {
  ComprehendClient,
```

```
  DetectDominantLanguageCommand,
  DetectSentimentCommand,
} from "@aws-sdk/client-comprehend";

/**
 * Determine the language and sentiment of the extracted text.
 *
 * @param {{ source_text: string}} extractTextOutput
 */
export const handler = async (extractTextOutput) => {
  const comprehendClient = new ComprehendClient({});

  const detectDominantLanguageCommand = new DetectDominantLanguageCommand({
    Text: extractTextOutput.source_text,
  });

  // The source language is required for sentiment analysis and
  // translation in the next step.
  const { Languages } = await comprehendClient.send(
    detectDominantLanguageCommand,
  );

  const languageCode = Languages[0].LanguageCode;

  const detectSentimentCommand = new DetectSentimentCommand({
    Text: extractTextOutput.source_text,
    LanguageCode: languageCode,
  });

  const { Sentiment } = await comprehendClient.send(detectSentimentCommand);

  return {
    sentiment: Sentiment,
    language_code: languageCode,
  };
};
```

```
import {
  DetectDocumentTextCommand,
  TextractClient,
} from "@aws-sdk/client-textract";

/**
```

```
 * Fetch the S3 object from the event and analyze it using Amazon Textract.
 *
 * @param {import("@types/aws-lambda").EventBridgeEvent<"Object Created">}
 eventBridgeS3Event
 */
export const handler = async (eventBridgeS3Event) => {
  const textractClient = new TextractClient();

  const detectDocumentTextCommand = new DetectDocumentTextCommand({
    Document: {
      S3Object: {
        Bucket: eventBridgeS3Event.bucket,
        Name: eventBridgeS3Event.object,
      },
    },
  });

  // Textract returns a list of blocks. A block can be a line, a page, word, etc.
  // Each block also contains geometry of the detected text.
  // For more information on the Block type, see https://docs.aws.amazon.com/
textract/latest/dg/API_Block.html.
  const { Blocks } = await textractClient.send(detectDocumentTextCommand);

  // For the purpose of this example, we are only interested in words.
  const extractedWords = Blocks.filter((b) => b.BlockType === "WORD").map(
    (b) => b.Text,
  );

  return extractedWords.join(" ");
};
```

```
import { PollyClient, SynthesizeSpeechCommand } from "@aws-sdk/client-polly";
import { S3Client } from "@aws-sdk/client-s3";
import { Upload } from "@aws-sdk/lib-storage";

/**
 * Synthesize an audio file from text.
 *
 * @param {{ bucket: string, translated_text: string, object: string}}
 sourceDestinationConfig
 */
export const handler = async (sourceDestinationConfig) => {
  const pollyClient = new PollyClient({});
```

```
  const synthesizeSpeechCommand = new SynthesizeSpeechCommand({
    Engine: "neural",
    Text: sourceDestinationConfig.translated_text,
    VoiceId: "Ruth",
    OutputFormat: "mp3",
  });

  const { AudioStream } = await pollyClient.send(synthesizeSpeechCommand);

  const audioKey = `${sourceDestinationConfig.object}.mp3`;

  // Store the audio file in S3.
  const s3Client = new S3Client();
  const upload = new Upload({
    client: s3Client,
    params: {
      Bucket: sourceDestinationConfig.bucket,
      Key: audioKey,
      Body: AudioStream,
      ContentType: "audio/mp3",
    },
  });

  await upload.done();
  return audioKey;
};
```

```
import {
  TranslateClient,
  TranslateTextCommand,
} from "@aws-sdk/client-translate";

/**
 * Translate the extracted text to English.
 *
 * @param {{ extracted_text: string, source_language_code: string}}
 textAndSourceLanguage
 */
export const handler = async (textAndSourceLanguage) => {
  const translateClient = new TranslateClient({});

  const translateCommand = new TranslateTextCommand({
```

```
    SourceLanguageCode: textAndSourceLanguage.source_language_code,
    TargetLanguageCode: "en",
    Text: textAndSourceLanguage.extracted_text,
  });

  const { TranslatedText } = await translateClient.send(translateCommand);

  return { translated_text: TranslatedText };
};
```

此範例中使用的服務

- Amazon Comprehend
- Lambda
- Amazon Polly
- Amazon Textract
- Amazon Translate

Amazon Transcribe 示例使用 SDK for JavaScript (v3)

下列程式碼範例說明如何透過使用 AWS SDK for JavaScript (v3) 搭配 Amazon Transcribe 來執行動作和實作常見案例。

Actions 是大型程式的程式碼摘錄，必須在內容中執行。雖然動作會顯示如何呼叫個別服務函數，但您可以在其相關案例中查看內容中的動作。

案例是程式碼範例，向您展示如何透過呼叫服務中的多個函式或與其他函式結合來完成特定工作 AWS 服務。

每個範例都包含完整原始程式碼的連結，您可以在其中找到如何在內容中設定和執行程式碼的指示。

主題

- 動作
- 案例

動作

DeleteMedicalTranscriptionJob

下列程式碼範例會示範如何使用DeleteMedicalTranscriptionJob。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

建立用戶端。

```
import { TranscribeClient } from "@aws-sdk/client-transcribe";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create an Amazon Transcribe service client object.
const transcribeClient = new TranscribeClient({ region: REGION });
export { transcribeClient };
```

刪除醫學轉錄作業。

```
// Import the required AWS SDK clients and commands for Node.js
import { DeleteMedicalTranscriptionJobCommand } from "@aws-sdk/client-transcribe";
import { transcribeClient } from "./libs/transcribeClient.js";

// Set the parameters
export const params = {
  MedicalTranscriptionJobName: "MEDICAL_JOB_NAME", // For example,
 'medical_transciption_demo'
};

export const run = async () => {
  try {
    const data = await transcribeClient.send(
      new DeleteMedicalTranscriptionJobCommand(params)
    );
    console.log("Success - deleted");
```

```
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考DeleteMedicalTranscriptionJob中的。

`DeleteTranscriptionJob`

下列程式碼範例會示範如何使用`DeleteTranscriptionJob`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

刪除轉錄作業。

```
// Import the required AWS SDK clients and commands for Node.js
import { DeleteTranscriptionJobCommand } from "@aws-sdk/client-transcribe";
import { transcribeClient } from "./libs/transcribeClient.js";

// Set the parameters
export const params = {
  TranscriptionJobName: "JOB_NAME", // Required. For example, 'transciption_demo'
};

export const run = async () => {
  try {
    const data = await transcribeClient.send(
      new DeleteTranscriptionJobCommand(params)
    );
```

```
    console.log("Success - deleted");
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

建立用戶端。

```
import { TranscribeClient } from "@aws-sdk/client-transcribe";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create an Amazon Transcribe service client object.
const transcribeClient = new TranscribeClient({ region: REGION });
export { transcribeClient };
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考DeleteTranscriptionJob中的。

`ListMedicalTranscriptionJobs`

下列程式碼範例會示範如何使用`ListMedicalTranscriptionJobs`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

建立用戶端。

```
import { TranscribeClient } from "@aws-sdk/client-transcribe";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create an Amazon Transcribe service client object.
```

```
const transcribeClient = new TranscribeClient({ region: REGION });
export { transcribeClient };
```

列出醫學轉錄作業。

```
// Import the required AWS SDK clients and commands for Node.js
import { StartMedicalTranscriptionJobCommand } from "@aws-sdk/client-transcribe";
import { transcribeClient } from "./libs/transcribeClient.js";

// Set the parameters
export const params = {
  MedicalTranscriptionJobName: "MEDICAL_JOB_NAME", // Required
  OutputBucketName: "OUTPUT_BUCKET_NAME", // Required
  Specialty: "PRIMARYCARE", // Required. Possible values are 'PRIMARYCARE'
  Type: "JOB_TYPE", // Required. Possible values are 'CONVERSATION' and 'DICTATION'
  LanguageCode: "LANGUAGE_CODE", // For example, 'en-US'
  MediaFormat: "SOURCE_FILE_FORMAT", // For example, 'wav'
  Media: {
    MediaFileUri: "SOURCE_FILE_LOCATION",
    // The S3 object location of the input media file. The URI must be in the same
 region
    // as the API endpoint that you are calling.For example,
    // "https://transcribe-demo.s3-REGION.amazonaws.com/hello_world.wav"
  },
};

export const run = async () => {
  try {
    const data = await transcribeClient.send(
      new StartMedicalTranscriptionJobCommand(params)
    );
    console.log("Success - put", data);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考ListMedicalTranscriptionJobs中的。

`ListTranscriptionJobs`

下列程式碼範例會示範如何使用`ListTranscriptionJobs`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

列出轉錄作業。

```
// Import the required AWS SDK clients and commands for Node.js

import { ListTranscriptionJobsCommand } from "@aws-sdk/client-transcribe";
import { transcribeClient } from "./libs/transcribeClient.js";

// Set the parameters
export const params = {
  JobNameContains: "KEYWORD", // Not required. Returns only transcription
  // job names containing this string
};

export const run = async () => {
  try {
    const data = await transcribeClient.send(
      new ListTranscriptionJobsCommand(params)
    );
    console.log("Success", data.TranscriptionJobSummaries);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

建立用戶端。

```
import { TranscribeClient } from "@aws-sdk/client-transcribe";
```

```
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create an Amazon Transcribe service client object.
const transcribeClient = new TranscribeClient({ region: REGION });
export { transcribeClient };
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考ListTranscriptionJobs中的。

`StartMedicalTranscriptionJob`

下列程式碼範例會示範如何使用`StartMedicalTranscriptionJob`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

建立用戶端。

```
import { TranscribeClient } from "@aws-sdk/client-transcribe";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create an Amazon Transcribe service client object.
const transcribeClient = new TranscribeClient({ region: REGION });
export { transcribeClient };
```

開始醫學轉錄作業。

```
// Import the required AWS SDK clients and commands for Node.js
import { StartMedicalTranscriptionJobCommand } from "@aws-sdk/client-transcribe";
import { transcribeClient } from "./libs/transcribeClient.js";

// Set the parameters
export const params = {
  MedicalTranscriptionJobName: "MEDICAL_JOB_NAME", // Required
```

```
  OutputBucketName: "OUTPUT_BUCKET_NAME", // Required
  Specialty: "PRIMARYCARE", // Required. Possible values are 'PRIMARYCARE'
  Type: "JOB_TYPE", // Required. Possible values are 'CONVERSATION' and 'DICTATION'
  LanguageCode: "LANGUAGE_CODE", // For example, 'en-US'
  MediaFormat: "SOURCE_FILE_FORMAT", // For example, 'wav'
  Media: {
    MediaFileUri: "SOURCE_FILE_LOCATION",
    // The S3 object location of the input media file. The URI must be in the same
 region
    // as the API endpoint that you are calling.For example,
    // "https://transcribe-demo.s3-REGION.amazonaws.com/hello_world.wav"
  },
};

export const run = async () => {
  try {
    const data = await transcribeClient.send(
      new StartMedicalTranscriptionJobCommand(params)
    );
    console.log("Success - put", data);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考StartMedicalTranscriptionJob中的。

