SDK v2 開發人員指南

AWS SDK for JavaScript

AWS SDK for JavaScript: SDK v2 開發人員指南

Table of Contents

..... ix
什麼是 AWS SDK for JavaScript？ 1
開發套件主要版本的維護與支援 1
使用軟體開發套件搭配 Node.js 2
搭配使用 SDK AWS Cloud9 2
搭配 AWS Amplify 使用 SDK 2
使用軟體開發套件搭配 Web 瀏覽器 2
常用案例 3
關於範例 3
開始使用 4
瀏覽器指令碼入門 4
使用案例 4
步驟 1：創建一個 Amazon Cognito 身份池 5
步驟 2：將政策新增至建立的 IAM 角色 6
步驟 3：建立 HTML 頁面 7
步驟 4：編寫瀏覽器指令碼 7
步驟 5：執行範例 9
完整範例 9
可能的增強功能 11
Node.js 入門 11
使用案例 11
先決條件任務 11
第 1 步：安裝 SDK 和依賴關係 12
步驟 2：設定您的認證 12
步驟 3：為專案建立 Package JSON 13
步驟 4：編寫 Node.js 程式碼 14
步驟 5：執行範例 15
使用AWS Cloud9使用適用於 JavaScript 的軟體開發套件 16
步驟 1：設定AWS使用的帳戶AWS Cloud9 16
步驟 2：設定AWS Cloud9開發環境 16
步驟 3：設定適用於 JavaScript 的軟體開發套件 17
設定適用於 Node.js 的軟體 JavaScript 發套件 17
在瀏覽器中設定適用於 JavaScript 的軟體開發套件 18
步驟 4：下載範例程式碼 18

步驟 5：執行範例程式碼和進行除錯 18
設定下列項目的 SDK JavaScript 19
必要條件 19
設定一個 AWS Node.js 環境 19
支援的 Web 瀏覽器 20
安裝 開發套件 21
使用 Bower 進行安裝 22
載入軟體開發套件 22
從第 1 版進行升級 23
自動轉換輸入/輸出上的 Base64 和時間戳記類型 23
移動響應. 數據。 RequestId 回應. 請求 24
公開的包裝函數元素 24
捨棄的用戶端屬性 30
設定下列項目的 SDK JavaScript 31
使用全域組態物件 31
設定全域組態 32
依服務設定組態 33
固定組態資料 34
設定 AWS 區域 34
在用戶端類別建構子 34
使用全域組態物件 34
使用環境變數 35
使用共用組態檔 35
設定區域的優先順序 35
指定自訂端點 36
端點字串格式 36
ap-northeast-3 區域的端點 36
用於的端點 MediaConvert 36
使用 SDK 驗證 AWS 37
啟動 AWS 存取入口網站會話 38
更多認證資訊 38
設定 登入資料 39
登入資料的最佳實務 39
在 Node.js 設定登入資料 40
在 Web 瀏覽器中設定登入資料 44
鎖定 API 版本 53

取得 API 版本 53
Node.js 的考量 54
使用內建 Node.js 模組 54
使用 NPM 套件 54
在 Node.js 中設定 maxSocket 55
在 Node.js 中重複使用 Keep-Alive 的連線 56
為 Node.js 設定代理 57
在 Node.js 中註冊憑證 57
瀏覽器指令碼考量 58
建立適用於瀏覽器的軟體開發套件 58
跨來源資源分享 (CORS) 61
使用 Webpack 進行綁定 65
安裝 Webpack 65
設定 Webpack 65
執行 Webpack 66
使用 Webpack 配套 67
匯入個別服務 68
綁定 Node.js 68
使用 服務 70
建立和呼叫服務物件 71
要求個別服務 71
建立服務物件 72
鎖定服務物件的API版本 73
指定服務物件參數 73
記錄 AWS SDK for JavaScript 通話 74
使用第三方記錄器 74
非同步呼叫服務 75
管理非同步呼叫 75
使用回呼函數 76
使用請求物件事件接聽程式 78
使用 async/await 83
使用 Promise 83
使用回應物件 85
在回應物件中存取傳回的資料 86
瀏覽傳回的資料分頁 87
存取來自回應物件的錯誤資訊 87

存取來源請求物件 87
使用 JSON 88
JSON as Service Object 參數 88
將資料傳回為 JSON 89
SDK對於 JavaScript 代碼示例 91
Amazon CloudWatch 示例 91
在 Amazon 中創建警報 CloudWatch 92
在 Amazon 中使用警報動作 CloudWatch 96
從 Amazon 獲取指標 CloudWatch 100
向 Amazon 活動發送 CloudWatch 活動 103
在 Amazon CloudWatch 日誌中使用訂閱篩選器 108
Amazon DynamoDB 範例 112
在 DynamoDB 中建立和使用表格 113
在 DynamoDB 中讀取和寫入單一項目 117
在 DynamoDB 中 Batch 讀取和寫入項目 121
查詢和掃描 DynamoDB 資料表 124
使用 DynamoDB 用戶端 127
Amazon EC2 範例 133
創建一個 Amazon EC2 實例 134
管理 Amazon EC2 執行個體 136
使用 Amazon EC2 金鑰對 142
搭配 Amazon EC2 使用區域和可用區域 145
在 Amazon EC2 中使用安全群組 147
在 Amazon EC2 中使用彈性 IP 地址 152
MediaConvert 例子 156
獲取特定於區域的端點 156
建立和管理任務 158
使用任務範本 165
AWSIAM 範例 174
管理 IAM 使用者 174
處理 IAM 政策 179
管理 IAM 存取金鑰 184
處理 IAM 伺服器憑證 189
管理 IAM 帳戶別名 193
Amazon Kinesis 示例 196
使用 Amazon Kinesis 擷取網頁捲動進度 197

Amazon S3 範例 204
Amazon S3 瀏覽器示例 204
Amazon S3 的例子 Node.js 232
Amazon SES 範例 252
管理實體 252
使用電子郵件範本 257
發送電子郵件使用 Amazon SES 263
使用 IP 地址篩選條件 269
使用接收規則 273
Amazon SNS 範例 278
管理主題 278
發佈訊息至主題 284
管理訂閱 285
傳送簡訊 291
Amazon SQS 範例 297
在 Amazon SQS 中使用佇列 298
在 Amazon SQS 中傳送和接收訊息 302
在 Amazon SQS 中管理可見性逾時 305
在 Amazon SQS 中啟用長輪詢 307
在 Amazon SQS 中使用無效字母佇列 311
教學課程 314
教學課程：在 Amazon EC2 執行個體上設定 Node.js 314
必要條件 314
程序 314
建立 Amazon Machine Image 316
相關資源 316
API 參考和變更記錄 317
開啟 SDK 變更記錄 GitHub 317
安全 318
資料保護 318
身分和存取權管理 319
物件 319
使用身分驗證 320
使用政策管理存取權 323
AWS 服務 如何使用 IAM 324
對 AWS 身分和存取權進行故障診斷 325

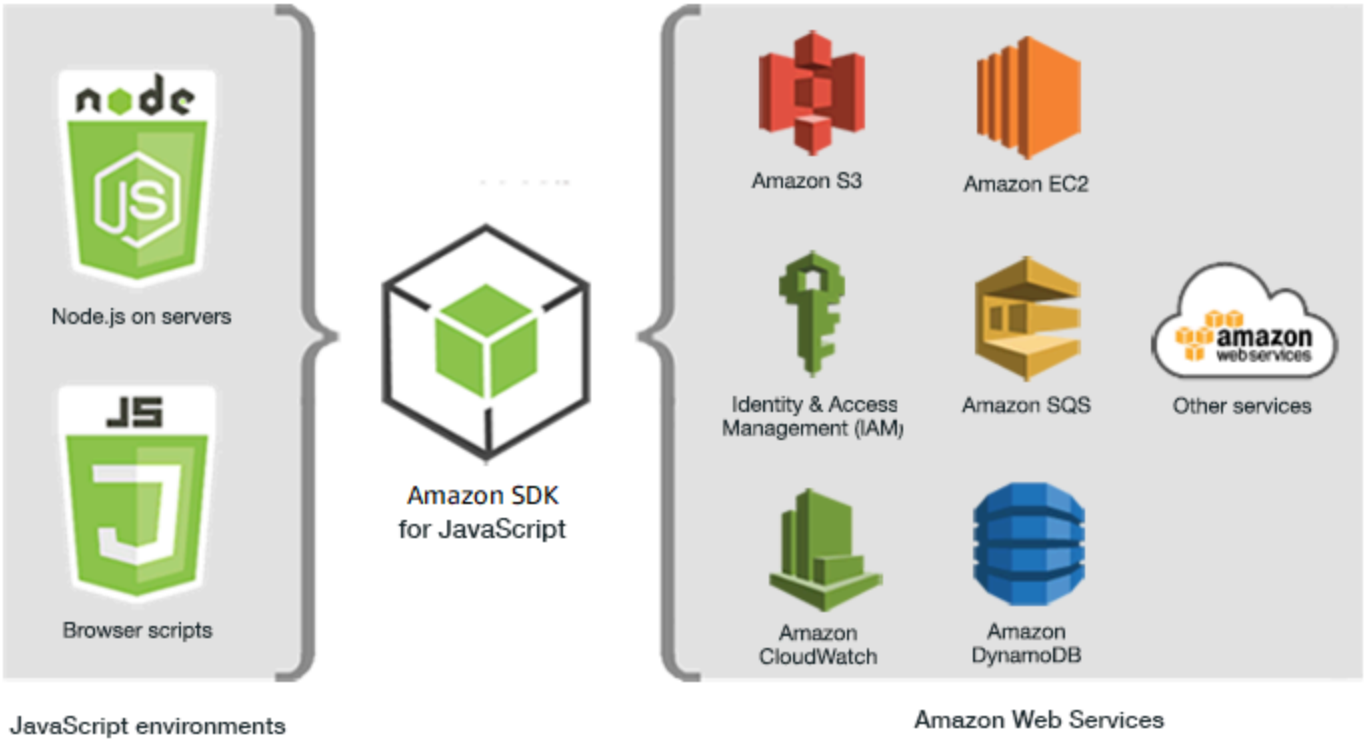
合規驗證 326
恢復能力 327
基礎設施安全性 328
強制執行 的最小版本 TLS 328
在 Node.js TLS中驗證和強制執行 329
在瀏覽器指令碼TLS中驗證和強制執行 331
其他資源 333
AWS軟體開發套件和工具參考指南 333
JavaScript 軟體開發套件 333
JavaScript 上的開發套件和開發人員指南 GitHub 333
JavaScript Gitter 上的開發套件 333
文件歷史記錄 334
文件歷史記錄 334
舊版更新 335

我們宣布即將推出 end-of-support AWS SDK for JavaScript v2。建議您遷移至 AWS SDK for JavaScript v3。如需日期、其他詳細資訊和如何遷移的資訊，請參閱連結公告。

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？

提AWS SDK for JavaScript供服AWS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。



並不是所有服務皆可立即於開發套件中使用。若要瞭解目前支援的服務AWS SDK for JavaScript，請參閱 https://github.com/aws/ aws-sdk-js /BLOB/ 主要服務。有關中的 SDK 的更多內容 JavaScript GitHub，敬請參閱其他資源。

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 開發套件及工具維護政策
- AWS 開發套件及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 Amazon EC2 執行個體上設定 Node.js，以便在伺服器上執行。此外，您也能使用 Node.js 編寫隨需 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。當您使用因 Node.js 和瀏覽器而異的特定 API 時，可以明顯看出這些差別。

搭配使用 SDK 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。如需有關如何用 AWS Cloud9於 Node.js 開發的範例，請參閱使用AWS Cloud9者指南AWS Cloud9中的 Node.js 範例。如需與 SDK AWS Cloud9 搭配使用的詳細資訊 JavaScript，請參閱搭配使用 AWS Cloud9 與 AWS SDK for JavaScript。

搭配 AWS Amplify 使用 SDK

對於以瀏覽器為基礎的 Web、行動應用程式和混合式應用程式，您也可以使用 AWSAmplify 程式庫 GitHub，以擴充 SDK JavaScript，提供宣告式介面。

> **Note**
>
> 框架，如 AWS Amplify 可能不會提供相同的瀏覽器支持作為 JavaScript SDK 的. 如需詳細資訊，請參閱架構文件。

使用軟體開發套件搭配 Web 瀏覽器

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。

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。當您使用因 Node.js 和瀏覽器而異的特定 API 時，可以明顯看出這些差別。

如需 AWS SDK for JavaScript 支援的瀏覽器清單，請參閱 支援的 Web 瀏覽器。

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 JavaScript 事情的幾個想法。

- 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。
- 使用 AWS Lambda 來封裝可從瀏覽器指令碼叫用的專屬邏輯，而不需額外下載和對使用者公開智慧財產權。

關於範例

您可以瀏覽 SDK 以取得AWS程式碼 JavaScript 範例程式庫中的範例。

開始使用 AWS SDK for JavaScript

提 AWS SDK for JavaScript 供瀏覽器指令碼或 Node.js 中對 Web 服務的存取。本節有兩個入門練習，說明如何在這些 JavaScript 環境 JavaScript 中使用 SDK。

您也可以 JavaScript 在 AWS Cloud9 IDE 中使用的 SDK 來開發 Node.js 應用程式。如需有關如何用 AWS Cloud9 於 Node.js 開發的範例，請參閱使用AWS Cloud9 者指南 AWS Cloud9中的 Node.js 範例。

主題

- 瀏覽器指令碼入門
- Node.js 入門

瀏覽器指令碼入門

這個瀏覽器指令碼範例會說明：

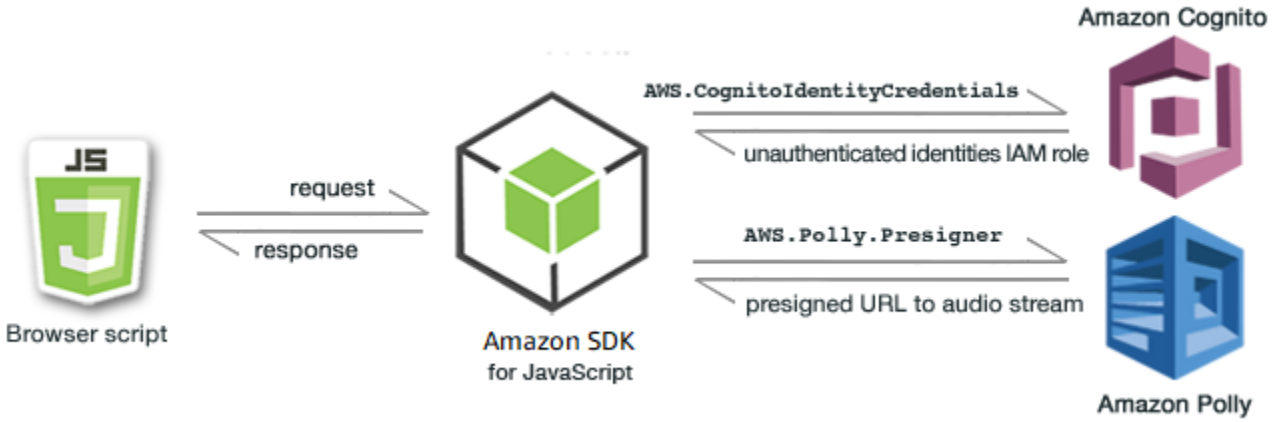
- 如何使用 Amazon Cognito 身份從瀏覽器腳本訪問 AWS 服務。
- 如何使用 Amazon Polly 將文本轉換為合成語音。
- 如何使用 presigner 物件建立預先簽章的 URL。