`StartTranscriptionJob`

下列程式碼範例會示範如何使用`StartTranscriptionJob`。

SDK對於 JavaScript（3）

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

開始轉錄作業。

```
// Import the required AWS SDK clients and commands for Node.js
import { StartTranscriptionJobCommand } from "@aws-sdk/client-transcribe";
import { transcribeClient } from "./libs/transcribeClient.js";

// Set the parameters
export const params = {
  TranscriptionJobName: "JOB_NAME",
  LanguageCode: "LANGUAGE_CODE", // For example, 'en-US'
  MediaFormat: "SOURCE_FILE_FORMAT", // For example, 'wav'
  Media: {
    MediaFileUri: "SOURCE_LOCATION",
    // For example, "https://transcribe-demo.s3-REGION.amazonaws.com/
hello_world.wav"
  },
  OutputBucketName: "OUTPUT_BUCKET_NAME"
};

export const run = async () => {
  try {
    const data = await transcribeClient.send(
      new StartTranscriptionJobCommand(params)
    );
    console.log("Success - put", data);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

建立用戶端。

```
import { TranscribeClient } from "@aws-sdk/client-transcribe";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create an Amazon Transcribe service client object.
const transcribeClient = new TranscribeClient({ region: REGION });
export { transcribeClient };
```

- 如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》。
- 如需詳API細資訊，請參閱AWS SDK for JavaScript API參考StartTranscriptionJob中的。

案例

建置 Amazon Transcribe 串流應用程式

下面的程式碼範例說明如何建置可即時記錄、轉錄和翻譯直播音訊並透過電子郵件傳送結果的應用程式。

SDK對於 JavaScript（3）

示範如何使用 Amazon Transcribe 建置可即時記錄、轉錄和轉譯即時音訊的應用程式，並使用 Amazon 簡易電子郵件服務 (Amazon) 將結果以電子郵件傳送給結果。SES

有關如何設置和運行的完整源代碼和說明，請參閱中的完整示例GitHub。

此範例中使用的服務

- Amazon Comprehend
- Amazon SES
- Amazon Transcribe
- Amazon Translate

Amazon Translate 示SDK例使用 JavaScript（v3）

下列程式碼範例說明如何透過使用 AWS SDK for JavaScript (v3) 搭配 Amazon Translate 來執行動作和實作常見案例。

案例是程式碼範例，向您展示如何透過呼叫服務中的多個函式或與其他函式結合來完成特定工作 AWS 服務。

每個範例都包含完整原始程式碼的連結，您可以在其中找到如何在內容中設定和執行程式碼的指示。

主題

案例

建置 Amazon Transcribe 串流應用程式

下面的程式碼範例說明如何建置可即時記錄、轉錄和翻譯直播音訊並透過電子郵件傳送結果的應用程式。

SDK對於 JavaScript（3）

示範如何使用 Amazon Transcribe 建置可即時記錄、轉錄和轉譯即時音訊的應用程式，並使用 Amazon 簡易電子郵件服務 (Amazon) 將結果以電子郵件傳送給結果。SES

有關如何設置和運行的完整源代碼和說明，請參閱中的完整示例GitHub。

此範例中使用的服務

- Amazon Comprehend
- Amazon SES
- Amazon Transcribe
- Amazon Translate

建立 Amazon Lex 聊天機器人

以下代碼示例演示瞭如何創建聊天機器人以吸引您的網站訪問者。

SDK對於 JavaScript（3）

示範如何使用 Amazon Lex 在 Web 應API用程式中建立 Chatbot，以吸引您的網站訪客。

如需有關如何設定和執行的完整原始程式碼和指示，請參閱開發人 AWS SDK for JavaScript 員指南中的建立 Amazon Lex 聊天機器人的完整範例。

此範例中使用的服務

- Amazon Comprehend
- Amazon Lex
- Amazon Translate

建立應用程式以分析客戶意見回饋

下列程式碼範例會示範如何建立可分析客戶評論卡、從其原始語言進行翻譯、判斷對方情緒，以及透過翻譯後的文字產生音訊檔案的應用程式。

SDK對於 JavaScript（3）

此範例應用程式會分析和存儲客戶的意見回饋卡。具體來說，它滿足了紐約市一家虛構飯店的需求。飯店以實體評論卡的形式收到賓客以各種語言撰寫的意見回饋。這些意見回饋透過 Web 用戶端上傳至應用程式。評論卡的影像上傳後，系統會執行下列步驟：

- 文字內容是使用 Amazon Textract 從影像中擷取。
- Amazon Comprehend 會決定擷取文字及其用語的情感。
- 擷取的文字內容會使用 Amazon Translate 翻譯成英文。
- Amazon Polly 會使用擷取的文字內容合成音訊檔案。

完整的應用程式可透過 AWS CDK 部署。如需原始程式碼和部署指示，請參閱中的專案 GitHub。下列摘錄顯示如何在 AWS SDK for JavaScript Lambda 函數內部使用。

```
import {
  ComprehendClient,
  DetectDominantLanguageCommand,
  DetectSentimentCommand,
} from "@aws-sdk/client-comprehend";

/**
 * Determine the language and sentiment of the extracted text.
 *
 * @param {{ source_text: string}} extractTextOutput
 */
export const handler = async (extractTextOutput) => {
  const comprehendClient = new ComprehendClient({});

  const detectDominantLanguageCommand = new DetectDominantLanguageCommand({
    Text: extractTextOutput.source_text,
  });

  // The source language is required for sentiment analysis and
  // translation in the next step.
  const { Languages } = await comprehendClient.send(
    detectDominantLanguageCommand,
  );
```

```
  const languageCode = Languages[0].LanguageCode;

  const detectSentimentCommand = new DetectSentimentCommand({
    Text: extractTextOutput.source_text,
    LanguageCode: languageCode,
  });

  const { Sentiment } = await comprehendClient.send(detectSentimentCommand);

  return {
    sentiment: Sentiment,
    language_code: languageCode,
  };
};
```

```
import {
  DetectDocumentTextCommand,
  TextractClient,
} from "@aws-sdk/client-textract";

/**
 * Fetch the S3 object from the event and analyze it using Amazon Textract.
 *
 * @param {import("@types/aws-lambda").EventBridgeEvent<"Object Created">}
 eventBridgeS3Event
 */
export const handler = async (eventBridgeS3Event) => {
  const textractClient = new TextractClient();

  const detectDocumentTextCommand = new DetectDocumentTextCommand({
    Document: {
      S3Object: {
        Bucket: eventBridgeS3Event.bucket,
        Name: eventBridgeS3Event.object,
      },
    },
  });

  // Textract returns a list of blocks. A block can be a line, a page, word, etc.
  // Each block also contains geometry of the detected text.
  // For more information on the Block type, see https://docs.aws.amazon.com/
textract/latest/dg/API_Block.html.
```

```
  const { Blocks } = await textractClient.send(detectDocumentTextCommand);

  // For the purpose of this example, we are only interested in words.
  const extractedWords = Blocks.filter((b) => b.BlockType === "WORD").map(
    (b) => b.Text,
  );

  return extractedWords.join(" ");
};
```

```
import { PollyClient, SynthesizeSpeechCommand } from "@aws-sdk/client-polly";
import { S3Client } from "@aws-sdk/client-s3";
import { Upload } from "@aws-sdk/lib-storage";

/**
 * Synthesize an audio file from text.
 *
 * @param {{ bucket: string, translated_text: string, object: string}}
 sourceDestinationConfig
 */
export const handler = async (sourceDestinationConfig) => {
  const pollyClient = new PollyClient({});

  const synthesizeSpeechCommand = new SynthesizeSpeechCommand({
    Engine: "neural",
    Text: sourceDestinationConfig.translated_text,
    VoiceId: "Ruth",
    OutputFormat: "mp3",
  });

  const { AudioStream } = await pollyClient.send(synthesizeSpeechCommand);

  const audioKey = `${sourceDestinationConfig.object}.mp3`;

  // Store the audio file in S3.
  const s3Client = new S3Client();
  const upload = new Upload({
    client: s3Client,
    params: {
      Bucket: sourceDestinationConfig.bucket,
      Key: audioKey,
      Body: AudioStream,
      ContentType: "audio/mp3",
```

```
    },
  });

  await upload.done();
  return audioKey;
};
```

```
import {
  TranslateClient,
  TranslateTextCommand,
} from "@aws-sdk/client-translate";

/**
 * Translate the extracted text to English.
 *
 * @param {{ extracted_text: string, source_language_code: string}}
 textAndSourceLanguage
 */
export const handler = async (textAndSourceLanguage) => {
  const translateClient = new TranslateClient({});

  const translateCommand = new TranslateTextCommand({
    SourceLanguageCode: textAndSourceLanguage.source_language_code,
    TargetLanguageCode: "en",
    Text: textAndSourceLanguage.extracted_text,
  });

  const { TranslatedText } = await translateClient.send(translateCommand);

  return { translated_text: TranslatedText };
};
```

此範例中使用的服務

- Amazon Comprehend
- Lambda
- Amazon Polly
- Amazon Textract
- Amazon Translate

本 AWS 產品或服務的安全性

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。安全是 AWS 與您之間共同的責任。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。

雲的安全性 — AWS 負責保護運行 AWS 雲中提供的所有服務的基礎設施，並為您提供可以安全使用的服務。我們的安全責任是我們的首要任務 AWS，並且我們的安全性有效性是由第三方審計師定期測試和驗證，作為AWS 合規計劃的一部分。

雲端安全性 — 您的責任取決於您使用的 AWS 服務，以及其他因素，包括資料的敏感性、組織的需求，以及適用的法律和法規。

本 AWS 產品或服務透過其支援的特定 Amazon 網路服務 (AWS) 服務，遵循共同的責任模式。如需 AWS 服務安全性資訊，請參閱AWS 服務安全性說明文件頁面和符合性計劃 AWS 遵循工作範圍的 AWS 服務。

主題

- 本 AWS 產品或服務中的資料保護
- 身分和存取權管理
- 本 AWS 產品或服務的合規驗證
- 本 AWS 產品或服務的復原能力
- 本 AWS 產品或服務的基礎架構安全性
- 強制執行最低TLS版本

本 AWS 產品或服務中的資料保護

AWS 共同責任模型適用於本 AWS 產品或服務中的資料保護。如此模型中所述， AWS 負責保護執行所有 AWS 雲端. 您負責維護在此基礎設施上託管內容的控制權。您也同時負責所使用 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FAQ。 如需歐洲資料保護的相關資訊，請參閱AWS 安全性GDPR部落格上的AWS 共同責任模型和部落格文章。

基於資料保護目的，我們建議您使用 AWS IAM Identity Center 或 AWS Identity and Access Management (IAM) 保護 AWS 帳戶 認證並設定個別使用者。如此一來，每個使用者都只會獲得授與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：

- 對每個帳戶使用多重要素驗證 (MFA)。
- 使用SSL/TLS與 AWS 資源溝通。我們需要 TLS 1.2 並推薦 TLS 1.3。
- 使用設定API和使用者活動記錄 AWS CloudTrail。
- 使用 AWS 加密解決方案以及其中的所有默認安全控制 AWS 服務。
- 使用進階的受管安全服務 (例如 Amazon Macie)，協助探索和保護儲存在 Amazon S3 的敏感資料。
- 如果 AWS 透過命令列介面或存取時需要 FIPS 140-3 驗證的密碼編譯模組API，請使用端點。FIPS 如需有關可用FIPS端點的詳細資訊，請參閱聯邦資訊處理標準 (FIPS) 140-3。

我們強烈建議您絕對不要將客戶的電子郵件地址等機密或敏感資訊，放在標籤或自由格式的文字欄位中，例如名稱欄位。這包括當您使用主控台、、API或 AWS 服務 使用本 AWS 產品或服務或其他產品時 AWS SDKs。 AWS CLI您在標籤或自由格式文字欄位中輸入的任何資料都可能用於計費或診斷日誌。如果您提供URL給外部伺服器，我們強烈建議您不要在中包含認證資訊，URL以驗證您對該伺服器的要求。

身分和存取權管理

AWS Identity and Access Management (IAM) 可協助系統管理員安全地控制 AWS 資源存取權。 AWS 服務 IAM管理員控制誰可以驗證（登錄）和授權（有權限）使用 AWS 資源。IAM是您 AWS 服務 可以免費使用的。

主題

- 物件
- 使用身分驗證
- 使用政策管理存取權
- 如何 AWS 服務 使用 IAM
- 疑難排解 AWS 身分和存取

物件

你如何使用 AWS Identity and Access Management （IAM）不同，具體取決於你在做的工作 AWS。

服務使用者 — 如果您 AWS 服務 用於執行工作，則管理員會為您提供所需的認證和權限。當您使用更多 AWS 功能來完成工作時，您可能需要其他權限。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，請參閱疑難排解 AWS 身分和存取或 AWS 服務 您正在使用的使用指南。

服務管理員 — 如果您負責公司的 AWS 資源，您可能擁有完整的存取權 AWS。決定您的服務使用者應該存取哪些 AWS 功能和資源是您的工作。然後，您必須向IAM管理員提交請求，才能變更服務使用者的權限。檢閱此頁面上的資訊，以瞭解的基本概念IAM。若要進一步瞭解貴公司如何IAM搭配使用 AWS，請參閱 AWS 服務 您使用的使用者指南。

IAM系統管理員 — 如果您是IAM系統管理員，您可能想要瞭解如何撰寫原則來管理存取權的詳細資訊 AWS。若要檢視您可以在中使用的以 AWS 身分識別為基礎的策略範例IAM，請參閱 AWS 服務 您正在使用的使用者指南。

使用身分驗證

驗證是您 AWS 使用身分認證登入的方式。您必須以IAM使用者身分或假設IAM角色來驗證 (登入 AWS)。 AWS 帳戶根使用者

您可以使用透過 AWS 身分識別來源提供的認證，以聯合身分識別身分登入。 AWS IAM Identity Center (IAM身分識別中心) 使用者、貴公司的單一登入驗證，以及您的 Google 或 Facebook 認證都是聯合身分識別的範例。當您以同盟身分登入時，您的管理員先前會使用IAM角色設定聯合身分識別。當您使 AWS 用同盟存取時，您會間接擔任角色。

根據您的使用者類型，您可以登入 AWS Management Console 或 AWS 存取入口網站。如需有關登入的詳細資訊 AWS，請參閱《AWS 登入 使用指南》 AWS 帳戶中的如何登入您的。

如果您 AWS 以程式設計方式存取，請 AWS 提供軟體開發套件 (SDK) 和命令列介面 (CLI)，以使用您的認證以密碼編譯方式簽署您的要求。如果您不使用 AWS 工具，則必須自行簽署要求。如需使用建議的方法自行簽署要求的詳細資訊，請參閱使用IAM者指南中的簽署 AWS API要求。

無論您使用何種身分驗證方法，您可能都需要提供額外的安全性資訊。例如， AWS 建議您使用多重要素驗證 (MFA) 來增加帳戶的安全性。若要深入瞭解，請參閱使用AWS IAM Identity Center 者指南中的多重要素驗證和使用多重要素驗證 (MFA) AWS的使用IAM者指南。

AWS 帳戶 根使用者

當您建立時 AWS 帳戶，您會從一個登入身分開始，該身分可完整存取該帳戶中的所有資源 AWS 服務和資源。此身分稱為 AWS 帳戶 root 使用者，可透過使用您用來建立帳戶的電子郵件地址和密碼登入來存取。強烈建議您不要以根使用者處理日常任務。保護您的根使用者憑證，並將其用來執行只能由根使用者執行的任務。如需需要您以 root 使用者身分登入的完整工作清單，請參閱《使用指南》中的〈需要 root 使用者認證的IAM工作〉。

聯合身分

最佳作法是要求人類使用者 (包括需要系統管理員存取權的使用者) 使用與身分識別提供者的同盟，才能使用臨時登入資料進行存取 AWS 服務 。

聯合身分識別是來自企業使用者目錄的使用者、Web 身分識別提供者、Identi ty Center 目錄，或使用透過身分識別來源提供的認證進行存取 AWS 服務 的任何使用者。 AWS Directory Service同盟身分存取時 AWS 帳戶，他們會假設角色，而角色則提供臨時認證。

對於集中式存取權管理，我們建議您使用 AWS IAM Identity Center。您可以在 IAM Identity Center 中建立使用者和群組，也可以連線並同步至您自己身分識別來源中的一組使用者和群組，以便在所有應用程式 AWS 帳戶 和應用程式中使用。如需IAM身分識別中心的相關資訊，請參閱IAM識別中心是什麼？在《AWS IAM Identity Center 使用者指南》中。

IAM 使用者和群組

IAM使用者是您內部的身分，具 AWS 帳戶 有單一人員或應用程式的特定權限。在可能的情況下，我們建議您仰賴臨時登入資料，而不要建立具有長期認證 (例如密碼和存取金鑰) 的IAM使用者。不過，如果您的特定使用案例需要使用IAM者的長期認證，建議您輪換存取金鑰。如需詳細資訊，請參閱《使用指南》中的「IAM定期輪換存取金鑰」以瞭解需要長期認證的使用案例。

IAM群組是指定IAM使用者集合的身分識別。您無法以群組身分簽署。您可以使用群組來一次為多名使用者指定許可。群組可讓管理大量使用者許可的程序變得更為容易。例如，您可以擁有一個名為的群組，IAMAdmins並授與該群組管理IAM資源的權限。

使用者與角色不同。使用者只會與單一人員或應用程式建立關聯，但角色的目的是在由任何需要它的人員取得。使用者擁有永久的長期憑證，但角色僅提供暫時憑證。要了解更多信息，請參閱《IAM用戶指南》中的創建用戶（而不是角色）的IAM時間。

IAM角色

IAM角色是您 AWS 帳戶 中具有特定權限的身份。它類似於用IAM戶，但不與特定人員相關聯。您可以 AWS Management Console 透過切換角色來暫時擔任中的角色。IAM您可以呼叫 AWS CLI 或 AWS API作業或使用自訂來擔任角色URL。如需有關使用角色方法的詳細資訊，請參閱《使用指南》中的IAM〈使用IAM角色〉。

IAM具有臨時認證的角色在下列情況下很有用：

- 聯合身分使用者存取 — 如需向聯合身分指派許可，請建立角色，並為角色定義許可。當聯合身分進行身分驗證時，該身分會與角色建立關聯，並獲授予由角色定義的許可。如需聯合角色的相關資訊，請參閱《使用指南》中的〈建立第三方身分識別提供IAM者的角色〉。如果您使用IAM身分識別中

心，則需要設定權限集。為了控制身分驗證後可以存取的內IAM容，IAMIdentity Center 會將權限集與中的角色相關聯。如需有關許可集的資訊，請參閱 AWS IAM Identity Center 使用者指南中的許可集。

- 暫時IAM使用者權限 — IAM 使用者或角色可以假定某個IAM角色，暫時取得特定工作的不同權限。
- 跨帳戶存取 — 您可以使用IAM角色允許不同帳戶中的某個人 (受信任的主體) 存取您帳戶中的資源。角色是授予跨帳戶存取權的主要方式。但是，對於某些策略 AWS 服務，您可以將策略直接附加到資源（而不是使用角色作為代理）。若要瞭解跨帳戶存取角色與以資源為基礎的政策之間的差異，請參閱《IAM使用指南》IAM中的〈跨帳號資源存取〉。
- 跨服務訪問 — 有些 AWS 服務 使用其他 AWS 服務功能。例如，當您在服務中撥打電話時，該服務通常會在 Amazon 中執行應用程式EC2或將物件存放在 Amazon S3 中。服務可能會使用呼叫主體的許可、使用服務角色或使用服務連結角色來執行此作業。
  - 轉寄存取工作階段 (FAS) — 當您使用使用IAM者或角色執行中的動作時 AWS，您會被視為主參與者。使用某些服務時，您可能會執行某個動作，進而在不同服務中啟動另一個動作。FAS會使用主參與者呼叫的權限 AWS 服務，並結合要求 AWS 服務 向下游服務發出要求。FAS只有當服務收到需要與其他 AWS 服務 資源互動才能完成的請求時，才會發出請求。在此情況下，您必須具有執行這兩個動作的許可。有關提出FAS請求時的策略詳細信息，請參閱轉發訪問會話。
  - 服務角色 — 服務角IAM色是服務代表您執行動作的角色。IAM管理員可以從中建立、修改和刪除服務角色IAM。如需詳細資訊，請參閱《IAM使用指南》 AWS 服務中的建立角色以將權限委派給
  - 服務連結角色 — 服務連結角色是連結至. AWS 服務服務可以擔任代表您執行動作的角色。服務連結角色會顯示在您的中， AWS 帳戶 且屬於服務所有。IAM管理員可以檢視 (但無法編輯服務連結角色) 的權限。
- 在 Amazon 上執行的應用程式 EC2 — 您可以使用IAM角色來管理在執行個體上EC2執行的應用程式以及發出 AWS CLI 或 AWS API請求的臨時登入資料。這比在EC2實例中存儲訪問密鑰更好。若要將 AWS 角色指派給EC2執行個體並讓其所有應用程式都能使用，請建立附加至執行個體的執行個體設定檔。執行個體設定檔包含角色，可讓執行個體上EC2執行的程式取得臨時登入資料。如需詳細資訊，請參閱使用者指南中的使用IAM角色將許可授與在 Amazon EC2 執行個體上執行的應IAM用程式。

要了解是否使用IAM角色還是用IAM戶，請參閱《用戶指南》中的「IAM創建IAM角色的時機（而不是用戶）」。

使用政策管理存取權

您可以透 AWS 過建立原則並將其附加至 AWS 身分識別或資源來控制中的存取。原則是一個物件 AWS ，當與身分識別或資源相關聯時，會定義其權限。 AWS 當主參與者 (使用者、root 使用者或角色

工作階段) 提出要求時，評估這些原則。政策中的許可決定是否允許或拒絕請求。大多數原則會 AWS 以JSON文件的形式儲存在中。如需有關JSON原則文件結構和內容的詳細資訊，請參閱《IAM使用指南》中的策略概觀。JSON

管理員可以使用 AWS JSON策略來指定誰可以存取什麼內容。也就是說，哪個主體在什麼條件下可以對什麼資源執行哪些動作。

預設情況下，使用者和角色沒有許可。若要授與使用者對所需資源執行動作的權限，IAM管理員可以建立IAM策略。然後，系統管理員可以將IAM原則新增至角色，使用者可以擔任這些角色。

IAM原則會定義動作的權限，不論您用來執行作業的方法為何。例如，假設您有一個允許 `iam:GetRole` 動作的政策。具有該原則的使用者可以從 AWS Management Console AWS CLI、或取得角色資訊 AWS API。

身分型政策

以身分識別為基礎的原則是您可以附加至身分識別 (例如使用者、使用IAM者群組或角色) 的JSON權限原則文件。這些政策可控制身分在何種條件下能對哪些資源執行哪些動作。若要瞭解如何建立以身分識別為基礎的策略，請參閱IAM使用指南中的建立IAM策略。

身分型政策可進一步分類成內嵌政策或受管政策。內嵌政策會直接內嵌到單一使用者、群組或角色。受管理的策略是獨立策略，您可以將其附加到您的 AWS 帳戶. 受管政策包括 AWS 受管政策和客戶管理的策略。若要了解如何在受管策略或內嵌策略之間進行選擇，請參閱《IAM使用手冊》中的「在受管策略和內嵌策略之間進行選擇」。

資源型政策

以資源為基礎的JSON策略是您附加至資源的政策文件。以資源為基礎的政策範例包括IAM角色信任政策和 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。對於附加政策的資源，政策會定義指定的主體可以對該資源執行的動作以及在何種條件下執行的動作。您必須在資源型政策中指定主體。主參與者可以包括帳戶、使用者、角色、同盟使用者或。 AWS 服務

資源型政策是位於該服務中的內嵌政策。您無法在以資源為基礎的策略IAM中使用 AWS 受管政策。

存取控制清單 (ACLs)

存取控制清單 (ACLs) 控制哪些主參與者 (帳戶成員、使用者或角色) 具有存取資源的權限。ACLs類似於以資源為基礎的策略，雖然它們不使用JSON政策文件格式。

Amazon S3 和 Amazon VPC 是支持服務的示例ACLs。 AWS WAF若要進一步了解ACLs，請參閱 Amazon 簡單儲存服務開發人員指南中的存取控制清單 (ACL) 概觀。

其他政策類型

AWS 支援其他較不常見的原則類型。這些政策類型可設定較常見政策類型授予您的最大許可。

- 權限界限 — 權限界限是一項進階功能，您可以在其中設定以身分識別為基礎的原則可授與給IAM實體 (IAM使用者或角色) 的最大權限。您可以為實體設定許可界限。所產生的許可會是實體的身分型政策和其許可界限的交集。會在 `Principal` 欄位中指定使用者或角色的資源型政策則不會受到許可界限限制。所有這類政策中的明確拒絕都會覆寫該允許。如需有關權限界限的詳細資訊，請參閱《IAM使用指南》中的IAM實體的權限界限。
- 服務控制策略 (SCPs) — SCPs 是指定中組織或組織單位 (OU) 最大權限的JSON策略 AWS Organizations。 AWS Organizations 是一種用於分組和集中管理您企業擁 AWS 帳戶 有的多個服務。如果您啟用組織中的所有功能，則可以將服務控制策略 (SCPs) 套用至您的任何或所有帳戶。SCP限制成員帳戶中實體的權限，包括每個帳戶 AWS 帳戶根使用者。如需有關 Organizations 的詳細資訊SCPs，請參閱AWS Organizations 使用指南中的服務控制原則。
- 工作階段政策 – 工作階段政策是一種進階政策，您可以在透過編寫程式的方式建立角色或聯合使用者的暫時工作階段時，作為參數傳遞。所產生工作階段的許可會是使用者或角色的身分型政策和工作階段政策的交集。許可也可以來自資源型政策。所有這類政策中的明確拒絕都會覆寫該允許。如需詳細資訊，請參閱《IAM使用指南》中的工作階段原則。

多種政策類型

將多種政策類型套用到請求時，其結果形成的許可會更為複雜、更加難以理解。若要瞭解如何在涉及多個原則類型時 AWS 決定是否允許要求，請參閱IAM使用指南中的原則評估邏輯。

如何 AWS 服務 使用 IAM

若要取得如何 AWS 服務 使用大部分IAM功能的高階檢視，請參閱《IAM使用者指南》IAM中的使用 AWS 服務。

要了解如何使用特定的 AWS 服務 與IAM，請參閱相關服務的用戶指南的安全部分。

疑難排解 AWS 身分和存取

使用下列資訊可協助您診斷及修正使用和時可能會遇到的 AWS 常見問題IAM。

主題

- 我沒有執行操作的授權 AWS
- 我沒有授權執行 iam：PassRole

- 我想允許我以外的人訪 AWS 帳戶 問我的 AWS 資源

我沒有執行操作的授權 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。

當使用mateojacksonIAM者嘗試使用主控台來檢視虛構*my-example-widget*資源的詳細資料，但沒有虛構的awes:*GetWidget*權限時，就會發生下列範例錯誤。