使用案例

Amazon Polly 是一種雲端服務，可將文字轉換為逼真的語音。您可以使用 Amazon Polly 開發可提高參與度和可存取性的應用程式。Amazon Polly 支援多種語言，並包含各種逼真的聲音。如需 Amazon Polly 的詳細資訊，請參閱 Amazon Polly 開發人員指南。

此範例顯示如何設定和執行簡單的瀏覽器指令碼，該指令碼會擷取您輸入的文字、將該文字傳送至 Amazon Polly，然後傳回文字合成音訊的 URL 供您播放。瀏覽器指令碼使用 Amazon Cognito 身分提供存取 AWS 服務所需的登入資料。您將看到 JavaScript 在瀏覽器腳本中加載和使用 SDK 的基本模式。

> **Note**
>
> 本範例的合成語音播放，必須使用支援 HTML 5 音訊的瀏覽器執行。



瀏覽器腳本使用 SDK 來通過使用以下 API JavaScript 來合成文本：

- `AWS.CognitoIdentityCredentials` 建構函數
- `AWS.Polly.Presigner` 建構函數
- `getSynthesizeSpeechUrl`

步驟 1：創建一個 Amazon Cognito 身份池

在本練習中，您會建立並使用 Amazon Cognito 身分集區，為 Amazon Polly 服務提供對瀏覽器指令碼的未經驗證存取。建立身分集區也會建立兩個 IAM 角色，一個用於支援身分識別提供者驗證的使用者，另一個用於支援未驗證的來賓使用者。

在本範例中，我們只會使用未經驗證的使用者角色，以專注進行任務重點。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

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，getStartedPool)。
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：編寫瀏覽器指令碼

建立 Amazon Cognito 身分集區後，您就可以為瀏覽器指令碼所需的 Amazon Polly 新增許可。

步驟 2：將政策新增至建立的 IAM 角色

若要啟用對 Amazon Polly 的瀏覽器指令碼存取以進行語音合成，請使用為 Amazon Cognito 身分集區建立的未經驗證 IAM 角色。這需要您將 IAM 政策新增至角色。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。

若要將 Amazon Polly 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，getStartedRole)，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，尋找並選取的核取方塊AmazonPollyReadOnly。

> **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。

建立 Amazon Cognito 身分集區，並將 Amazon Polly 的許可新增至未經驗證使用者的 IAM 角色後，您就可以準備好建立網頁和瀏覽器指令碼。

步驟 3：建立 HTML 頁面

範例應用程式是由單一 HTML 頁面所組成，其中包含使用者界面和瀏覽器指令碼。若要開始，請建立 HTML 文件並將下列內容複製到該文件。該頁面包括輸入欄位和按鈕、用來播放合成語音的 <audio> 元素，以及用來顯示訊息的 <p> 元素。(請注意，本頁面底部會顯示完整範例。)

如需 <audio> 元素的詳細資訊，請參閱音訊。

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>AWS SDK for JavaScript - Browser Getting Started Application</title>
  </head>

  <body>
    <div id="textToSynth">
      <input autofocus size="23" type="text" id="textEntry" value="It's very good to
 meet you."/>
      <button class="btn default" onClick="speakText()">Synthesize</button>
      <p id="result">Enter text above then click Synthesize</p>
    </div>
    <audio id="audioPlayback" controls>
      <source id="audioSource" type="audio/mp3" src="">
    </audio>
    <!-- (script elements go here) -->
 </body>
</html>
```

儲存 HTML 檔案，然後將其命名為 polly.html。當您為應用程式建立使用者界面後，就能夠開始新增可執行該應用程式的瀏覽器指令碼程式碼。

步驟 4：編寫瀏覽器指令碼

創建瀏覽器腳本時要做的第一件事是 JavaScript 通過在頁面中的<script>元素後面添加<audio>元素來包含 SDK。若要尋找目前的 SDK_ 版本編號，請參閱 API 參考指南中適用於 SDK 的 API 參考。JavaScript AWS SDK for JavaScript

```
<script src="https://sdk.amazonaws.com/js/aws-sdk-SDK_VERSION_NUMBER.min.js"></script>
```

接著，在軟體開發套件項目後面新增新的 <script type="text/javascript"> 元素。您需要將這個元素新增至瀏覽器指令碼。設置 SDK 的 AWS 區域和憑據。接下來，建立一個名為 speakText() 的函數，此函數會成為按鈕叫用的事件處理常式。

若要與 Amazon Polly 合成語音，您必須提供各種參數，包括輸出的聲音格式、取樣率、要使用的語音 ID 以及要播放的文字。在您最初建立參數時，請將 Text: 參數設定為空白字串；系統隨後會將 Text: 參數設定為您從網頁 <input> 元素擷取的數值。將下列程式碼#####和##取代為中註明的值。步驟 1：創建一個 Amazon Cognito 身份池

```
    <script type="text/javascript">

        // Initialize the Amazon Cognito credentials provider
        AWS.config.region = 'REGION';
        AWS.config.credentials = new AWS.CognitoIdentityCredentials({IdentityPoolId:
'IDENTITY_POOL_ID'});

        // Function invoked by button click
        function speakText() {
            // Create the JSON parameters for getSynthesizeSpeechUrl
            var speechParams = {
                OutputFormat: "mp3",
                SampleRate: "16000",
                Text: "",
                TextType: "text",
                VoiceId: "Matthew"
            };
            speechParams.Text = document.getElementById("textEntry").value;
```

Amazon Polly 將合成語音作為音頻流返回。在瀏覽器中播放該音頻的最簡單方法是讓 Amazon Polly 在預先簽署的 URL 上提供音頻，然後您可以將其設置為網頁中<audio>元素的src屬性。

請建立新的 AWS.Polly 服務物件。然後，建立要用來建構預先簽章 URL 的 AWS.Polly.Presigner 物件，以便從中擷取合成語音音訊。您必須將定義的語音參數及建立的 AWS.Polly 服務物件傳遞至 AWS.Polly.Presigner 建構函數。

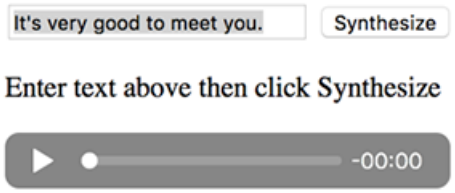
Presigner 物件建立完畢後，請呼叫該物件的 getSynthesizeSpeechUrl 方法來傳遞語音參數。如果成功，這個方法會傳回合成語音的 URL，您稍後能將其指派給 <audio> 元素以供播放使用。

```
            // Create the Polly service object and presigner object
            var polly = new AWS.Polly({apiVersion: '2016-06-10'});
            var signer = new AWS.Polly.Presigner(speechParams, polly)
```

```
            // Create presigned URL of synthesized speech file
            signer.getSynthesizeSpeechUrl(speechParams, function(error, url) {
            if (error) {
                document.getElementById('result').innerHTML = error;
            } else {
                document.getElementById('audioSource').src = url;
                document.getElementById('audioPlayback').load();
                document.getElementById('result').innerHTML = "Speech ready to play.";
            }
          });
        }
    </script>
```

步驟 5：執行範例

請將 `polly.html` 載入 Web 瀏覽器，即可執行範例應用程式。瀏覽器中出現的內容應如下所示。



在輸入方塊中輸入您想轉成語音的詞句，接著選擇 Synthesize (合成)。當音訊準備好播放時，將出現一則訊息。您可以使用音訊播放器控制項來聆聽合成語音。

完整範例

以下是含有瀏覽器指令碼的完整 HTML 頁面，它也可以在這裡使用 GitHub。

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>AWS SDK for JavaScript - Browser Getting Started Application</title>
  </head>

  <body>
    <div id="textToSynth">
      <input autofocus size="23" type="text" id="textEntry" value="It's very good to
 meet you."/>
      <button class="btn default" onClick="speakText()">Synthesize</button>
```

```
      <p id="result">Enter text above then click Synthesize</p>
    </div>
    <audio id="audioPlayback" controls>
      <source id="audioSource" type="audio/mp3" src="">
    </audio>
    <script src="https://sdk.amazonaws.com/js/aws-sdk-2.410.0.min.js"></script>
    <script type="text/javascript">

        // Initialize the Amazon Cognito credentials provider
        AWS.config.region = 'REGION';
        AWS.config.credentials = new AWS.CognitoIdentityCredentials({IdentityPoolId:
 'IDENTITY_POOL_ID'});

        // Function invoked by button click
        function speakText() {
            // Create the JSON parameters for getSynthesizeSpeechUrl
            var speechParams = {
                OutputFormat: "mp3",
                SampleRate: "16000",
                Text: "",
                TextType: "text",
                VoiceId: "Matthew"
            };
            speechParams.Text = document.getElementById("textEntry").value;

            // Create the Polly service object and presigner object
            var polly = new AWS.Polly({apiVersion: '2016-06-10'});
            var signer = new AWS.Polly.Presigner(speechParams, polly)

            // Create presigned URL of synthesized speech file
            signer.getSynthesizeSpeechUrl(speechParams, function(error, url) {
            if (error) {
                document.getElementById('result').innerHTML = error;
            } else {
                document.getElementById('audioSource').src = url;
                document.getElementById('audioPlayback').load();
                document.getElementById('result').innerHTML = "Speech ready to play.";
            }
          });
        }
    </script>
  </body>
</html>
```

可能的增強功能

以下是此應用程序的變化，您可以用來在瀏覽器腳本 JavaScript 中進一步探索使用 SDK。

- 實驗其他音訊輸出格式。
- 添加選項以選擇任何由 Amazon Polly 提供的各種聲音。
- 整合 Facebook 或 Amazon 等身分提供者，以便與經過身份驗證的 IAM 角色搭配使用。

Node.js 入門

這個 Node.js 程式碼範例會說明：

- 如何為專案建立 `package.json` 資訊清單。
- 如何安裝並加入專案所使用的模組。
- 如何從`AWS.S3`用戶端類別建立 Amazon 簡易儲存服務 (Amazon S3) 服務物件。
- 如何建立 Amazon S3 儲存貯體並將物件上傳到該儲存貯體。

使用案例

此範例顯示如何設定和執行建立 Amazon S3 儲存貯體的簡單 Node.js 模組，然後在其中新增文字物件。

由於 Amazon S3 中的儲存貯體名稱必須是全域唯一的，因此此範例包含第三方 Node.js 模組，可產生唯一 ID 值，您可以將其合併到儲存貯體名稱中。這個額外的模組名為 `uuid`。

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：

- 建立可用來開發 Node.js 模組的工作目錄，並將該目錄命名為 `awsnodesample`。請注意，目錄必須建立在可由應用程式更新的位置。例如，請勿在 Windows 環境的「C:\Program Files」下建立目錄。

- 安裝 Node.js。如需詳細資訊，請參閱 Node.js 網站。您可以前往 https://nodejs.org/en/download/current/，尋找並下載適用於各種作業系統的 Node.js 目前版本和 LTS 版本。

內容

- 第 1 步：安裝 SDK 和依賴關係
- 步驟 2：設定您的認證
- 步驟 3：為專案建立 Package JSON
- 步驟 4：編寫 Node.js 程式碼
- 步驟 5：執行範例

第 1 步：安裝 SDK 和依賴關係

您可以使用 npm (Node.js JavaScript 套件管理員) 來安裝套件的 SDK。

移至套件中的 `awsnodesample` 目錄，並將下列指令輸入命令列。

```
npm install aws-sdk
```

此命令會在您的專案 JavaScript 中安裝 SDK，並更新`package.json`以將 SDK 列為專案相依性。您可以在 npm 網站搜尋「aws-sdk」，找到有關此套件的資訊。

接下來，請將 `uuid` 模組安裝至專案，方法是在命令列中輸入下列指令；該指令會隨即安裝模組並更新 `package.json`。如需 `uuid` 的詳細資訊，請參閱 https://www.npmjs.com/package/uuid 上的模組頁面。

```
npm install uuid
```

系統會在專案的 `node_modules` 子目錄中安裝這些套件及其相關聯的程式碼。

如需有關安裝 Node.js 套件的詳細資訊，請參閱在本機下載和安裝套件和在 npm (Node.js 套件管理員) 網站上建立 Node.js 模組。如需有關下載和安裝的資訊 AWS SDK for JavaScript，請參閱為下列項目安裝 SDK JavaScript。

步驟 2：設定您的認證

您需要提供認證，以 AWS 便 SDK 只能存取您的帳戶及其資源。如需取得帳戶登入資料的詳細資訊，請參閱使用 SDK 驗證 AWS。

建議您建立一個共用登入資料檔案，以便保留此資訊。如要瞭解如何作業，請參閱從共用登入資料檔案中在 Node.js 中載入登入資料。登入資料檔案應該如下方範例所示。

```
[default]
aws_access_key_id = YOUR_ACCESS_KEY_ID
aws_secret_access_key = YOUR_SECRET_ACCESS_KEY
```

您可以使用 Node.js 執行下列程式碼，判斷是否已正確設定認證：

```
var AWS = require("aws-sdk");

AWS.config.getCredentials(function(err) {
  if (err) console.log(err.stack);
  // credentials not loaded
  else {
    console.log("Access key:", AWS.config.credentials.accessKeyId);
  }
});
```

同樣地，如果您已在config檔案中正確設定區域，則可以將AWS_SDK_LOAD_CONFIG環境變數設定為任何值並使用下列程式碼來顯示該值：

```
var AWS = require("aws-sdk");

console.log("Region: ", AWS.config.region);
```