```
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。

我沒有授權執行 iam：PassRole

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 允許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。

當名為的使用IAM者marymajor嘗試使用主控台執行中的動作時，就會發生下列範例錯誤 AWS。但是，動作請求服務具備服務角色授予的許可。Mary 沒有將角色傳遞至該服務的許可。

```
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。

我想允許我以外的人訪 AWS 帳戶 問我的 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。對於支援以資源為基礎的政策或存取控制清單 (ACLs) 的服務，您可以使用這些政策授與人員存取您的資源。

如需進一步了解，請參閱以下內容：

- 若要瞭解是否 AWS 支援這些功能，請參閱如何 AWS 服務 使用 IAM。
- 若要瞭解如何提供您所擁有資源 AWS 帳戶 的存取權，請參閱《IAM使用者指南》中 AWS 帳戶 的〈提供存取權給其他IAM使用者〉。
- 若要瞭解如何將資源存取權提供給第三方 AWS 帳戶，請參閱《IAM使用指南》中的提供第三方 AWS 帳戶 擁有的存取權。
- 若要瞭解如何透過身分聯盟提供存取權，請參閱《使用指南》中的提供對外部驗證使用IAM者的存取權 (身分聯合)。
- 若要瞭解針對跨帳號存取使用角色與以資源為基礎的政策之間的差異，請參閱《使用IAM者指南》IAM中的〈跨帳號資源存取〉。

本 AWS 產品或服務的合規驗證

若要瞭解 AWS 服務 是否屬於特定規範遵循方案的範圍內，請參閱AWS 服務 遵循規範計劃方案中的，並選擇您感興趣的合規方案。如需一般資訊，請參閱AWS 規範計劃AWS。

您可以使用下載第三方稽核報告 AWS Artifact。如需詳細資訊，請參閱下載中的報告中的 AWS Artifact。

您在使用時的合規責任取決 AWS 服務 於資料的敏感性、公司的合規目標以及適用的法律和法規。AWS 提供下列資源以協助遵循法規：

- 安全性與合規性快速入門指南 — 這些部署指南討論架構考量，並提供部署以安全性和合規性 AWS 為重點的基準環境的步驟。
- 在 Amazon Web Services 上進行HIPAA安全與合規架構 — 本白皮書說明公司如何使用建立符合資格的應 AWS 用程HIPAA式。

> **Note**
>
> 並非所有 AWS 服務 人都HIPAA符合資格。如需詳細資訊，請參閱合HIPAA格服務參考。

- AWS 合規資源AWS — 此工作簿和指南集合可能適用於您的產業和所在地。
- AWS 客戶合規指南 — 透過合規的角度瞭解共同的責任模式。這份指南總結了在多個架構 (包括美國國家標準 AWS 服務 與技術研究所 (NIST)、支付卡產業安全標準委員會 () 和國際標準化組織 ()PCI) 中保護安全控制指引的最佳做法，並將其對應至安全性控制。ISO
- 使用AWS Config 開發人員指南中的規則評估資源 — 此 AWS Config 服務會評估您的資源組態符合內部實務、產業準則和法規的程度。

- AWS Security Hub— 這 AWS 服務 提供了內部安全狀態的全面視圖 AWS。Security Hub 使用安全控制，可評估您的 AWS 資源並檢查您的法規遵循是否符合安全業界標準和最佳實務。如需支援的服務和控制清單，請參閱 Security Hub controls reference。
- Amazon GuardDuty — 透過監控環境中的 AWS 帳戶可疑和惡意活動，藉此 AWS 服務 偵測您的工作負載、容器和資料的潛在威脅。 GuardDuty 可協助您因應各種合規性需求 PCIDSS，例如符合特定合規性架構所要求的入侵偵測需求。
- AWS Audit Manager— 這 AWS 服務 有助於您持續稽核您的 AWS 使用情況，以簡化您管理風險的方式，以及遵守法規和業界標準的方式。

本 AWS 產品或服務透過其支援的特定 Amazon 網路服務 (AWS) 服務，遵循共同的責任模式。如需 AWS 服務安全性資訊，請參閱AWS 服務安全性說明文件頁面和符合性計劃 AWS 遵循工作範圍的 AWS 服務。

本 AWS 產品或服務的復原能力

AWS 全球基礎架構是圍繞 AWS 區域 和可用區域建立的。

AWS 區域 提供多個實體分離和隔離的可用區域，這些區域透過低延遲、高輸送量和高度備援的網路連線。

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。

如需區域和可用區域的相關 AWS 資訊，請參閱AWS 全域基礎結構。

本 AWS 產品或服務透過其支援的特定 Amazon 網路服務 (AWS) 服務，遵循共同的責任模式。如需 AWS 服務安全性資訊，請參閱AWS 服務安全性說明文件頁面和符合性計劃 AWS 遵循工作範圍的 AWS 服務。

本 AWS 產品或服務的基礎架構安全性

此 AWS 產品或服務使用受管理的服務，因此受到 AWS 全球網路安全性的保護。有關 AWS 安全服務以及如何 AWS 保護基礎結構的詳細資訊，請參閱AWS 雲端安全 若要使用基礎架構安全性的最佳做法來設計您的 AWS 環境，請參閱安全性支柱架構良 AWS 好的架構中的基礎結構保護。

您使用 AWS 已發佈的API呼叫，透過網路存取本「 AWS 產品」或「服務」。使用者端必須支援下列專案：

- 傳輸層安全性 (TLS)。我們需要 TLS 1.2 並推薦 TLS 1.3。

- 具有完美前向保密（ ）的密碼套件，例如（短暫的迪菲-赫爾曼PFS）或DHE（橢圓曲線短暫迪菲-赫爾曼）。ECDHE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主體相關聯的秘密存取金鑰來簽署。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。

本 AWS 產品或服務透過其支援的特定 Amazon 網路服務 (AWS) 服務，遵循共同的責任模式。如需 AWS 服務安全性資訊，請參閱AWS 服務安全性說明文件頁面和符合性計劃 AWS 遵循工作範圍的 AWS 服務。

強制執行最低TLS版本

若要在與 AWS 服務通訊時增加安全性，請將設定 AWS SDK for JavaScript 為使用 TLS 1.2 或更新版本。

> **Important**
>
> AWS SDK for JavaScript v3 會自動交涉指定 AWS 服務端點所支援的最高層級TLS版本。您可以選擇性地強制執行應用程式所需的最低TLS版本，例如 TLS 1.2 或 1.3，但請注意，某些 AWS 服務端點不支援 TLS 1.3，因此如果您強制執行 TLS 1.3，某些呼叫可能會失敗。

傳輸層安全性 (TLS) 是網頁瀏覽器和其他應用程式所使用的通訊協定，以確保透過網路交換資料的隱私性和完整性。

TLS在 Node.js 中驗證和強制執行

當您 AWS SDK for JavaScript 搭配 Node.js 使用時，會使用基礎 Node.js 安全性層來設定TLS版本。

Node.js 12.0.0 及更新版本使用開啟 SSL 1.1.1b 的最低版本，此版本支援 1.3。TLS AWS SDK for JavaScript v3 預設會在可用時使用 TLS 1.3，但如果需要，預設為較低的版本。

驗證開啟SSL和的版本 TLS

若要取得您電腦上 Node.js 所SSL使用的開啟版本，請執行下列命令。

```
node -p process.versions
```

清單SSL中的「開啟」版本是 Node.js 使用的版本，如下列範例所示。

```
openssl: '1.1.1b'
```

若要取得您的電腦上 Node.js TLS 使用的版本，請啟動節點外殼，並依序執行下列命令。

```
> var tls = require("tls");
> var tlsSocket = new tls.TLSSocket();
> tlsSocket.getProtocol();
```

最後一個指令會輸出TLS版本，如下列範例所示。