步驟 3：為專案建立 Package JSON

awsnodesample 專案目錄建立完畢後，您便能建立和新增 package.json 檔案，藉此保留 Node.js 專案的中繼資料。如需有關在 Node.js 專案package.json中使用的詳細資訊，請參閱建立封裝. json 檔案。

在專案目錄中，建立稱為 package.json 的新檔案。接著，將下列 JSON 新增至檔案。

```
{
  "dependencies": {},
  "name": "aws-nodejs-sample",
  "description": "A simple Node.js application illustrating usage of the SDK for
 JavaScript.",
  "version": "1.0.1",
```

```
  "main": "sample.js",
  "devDependencies": {},
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "NAME",
  "license": "ISC"
}
```

儲存檔案。在您安裝所需模組時，系統會完成檔案的 dependencies 部分。您可以在這裡找到一個 JSON 文件，該文件顯示了這些依賴關係的示例 GitHub。

步驟 4：編寫 Node.js 程式碼

建立名為 sample.js 的新檔案，其中包含範例程式碼。首先添加require函數調用以包含用於 JavaScript和uuid模塊的 SDK，以便您可以使用它們。

在這種情況下，透過將唯一 ID 值附加到可識別的前綴，建立用於建立 Amazon S3 儲存貯體的唯一儲存貯體名稱。'node-sdk-sample-'請呼叫 uuid 模組來產生唯一 ID。然後，為 Key 參數建立一個名稱，該參數可用來上傳物件至儲存貯體。

建立 promise 物件，以便呼叫 AWS.S3 服務物件的 createBucket 方法。收到成功回應後，建立將文字上傳至新建儲存貯體所需的參數。使用另一個 promise 來呼叫 putObject 方法，並上傳文字物件至儲存貯體。

```
// Load the SDK and UUID
var AWS = require("aws-sdk");
var uuid = require("uuid");

// Create unique bucket name
var bucketName = "node-sdk-sample-" + uuid.v4();
// Create name for uploaded object key
var keyName = "hello_world.txt";

// Create a promise on S3 service object
var bucketPromise = new AWS.S3({ apiVersion: "2006-03-01" })
  .createBucket({ Bucket: bucketName })
  .promise();

// Handle promise fulfilled/rejected states
bucketPromise
  .then(function (data) {
```

```
    // Create params for putObject call
    var objectParams = {
      Bucket: bucketName,
      Key: keyName,
      Body: "Hello World!",
    };
    // Create object upload promise
    var uploadPromise = new AWS.S3({ apiVersion: "2006-03-01" })
      .putObject(objectParams)
      .promise();
    uploadPromise.then(function (data) {
      console.log(
        "Successfully uploaded data to " + bucketName + "/" + keyName
      );
    });
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

您可以在這裡找到此範例程式碼 GitHub。

步驟 5：執行範例

輸入下列命令執行範例。

```
node sample.js
```

如果上傳成功，您就會在命令列看到確認訊息。您也能夠在 Amazon S3 主控台中尋找儲存貯體，以及上傳的文字物件。

搭配使用 AWS Cloud9 與 AWS SDK for JavaScript

您可以使用AWS Cloud9使用AWS SDK for JavaScript在瀏覽器程式碼中寫入和執行 JavaScript，也能寫入和執行您的 Node.js 程式碼和進行除錯，這些動作只要透過瀏覽器即可執行。AWS Cloud9包含了程式碼編輯器和終端等工具，另外還有 Node.js 程式碼的調試器。由於 AWS Cloud9 IDE 為雲端式，因此您可以在辦公室、家中或任何地點，使用連線到網際網路的機器，來進行專案的作業。若要取得的一般資訊，AWS Cloud9，請參AWS Cloud9使用者指南。

請按照下列步驟設定AWS Cloud9，取得適用於 JavaScript 的軟體開發套件：

內容

- 步驟 1：設定AWS使用的帳户AWS Cloud9
- 步驟 2：設定AWS Cloud9開發環境
- 步驟 3：設定適用於 JavaScript 的軟體開發套件
  - 設定適用於 Node.js 的軟體 JavaScript 發套件
  - 在瀏覽器中設定適用於 JavaScript 的軟體開發套件
- 步驟 4：下載範例程式碼
- 步驟 5：執行範例程式碼和進行除錯

步驟 1：設定AWS使用的帳户AWS Cloud9

開始使用AWS Cloud9，方法是登入AWS Cloud9主控台作為AWS Identity and Access Management(IAM) 實體（如 IAM 用户），具備AWS Cloud9在您的AWS帳户。

若要在您的AWS帳户來訪問AWS Cloud9，然後登錄到AWS Cloud9主控台，請參的小組設定AWS Cloud9中的AWS Cloud9使用者指南。

步驟 2：設定AWS Cloud9開發環境

當您登入 AWS Cloud9 主控台之後，使用主控台建立 AWS Cloud9 開發環境。在建立環境之後，AWS Cloud9 會開啟該環境的整合開發環境 (IDE)。

請參閱在 中建立環境AWS Cloud9中的AWS Cloud9使用者指南，取得詳細資訊。

> **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。此選項告訴AWS Cloud9來創建環境、啟動 Amazon EC2 實例，然後將新的實例連接到新的環境。這是開始使用 AWS Cloud9 最快的方式。

步驟 3：設定適用於 JavaScript 的軟體開發套件

AfterAWS Cloud9，請按照下列其中一個或兩個過程，使用 IDE 在環境中設定適用於 JavaScript 的開發套件。

設定適用於 Node.js 的軟體 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，來使用 npm 來安裝適用於 JavaScript 的軟體開發套件。

```
npm install aws-sdk
```

如果 IDE 找不到 npm，請依序逐一執行下列命令，以便安裝 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 不負責控制以下程式碼。在您執行前，請務必驗證其真確性及完整性。您可以在這裡找到與此程式碼的更多相關資訊：nvm GitHub 儲存庫。

```
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).
```

在瀏覽器中設定適用於 JavaScript 的軟體開發套件

您無需另行安裝適用於 JavaScript 的開發套件，也能在瀏覽器指令碼中使用。您可以將適用於 JavaScript 的託管軟體開發套件包裝到AWS，使用 HTML 頁面上的指令碼。

您可以從 GitHub 下載適用於 JavaScript 的當前軟體開發套件的精簡和非精簡分發版本，https://github.com/aws/aws-sdk-js/tree/master/dist。

步驟 4：下載範例程式碼

使用您在上一個步驟中所開啟的終端，來將適用於 JavaScript 的軟體開發套件的範例程式碼下載到AWS Cloud9開發環境。(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。此命令會下載官方網站上使用的所有程式碼範例的副本。AWS開發套件文件到環境的根目錄。

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

要查找適用於 JavaScript 的軟件開發工具包的代碼示例，請使用環境窗口以打開*ENVIRONMENT_NAME*\aws-doc-sdk-examples\javascript\example_code，其中####是您的名稱AWS Cloud9開發環境。

若要瞭解如何使用這些程式碼範例和其他範例，請參適用於 JavaScript 的開發套件程式碼範例。

步驟 5：執行範例程式碼和進行除錯

若要在AWS Cloud9開發環境，請參執行您的程式碼中的AWS Cloud9使用者指南。

若要調試 Node.js 代碼，請參閱除錯您的程式碼中的AWS Cloud9使用者指南。

設定下列項目的 SDK JavaScript

本節中的主題說明如何安裝 SDK，以便 JavaScript 在網頁瀏覽器和 Node.js 中使用。該主題還會示範如何載入軟體開發套件，以便存取軟體開發套件支援的 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
- 為下列項目安裝 SDK JavaScript
- 載入下列項目的 SDK JavaScript
- JavaScript 從版本 1 升級的 SDK

必要條件

在使用之前 AWS SDK for JavaScript，請確定您的程式碼是否需要在 Node.js 或網頁瀏覽器中執行。之後，請執行下列操作：

- 若要使用 Node.js，請在伺服器上安裝 Node.js (如果尚未安裝)。
- 若要使用 Web 瀏覽器，則請識別需要支援的瀏覽器版本。

主題

- 設定一個 AWS Node.js 環境
- 支援的 Web 瀏覽器

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：

- 選擇已預先安裝 Node.js 的 Amazon 機器映像 (AMI)，並使用該 AMI 建立一個 Amazon EC2 執行個體。建立您的亞馬遜 EC2 執行個體時，請從中選擇您的 AMI AWS Marketplace。搜 AWS Marketplace 尋 Node.js 並選擇包含預先安裝的 Node.js (32 位元或 64 位元) 版本的 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，請參閱教學課程：在 Amazon EC2 執行個體上設定 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. 如需將 Node.js 搭配 Elastic Beanstalk 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〉。

支援的 Web 瀏覽器

的 SDK 支 JavaScript 援所有現代網頁瀏覽器，包括以下最低版本：

瀏覽器	版本
Google Chrome	28.0 版及更新版本
Mozilla Firefox	26.0 版及更新版本
Opera	17.0 版及更新版本
Microsoft Edge	25.10 版及更新版本
Windows Internet Explorer	N/A
Apple Safari	第 5 版及更新版本
Android 瀏覽器	4.3 版及更新版本

> **Note**
>
> 框架，如 AWS Amplify 可能不會提供相同的瀏覽器支持作為 JavaScript SDK 的. 如需詳細資訊，請參閱架構文件。

為下列項目安裝 SDK JavaScript

您是否以及如何安裝，取 AWS SDK for JavaScript 決於程式碼是否在 Node.js 模組或瀏覽器指令碼中執行。

並不是所有服務皆可立即於開發套件中使用。若要瞭解目前支援的服務 AWS SDK for JavaScript，請參閱 https://github.com/aws/ aws-sdk-js /BLOB/ 主要服務

Node

若要安裝 Node.js 的偏好方式是使用 npm，也就是 Node.js 套件管理員。 AWS SDK for JavaScript 若要這麼做，請在命令列中輸入以下指令。

```
npm install aws-sdk
```

如果您看見以下錯誤訊息：

```
npm WARN deprecated node-uuid@1.4.8: Use uuid module instead
```

請在命令列中輸入這些命令：

```
npm uninstall --save node-uuid
npm install --save uuid
```

Browser

您無需另行安裝，也能在瀏覽器指令碼中使用軟體開發套件。您可以使用 HTML 頁面中的指令碼直接從 Amazon Web Services 載入託管的 SDK 套件。託管 SDK 套件支援強制執行跨來源資源共用 (CORS) 的 AWS 服務子集。如需詳細資訊，請參閱 載入下列項目的 SDK JavaScript。

您可以建立軟體開發套件的自訂建置，然後在當中選取特定 Web 服務和要使用的版本。接著，下載適用於本機開發的自訂軟體開發套件封裝並予以託管，以供應用程式使用。如需建立軟體開發套件自訂建置的詳細資訊，請參閱建立適用於瀏覽器的軟體開發套件。

您可以從以下位置下載當前的縮小和非縮小分發版本： AWS SDK for JavaScript GitHub

https://github.com/aws/ aws-sdk-js /樹/主/區

使用 Bower 進行安裝

Bower 是適用於 Web 的套件管理工具。Bower 安裝完畢後，您即可使用該工具來安裝軟體開發套件。若要使用 Bower 安裝軟體開發套件，請在終端機視窗中輸入以下指令：

```
bower install aws-sdk-js
```

載入下列項目的 SDK JavaScript

載入 SDK 的方式 JavaScript 取決於您要載入它以在網頁瀏覽器或 Node.js 中執行。

並不是所有服務皆可立即於開發套件中使用。若要瞭解目前支援的服務 AWS SDK for JavaScript，請參閱 https://github.com/aws/ aws-sdk-js /BLOB/ 主要服務

Node.js

安裝 SDK 之後，您可以使用將 AWS 套件載入節點應用程式中`require`。

```
var AWS = require('aws-sdk');
```

React Native

若要在 React Native 專案中運用軟體開發套件，請先透過 npm 安裝軟體開發套件：

```
npm install aws-sdk
```

在應用程式中使用下列程式碼，以參考 React Native 相容的軟體開發套件版本：

```
var AWS = require('aws-sdk/dist/aws-sdk-react-native');
```

Browser

開始使用 SDK 的最快方法是直接從 Amazon Web Services 載入託管的 SDK 套件。若要這麼做，請以下列格式，將 `<script>` 元素加入您的 HTML 網頁：

```
<script src="https://sdk.amazonaws.com/js/aws-sdk-SDK_VERSION_NUMBER.min.js"></
script>
```

若要尋找目前的 SDK_ 版本編號，請參閱 API 參考指南中適用於 SDK 的 API 參考。 JavaScript AWS SDK for JavaScript

當您將軟體開發套件載入至頁面後，便可從全域變數 AWS (或 window.AWS) 取得該軟體開發套件。

如果您是使用 browserify 來綁定程式碼和模組相依性，即可使用 require 載入軟體開發套件，如同在 Node.js 中的做法一般。

JavaScript 從版本 1 升級的 SDK

下列注意事項可協助您將 SDK JavaScript 從版本 1 升級至版本 2。

自動轉換輸入/輸出上的 Base64 和時間戳記類型

現在，軟體開發套件能代表使用者自動編碼 base64 編碼值和時間戳記值，也可進行解碼。如果 base64 或時間戳記值是藉由請求傳送，或是允許 base64 編碼值的回應所傳回，則上述變更會影響這些操作。

您不再需要先前轉換為 base64 的使用者程式碼，系統現在會在伺服器回應中以緩衝物件的形式傳回 base64 編碼值，也會將其做為緩衝輸入傳遞。例如，下方的第 1 版 SQS.sendMessage 參數：

```
var params = {
   MessageBody: 'Some Message',
   MessageAttributes: {
      attrName: {
         DataType: 'Binary',
         BinaryValue: new Buffer('example text').toString('base64')
      }
   }
};
```

可以重寫如下。

```
var params = {
   MessageBody: 'Some Message',
```

```
   MessageAttributes: {
      attrName: {
         DataType: 'Binary',
         BinaryValue: 'example text'
      }
   }
};
```

以下是系統讀取訊息的方式。

```
sqs.receiveMessage(params, function(err, data) {
  // buf is <Buffer 65 78 61 6d 70 6c 65 20 74 65 78 74>
  var buf = data.Messages[0].MessageAttributes.attrName.BinaryValue;
  console.log(buf.toString()); // "example text"
});
```

移動響應. 數據。 RequestId 回應. 請求

現在，軟體開發套件會在 `response` 物件上的同一位置存放所有服務請求 ID，而不是存放在 `response.data` 屬性內部。針對以不同方式公開請求 ID 的所有服務，這項改變能提高一致性。另外，將 `response.data.RequestId` 屬性重新命名為 `response.requestId` (回呼函數內的 `this.requestId`)，也是一項重大變更。

請在程式碼中變更以下內容：

```
svc.operation(params, function (err, data) {
  console.log('Request ID:', data.RequestId);
});
```

變更為：

```
svc.operation(params, function () {
  console.log('Request ID:', this.requestId);
});
```

公開的包裝函數元素

如果使用 `AWS.ElastiCache`、`AWS.RDS` 或 `AWS.Redshift`，則部分操作的回應，您必須透過最上層輸出屬性來存取。

舉例而言，RDS.describeEngineDefaultParameters 方法以往會傳回下列內容。

```
{ Parameters: [ ... ] }
```

現在，其傳回的內容如下所示。

```
{ EngineDefaults: { Parameters: [ ... ] } }
```

下表會顯示每個服務受影響的操作清單。

用戶端類別	操作
AWS.ElastiCache	authorizeCacheSecurityGroupIngress createCacheCluster createCacheParameterGroup createCacheSecurityGroup createCacheSubnetGroup createReplicationGroup deleteCacheCluster deleteReplicationGroup describeEngineDefaultParameters modifyCacheCluster modifyCacheSubnetGroup modifyReplicationGroup purchaseReservedCacheNodesOffering rebootCacheCluster

用戶端類別	操作
	revokeCacheSecurityGroupIngress

用戶端類別	操作
AWS.RDS	addSourceIdentifierToSubscr iption authorizeDBSecurityGroupIngress copyDBSnapshot createDBInstance createDBInstanceReadReplica createDBParameterGroup createDBSecurityGroup createDBSnapshot createDBSubnetGroup createEventSubscription createOptionGroup deleteDBInstance deleteDBSnapshot deleteEventSubscription describeEngineDefaultParameters modifyDBInstance modifyDBSubnetGroup modifyEventSubscription modifyOptionGroup promoteReadReplica purchaseReservedDBInstances Offering

用戶端類別	操作
	rebootDBInstance removeSourceIdentifierFromSubscription restoreDBInstanceFromDBSnapshot restoreDBInstanceToPointInTime revokeDBSecurityGroupIngress

用戶端類別	操作
AWS.Redshift	authorizeClusterSecurityGroupIngress authorizeSnapshotAccess copyClusterSnapshot createCluster createClusterParameterGroup createClusterSecurityGroup createClusterSnapshot createClusterSubnetGroup createEventSubscription createHsmClientCertificate createHsmConfiguration deleteCluster deleteClusterSnapshot describeDefaultClusterParameters disableSnapshotCopy enableSnapshotCopy modifyCluster modifyClusterSubnetGroup modifyEventSubscription modifySnapshotCopyRetentionPeriod

用戶端類別	操作
	purchaseReservedNodeOffering rebootCluster restoreFromClusterSnapshot revokeClusterSecurityGroupIngress revokeSnapshotAccess rotateEncryptionKey

捨棄的用戶端屬性

.Client 和 .client 屬性皆已從服務物件中移除。如果您還在使用服務類別的 .Client 屬性，或是服務物件執行個體的 .client 屬性，請從程式碼中移除這些屬性。

下列程式碼與 SDK 版本 1 搭配使用，用於 JavaScript：

```
var sts = new AWS.STS.Client();
// or
var sts = new AWS.STS();

sts.client.operation(...);
```

您應將其變更為下列程式碼。

```
var sts = new AWS.STS();
sts.operation(...)
```

設定下列項目的 SDK JavaScript

您必須先設 JavaScript 定 SDK，才能使用 API 叫用 Web 服務。您必須至少設定這些設定：

- 您將請求服務的區域。
- 登入資料，可授予您存取軟體開發套件資源的許可。

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 SDK 和工具參考指南還包含許多 SDK 中常見的設置，功能和其他基礎概念。 AWS

本節中的主題說明了針 JavaScript 對 Node.js 設定 SDK 並在網頁瀏覽器中 JavaScript 執行的各種方法。

主題

- 使用全域組態物件
- 設定 AWS 區域
- 指定自訂端點
- 使用 SDK 驗證 AWS
- 設定 登入資料
- 鎖定 API 版本
- Node.js 的考量
- 瀏覽器指令碼考量
- 使用 Webpack 綁定應用程式

使用全域組態物件

有兩種方式可以設定軟體開發套件：

- 使用 `AWS.Config` 設定全域組態。
- 將額外的組態資訊傳遞給服務物件。

使用 AWS.Config 設定全域組態的入門通常比較簡單，但服務層級組態可提供對個別服務的進一步控制。AWS.Config 指定的全球組態會為您後續建立的服務物件提供預設設定，可讓設定更簡單。然而，您可以在需求隨全域組態而變化時，更新個別服務物件的組態。

設定全域組態

在您的程式碼中載入 aws-sdk 套件後，您可以使用 AWS 全域變述來存取軟體開發套件的類別並與個別服務互動。該軟體開發套件包含的全域組態物件 AWS.Config，是您可以用來指定應用程式所需的軟體開發套件組態設定。

請根據您的應用程式需求設定 AWS.Config 屬性，來設定軟體開發套件。下表摘要說明一般用來設定軟體開發套件組態的 AWS.Config 屬性。

組態選項	描述
credentials	「必要」。指定判斷服務和資源存取所用的登入資料。
region	「必要」。指定為服務提出請求的區域。
maxRetries	選用。指定特定請求重試的次數上限。
logger	選用。指定要將偵錯資訊寫入其中的記錄器物件。
update	選用。使用新值更新目前組態。

如需有關設定物件的詳細資訊，請參閱 API 參考Class: AWS.Config中的。

全域組態範例

您必須在 AWS.Config 中設定區域和登入資料。您可以將這些屬性設為 AWS.Config 建構子的一部分，如下列瀏覽器指令碼範例所示：

```
var myCredentials = new
 AWS.CognitoIdentityCredentials({IdentityPoolId:'IDENTITY_POOL_ID'});
var myConfig = new AWS.Config({
  credentials: myCredentials, region: 'us-west-2'
});
```

您也可以在使用 update 方法建立 AWS.Config 後，設定這些屬性，如下更新該區域的範例所示：

```
myConfig = new AWS.Config();
myConfig.update({region: 'us-east-1'});
```

您可以呼叫 AWS.config 的靜態 getCredentials 方法，取得您的預設登入資料：

```
var AWS = require("aws-sdk");

AWS.config.getCredentials(function(err) {
  if (err) console.log(err.stack);
  // credentials not loaded
  else {
    console.log("Access key:", AWS.config.credentials.accessKeyId);
  }
});
```

同樣地，如果您已在config檔案中正確設定區域，您可以透過將AWS_SDK_LOAD_CONFIG環境變數設定為任何值並呼叫 static region 屬性來取得該值AWS.config：

```
var AWS = require("aws-sdk");

console.log("Region: ", AWS.config.region);
```

依服務設定組態

您在 SDK 中使用的每個服務都 JavaScript 是透過屬於該服務 API 一部分的服務物件存取。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。當您在服務物件上設定組態值時，建構子會採用 AWS.Config 所用的所有組態值 (包含登入資料)。

例如，如果您需要存取多個區域中的 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 AWS.EC2({region: 'ap-southeast-2', maxRetries: 15, apiVersion:
 '2014-10-01'});
var ec2_regionB = new AWS.EC2({region: 'us-east-1', maxRetries: 15, apiVersion:
 '2014-10-01'});
```

您也可以在使用 AWS.Config 設定軟體開發套件時，設定服務專屬的組態值。全域組態物件支援許多服務特定的組態選項。如需有關服務特定組態的詳細資訊，請參閱 AWS SDK for JavaScript API 參考Class: AWS.Config中的。

固定組態資料

全域組態變更適用於所有新建立服務物件的請求。新建立服務物件的設定會先使用目前全域組態資料，然後再使用任何本機組態選項。您對全域 AWS.config 物件所做的更新不會套用至先前建立的服務物件。

您必須使用新組態資料來手動更新現有服務物件，或是您必須建立和使用具有新組態資料的新服務物件。下列範例會使用新的組態資料建立新的 Amazon S3 服務物件：

```
s3 = new AWS.S3(s3.config);
```

設定 AWS 區域

「地區」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。您可以在為其配置 SDK 時指定「區域」， JavaScript 以便 SDK 存取該區域中的資源。某些 服務僅在特定區域提供。

的 SDK 預設 JavaScript 不會選取 [地區]。然而，您可以使用環境變數 (一個共用的 config 檔案) 或是全域組態物件來設定區域。

在用戶端類別建構子

當您初始化服務物件時，您可以將該資源的區域指定為用戶端類別建構子中的一部分，如此處所示。

```
var s3 = new AWS.S3({apiVersion: '2006-03-01', region: 'us-east-1'});
```

使用全域組態物件

若要在 JavaScript 程式碼中設定 Region，請更新AWS.Config全域設定物件，如下所示。

```
AWS.config.update({region: 'us-east-1'});
```

如需有關目前區域和每個區域中可用服務的詳細資訊，請參閱AWSAWS 一般參考.

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，用於 JavaScript 讀取並使用它的 SDK。

使用共用組態檔

與共用組態檔讓您存放軟體開發套件所用的登入資料的方式類似，您可以在軟體開發套件所用、名為 config 的共用檔中保留區域和其他組態設定。如果AWS_SDK_LOAD_CONFIG環境變數已設定為任何值，SDK 會在載入config檔案時 JavaScript 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 或 Unix 使用者：~/.aws/config
- Windows 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-east-1
   output=json
```

如需有關使用共用設定和認證檔案的詳細資訊，請參閱使AWS Command Line Interface 用指南中的從共用登入資料檔案中在 Node.js 中載入登入資料或「組態與認證檔案」。

設定區域的優先順序

區域設定的優先順序如下：

- 如將某區域傳遞至用戶端類別建構子，則會使用該區域。如果沒有，則...
- 如在全域建構子物件上設定某區域，則會使用該區域。如果沒有，則...
- 如果 AWS_REGION 環境變數是真值，則會使用該區域。如果沒有，則...
- 如果 AMAZON_REGION 環境變數是真值，則會使用該區域。如果沒有，則...
- 如果AWS_SDK_LOAD_CONFIG環境變數設定為任何值，且共用認證檔案 (~/.aws/credentials或由指示的路徑AWS_SHARED_CREDENTIALS_FILE) 包含已設定描述檔的 [區域]，則會使用該 [區域]。如果沒有，則...

- 如果將AWS_SDK_LOAD_CONFIG環境變數設定為任何值，且組態檔案 (~/.aws/config或由指示的路徑AWS_CONFIG_FILE) 包含已設定描述檔的「區域」(Region)，則會使用該「區域」(Region)。

指定自訂端點

對 SDK 中的 API 方法的呼叫 JavaScript 會對服務端點 URI 進行。依預設，會透過您為程式碼設定的區域來建立這些端點。然而，這些是您需要為 API 呼叫指定自訂端點的情況。

端點字串格式

端點值應是以下格式的字串：

https://*{service}.{region}*.amazonaws.com

ap-northeast-3 區域的端點

不會依區域列舉 API 傳回在日本的 ap-northeast-3 區域，像是 EC2.describeRegions。若要為此區域定義端點，請遵循先前所述的格式。所以這個區域的 Amazon EC2 端點將是

ec2.ap-northeast-3.amazonaws.com

用於的端點 MediaConvert

您需要建立要搭配使用的自訂端點 MediaConvert。會為每個客戶帳戶指派其專屬的端點，而您必須使用該端點。以下是如何搭配使用自訂端點的範例 MediaConvert。

```
// Create MediaConvert service object using custom endpoint
var mcClient = new AWS.MediaConvert({endpoint: 'https://abcd1234.mediaconvert.us-
west-1.amazonaws.com'});

var getJobParams = {Id: 'job_ID'};

mcClient.getJob(getJobParams, function(err, data)) {
   if (err) console.log(err, err.stack); // an error occurred
   else console.log(data); // successful response
};
```

若要取得帳戶 API 端點，請參閱 API 參考的 MediaConvert.describeEndpoints。

請確認您在程式碼中指定的區域與自訂端點 URI 中的區域相同。區域設定和自訂端點 URI 之間的不相符可能會造成 API 失敗。

如需詳細資訊 MediaConvert，請參閱 API 參考或AWS Elemental MediaConvert 使用者指南中的AWS.MediaConvert類別。

使用 SDK 驗證 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。

我們建議在本地開發，並且沒有給予雇主身份驗證方法的新用戶應該設置 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
```

的 SDK JavaScript 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 身分中心驗證。

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。

存取時使用臨時登入資料 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。
- 若要深入瞭解 JavaScript 認證提供者的其他 SDK，請參閱 AWS SDK 和工具參考指南中的標準化認證提供者。

設定 登入資料

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 程式碼都必須先取得有效的認證，才能透過 API 存取服務。您可以使用 `AWS.Config` 或依服務，透過將登入資料直接傳遞至服務物件，在組態物件上全域設定登入資料。

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。

登入資料的最佳實務

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。

如需授與最低權限的詳細資訊，請參閱 IAM 使用者指南中最佳做法主題的授與最低權限一節。

> **Warning**
>
> 如果這麼做是可行的，我們建議您不要將登入資料寫死在應用程式或瀏覽器指令碼中。硬式編碼認證會造成暴露敏感資訊的風險。

如需如何管理存取金鑰的詳細資訊，請參閱《管理 AWS 存取金鑰的最佳做法》 AWS 一般參考。

主題

- 在 Node.js 設定登入資料

- 在 Web 瀏覽器中設定登入資料

在 Node.js 設定登入資料

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 中取得登入資料時，對於依賴不只一個來源 (像是環境變數和您載入的 JSON 檔案) 時請小心。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。

以下是您可以提供登入資料的方法，依建議順序列出：

1. 從 Amazon EC2 的 AWS Identity and Access Management (IAM) 角色載入
2. 從共享登入資料檔案 (`~/.aws/credentials`) 載入。
3. 從環境變數中載入
4. 從磁碟上的 JSON 檔案載入
5. SDK 提供的其他憑證提供者類別 JavaScript

如果開發套件有不只一個登入資料來源，預設的選擇順序如下：

1. 透過服務客戶建構函數明確設定的登入資料
2. 環境變數
3. 共享登入資料檔案
4. 從 ECS 登入資料提供者 (若適用的話) 載入的登入資料
5. 透過使用共用 AWS 設定檔或共用認證檔案中指定的認證程序取得的認證。如需詳細資訊，請參閱 the section called “使用所設定登入資料程序的登入資料”。
6. 使用 Amazon EC2 執行個體的登入資料提供者從 AWS IAM 載入的登入資料 (如果在執行個體中繼資料中設定)