```
'TLSv1.3'
```

Node.js 預設會使用此版本的TLS，TLS如果呼叫不成功，則會嘗試交涉另一個版本的。

強制執行的最低版本 TLS

Node.js 會在呼叫失敗TLS時交涉的版本。您可以在此協商期間強制執行允許的最低TLS版本，無論是從命令列執行指令碼或 JavaScript 程式碼中的每個要求時。

若要從命令列指定最低TLS版本，您必須使用 Node.js 版本 11.0.0 或更新版本。若要安裝特定的 Node.js 版本，請先使用節點版本管理員安裝和更新中的步驟安裝節點版本管理員 (nvm)。然後執行以下命令來安裝和使用特定版本的 Node.js。

```
nvm install 11
nvm use 11
```

Enforce TLS 1.2

若要強制執行 TLS 1.2 為允許的最小版本，請在執行指令碼時指定`--tls-min-v1.2`引數，如下列範例所示。

```
node --tls-min-v1.2 yourScript.js
```

若要為 JavaScript 程式碼中的特定要求指定允許的最小TLS版本，請使用`http0ptions`參數來指定通訊協定，如下列範例所示。

```
import https from "https";
import { NodeHttpHandler } from "@smithy/node-http-handler";
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
```

```
const client = new DynamoDBClient({
    region: "us-west-2",
    requestHandler: new NodeHttpHandler({
        httpsAgent: new https.Agent(
            {
                secureProtocol: 'TLSv1_2_method'
            }
        )
    })
});
```

Enforce TLS 1.3

若要強制執行 TLS 1.3 為允許的最小版本，請在執行指令碼時指定`--tls-min-v1.3`引數，如下列範例所示。

```
node --tls-min-v1.3 yourScript.js
```

若要為 JavaScript 程式碼中的特定要求指定允許的最小TLS版本，請使用`httpOptions`參數來指定通訊協定，如下列範例所示。

```
import https from "https";
import { NodeHttpHandler } from "@smithy/node-http-handler";
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({
    region: "us-west-2",
    requestHandler: new NodeHttpHandler({
        httpsAgent: new https.Agent(
            {
                secureProtocol: 'TLSv1_3_method'
            }
        )
    })
});
```

在瀏覽器腳本TLS中驗證和強制執行

當您在瀏覽器指令碼 JavaScript 中使用 SDK for 時，瀏覽器設定會控制所使TLS用的版本。瀏覽器TLS使用的版本無法通過腳本發現或設置，並且必須由用戶配置。若要驗證並強制執行瀏覽器指令碼中TLS使用的版本，請參閱您特定瀏覽器的指示。

Microsoft Internet Explorer

1. 打開 IE 瀏覽器。
2. 從功能表列選擇 [工具]-[網際網路選項]-[進階] 標籤。
3. 向下滾動到安全類別，手動選中使用 TLS 1.2 的選項框。
4. 按一下 OK (確定)。
5. 關閉瀏覽器並重新啟動 IE 瀏覽器。

Microsoft Edge

1. 在視窗功能表搜尋方塊中，輸入 *Internet options*.
2. 在最符合下，按一下網際網路選項。
3. 在 [網際網路內容] 視窗的 [進階] 索引標籤上，向下捲動至 [安全性] 區段。
4. 勾選「使用者 TLS 1.2」核取方塊。
5. 按一下 OK (確定)。

Google Chrome

1. 打開谷歌瀏覽器。
2. 按一下 Alt F 並選取「設定」。
3. 向下捲動並選取 [顯示進階設定...] 。
4. 向下滾動到系統部分，然後單擊打開代理設置... 。
5. 選取 [進階] 索引標籤。
6. 向下滾動到安全類別，手動選中使用 TLS 1.2 的選項框。
7. 按一下 OK (確定)。
8. 關閉瀏覽器並重新啟動谷歌瀏覽器。

Mozilla Firefox

1. 打開火狐。
2. 在網址列中，輸入「關於:組態」，然後按 Enter 鍵。
3. 在「搜尋」欄位中，輸入 tls。尋找並連按兩下安全性 .tls.min 版本的項目。
4. 將整數值設置為 3 以強制 TLS 1.2 為默認協議。

5. 按一下 OK (確定)。
6. 關閉瀏覽器並重新啟動火狐瀏覽器。

Apple Safari

沒有啟用SSL通訊協定的選項。如果您使用的是 Safari 版本 7 或更高版本，則會自動啟用 TLS 1.2。

從第 2.x 版移轉至第 3.x 版 AWS SDK for JavaScript

AWS SDK for JavaScript 版本 3 是版本 2 的主要重寫。本節說明兩個版本之間的差異，並說明如何從版本 2 移轉至版本 3 SDK 的版本 JavaScript。

使用代碼模式將您SDK的代碼遷移到 JavaScript v3

AWS SDK for JavaScript 版本 3 (v3) 隨附用戶端組態和公用程式的現代化界面，包括登入資料、Amazon S3 多部分上傳、DynamoDB 文件用戶端、服務員等。您可以在 AWS SDK for JavaScript GitHub repo 的遷移指南中找到 v2 中的每個更改的內容和 v3 等價物。

為了充分利用 AWS SDK for JavaScript v3，我們建議您使用下面描述的 codemod 腳本。

使用代碼模式遷移現有的 v2 代碼

中的 codemod 指令碼集合有aws-sdk-js-codemod助於移轉您現有的 AWS SDK for JavaScript (v2) 應用程式以使用 v3 APIs。您可以按如下方式運行轉換。

```
$ npx aws-sdk-js-codemod -t v2-to-v3 PATH...
```

例如，假設您有下列程式碼，它會從 v2 建立 Amazon DynamoDB 用戶端並呼叫`listTables`作業。

```
// example.ts
import AWS from "aws-sdk";

const region = "us-west-2";
const client = new AWS.DynamoDB({ region });
await client.listTables({}).promise()
  .then(console.log)
  .catch(console.error);
```

您可以如下運行我們`example.ts`的`v2-to-v3`變換。

```
$ npx aws-sdk-js-codemod -t v2-to-v3 example.ts
```

轉換會將 DynamoDB 匯入轉換為 v3、建立 v3 用戶端，並依照下列方式呼叫`listTables`作業。

```
// example.ts
import { DynamoDB } from "@aws-sdk/client-dynamodb";
```

```
const region = "us-west-2";
const client = new DynamoDB({ region });
await client.listTables()
  .then(console.log)
  .catch(console.error);
```

我們已針對常見使用案例實作轉換。如果您的代碼無法正確轉換，請創建錯誤報告或功能請求，其中包含示例輸入代碼和觀察/預期的輸出代碼。如果您的特定使用案例已在現有問題中回報，請透過 upvote 展示您的支援。

第 3 版中的新功能

JavaScript (v3) SDK 的版本 3 包含下列新功能。

模塊化軟件包

用戶現在可以為每個服務使用單獨的軟件包。

新的中介軟體堆疊

使用者現在可以使用中介軟體堆疊來控制作業呼叫的生命週期。

此外，寫SDK在中 TypeScript，它具有許多優點，例如靜態類型。

> **Important**
>
> 本指南中 v3 的程式碼範例以 ECMAScript 6 (ES6) 撰寫。ES6帶來了新的語法和新功能，使您的代碼更加現代化和可讀性，並執行更多操作。ES6需要您使用 Node.js 版本 13.x 或更高版本。要下載並安裝最新版本的 Node.js，請參閱 Node.js 下載。如需詳細資訊，請參閱JavaScript ES6/共同語法。

模塊化軟件包

SDKfor JavaScript (v2) 的版本 2 需要您使用整個 AWS SDK，如下所示。

```
var AWS = require("aws-sdk");
```

如SDK果您的應用程序使用許多 AWS 服務，則加載整個不是問題。但是，如果您只需要使用少數 AWS 服務，則意味著使用不需要或不使用的代碼來增加應用程序的大小。

在 v3 中，您只能載入並使用所需的個別 AWS 服務。這會顯示在下列範例中，可讓您存取 Amazon DynamoDB (DynamoDB)。

```
import { DynamoDB } from "@aws-sdk/client-dynamodb";
```

您不僅可以載入和使用個別 AWS 服務，還可以只載入和使用所需的服務命令。這會顯示在下列範例中，可讓您存取 DynamoDB 用戶端和命令。`ListTablesCommand`

```
import {
  DynamoDBClient,
  ListTablesCommand
} from "@aws-sdk/client-dynamodb";
```

Important

您不應該將子模塊導入到模塊中。例如，下列程式碼可能會導致錯誤。

```
import { CognitoIdentity } from "@aws-sdk/client-cognito-identity/
CognitoIdentity";
```

以下是正確的代碼。

```
import { CognitoIdentity } from "@aws-sdk/client-cognito-identity";
```

比較程式碼大小

在版本 2 (v2) 中，列出`us-west-2`區域中所有 Amazon DynamoDB 表的簡單程式碼範例可能如下所示。

```
var AWS = require("aws-sdk");
// Set the Region
AWS.config.update({ region: "us-west-2" });
// Create DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

// Call DynamoDB to retrieve the list of tables
ddb.listTables({ Limit: 10 }, function (err, data) {
  if (err) {
    console.log("Error", err.code);
```

```
  } else {
    console.log("Tables names are ", data.TableNames);
  }
});
```

v3 看起來像下面這樣。

```
import {
  DynamoDBClient,
  ListTablesCommand
} from "@aws-sdk/client-dynamodb";

const dbclient = new DynamoDBClient({ region: "us-west-2" });

try {
  const results = await dbclient.send(new ListTablesCommand);

  for (const item of results.TableNames) {
    console.log(item);
  }
} catch (err) {
  console.error(err)
}
```

該aws-sdk軟件包增加了大約 40 MB 到您的應用程序。替換var AWS = require("aws-sdk")為可將該開銷import {DynamoDB} from "@aws-sdk/client-dynamodb"減少到大約 3 MB。限制只匯入 DynamoDB 用戶端和ListTablesCommand命令，可將額外負荷降低到 100 KB 以下。

```
// Load the DynamoDB client and ListTablesCommand command for Node.js
import {
  DynamoDBClient,
  ListTablesCommand
} from "@aws-sdk/client-dynamodb";
const dbclient = new DynamoDBClient({});
```

在 v3 中調用命令

您可以使用 v2 或 v3 命令在 v3 中執行操作。若要使用 v3 命令，請匯入命令和所需的 AWS 服務套件用戶端，並使用使用異步/等待模式的.send方法執行命令。

若要使用 v2 命令，您可以匯入必要的 AWS 服務套件，並使用回呼或異步/等待模式直接在套件中執行 v2 命令。

使用 v3 命令

v3 會為每個 AWS 服務套件提供一組指令，讓您能夠執行該 AWS 服務的作業。安裝 AWS 服務後，您可以瀏覽項目中的可用命令 `node_modules/@aws-sdk/client-PACKAGE_NAME/commands folder.`

您必須匯入要使用的指令。例如，下列程式碼會載入 DynamoDB 服務和命令。`CreateTableCommand`

```
import { DynamoDB, CreateTableCommand } from "@aws-sdk/client-dynamodb";
```

若要以建議的異步/等待模式呼叫這些命令，請使用下列語法。

```
CLIENT.send(new XXXCommand);
```

例如，下列範例會使用建議的異步/等待模式建立 DynamoDB 資料表。

```
import { DynamoDB, CreateTableCommand } from "@aws-sdk/client-dynamodb";
const dynamodb = new DynamoDB({ region: "us-west-2" });
const tableParams = {
  TableName: TABLE_NAME
};

try {
  const data = await dynamodb.send(new CreateTableCommand(tableParams));
  console.log("Success", data);
} catch (err) {
  console.log("Error", err);
};
```

使用 v2 命令

若要在中使用 v2 命令 JavaScript，請匯入完整的 AWS Service 套件，如下列程式碼所示。SDK