如需詳細資訊，請參閱 API 參考資料`Class: AWS.CredentialProviderChain`中的`Class: AWS.Credentials`和。

> **Warning**
>
> 雖然可以這樣做，但我們不建議您在應用程式中對 AWS 憑證進行硬式編碼。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。

本節的主題說明如何在 Node.js 中載入登入資料。

主題

- 從 Amazon EC2 的 IAM 角色載入 Node.js 中的登入資料
- 載入 Node.js Lambda 函數的登入資料
- 從共用登入資料檔案中在 Node.js 中載入登入資料
- 從環境變數在 Node.js 中載入登入資料
- 從 JSON 檔案中在 Node.js 中載入登入資料
- 使用所設定的登入資料程序在 Node.js 中載入登入資料

從 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AWS 開發套件和工具參考指南中的 Amazon EC2 執行個體使用 IAM 角色。

載入 Node.js Lambda 函數的登入資料

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許可：使用 IAM 角色 (執行角色)。

從共用登入資料檔案中在 Node.js 中載入登入資料

您可以將 AWS 認證資料保存在 SDK 和命令列介面所使用的共用檔案中。當 SDK 加 JavaScript 載時，它會自動搜索名為「憑據」的共享憑據文件。共用登入資料檔案保存位置取決於您的作業系統：

- 在 Linux、Unix 和 macOS 上的共用登入資料檔案：`~/.aws/credentials`
- 在 Windows 上的共用登入資料檔案：`C:\Users\USER_NAME\.aws\credentials`

如果您還沒有共用的登入資料檔案，請參閱使用 SDK 驗證 AWS。完成這些指示之後，您應該會在登入資料檔案中看到類似下列的文字，其中 *<YOUR_ACCESS_KEY_ID>* 是您的存取金鑰 ID, 而 *<YOUR_SECRET_ACCESS_KEY>* 是您的私密存取金鑰：

```
[default]
aws_access_key_id = <YOUR_ACCESS_KEY_ID>
aws_secret_access_key = <YOUR_SECRET_ACCESS_KEY>
```

如需使用此檔案的使用範例，請參閱Node.js 入門。

[default] 區段標題指定預設設定檔和登入資料的相關值。您可以在相同的共用組態檔中建立其他設定檔，每個設定檔包含其專屬的登入資料資訊。下列範例顯示具有預設設定檔和兩個其他設定檔的組態檔：

```
[default] ; default profile
aws_access_key_id = <DEFAULT_ACCESS_KEY_ID>
aws_secret_access_key = <DEFAULT_SECRET_ACCESS_KEY>

[personal-account] ; personal account profile
aws_access_key_id = <PERSONAL_ACCESS_KEY_ID>
aws_secret_access_key = <PERSONAL_SECRET_ACCESS_KEY>

[work-account] ; work account profile
aws_access_key_id = <WORK_ACCESS_KEY_ID>
aws_secret_access_key = <WORK_SECRET_ACCESS_KEY>
```

依預設，軟體開發套件會檢查 AWS_PROFILE 環境變數來判斷要使用哪個設定檔。如果未在您的環境中設定 AWS_PROFILE 變數，軟體開發套件會使用 [default] 設定檔的登入資料。若要使用其他設定檔的登入資料，請設定或變更 AWS_PROFILE 環境變數的值。例如上述的組態檔案，若要使用工作帳戶的登入資料，請將 AWS_PROFILE 環境變數設定為 work-account (對應您的作業系統)。

> **Note**
>
> 設定環境變數之後，請務必採取適當的動作 (根據作業系統需求)，讓您的變數可在 shell 或命令環境使用。

設定環境變數後 (如果需要)，您可以執行使用 SDK 的 JavaScript 檔案，例如名為的檔案script.js。

```
$ node script.js
```

您也可以擇一在載入軟體開發套件前設定 process.env.AWS_PROFILE，或選取以下範例中所示的登入資料供應商，來明確選取軟體開發套件所用的設定檔：

```
var credentials = new AWS.SharedIniFileCredentials({profile: 'work-account'});
AWS.config.credentials = credentials;
```

從環境變數在 Node.js 中載入登入資料

SDK 會自動偵測設定為環境中變數的 AWS 認證，並將其用於 SDK 請求，無需在應用程式中管理認證。您設為提供登入資料的環境變數為：

- AWS_ACCESS_KEY_ID
- AWS_SECRET_ACCESS_KEY
- AWS_SESSION_TOKEN

有關設置環境變量的更多詳細信息，請參閱 AWS SDK 和工具參考指南中的環境變量支持。

從 JSON 檔案中在 Node.js 中載入登入資料

您可以使用 AWS.config.loadFromPath，從磁碟上的 JSON 文件載入組態和登入資料。指定的路徑是相對於程序的目前工作目錄。例如，若要從包含以下內容的 'config.json' 檔案載入登入資料：

```
{ "accessKeyId": <YOUR_ACCESS_KEY_ID>, "secretAccessKey": <YOUR_SECRET_ACCESS_KEY>,
 "region": "us-east-1" }
```

然後使用下面的代碼：

```
var AWS = require("aws-sdk");
AWS.config.loadFromPath('./config.json');
```

> **Note**
>
> 從 JSON 文件載入組態資料會重設所有現有組態資料。在使用此技術後新增其他組態資料。從瀏覽器指令碼不支援的 JSON 文件中載入登入資料。

使用所設定的登入資料程序在 Node.js 中載入登入資料

您可以使用沒有內建在軟體開發套件中的方法取得登入資料。若要這麼做，請在共用設定檔或共用 AWS 認證檔案中指定認證程序。如果AWS_SDK_LOAD_CONFIG環境變數設定為任何值，SDK 會偏好在組態檔案中指定的處理序，而不是認證檔案中指定的程序 (如果有的話)。

如需有關在共用 AWS 設定檔或共用認證檔案中指定認證程序的詳細資訊，請參閱AWS CLI 命令參考，特別是有關來自外部處理程序的 Sourcing 認證的資訊。

如需使用 AWS_SDK_LOAD_CONFIG 環境變數的相關資訊，請參閱本文件中的the section called “使用共用組態檔”。

在 Web 瀏覽器中設定登入資料

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。以下是您可以提供登入資料的方法，依建議順序列出：

1. 使用 Amazon Cognito 身分來對使用者進行身份驗證並提供登入資料
2. 使用 Web 聯合身分
3. 在指令碼中將程式碼寫死

> **Warning**
>
> 不建議您使用硬式編碼AWS憑據。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 份對用户進行身份驗證
- 使用 Web 聯合身分來驗證使用者
- Web 聯合身分範例

使用 Amazon Cognito 份對用户進行身份驗證

推薦的方法來獲得AWS您的瀏覽器腳本的憑據是使用 Amazon Cognito 身份證書對象，AWS.CognitoIdentityCredentials。Amazon Cognito 支持通過第三方身份提供商對用户進行身份驗證。

若要使用 Amazon Cognito 身分，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會專門識別每個使用者帳戶。Amazon Cognito 身分不是憑證。它們使用 AWS Security Token Service (AWS STS) 中支援的 Web 聯合身分來交換登入資料。

Amazon Cognito 可以幫助您在多個身分供應商中使用`AWS.CognitoIdentityCredentials`物件。接著，會將載入的身分交換為在 AWS STS 中的登入資料。

配置亞 Amazon Cognito 份證書對象

如果您尚未建立身分集區，請在Amazon Cognito 主控台，然後再配置`AWS.CognitoIdentityCredentials`。為身分集區同時建立經授權和未經授權的 IAM 角色並將其建立關聯。

未經授權的使用者的身分未經驗證，因此這個角色適用於您應用程式的訪客使用者，或使用者身分是否已驗證並不重要的情況。已驗證使用者透過驗證其身分的第三方身分供應商來登入應用程式。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。

在您以連接的身分供應商設定身分集區之後，您可以使用 `AWS.CognitoIdentityCredentials` 來驗證使用者。若要設定您的應用程式登入資料使用 `AWS.CognitoIdentityCredentials`，請將 `credentials` 屬性設定為 `AWS.Config` 或每個服務的組態。以下範例使用 `AWS.Config`：

```
AWS.config.credentials = new AWS.CognitoIdentityCredentials({
  IdentityPoolId: 'us-east-1:1699ebc0-7900-4099-b910-2df94f52a030',
  Logins: { // optional tokens, used for authenticated login
    'graph.facebook.com': 'FBTOKEN',
    'www.amazon.com': 'AMAZONTOKEN',
    'accounts.google.com': 'GOOGLETOKEN'
  }
});
```

選用的 `Logins` 屬性是身分供應商名稱與這些身分供應商的身分權杖的對應。您從身分供應商取得權杖的方式，取決於您使用的供應商。例如，如果 Facebook 是您的身分供應商之一，您可以使用 `FB.login`Facebook 開發套件的 函數來取得身分供應商權杖：

```
FB.login(function (response) {
  if (response.authResponse) { // logged in
    AWS.config.credentials = new AWS.CognitoIdentityCredentials({
      IdentityPoolId: 'us-east-1:1699ebc0-7900-4099-b910-2df94f52a030',
      Logins: {
```

```
        'graph.facebook.com': response.authResponse.accessToken
      }
    });

    s3 = new AWS.S3; // we can now create our service object

    console.log('You are now logged in.');
  } else {
    console.log('There was a problem logging you in.');
  }
});
```

將未經驗證的使用者切換為經驗證的使用者

Amazon Cognito 可支援已驗證和未驗證的使用者。未驗證的使用者即使沒有以任何身分供應商登入，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。每個未驗證使用者在 Amazon Cognito 中都有專屬身分 (即使使用者尚未個別登入和驗證身分亦同)。

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。在這種情況下，您的預設組態可能如下所示：

```
// set the default config object
var creds = new AWS.CognitoIdentityCredentials({
 IdentityPoolId: 'us-east-1:1699ebc0-7900-4099-b910-2df94f52a030'
});
AWS.config.credentials = creds;
```

切換到已驗證使用者

當未驗證使用者登入身分供應商，且您有一個字符時，您可以透過呼叫自訂函數更新登入資料物件並新增 Logins 字符，將使用者從未驗證切換為已驗證：

```
// 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;
```

```
}
```

您也可建立 CognitoIdentityCredentials 物件。如果您建立此物件，您必須為建立的任何現有服務物件重設登入資料屬性。服務物件僅會在物件初始化時從全域組態進行讀取。

如需有關的詳細資訊CognitoIdentityCredentials對象，請參閱AWS.CognitoIdentityCredentials中的AWS SDK for JavaScriptAPI 參考。

使用 Web 聯合身分來驗證使用者

您可以直接配置單個身份提供程序以訪問AWS使用 web 聯合身分的資源。AWS目前透過各種身分供應商，使用 web 聯合身分對使用者進行身份驗證：

- Login with Amazon
- Facebook 登入
- Google 登入

您必須先使用應用程式支援的供應商註冊應用程式。接下來，建立 IAM 角色並為其設定許可。接下來，系統會使用您建立的 IAM 角色來透過個別身分供應商授予您為其設定的許可。例如，您可以設定一個角色，來允許透過 Facebook 登入的使用者擁有對您控制之特定 Amazon S3 存取的讀取存取。

在同時擁有包含設定權限的 IAM 角色和使用所選身分供應商註冊的應用程式後，您可以設定軟體開發套件以使用協助程式來取得 IAM 角色的登入資料，所下所示：

```
AWS.config.credentials = new AWS.WebIdentityCredentials({
   RoleArn: 'arn:aws:iam::<AWS_ACCOUNT_ID>/:role/<WEB_IDENTITY_ROLE_NAME>',
   ProviderId: 'graph.facebook.com|www.amazon.com', // this is null for Google
   WebIdentityToken: ACCESS_TOKEN
});
```

ProviderId 參數中的值取決於指定的身分供應商。WebIdentityToken 參數中的值是透過身分供應商成功登入時所擷取的存取字符。如需為每個身分供應商設定和擷取存取字符的詳細資訊，請參閱身分供應商的文件。

步驟 1：向身分供應商註冊

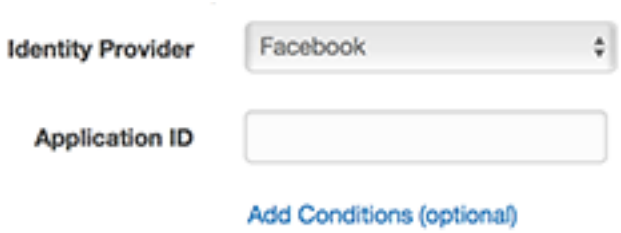
若要開始，請向您選擇要支援的身分供應商註冊應用程式。系統會要求您提供可辨識您應用程式和其作者的資訊。這可確保身分供應商了解誰在接收其使用者資訊。在每個案例中，身分供應商會發行您用來設定使用者角色的應用程式 ID。

步驟 2：為身分供應商建立 IAM 角色

從身分供應商取得應用程式 ID 後，前往 IAM 主控台https://console.aws.amazon.com/iam/創建新的 IAM 角色。

為身分供應商建立 IAM 角色

1. 前往主控台的 Roles (角色) 區段，然後選擇 Create New Role (新建角色)。
2. 為新角色輸入可協助您追蹤其用量的名稱，例如 **`facebookIdentity`**，然後選擇 Next Step (下一步)。
3. 在 Select Role Type (選擇角色類型) 中，選擇 Role for Identity Provider Access (身分供應商存取的角色)。
4. 針對 Grant access to web identity providers (將存取授予 web 身分供應商)，選擇 Select (選取)。
5. 來自身分供應商列表中，選擇您要用於此 IAM 角色的身分供應商。



6. 輸入在 Application ID (應用程式 ID) 中由身分供應商提供的應用程式 ID，然後選擇 Next Step (下一步)。
7. 為您要公開的資源設定許可，允許對特定資源進行特定操作。如需 IAM 許可的詳細資訊，請參概觀AWSIAM 許可中的IAM User Guide。檢視，並視需要自訂角色的信任關係，然後選擇 Next Step (下一步)。
8. 連接您需要的額外政策，然後選擇 Next Step (下一步)。如需 IAM 政策的詳細資訊，請參IAM 政策概觀中的IAM User Guide。
9. 檢閱新角色，然後選擇 Create Role (建立角色)。

您可以將其他限制條件提供給該角色，像是將其範圍限制在特定的使用者 ID。如果該角色將寫入許可授予您的資源，請確保您正確地將角色範圍限制在含正確權限的使用者，否則具有 Amazon、Facebook 或 Google 身分的任何使用者都能夠修改您應用程式中的資源。

如需在 IAM 中使用 web 聯合身分的詳細資訊，請參關於 Web 聯合身分中的IAM User Guide。

步驟 3：在登入後取得供應商存取字符

使用身分供應商的軟體開發套件來為應用程式設定登入動作。您可以透過啟用使用者登入 (使用 OAuth 或 OpenID) 的身分供應商，來下載和安裝 JavaScript 軟體開發套件。如需如何在應用程式中下載和設定軟體開發套件程式碼的詳細資訊，請參閱身分供應商的軟體開發套件文件

- Login with Amazon
- Facebook 登入
- Google 登入

步驟 4：取得臨時登入資料

在應用程式、角色和資源許可都設定後，將該程式碼新增至應用程式來取得暫時登入資料。系統會使用 Web 聯合身分透過 AWS Security Token Service 來提供這些登入資料。使用者登入身分供應商，而傳回存取字符。設定`AWS.WebIdentityCredentials`物件，使用您為此身分供應商建立的 IAM 角色的 ARN：

```
AWS.config.credentials = new AWS.WebIdentityCredentials({
    RoleArn: 'arn:aws:iam::<AWS_ACCOUNT_ID>:role/<WEB_IDENTITY_ROLE_NAME>',
    ProviderId: 'graph.facebook.com|www.amazon.com', // Omit this for Google
    WebIdentityToken: ACCESS_TOKEN // Access token from identity provider
});
```

後續建立的服務物件會擁有適當的登入資料。在設定 `AWS.config.credentials` 屬性前建立的物件不會有目前的登入資料。

您也可以建立 `AWS.WebIdentityCredentials`，再擷取存取字符。此可讓您建立依靠登入資料的服務物件，再載入存取字符。若要這麼做，請建立不含 `WebIdentityToken` 參數的登入資料物件：

```
AWS.config.credentials = new AWS.WebIdentityCredentials({
  RoleArn: 'arn:aws:iam::<AWS_ACCOUNT_ID>:role/<WEB_IDENTITY_ROLE_NAME>',
  ProviderId: 'graph.facebook.com|www.amazon.com' // Omit this for Google
});

// Create a service object
var s3 = new AWS.S3;
```

接著會透過包含該存取字符的身分供應商軟體開發套件，在回呼中設定 `WebIdentityToken`：

```
AWS.config.credentials.params.WebIdentityToken = accessToken;
```

Web 聯合身分範例

這裡有一些使用 web 聯合身分來在瀏覽器 JavaScript 中取得登入資料的範例。這些範例的執行必須透過 http:// 或 https:// 主機結構描述，來確保身分供應商可以重新導向至您的應用程式。

Login with Amazon 範例

以下程式碼說明如何使用 Login with Amazon 做為身分供應商。

```
<a href="#" id="login">
  <img border="0" alt="Login with Amazon"
    src="https://images-na.ssl-images-amazon.com/images/G/01/lwa/
btnLWA_gold_156x32.png"
    width="156" height="32" />
</a>
<div id="amazon-root"></div>
<script type="text/javascript">
  var s3 = null;
  var clientId = 'amzn1.application-oa2-client.1234567890abcdef'; // client ID
  var roleArn = 'arn:aws:iam::<AWS_ACCOUNT_ID>:role/<WEB_IDENTITY_ROLE_NAME>';

  window.onAmazonLoginReady = function() {
    amazon.Login.setClientId(clientId); // set client ID

    document.getElementById('login').onclick = function() {
      amazon.Login.authorize({scope: 'profile'}, function(response) {
        if (!response.error) { // logged in
          AWS.config.credentials = new AWS.WebIdentityCredentials({
            RoleArn: roleArn,
            ProviderId: 'www.amazon.com',
            WebIdentityToken: response.access_token
          });

          s3 = new AWS.S3();

          console.log('You are now logged in.');
        } else {
          console.log('There was a problem logging you in.');
        }
      });
    };
```

```
  };

  (function(d) {
    var a = d.createElement('script'); a.type = 'text/javascript';
    a.async = true; a.id = 'amazon-login-sdk';
    a.src = 'https://api-cdn.amazon.com/sdk/login1.js';
    d.getElementById('amazon-root').appendChild(a);
  })(document);
</script>
```

Facebook Login 範例

以下程式碼說明如何使用 Facebook Login 做為身分供應商：

```
<button id="login">Login</button>
<div id="fb-root"></div>
<script type="text/javascript">
var s3 = null;
var appId = '1234567890'; // Facebook app ID
var roleArn = 'arn:aws:iam::<AWS_ACCOUNT_ID>:role/<WEB_IDENTITY_ROLE_NAME>';

window.fbAsyncInit = function() {
  // init the FB JS SDK
  FB.init({appId: appId});

  document.getElementById('login').onclick = function() {
    FB.login(function (response) {
      if (response.authResponse) { // logged in
        AWS.config.credentials = new AWS.WebIdentityCredentials({
          RoleArn: roleArn,
          ProviderId: 'graph.facebook.com',
          WebIdentityToken: response.authResponse.accessToken
        });

        s3 = new AWS.S3;

        console.log('You are now logged in.');
      } else {
        console.log('There was a problem logging you in.');
      }
    });
  };
};
```

```
// Load the FB JS SDK asynchronously
(function(d, s, id){
   var js, fjs = d.getElementsByTagName(s)[0];
   if (d.getElementById(id)) {return;}
   js = d.createElement(s); js.id = id;
   js.src = "//connect.facebook.net/en_US/all.js";
   fjs.parentNode.insertBefore(js, fjs);
 }(document, 'script', 'facebook-jssdk'));
</script>
```

Google+ Sign-in 範例

以下程式碼說明如何使用 Google+ Sign-in 做為身分供應商。透過 Google 用於 web 聯合身分的存取字符會存放在 `response.id_token`，而不是與其他身分供應商一樣放在 `access_token`。

```
<span
  id="login"
  class="g-signin"
  data-height="short"
  data-callback="loginToGoogle"
  data-cookiepolicy="single_host_origin"
  data-requestvisibleactions="http://schemas.google.com/AddActivity"
  data-scope="https://www.googleapis.com/auth/plus.login">
</span>
<script type="text/javascript">
  var s3 = null;
  var clientID = '1234567890.apps.googleusercontent.com'; // Google client ID
  var roleArn = 'arn:aws:iam::<AWS_ACCOUNT_ID>:role/<WEB_IDENTITY_ROLE_NAME>';

  document.getElementById('login').setAttribute('data-clientid', clientID);
  function loginToGoogle(response) {
    if (!response.error) {
      AWS.config.credentials = new AWS.WebIdentityCredentials({
        RoleArn: roleArn, WebIdentityToken: response.id_token
      });

      s3 = new AWS.S3();

      console.log('You are now logged in.');
    } else {
      console.log('There was a problem logging you in.');
    }
  }
```

```
 (function() {
   var po = document.createElement('script'); po.type = 'text/javascript'; po.async =
true;
   po.src = 'https://apis.google.com/js/client:plusone.js';
   var s = document.getElementsByTagName('script')[0]; s.parentNode.insertBefore(po,
s);
 })();
</script>
```

鎖定 API 版本

AWS 服務具有 API 版本號，以跟踪 API 兼容性。 AWS 服務中的 API 版本由YYYY-mm-dd格式化的日期字串識別。例如，Amazon S3 的當前 API 版本是2006-03-01。

如果您依賴在生產程式碼中某服務的 API 版本，我們建議您將其鎖定。此能夠隔離您的應用程式，避免受到因對軟體開發套件更新而造成的服務變更。如果您在建立服務物件時不指定 API 版本，軟體開發套件依預設會使用最新的 API 版本。這可能會使得應用程式參考更新的 API，內含會對您應用程式造成負面影響的變更。

若要鎖定您要用於服務的 API 版本，請在建構服務物件時傳遞 `apiVersion` 參數。在以下範例中，已將新建立的 `AWS.DynamoDB` 服務物件鎖定至 `2011-12-05` API 版本：

```
var dynamodb = new AWS.DynamoDB({apiVersion: '2011-12-05'});
```

您可以透過在 `AWS.Config` 中指定 `apiVersions` 參數，來全域設定一組服務 API 版本。例如，若要設定特定版本的 DynamoDB 和 Amazon EC2 API 以及目前的亞 Amazon Redshift API，請按如下方式進行設 `apiVersions`定：

```
AWS.config.apiVersions = {
  dynamodb: '2011-12-05',
  ec2: '2013-02-01',
  redshift: 'latest'
};
```

取得 API 版本

若要取得服務的 API 版本，請參閱服務參考頁面 (例如 https://docs.aws.amazon.com/AWSJavaScriptSDK/latest/AWS/S3.html Amazon S3) 上的「鎖定 API 版本」一節。

Node.js 的考量

雖然 Node.js 程式碼是 JavaScript， AWS SDK for JavaScript 在 Node.js 中使用可能與在瀏覽器指令碼中使用 SDK 不同。有些在 Node.js 中可運作的 API 方法無法在瀏覽器指令碼中運作，反之亦然。是否能成功使用某些 API，取決於您對常見 Node.js 程式碼編寫方式的熟悉程度，例如匯入及使用 `File System (fs)` 模組之類的其他 Node.js 模組。

使用內建 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，請使用以下資訊。

```
var http = require('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 檔案的程式碼。

```
// include File System module
var fs = require('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完整清單，請參閱 Node.js 網站的 Node.js 6.11.1 版文件。

使用 NPM 套件

除了內建模組，您可以透過 npm (Node.js 套件管理工具) 來包含和納入第三方的程式碼。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 npm 與目前可用套件清單的詳細資訊，請參閱 https://www.npmjs.com。您也可以在此處瞭解其他 Node.js 套件的相關資訊 GitHub。

您可以使用 AWS SDK for JavaScript is 的 npm 套件的其中一個範例`browserify`。如需詳細資訊，請參閱 使用 Browserify 來將軟體開發套件建立為相依性。另一個範例是 `webpack`。如需詳細資訊，請參閱 使用 Webpack 綁定應用程式。

主題

- 在 Node.js 中設定 maxSocket
- 在 Node.js 中重複使用 Keep-Alive 的連線
- 為 Node.js 設定代理
- 在 Node.js 中註冊憑證

在 Node.js 中設定 maxSocket

您可以在 Node.js 中，依來源設定連線數量上限。如果已設定 `maxSockets`，低階 HTTP 用戶端會在請求可供使用時將它們排入佇列並指派至通訊端。

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。

以下範本示範如何為您建立的所有服務物件設定 `maxSockets`。此範例可允許對每個服務端點進行最多 25 個並行連線。

```
var AWS = require('aws-sdk');
var https = require('https');
var agent = new https.Agent({
   maxSockets: 25
});

AWS.config.update({
   httpOptions:{
      agent: agent
   }
});
```

您可以為每個服務完成相同的設定。

```
var AWS = require('aws-sdk');
var https = require('https');
var agent = new https.Agent({
   maxSockets: 25
});

var dynamodb = new AWS.DynamoDB({
   apiVersion: '2012-08-10'
   httpOptions:{
      agent: agent
```

```
    }
});
```

使用 https 的預設時，軟體開發套件會從 globalAgent 取得 maxSockets 值。若沒有定義 maxSockets 值或是該值為 Infinity，軟體開發套件會假設 maxSockets 的值為 50。

如需在 Node.js 中設定 maxSockets 的詳細資訊，請參閱 Node.js online documentation。

在 Node.js 中重複使用 Keep-Alive 的連線

根據預設，預設 Node.js HTTP/HTTPS 代理程式會為每個新的請求建立新的 TCP 連線。若要避免建立新連線的成本，您可以重複使用現有的連線。

對於如 DynamoDB 查詢等短期操作，設定 TCP 連線的延遲額外負荷可能大於該操作本身。此外，由於靜態 DynamoDB 加密已與 AWS KMS 整合，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。

配置 SDK 以重複使用 TCP 連接的最簡單方法是將 AWS_NODEJS_CONNECTION_REUSE_ENABLED環境變量設置為1。 JavaScript 此功能已在 2.463.0 版本中加入。

或者，您可以將 HTTP 或 HTTPS 代理程式的 keepAlive 屬性設為 true，如下列範例所示。

```
const AWS = require('aws-sdk');
// http or https
const http = require('http');
const agent = new http.Agent({
  keepAlive: true,
// Infinity is read as 50 sockets
  maxSockets: Infinity
});

AWS.config.update({
  httpOptions: {
    agent
  }
});
```

下列範例顯示如何僅keepAlive針對 DynamoDB 用戶端進行設定：

```
const AWS = require('aws-sdk')
// http or https
const https = require('https');
```

```
const agent = new https.Agent({
  keepAlive: true
});

const dynamodb = new AWS.DynamoDB({
  httpOptions: {
    agent
  }
});
```

如果已啟用 keepAlive，您也可以使用 keepAliveMsecs 設定 TCP Keep-Alive 封包的初始延遲 (預設為 1000ms)。請參閱 Node.js 文件了解詳細資訊。

為 Node.js 設定代理

如果您無法直接連線到網際網路，用於 JavaScript 支援透過第三方 HTTP 代理程式 (例如代理代理程式) 使用 HTTP 或 HTTPS 代理伺服器的 SDK。若要安裝 proxy 代理程式，請在命令列中輸入下列指令。

```
npm install proxy-agent --save
```

如果您決定使用不同的代理，先遵循該代理的安裝和設定指示。若要在應用程式中使用此代理或另一個第三方代理，您必須設定 AWS.Config 的 httpOptions 屬性來指定您選擇的代理。此範例說明 proxy-agent。

```
var AWS = require("aws-sdk");
var ProxyAgent = require('proxy-agent').ProxyAgent;
AWS.config.update({
  httpOptions: { agent: new ProxyAgent('http://internal.proxy.com') }
});
```

如需其他代理程式庫的更多資訊，請參閱 npm (Node.js 套件管理工具)。

在 Node.js 中註冊憑證

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)。接著會使用新建立的 https.Agent 來更新軟體開發套件組態。

```
var fs = require('fs');
var https = require('https');
var certs = [
  fs.readFileSync('/path/to/cert.pem')
];

AWS.config.update({
  httpOptions: {
    agent: new https.Agent({
      rejectUnauthorized: true,
      ca: certs
    })
  }
});
```