```
const { DynamoDB } = require('@aws-sdk/client-dynamodb');
```

若要以建議的異步/等待模式呼叫 v2 命令，請使用下列語法。

```
client.command(parameters);
```

下列範例使用 v2 命`createTable`令，使用建議的異步/等待模式建立 DynamoDB 資料表。

```
const { DynamoDB } = require('@aws-sdk/client-dynamodb');
const dynamoDB = new DynamoDB({ region: 'us-west-2' });
var tableParams = {
  TableName: TABLE_NAME
};
async function run() => {
  try {
    const data = await dynamoDB.createTable(tableParams);
    console.log("Success", data);
  }
  catch (err) {
    console.log("Error", err);
  }
};
run();
```

下列範例使用 v2 `createBucket` 命令，使用回呼模式建立 Amazon S3 儲存貯體。

```
const { S3 } = require('@aws-sdk/client-s3');
const s3 = new S3({ region: 'us-west-2' });
var bucketParams = {
  Bucket : BUCKET_NAME
};
function run() {
  s3.createBucket(bucketParams, function (err, data) {
    if (err) {
      console.log("Error", err);
    } else {
      console.log("Success", data.Location);
    }
  })
};
run();
```

新的中介軟體堆疊

v2 的可讓您SDK透過將事件偵聽程式附加至要求，在整個生命週期的多個階段中修改要求。這種方法可能會使得在請求的生命週期中難以調試出錯的問題。

在 v3 中，您可以使用新的中介軟體堆疊來控制作業呼叫的生命週期。這種方法提供了幾個好處。堆疊中的每個中介軟體階段會在對要求物件進行任何變更之後，呼叫下一個中介軟體階段。這也使得堆疊中的偵錯問題變得更加容易，因為您可以準確地看到哪些中介軟體階段被呼叫導致錯誤。

下列範例會使用中介軟體將自訂標頭新增至 Amazon DynamoDB 用戶端 (我們先前建立並顯示)。第一個引數是接受的函數`next`，這是堆棧中的下一個中間件階段調用`context`，並且它是一個包含有關被調用操作的一些信息的對象。該函數返回一個接受的函數`args`，該函數是包含傳遞給操作和請求的參數的對象。它返回從調用`args`下一個中間件的結果。

```
dbclient.middlewareStack.add(
  (next, context) => args => {
    args.request.headers["Custom-Header"] = "value";
    return next(args);
  },
  {
    name: "my-middleware",
    override: true,
    step: "build"
  }
);

dbclient.send(new PutObjectCommand(params));
```

AWS SDK for JavaScript v2 和 v3 之間有什麼不同

本節捕獲從 AWS SDK for JavaScript v2 到 v3 的顯著更改。因為 v3 是 v2 的模塊化重寫，v2 和 v3 之間的一些基本概念是不同的。您可以在我們的部落格文章中瞭解這些變更。以下博客文章將幫助您加快速度：

- 模組化封裝 AWS SDK for JavaScript
- 在模組化中介紹中介軟體堆疊 AWS SDK for JavaScript

從 AWS SDK for JavaScript v2 到 v3 的界面變化的摘要如下。目標是幫助您輕鬆找到您已經熟悉的 v2 API 的 v3 等價物。

主題

- 用戶端建構函式
- 憑證提供者
- Amazon S3 考量

- 文件用戶端
- 服務員和簽署者
- 特定服務客戶的注意事項

用戶端建構函式

此列表由 v2 配置參數索引。

- computeChecksums
  - v2：當服務接受承載主體時，是否計算裝載主體的MD5總和檢查 (目前僅在 S3 支援)。
  - v3：S3（PutObject等）的適用命令將自動計算請求有效負載MD5的校驗和。 PutBucketCors您也可以在命令的ChecksumAlgorithm參數中指定不同的總和檢查碼演算法，以使用不同的總和檢查碼演算法。您可以在 S3 功能公告中找到更多資訊。
- convertResponseTypes
  - v2：在解析響應數據時是否轉換類型。
  - V3：已棄用。此選項被認為不是類型安全的，因為它不會轉換響應中的時間戳或 base64 二進製文件之類的類型。JSON
- correctClockSkew
  - v2：是否套用時脈偏斜校正，並重試因為用戶端時鐘傾斜而失敗的要求。
  - V3：已棄用。SDK一律套用時脈偏斜校正。
- systemClockOffset
  - v2：套用至所有簽署時間的位移值 (以毫秒為單位)。
  - 第三版：沒有變化。
- credentials
  - v2：用來簽署請求的 AWS 憑據。
  - 第三版：沒有變化。它也可以是返回憑據的異步函數。如果函數返回一個expiration (Date)，該函數將在到期日期時間臨近時再次調用。有關AwsAuthInputConfig憑據，請API參閱 v3 參考。
- endpointCacheSize
  - v2：儲存端點探查作業中端點的全域快取大小。
  - 第三版：沒有變化。
- endpointDiscoveryEnabled
  - v2：是否通過服務動態給出的端點調用操作。
  - 第三版：沒有變化。
- hostPrefixEnabled

- v2：是否編組請求參數到主機名的前綴。
- V3：已棄用。SDK必要時始終注入主機名前綴。

- `httpOptions`

  一組選項傳遞給低級別的HTTP請求。這些選項在 v3 中以不同的方式彙總。您可以通過提供一個新的配置它們`requestHandler`。以下是在 Node.js 運行時設置 http 選項的示例。您可以在 v3 API 參考中找到更多 NodeHttpHandler。

  默認情況HTTPS下，所有 v3 請求都使用。您只需要提供自定義httpsAgent。

```
const { Agent } = require("https");
const { Agent: HttpAgent } = require("http");
const { NodeHttpHandler } = require("@smithy/node-http-handler");
const dynamodbClient = new DynamoDBClient({
    requestHandler: new NodeHttpHandler({
        httpsAgent: new Agent({
            /*params*/
        }),
        connectionTimeout: /*number in milliseconds*/,
        socketTimeout: /*number in milliseconds*/
    }),
});
```

  如果您傳遞使用 http 的自定義端點，則需要提供httpAgent。

```
const { Agent } = require("http");
const { NodeHttpHandler } = require("@smithy/node-http-handler");

const dynamodbClient = new DynamoDBClient({
    requestHandler: new NodeHttpHandler({
        httpAgent: new Agent({
            /*params*/
        }),
    }),
    endpoint: "http://example.com",
});
```

  如果客戶端在瀏覽器中運行，則可以使用不同的選項集。您可以在 v3 API 參考中找到更多 FetchHttpHandler。

```
const { FetchHttpHandler } = require("@smithy/fetch-http-handler");
```

```
const dynamodbClient = new DynamoDBClient({
    requestHandler: new FetchHttpHandler({
        requestTimeout: /* number in milliseconds */
    }),
});
```

的httpOptions每個選項指定如下：

- proxy
  - v2：通URL過代理請求。
  - v3：您可以在配置 Node.js 的代理之後使用代理設置代理。
- agent
  - v2：用來執行HTTP要求的代理程式物件。用於連接池。
  - v3：您可以配置httpAgent或httpsAgent如上面的示例所示。
- connectTimeout
  - v2：將套接字設置為在connectTimeout毫秒後與服務器建立連接失敗後超時。
  - v3：connectionTimeout在NodeHttpHandler選項中可用。
- timeout
  - v2：在自動終止之前，請求可以花費的毫秒數。
  - v3：socketTimeout在NodeHttpHandler選項中可用。
- xhrAsync
  - v2：是否SDK將發送異步HTTP請求。
  - V3：已棄用。請求始終是異步的。
- xhrWithCredentials
  - v2：設置對XMLHttpRequest象的 withCredentials "" 屬性。
  - v3：不可用。SDK會繼承預設的擷取組態。
- logger
  - v2：響應（如流）或 .write().log()（如控制台對象）以記錄有關請求的信息的對象。
  - 第三版：沒有變化。v3 中提供更精細的記錄檔。
- maxRedirects
  - v2：服務要求所遵循的最大重新導向量。
  - V3：已棄用。SDK不會跟隨重定向以避免意外的跨區域請求。
- maxRetries
  - v2：為服務要求執行的重試次數上限。
  - 第三版：已變更為maxAttempts。在 v3 API 參考中查看更多 RetryInputConfig. 請注意，maxAttempts應該是maxRetries + 1。

- paramValidation
  - v2：在發送請求之前，是否應根據操作說明驗證輸入參數。
  - V3：已棄用。SDK在運行時不會在客戶端上進行驗證。
- region
  - v2：要傳送服務要求的目標區域。
  - 第三版：沒有變化。它也可以是返回區域字符串的異步函數。
- retryDelayOptions
  - v2：一組選項，用於配置可重試錯誤的重試延遲。
  - V3：已棄用。SDK支持與retryStrategy客戶端構造函數選項更靈活的重試策略。在 v3 API 參考中查看更多內容。
- s3BucketEndpoint
  - v2：提供的端點是否位址個別值區 (如果解決根API端點，則為 false)。
  - 第三版：已變更為bucketEndpoint。在 v3 API 參考中查看更多 bucketEndpoint. 請注意，當設置為時true，您在 request 參數中指定Bucket請求端點，原始端點將被覆蓋。而在 v2 中，客戶端構造函數中的請求端點覆蓋Bucket請求參數。
- s3DisableBodySigning
  - v2：是否在使用簽章版本 v4 時停用 S3 主體簽署。
  - v3：重新命名為applyChecksum。
- s3ForcePathStyle
  - v2：是否強制 S3 對象URLs的路徑樣式。
  - v3：重新命名為forcePathStyle。
- s3UseArnRegion
  - v2：是否覆蓋請求的ARN資源推斷的區域的請求區域。
  - v3：重新命名為useArnRegion。
- s3UsEast1RegionalEndpoint
  - v2：當區域設為「us-east-1」時，是否將 s3 請求發送到全球端點還是「us-east-1」區域端點。
  - V3：已棄用。如果區域設定為，S3 用戶端將一律使用區域端點us-east-1。您可以將區域設定為aws-global將請求傳送到 S3 全球端點。
- signatureCache
  - v2：是否緩存用於簽署請求的簽名（覆蓋API配置）。
  - V3：已棄用。SDK總是緩存散列簽名密鑰。
- signatureVersion
  - v2：用來簽署請求的簽名版本（覆蓋API配置）。
  - V3：已棄用。v2 中支持的簽名 V2 SDK 已被棄用 AWS。v3 僅支持簽名 v4。

- sslEnabled
  - v2：SSL是否為請求啟用。
  - v3：重新命名為tls。
- stsRegionalEndpoints
  - v2：是否將 sts 請求發送到全球端點還是區域端點。
  - V3：已棄用。STS如果設定為特定區域，用戶端將一律使用地區端點。您可以將區域設置為aws-global將請求發送到STS全局端點。
- useAccelerateEndpoint
  - v2：是否搭配 S3 服務使用加速端點。
  - 第三版：沒有變化。

憑證提供者

在 v2 中，的 SDK 提供了可 JavaScript 供選擇的認證提供者清單，以及 Node.js 上預設可用的認證提供者鏈結，該鏈會嘗試從所有最常見的提供者載入認 AWS 證。 JavaScript v3 的 SDK 簡化了憑證提供者的介面，使其更容易使用和寫入自訂憑證提供者。在新的憑據提供程序鏈之上， JavaScript v3 的 SDK 都提供了旨在提供相當於 v2 的憑據提供程序列表。

以下是 v2 中的所有憑據提供程序及其在 v3 中的等價物。

預設認證提供者

如果您沒有明確提供認證，則預設 AWS 認證提供者是用於 JavaScript 解析認證的 SDK 的方式。

- v2：CredentialProviderChain在 Node.js 中，按照以下順序解析來源的憑據：
  - 环境变量
  - 共用認證檔
  - ECS 容器認證
  - 產生外部處理
  - 來自指定文件的 OIDC 令牌
  - EC2 實例中繼資料

  如果上述其中一個憑證提供者無法解析 AWS 認證，則鏈結會退回到下一個提供者，直到解析有效的認證為止，並且當所有提供者失敗時，鏈結將拋出錯誤。

  在瀏覽器和 React Native 執行階段中，憑證鏈結是空的，而且必須明確設定認證。
- v3：default Provider。在 v3 中，憑證來源和後援順序不會改變。它還支持AWS IAM Identity Center憑據。

暫時登入資料

- v2：ChainableTemporaryCredentials代表從中擷取的臨時證明資料AWS.STS。
  如果沒有任何額外的參數，認證將從AWS.STS.getSessionToken()操作中獲取。
  如果提供 IAM 角色，則會改為使用此AWS.STS.assumeRole()作業來擷取角色的認證。 AWS.ChainableTemporaryCredentials與處理主認證和重新整理的方式不同。AWS.TemporaryCredentials AWS.ChainableTemporaryCredentials使用使用者傳遞的主要證明資料重新整理過期的證明資料，以支援 STS 證明資料鏈結。但是，在實例化期間AWS.TemporaryCredentials遞歸地收合 MasterLidentity，排除重新整理需要中繼臨時認證的認證的能力。

  原始版本TemporaryCredentials已被棄用，以支持 v2。ChainableTemporaryCredentials
- 第三版：fromTemporaryCredentials。您可以fromTemporaryCredentials()從@aws-sdk/credential-providers包中打電話。範例如下：

```
import { FooClient } from "@aws-sdk/client-foo";
import { fromTemporaryCredentials } from "@aws-sdk/credential-providers"; // ES6
 import
// const { FooClient } = require("@aws-sdk/client-foo");
// const { fromTemporaryCredentials } = require("@aws-sdk/credential-providers"); //
 CommonJS import

const sourceCredentials = {
  // A credential can be a credential object or an async function that returns a
 credential object
};
const client = new FooClient({
  credentials: fromTemporaryCredentials({
    masterCredentials: sourceCredentials,
    params: { RoleArn },
  }),
});
```

Amazon Cognito 身份憑據

從 Amazon Cognito 身分識別服務 (通常用於瀏覽器) 載入登入資料。

- v2：CognitoIdentityCredentials代表使用 Amazon Cognito 身分識別服務從 STS 網路身分聯合擷取的登入資料。

- v3：Cognito Identity Credential Provider該@aws/credential-providers軟件包提供了兩個憑據提供程序函數，其中一個fromCognitoIdentity函數接受身份 ID 和調用cognitoIdentity:GetCredentialsForIdentity，而另一個fromCognitoIdentityPool採用身份池 ID，cognitoIdentity:GetId在第一次調用時調用，然後調fromCognitoIdentity用。後者的後續調用不會重新調用。 GetId

  提供者實作 Amazon Cognito 開發人員指南中所述的「簡化流程」。不支持涉及呼叫cognito:GetOpenIdToken然後調sts:AssumeRoleWithWebIdentity用的「經典流程」。如果您需要，請向我們打開功能請求。

```
// fromCognitoIdentityPool example
import { fromCognitoIdentityPool } from "@aws-sdk/credential-providers"; // ES6
 import
// const { fromCognitoIdentityPool } = require("@aws-sdk/credential-providers"); //
 CommonJS import

const client = new FooClient({
  region: "us-east-1",
  credentials: fromCognitoIdentityPool({
    clientConfig: cognitoIdentityClientConfig, // Optional
    identityPoolId: "us-east-1:1699ebc0-7900-4099-b910-2df94f52a030",
    customRoleArn: "arn:aws:iam::1234567890:role/MYAPP-CognitoIdentity", // Optional
    logins: {
      // Optional
      "graph.facebook.com": "FBTOKEN",
      "www.amazon.com": "AMAZONTOKEN",
      "api.twitter.com": "TWITTERTOKEN",
    },
  }),
});
```

```
// fromCognitoIdentity example
import { fromCognitoIdentity } from "@aws-sdk/credential-providers"; // ES6 import
// const { fromCognitoIdentity } = require("@aws-sdk/credential-provider-cognito-
identity"); // CommonJS import

const client = new FooClient({
  region: "us-east-1",
  credentials: fromCognitoIdentity({
    clientConfig: cognitoIdentityClientConfig, // Optional
    identityId: "us-east-1:128d0a74-c82f-4553-916d-90053e4a8b0f",
```

```
    customRoleArn: "arn:aws:iam::1234567890:role/MYAPP-CognitoIdentity", // Optional
    logins: {
      // Optional
      "graph.facebook.com": "FBTOKEN",
      "www.amazon.com": "AMAZONTOKEN",
      "api.twitter.com": "TWITTERTOKEN",
    },
  }),
});
```

EC2 中繼資料 (IMDS) 登入資料

代表從 Amazon EC2 執行個體上的中繼資料服務收到的登入資料。

- 第二版：EC2MetadataCredentials
- v3: fromInstanceMetadata: 建立將從 Amazon EC2 執行個體中繼資料服務取得登入資料的認證提供者。

```
import { fromInstanceMetadata } from "@aws-sdk/credential-providers"; // ES6 import
// const { fromInstanceMetadata } = require("@aws-sdk/credential-providers"); //
 CommonJS import

const client = new FooClient({
  credentials: fromInstanceMetadata({
    maxRetries: 3, // Optional
    timeout: 0, // Optional
  }),
});
```

ECS 認證

代表從指定 URL 接收的認證。此提供者將從AWS_CONTAINER_CREDENTIALS_RELATIVE_URI或AWS_CONTAINER_CREDENTIALS_FULL_URI環境變數指定的 URI 要求臨時認證。

- 第 2 版：ECSCredentials或RemoteCredentials。
- v3：fromContainerMetadata建立將從 Amazon ECS 容器中繼資料服務取得登入資料的認證提供者。

```
import { fromContainerMetadata } from "@aws-sdk/credential-providers"; // ES6 import

const client = new FooClient({
  credentials: fromContainerMetadata({
    maxRetries: 3, // Optional
    timeout: 0, // Optional
  }),
});
```

檔案系統認證

- v2：FileSystemCredentials代表來自磁盤上 JSON 文件的憑據。
- V3：已棄用。您可以明確讀取 JSON 文件並提供給客戶端。如果您需要，請向我們打開功能請求。

SAML 認證提供者

- v2：SAMLCredentials代表從 STS SAML 支援擷取的認證。
- v3：不可用。如果您需要，請向我們打開功能請求。

共用認證檔案認證

從共用認證檔案載入認證 (預設為環境變數~/.aws/credentials或
由AWS_SHARED_CREDENTIALS_FILE環境變數定義)。此檔案在不同的 AWS SDK 和工具中受到支援。您可以參考共用設定和憑證檔案文件以取得更多資訊。

- 第二版：SharedIniFileCredentials
- 第三版：fromIni。

```
import { fromIni } from "@aws-sdk/credential-providers";
// const { fromIni } from("@aws-sdk/credential-providers");

const client = new FooClient({
  credentials: fromIni({
    configFilepath: "~/.aws/config", // Optional
    filepath: "~/.aws/credentials", // Optional
    mfaCodeProvider: async (mfaSerial) => {
      // implement a pop-up asking for MFA code
```

```
      return "some_code";
    }, // Optional
    profile: "default", // Optional
    clientConfig: { region }, // Optional
  }),
});
```

網路身分認證

使用 OIDC 權杖從磁碟上的檔案擷取認證。它在 EKS 中常用。

- 第 2 版：TokenFileWebIdentityCredentials。
- 第三版：fromTokenFile

```
import { fromTokenFile } from "@aws-sdk/credential-providers"; // ES6 import
// const { fromTokenFile } from("@aws-sdk/credential-providers"); // CommonJS import

const client = new FooClient({
  credentials: fromTokenFile({
    // Optional. If skipped, read from `AWS_ROLE_ARN` environmental variable
    roleArn: "arn:xxxx",
    // Optional. If skipped, read from `AWS_ROLE_SESSION_NAME` environmental variable
    roleSessionName: "session:a",
    // Optional. STS client config to make the assume role request.
    clientConfig: { region },
  }),
});
```

Web 身分同盟認證

從 STS Web 身分同盟支援擷取認證。

- 第二版：WebIdentityCredentials
- 第三版：fromWebToken

```
import { fromWebToken } from "@aws-sdk/credential-providers"; // ES6 import
// const { fromWebToken } from("@aws-sdk/credential-providers"); // CommonJS import
```

```
const client = new FooClient({
  credentials: fromWebToken({
    // Optional. If skipped, read from `AWS_ROLE_ARN` environmental variable
    roleArn: "arn:xxxx",
    // Optional. If skipped, read from `AWS_ROLE_SESSION_NAME` environmental variable
    roleSessionName: "session:a",
    // Optional. STS client config to make the assume role request.
    clientConfig: { region },
  }),
});
```

Amazon S3 考量

Amazon S3 多部分上傳

在 v2 中，Amazon S3 用戶端包含一項操作，該upload()操作支援使用 Amazon S3 提供的多部分上傳功能上傳大型物件。

在 v3 中，該 @aws-sdk/lib-storage軟件包是可用的。它支持 V2 upload() 操作提供的所有功能，並支持 Node.js 和瀏覽器運行時。

Amazon S3 預簽名網址

在 v2 中，Amazon S3 用戶端包含用於產生 URL 的getSignedUrl()和getSignedUrlPromise()操作，讓使用者可以用來從 Amazon S3 上傳或下載物件。

在 v3 中，該 @aws-sdk/s3-request-presigner軟件包是可用的。此套件包含getSignedUrl()和 getSignedUrlPromise()作業的功能。這篇博客文章討論了這個軟件包的詳細信息。

Amazon S3 區域重定向

如果將不正確的區域傳遞給 Amazon S3 用戶端，並擲回後續 PermanentRedirect (狀態 301) 錯誤，v3 中的 Amazon S3 用戶端會支援區域重新導向 (先前稱為 v2 中的 Amazon S3 全球用戶端)。您可以使用用戶端組態中的followRegionRedirects旗標，讓 Amazon S3 用戶端遵循區域重新導向，並支援其作為全球用戶端的功能。

> **Note**
>
> 請注意，當收到狀態為 301 的`PermanentRedirect`錯誤時，此功能可能會導致額外的延遲，因為失敗的請求會以更正的區域重試。只有在您不提前知道儲存貯體的區域時，才應使用此功能。

Amazon S3 串流和緩衝回應

v3 SDK 更喜歡不緩衝潛在的大響應。這在 Amazon S3 `GetObject` 操作中通常遇到，該操作`Buffer`在 v2 中返回一個，但`Stream`在 v3 中返回 a。

對於 Node.js，您必須使用串流或垃圾收集用戶端或其要求處理常式，藉由釋放通訊端來保持對新流量的連線開啟。

```
// v2
const get = await s3.getObject({ ... }).promise(); // this buffers consumes the stream
 already.
```

```
// v3, consume the stream to free the socket
const get = await s3.getObject({ ... }); // object .Body has unconsumed stream
const str = await get.Body.transformToString(); // consumes the stream