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

- 建立適用於瀏覽器的軟體開發套件
- 跨來源資源分享 (CORS)

建立適用於瀏覽器的軟體開發套件

的 SDK 會以 JavaScript JavaScript 檔案的形式提供，其中包含預設服務集的支援。此檔案通常會使用 `<script>` 標籤載入瀏覽器指令碼，參考託管的軟體開發套件套件。但是，您可能需要預設服務之外其他服務的支援，否則便需要自訂軟體開發套件。

如果您在瀏覽器中強制執行 CORS 的環境之外使用 SDK，並且想要存取 SDK 提供的所有服務 JavaScript，則可以透過複製存放庫並執行建置 SDK 預設託管版本的相同建置工具來在本機建置 SDK 的自訂副本。以下區段說明使用額外服務和 API 版本來建立軟體開發套件的步驟。

主題

- 使用 SDK 產生器建置適用的 SDK JavaScript
- 使用 CLI 建置適用於下列項目的開發套件 JavaScript
- 建立特定服務和 API 版本
- 使用 Browserify 來將軟體開發套件建立為相依性

使用 SDK 產生器建置適用的 SDK JavaScript

創建自己的構建的最簡單方法 AWS SDK for JavaScript 是使用 SDK 構建器 Web 應用程序 https://sdk.amazonaws.com/builder/js。使用軟體開發套件建置器，來指定要包含在建置中的服務和其 API 版本。

選擇 Select all services (選取所有服務)，或選擇 Select default services (選取預設服務) 做為您可以新增或移除服務的起點。選擇 Development (開發) 來取得可讀性更高的程式碼，或選擇 Minified (壓縮) 來建立要部署的壓縮建置。在您選擇要包含的服務和版本後，請選擇 Build (建置) 來建置及下載您的自訂軟體開發套件。

使用 CLI 建置適用於下列項目的開發套件 JavaScript

若要建置 JavaScript 使用的 SDK AWS CLI，您必須先複製包含 SDK 來源的 Git 儲存庫。您必須在電腦上安裝 Git 和 Node.js。

首先，從中克隆存儲庫 GitHub 並將目錄更改到目錄中：

```
git clone https://github.com/aws/aws-sdk-js.git
cd aws-sdk-js
```

複製儲存庫後，下載同時適用於軟體開發套件和建置工具的相依性模組：

```
npm install
```

您就可以立即建立軟體開發套件的套件版本。

從命令列建立

建置器工具位於 `dist-tools/browser-builder.js` 中。輸入以下資訊來執行此指令碼：

```
node dist-tools/browser-builder.js > aws-sdk.js
```

此命令會建立 aws-sdk.js 檔案。此檔案未壓縮。依預設，此套件僅包含一套標準的服務。

壓縮建置輸出

為了減少網路上的資料量，可以透過稱為縮小的程序來壓縮 JavaScript 檔案。壓縮指令碼評論、不必要的空間和有助於人類讀取的其他特性，但不會影響程式碼執行。建置器工具可產生解壓縮或已壓縮的輸出。若要壓縮建置輸出，請設定 `MINIFY` 環境變數：

```
MINIFY=1 node dist-tools/browser-builder.js > aws-sdk.js
```

建立特定服務和 API 版本

您可以選取要在軟體開發套件中建立哪些服務。若要選取服務，請指定做為參數的服務名稱 (以逗號分隔)。例如，若要僅建置 Amazon S3 和 Amazon EC2，請使用以下命令：

```
node dist-tools/browser-builder.js s3,ec2 > aws-sdk-s3-ec2.js
```

您也可以透過在服務名稱後新增版本名稱，來選取服務建置的特定 API 版本。例如，若要建立兩個 API 版本的 Amazon DynamoDB，請使用下列命令：

```
node dist-tools/browser-builder.js dynamodb-2011-12-05,dynamodb-2012-08-10
```

服務標識符和 API 版本可以在服務特定的配置文件中找到：https://github.com/aws/ aws-sdk-js /樹/主/ api。

建置所有服務

您可以透過包含 `all` 參數來建立所有服務和 API 版本：

```
node dist-tools/browser-builder.js all > aws-sdk-full.js
```

建立特定服務

若要自訂在建置中包含的特定一組服務，請將 `AWS_SERVICES` 環境變數傳遞給 Browserify 命令，其中包含您需要的服務清單。下列範例會建置 Amazon EC2、Amazon S3 和 DynamoDB 服務。

```
$ AWS_SERVICES=ec2,s3,dynamodb browserify index.js > browser-app.js
```

使用 Browserify 來將軟體開發套件建立為相依性

Node.js 擁有以模型為基礎的機制，可用來包含來自第三方開發人員的程式碼和功能。在 Web 瀏覽器中 JavaScript 運行本身不支持這種模塊化方法。然而，您可以使用名為 Browserify 的工具，來使用 Node.js 模組方法和使用瀏覽器中為 Node.js 所寫的模組。Browserify 會將瀏覽器指令碼的模組相依性建置到可在瀏覽器中使用的單一、獨立 JavaScript 檔案。

您可以使用 Browserify 來建立做為任何瀏覽器指令碼程式庫相依性的軟體開發套件。例如，以下 Node.js 程式碼需要軟體開發套件：

```
var AWS = require('aws-sdk');
var s3 = new AWS.S3();
s3.listBuckets(function(err, data) { console.log(err, data); });
```

您可以使用 Browserify 來將此範例程式碼編譯為與瀏覽器相容的版本：

```
$ browserify index.js > browser-app.js
```

可透過 `browser-app.js` 在瀏覽器中取得應用程式 (包含其軟體開發套件相依性)。

如需 Browserify 的詳細資訊，請參閱 Browserify 網站。

跨來源資源分享 (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 JavaScript 環境會強制執行 CORS 安全性，您就必須使用服務設定 CORS。

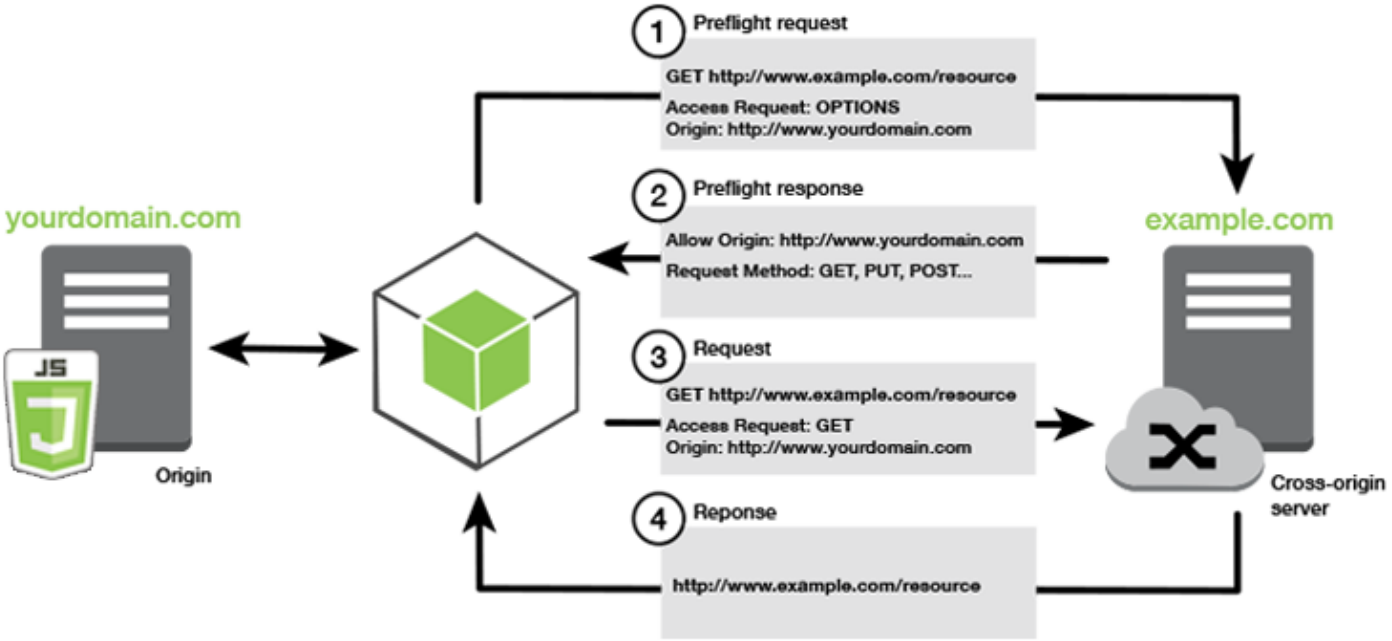
CORS 會根據以下要素判斷是否允許在跨來源請求中共享資源：

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

CORS 的運作方式

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。



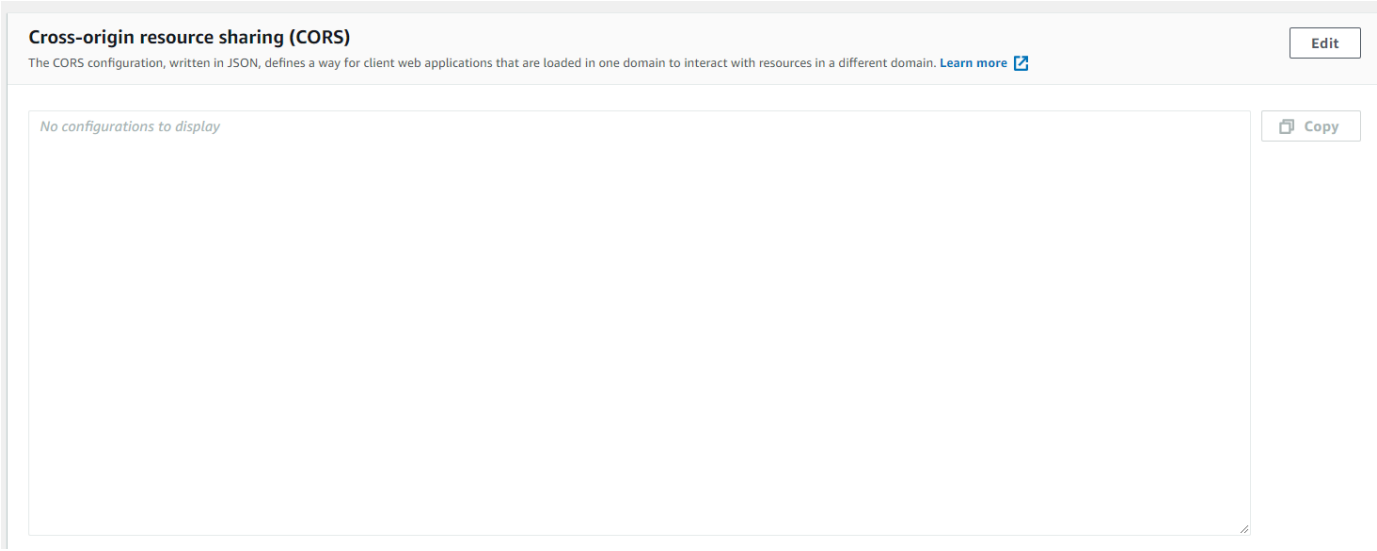
是否需要設定 CORS

您需要提供 CORS 組態 Amazon S3 儲存貯體，才能對其執行操作。某些 JavaScript 環境可能不會強制執行 CORS，所以不需要設定 CORS。例如，若您是從 Amazon S3 儲存貯體託管應用程式，並透過`*.s3.amazonaws.com`或其他特定端點，請求將不會存取外部網域。因此，這個組態就不需要 CORS。在這種情況下，CORS 仍可用於 Amazon S3 以外的其他服務。

為 Amazon S3 儲存貯體設定 CORS

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

1. 在 Amazon S3 主控台中，選擇要編輯的儲存貯體。
2. 選取Permissions (許可)選項卡，然後罵到跨來源資源分享 (CORS)面板。



3. 選擇Edit (編輯)，然後在CORS 組態編輯器，然後選擇Save。

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> 會指定允許提出跨網域請求的網域來源。
- <AllowedMethod> 會指定跨網域請求中允許的請求類型 (GET、PUT、POST、DELETE、HEAD)。
- <AllowedHeader> 會指定預檢請求中允許使用的標頭。

有關示例配置，請參閱如何在儲存貯體上設定 CORS？中的Amazon Simple Storage Service 用户指南。

CORS 組態範例

以下 CORS 組態範例允許使用者從網域 example.org 檢視、新增、移除或更新儲存貯體內的物件，但仍建議您將 <AllowedOrigin> 範圍限定在網站的網域。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，它能讓瀏覽器安全模型允許向 Amazon S3 發送請求。您需要透過儲存貯體許可或 IAM 角色許可來設定許可。

您可以使用ExposeHeader，讓軟體開發套件能讀取 Amazon S3 返回的響應標頭。舉例而言，如果您想讀取來自 PUT 或分段上傳的 ETag 標頭，就需要在組態中加入 ExposeHeader 標籤，如上述範例所示。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。

使用 Webpack 綁定應用程式

瀏覽器指令碼或 Node.js 中的 Web 應用程式會使用程式碼模組來建立依存項目。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。若要管理依存項目，則可使用 webpack 等模組綁定程式。

webpack 模組綁定程式會剖析應用程式的程式碼，進而搜尋 import 或 require 陳述式以建立配套，其中包含應用程式需要的所有資產。如此一來，即可透過網頁輕鬆提供資產。您可以將當做輸出配套所含的其中一個依存項目，並將其加入 webpack。

如需 webpack 的詳細資訊，請參閱 GitHub 上的 webpack 模組綁定程式。

安裝 Webpack

您必須先安裝 npm (Node.js 套件管理工具)，才能安裝 webpack 模組綁定程式。若要安裝 webpack CLI 與 JavaScript 模組，請輸入下列命令。

```
npm install webpack
```

您可能也需要安裝允許載入 JSON 檔案的 webpack 外掛程式。輸入下列命令，即可安裝 JSON 載入器外掛程式。

```
npm install json-loader
```

設定 Webpack

根據預設，webpack 會在專案的根目錄中搜尋名為 webpack.config.js 的 JavaScript 檔案；此檔案能夠指定組態選項。以下是 webpack.config.js 組態檔案的範例。

```
// 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'
  },
  module: {
    /**
      * Tell webpack how to load 'json' files.
      * When webpack encounters a 'require()' statement
      * where a 'json' file is being imported, it will use
      * the json-loader.
      */
    loaders: [
      {
        test: /\.json$/,
        loaders: ['json']
      }
    ]
  }
}
```