// other ways to consume the stream include writing it to a file,
// passing it to another consumer like an upload, or buffering to
// a string or byte array.
```

如需詳細資訊，請參閱通訊端耗盡一節。

文件用戶端

v3 中文件 DynamoDB 端的基本用法

- 在 v2 中，您可以使用`AWS.DynamoDB.DocumentClient`類別來呼叫具有陣列、編號和物件等原生 JavaScript 類型的 DynamoDB API。因此，它透過抽象化屬性值的概念，簡化了 Amazon DynamoDB 中的項目的處理。
- 在 v3 中，等效的用`@aws-sdk/lib-dynamodb`戶端可用。它類似於來自 v3 SDK 的一般服務用戶端，不同之處在於它需要在其建構函式中使用基本 DynamoDB 用戶端。

範例：

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb"; // ES6 import
// const { DynamoDBClient } = require("@aws-sdk/client-dynamodb"); // CommonJS import
import { DynamoDBDocumentClient, PutCommand } from "@aws-sdk/lib-dynamodb"; // ES6
 import
// const { DynamoDBDocumentClient, PutCommand } = require("@aws-sdk/lib-dynamodb"); //
 CommonJS import

// Bare-bones DynamoDB Client
const client = new DynamoDBClient({});

// Bare-bones document client
const ddbDocClient = DynamoDBDocumentClient.from(client); // client is DynamoDB client

await ddbDocClient.send(
  new PutCommand({
    TableName,
    Item: {
      id: "1",
      content: "content from DynamoDBDocumentClient",
    },
  })
);
```

Undefined編組時的值

- 在 v2 中，在 DynamoDB 的封送程序期間，物件中的undefined值會自動省略。
- 在 v3 中，中的預設封送行為@aws-sdk/lib-dynamodb已變更：具有undefined值的物件不再省略。若要與 v2 的功能保持一致，開發人員必須true在 DynamoDB 文件用戶端marshallOptions的中明確設定removeUndefinedValues為。

範例：

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { DynamoDBDocumentClient, PutCommand } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});

// The DynamoDBDocumentClient is configured to handle undefined values properly
const ddbDocClient = DynamoDBDocumentClient.from(client, {
```

```
  marshallOptions: {
    removeUndefinedValues: true
  }
});

await ddbDocClient.send(
  new PutCommand({
    TableName,
    Item: {
      id: "123",
      content: undefined // This value will be automatically omitted
    };
  })
);
```

套件 README 中提供了更多範例和組態。

服務員和簽署者

本頁描述 v3 中服務員和簽署者的使用情況。 AWS SDK for JavaScript

等待程式

在 v2 中，所有服務員都綁定到服務客戶端類，並且您需要在服務員的輸入中指定客戶端將等待的設計狀態。例如，您需`waitFor("bucketExists")`要打電話等待新創建的存儲桶準備就緒。

在 v3 中，如果您的應用程序不需要服務員，則不需要導入服務員。此外，您只能導入您需要等待所需的特定狀態的服務員。因此，您可以減少捆綁包大小並提高性能。以下是在創建後等待存儲桶準備就緒的示例：

```
import { S3Client, CreateBucketCommand, waitUntilBucketExists } from "@aws-sdk/client-
s3"; // ES6 import
// const { S3Client, CreateBucketCommand, waitUntilBucketExists } = require("@aws-sdk/
client-s3"); // CommonJS import

const Bucket = "BUCKET_NAME";
const client = new S3Client({ region: "REGION" });
const command = new CreateBucketCommand({ Bucket });

await client.send(command);
await waitUntilBucketExists({ client, maxWaitTime: 60 }, { Bucket });
```

您可以在 v3 服務員的博客文章中找到如何配置服務員的所有內容。 AWS SDK for JavaScript

Amazon CloudFront 簽署者

在 v2 中，您可以使用. 簽署存取受限制的 Amazon CloudFront 分發的請求`AWS.CloudFront.Signer`。

在 v3 中，您具有`@aws-sdk/cloudfront-signer`包中提供的相同實用程序。

Amazon RDS 簽名者

在 v2 中，您可 Amazon 使用 `AWS.RDS.Signer`.

在 v3 中，類似的實用程序類在 `@aws-sdk/rds-signer`包中可用。

Amazon Polly 簽名者

在 v2 中，您可以為 Amazon Polly 服務合成的語音產生已簽署的 URL。 `AWS.Polly.Presigner`

在 v3 中，類似的實用程序功能在 `@aws-sdk/polly-request-presigner`包中可用。

特定服務客戶的注意事項

AWS Lambda

Lambda 調用響應類型在 v2 和 v3 中有所不同。

```
// v2
import { Lambda } from "@aws-sdk/client-lambda";
import AWS from "aws-sdk";

const lambda = new AWS.Lambda({ REGION });
const invoke = await lambda.invoke({
  FunctionName: "echo",
  Payload: JSON.stringify({ message: "hello" }),
}).promise();

// in v2, Lambda::invoke::Payload is automatically converted to string via a
// specific code customization.
const payloadIsString = typeof invoke.Payload === "string";
console.log("Invoke response payload type is string:", payloadIsString);

const payloadObject = JSON.parse(invoke.Payload);
```

```
console.log("Invoke response object", payloadObject);
```

```
// v3
const lambda = new Lambda({ REGION });
const invoke = await lambda.invoke({
  FunctionName: "echo",
  Payload: JSON.stringify({ message: "hello" }),
});

// in v3, Lambda::invoke::Payload is not automatically converted to a string.
// This is to reduce the number of customizations that create inconsistent behaviors.
const payloadIsByteArray = invoke.Payload instanceof Uint8Array;
console.log("Invoke response payload type is Uint8Array:", payloadIsByteArray);

// To maintain the old functionality, only one additional method call is needed:
// v3 adds a method to the Uint8Array called transformToString.
const payloadObject = JSON.parse(invoke.Payload.transformToString());
console.log("Invoke response object", payloadObject);
```

Amazon SQS

MD5 校验和

若要略過郵件主體的 MD5 總和檢查碼的計算，請在配置物件上設md5定為 false。否則，SDK 預設會計算傳送訊息的總和檢查碼，以及驗證擷取之郵件的總和檢查碼。

```
// Example: Skip MD5 checksum in Amazon SQS
import { SQS } from "@aws-sdk/client-sqs";

new SQS({
  md5: false // note: only available in v3.547.0 and higher
});
```

`QueueUrl`在 Amazon SQS 操作中使用以此作為輸入參數的自訂項目時，可以在 v2 中提供自訂`QueueUrl`來覆寫 Amazon SQS 用戶端的預設端點。

多區域訊息

您應該在 v3 中每個區域使用一個客戶端。該 AWS 區域旨在在客戶端級別進行初始化，並且在請求之間不會更改。

```
import { SQS } from "@aws-sdk/client-sqs";
```

```
const sqsClients = {
  "us-east-1": new SQS({ region: "us-east-1" }),
  "us-west-2": new SQS({ region: "us-west-2" }),
};

const queues = [
  { region: "us-east-1", url: "https://sqs.us-east-1.amazonaws.com/{AWS_ACCOUNT}/
MyQueue" },
  { region: "us-west-2", url: "https://sqs.us-west-2.amazonaws.com/{AWS_ACCOUNT}/
MyOtherQueue" },
];

for (const { region, url } of queues) {
  const params = {
    MessageBody: "Hello",
    QueueUrl: url,
  };
  await sqsClients[region].sendMessage(params);
}
```

自訂端點

在 v3 中，使用自訂端點 (即與預設公有 Amazon SQS 端點不同的端點) 時，您應始終在 Amazon SQS 用戶端和欄位上設定 `QueueUrl`端點。

```
import { SQS } from "@aws-sdk/client-sqs";

const sqs = new SQS({
  // client endpoint should be specified in v3 when not the default public SQS endpoint
 for your region.
  // This is required for versions <= v3.506.0
  // This is optional but recommended for versions >= v3.507.0 (a warning will be
 emitted)
  endpoint: "https://my-custom-endpoint:8000/",
});

await sqs.sendMessage({
  QueueUrl: "https://my-custom-endpoint:8000/1234567/MyQueue",
  Message: "hello",
});
```

如果您不使用自訂端點，則不需要在用戶端`endpoint`上進行設定。

```
import { SQS } from "@aws-sdk/client-sqs";

const sqs = new SQS({
  region: "us-west-2",
});

await sqs.sendMessage({
  QueueUrl: "https://sqs.us-west-2.amazonaws.com/1234567/MyQueue",
  Message: "hello",
});
```

AWS SDK for JavaScript 版本 3 的文件歷史記錄

文件歷史記錄

下表說明AWS SDK for JavaScript自 2020 年 10 月 20 日之後的 V3 版本中的重要變更。如需有關此文件更新的通知，您可以訂閱RSS摘要。

變更	描述	日期
公告	更新了頂部橫幅與互聯網資源管理器 11 的 end-of-support 提醒。	2022 年 9 月 23 日
次要更新	次要更新，以清晰度和解決斷開的鏈接。已新增感知連結AWS SDKs和工具參考指南。	2022 年 8 月 22 日
強制執行最低TLS版本	添加了關於 TLS 1.3 的信息。	2022 年 3 月 31 日
更新 AWS Lambda 教程	已新增教學課程，說明如何建立以瀏覽器為基礎的應用程式，以將資料提交至 Amazon DynamoDB 表格。	2020 年 10 月 20 日
在 Node.js 主題中設置憑據已更新	更新有關在 Node.js 中為 AWS SDK for JavaScript V3 設定認證的主題。	2020 年 10 月 20 日
遷移到第 3 版	已新增說明如何移轉至 AWS SDK for JavaScript v3 的主題。	2020 年 10 月 20 日
開始使用	更新了在瀏覽器中開始使用和開始使用 AWS SDK for JavaScript V3 Node.js 的主題。	2020 年 10 月 20 日

瀏覽器構建	有關 AWS 瀏覽器生成器的信息已被刪除，因為它不是 AWS SDK for JavaScript V3 所需的。	2020 年 10 月 20 日
Amazon Transcribe 服務示例已更新	更新了 V3 的 Amazon Transcribe 服務示例 AWS SDK for JavaScript 。	2020 年 10 月 20 日
Amazon 簡易通知服務服務範例已更新	更新了 AWS SDK for JavaScript V3 的 Amazon 簡單通知服務服務示例。	2020 年 10 月 20 日
Amazon 簡單電子郵件服務服務示例	更新了 AWS SDK for JavaScript V3 的 Amazon 簡單電子郵件服務服務示例。	2020 年 10 月 20 日
Amazon Redshift 服務示例已更新	更新了 AWS SDK for JavaScript V3 的 Amazon Redshift 服務示例。	2020 年 10 月 20 日
Amazon Lex 服務示例更新	更新了 AWS SDK for JavaScript V3 的 Amazon Lex 服務示例。	2020 年 10 月 20 日
Amazon DynamoDB 務範例已更新	已更新 V3 的 Amazon DynamoDB 服務範例 AWS SDK for JavaScript 。	2020 年 10 月 20 日
AWS Elemental MediaConvert 服務範例已更新	更新了 AWS SDK for JavaScript V3 的 AWS Elemental MediaConvert 服務示例。	2020 年 10 月 20 日
AWS Lambda 服務範例已更新	更新了 AWS SDK for JavaScript V3 的 AWS Lambda 服務示例。	2020 年 10 月 20 日

AWS SDK for JavaScript V3 開發人員指南預覽	已發行 AWS SDK for JavaScript V3 開發人員指南的預先發行版本。	2020 年 10 月 19 日