本範例會將 `browser.js` 指定為進入點。進入點指的是 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`，這個輸出檔案將包含應用程式需要執行的所有 JavaScript。如果進入點中指定的程式碼匯入或需要其他模組 (如用於 JavaScript 的軟體開發套件)，則您無需在組態內指定該程式碼，即可予以綁定。

在先前安裝的 `json-loader` 外掛程式中，組態可用來指定 webpack 匯入 JSON 檔案的方式。webpack 依預設僅支援 JavaScript，但其會使用載入器來新增支援匯入其他檔案類型。由於用來建立 JavaScript 的軟體開發套件廣泛利用 JSON 檔案，因此在`json-loader`不包含。

執行 Webpack

若要使用 webpack 建置應用程式，請將以下內容新增至 `package.json` 檔案中的 `scripts` 物件。

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

此處 `package.json` 範例會示範新增 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": "^2.6.1"
  },
  "devDependencies": {
    "json-loader": "^0.5.4",
    "webpack": "^1.13.2"
  }
}
```

請輸入下列命令以建置應用程式。

```
npm run build
```

接著，webpack 模組綁定程式會在專案的根目錄中產生您所指定的 JavaScript 檔案。

使用 Webpack 配套

若要在瀏覽器指令碼中使用配套，您可以使用 `<script>` 標籤來採納該配套，如下方範例所示。

```
<!DOCTYPE html>
<html>
    <head>
        <title>AWS SDK with webpack</title>
    </head>
    <body>
        <div id="list"></div>
        <script src="bundle.js"></script>
    </body>
</html>
```

匯入個別服務

webpack 的其中一個優點，就是其可剖析程式碼中的依存項目，且只會綁定應用程式所需的程式碼。如果您使用的是用來建立 JavaScript 的軟體開發套件，則只綁定應用程式實際使用的軟體開發套件部分能大幅縮減 webpack 輸出的大小。

請參考下方用來建立 Amazon S3 服務物件的程式碼範例。

```
// Import the AWS SDK
var AWS = require('aws-sdk');

// Set credentials and Region
// This can also be done directly on the service client
AWS.config.update({region: 'us-west-1', credentials: {YOUR_CREDENTIALS}});

var s3 = new AWS.S3({apiVersion: '2006-03-01'});
```

`require()` 函數會指定整個軟體開發套件。通過此程式碼所產生的 webpack 配套會包含完整的軟體開發套件，但在只有使用 Amazon S3 客户端類別的情況下，並不需要完整的軟體開發套件。如果僅包含 Amazon S3 服務所需的軟體開發套件部分，就能大幅縮減該配套的大小。即使是設定組態時也不需要完整的軟體開發套件，這是因為您可以在 Amazon S3 服務物件上設定組態資料。

當同一個程式碼只含有軟體開發套件的 Amazon S3 部分時，便會如下所示。

```
// Import the Amazon S3 service client
var S3 = require('aws-sdk/clients/s3');

// Set credentials and Region
var s3 = new S3({
    apiVersion: '2006-03-01',
    region: 'us-west-1',
    credentials: {YOUR_CREDENTIALS}
  });
```

綁定 Node.js

您能夠在組態中將 Node.js 指定為目標，藉此使用 webpack 產生要在其中執行的配套。

```
target: "node"
```

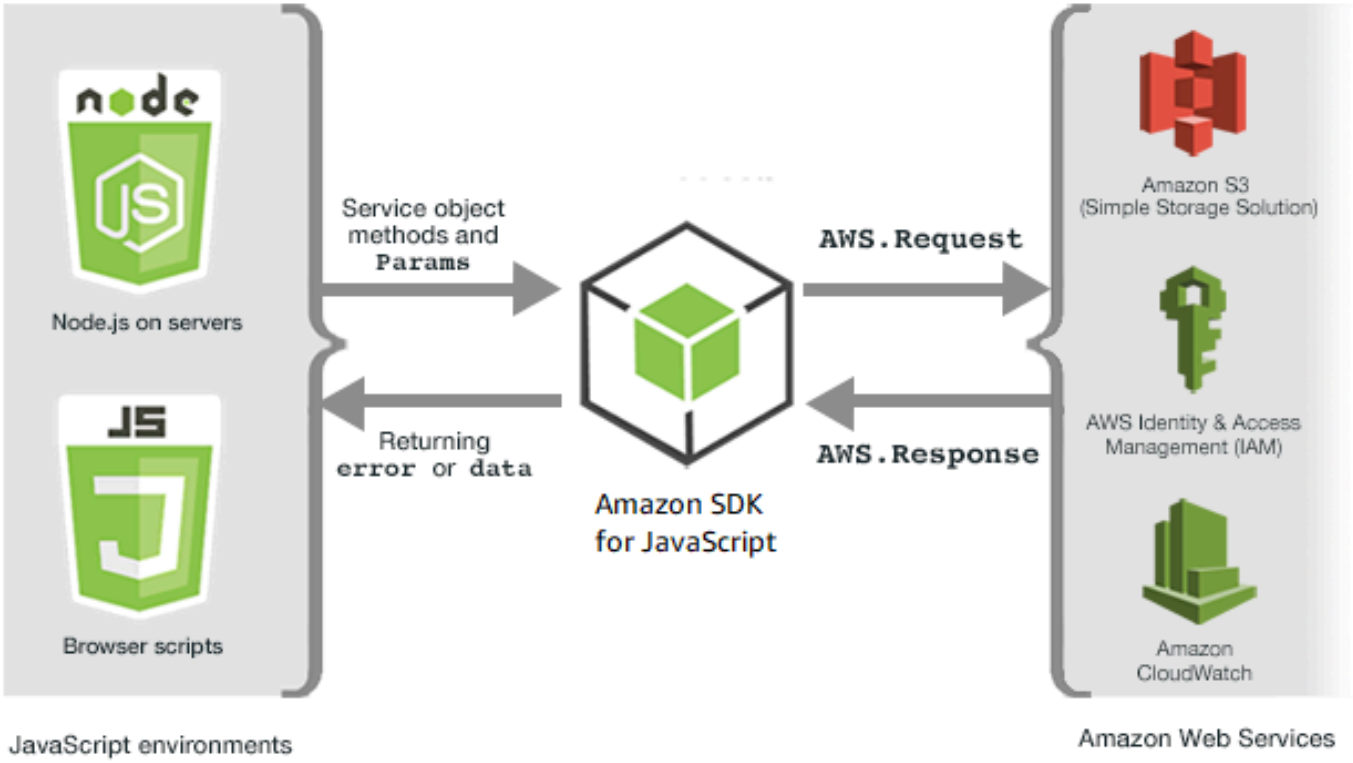
當您在磁碟空間有限的環境中執行 Node.js 應用程式時，這個做法十分實用。下方為 `webpack.config.js` 組態範例，而該組態會將 Node.js 指定為輸出目標。

```
// Import path for resolving file paths
var path = require('path');
module.exports = {
  // Specify the entry point for our app
  entry: [
    path.join(__dirname, 'node.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",
  module: {
    /**
      * Tell webpack how to load JSON files.
      * When webpack encounters a 'require()' statement
      * where a JSON file is being imported, it will use
      * the json-loader
      */
    loaders: [
      {
        test: /\.json$/,
        loaders: ['json']
      }
    ]
  }
}
```

在 SDK 中使用 中的 服務 JavaScript

AWS SDK for JavaScript 可透過一系列用戶端類別提供其支援的 服務存取權。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，提供低階APIs使用服務功能和資源。例如，Amazon DynamoDB APIs可透過 `AWS.DynamoDB`類別使用。

透過 SDK為 公開的服務會 JavaScript 遵循請求回應模式，以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 請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操作的完整請求和回應生命週期，包括任何嘗試的重試。物件會將請求封裝在 中SDK`AWS.Request`。回應SDK由 物件封裝在 中，該`AWS.Response`物件是透過多種技術之一提供給請求者，例如回呼函數或 JavaScript 承諾。

主題

- 建立和呼叫服務物件
- 記錄 AWS SDK for JavaScript 通話

- 非同步呼叫服務
- 使用回應物件
- 使用 JSON

建立和呼叫服務物件

JavaScript API 支援大多數可用的 AWS 服務。中的每個服務類別 JavaScript API都會提供其服務中每個API呼叫的存取權。如需 中服務類別、操作和參數的詳細資訊 JavaScript API，請參閱API參考 。

在 Node.js SDK中使用 時，您可以使用 將SDK套件新增至您的應用程式`require`，這可提供所有目前服務的支援。

```
var AWS = require('aws-sdk');
```

SDK 搭配瀏覽器使用 時 JavaScript，您可以使用 AWS託管SDK套件將SDK套件載入瀏覽器指令碼。若要載入SDK套件，請新增下列`<script>`元素：

```
<script src="https://sdk.amazonaws.com/js/aws-sdk-SDK_VERSION_NUMBER.min.js"></script>
```

若要尋找目前的 SDK_VERSION_NUMBER，請參閱API參考指南 JavaScript 中的 SDK 的參考。AWS SDK for JavaScript API

預設託管SDK套件可支援可用服務的子集 AWS 。如需瀏覽器託管SDK套件中預設服務的清單，請參閱API 參考中的支援服務。如果停用CORS安全性檢查，您可以將 SDK與其他 服務搭配使用。在此情況下，您可以建置自訂版本的 SDK，以包含您所需的其他服務。如需建置自訂版本的詳細資訊SDK，請參閱 建立適用於瀏覽器的軟體開發套件。

要求個別服務

JavaScript 如先前所示需要 SDK 的 將整個 包含在程式碼SDK中。或者，您也能選擇僅需加入程式碼所使用的個別服務。請考慮下列用來建立 Amazon S3 服務物件的程式碼。

```
// Import the AWS SDK
var AWS = require('aws-sdk');

// Set credentials and Region
// This can also be done directly on the service client
AWS.config.update({region: 'us-west-1', credentials: {YOUR_CREDENTIALS}});
```

```
var s3 = new AWS.S3({apiVersion: '2006-03-01'});
```

在上一個範例中，require函數會指定整個 SDK。如果只包含 SDK Amazon S3 服務所需的 部分，則透過網路傳輸的程式碼量以及程式碼的記憶體負荷會大幅減少。若需要加入個別服務，請呼叫 require 函數，其中應包括全部小寫的服務建構函數，如下所示。

```
require('aws-sdk/clients/SERVICE');
```

以下是建立先前 Amazon S3 服務物件的程式碼，其僅包含 的 Amazon S3 部分時看起來的樣子SDK。

```
// Import the Amazon S3 service client
var S3 = require('aws-sdk/clients/s3');

// Set credentials and Region
var s3 = new S3({
    apiVersion: '2006-03-01',
    region: 'us-west-1',
    credentials: {YOUR_CREDENTIALS}
  });
```

您仍然可以存取全域 AWS 命名空間，而無需連接每個服務。

```
require('aws-sdk/global');
```

這個技術很適合用來將相同組態套用至多個個別服務。舉例而言，您可以提供相同的登入資料給所有服務。要求個別服務應該能減少 Node.js 環境中的載入時間，以及記憶體耗用情形。使用綁定工具如 Browserify 或 Webpack 完成時，需要個別服務會導致 SDK變成完整大小的一小部分。這有助於解決記憶體或磁碟空間受限的環境，例如 IoT 裝置或 Lambda 函數。

建立服務物件

若要透過 存取服務功能 JavaScript API，請先建立服務物件，透過該物件存取基礎用戶端類別提供的一組功能。普遍來說，系統會提供一個用戶端類別給每個服務，但某些服務會將功能存取權限分成多個用戶端類別。

您必須針對提供功能存取權的類別建立執行個體，才能使用該功能。下列範例顯示從AWS.DynamoDB用戶端類別為 DynamoDB 建立服務物件。

```
var dynamodb = new AWS.DynamoDB({apiVersion: '2012-08-10'});
```

依預設，服務物件會設定為同時用於設定 的全域設定SDK。然而，您可以利用服務物件特定的執行時間組態資料來設定該服務物件。系統會先套用全域組態設定，隨後再套用服務特定的組態資料。

在下列範例中，Amazon EC2服務物件是使用特定區域的組態建立，但否則會使用全域組態。

```
var ec2 = new AWS.EC2({region: 'us-west-2', apiVersion: '2014-10-01'});
```

本工具除了支援套用服務特定組態至個別服務物件以外，也能讓您將服務特定組態套用至所有指定類別新建的服務物件。例如，若要設定從 Amazon EC2類別建立的所有服務物件以使用美國西部 （奧勒岡）（us-west-2）區域，請將下列項目新增至AWS.config全域組態物件。

```
AWS.config.ec2 = {region: 'us-west-2', apiVersion: '2016-04-01'};
```

鎖定服務物件的API版本

您可以在建立物件時指定 apiVersion選項，將服務物件鎖定至特定API版本的服務。在下列範例中，會建立鎖定至特定API版本的 DynamoDB 服務物件。

```
var dynamodb = new AWS.DynamoDB({apiVersion: '2011-12-05'});
```

如需鎖定服務物件API版本的詳細資訊，請參閱 鎖定 API 版本。

指定服務物件參數

呼叫服務物件的方法時， JSON會視需要在 中傳遞參數API。例如，在 Amazon S3 中，若要取得指定儲存貯體和金鑰的物件，請將下列參數傳遞至 getObject方法。如需傳遞JSON參數的詳細資訊，請參閱 使用 JSON。

```
s3.getObject({Bucket: 'bucketName', Key: 'keyName'});
```

如需 Amazon S3 參數的詳細資訊，請參閱 API Class: AWS.S3參考中的 。

此外，當您使用 params 參數建立服務物件時，也能將數值繫結至個別參數。服務物件的 params 參數值是一個映射屬性，可指定該服務物件定義的一個或多個參數值。下列範例顯示 Amazon S3 服務物件繫結至名為 的儲存貯體的Bucket參數amzn-s3-demo-bucket。

```
var s3bucket = new AWS.S3({params: {Bucket: 'amzn-s3-demo-bucket'}, apiVersion:
 '2006-03-01' });
```

一旦將服務物件繫結至儲存貯體，s3bucket 服務物件即會視 amzn-s3-demo-bucket 參數值為預設值，後續操作中無需再加以指定。如果將物件用於參數值不適用的操作，系統就會忽略任何繫結的參數值。您能夠指定新數值，以便在呼叫服務物件時覆寫此繫結參數。

```
var s3bucket = new AWS.S3({ params: {Bucket: 'amzn-s3-demo-bucket'}, apiVersion:
 '2006-03-01' });
s3bucket.getObject({Key: 'keyName'});
// ...
s3bucket.getObject({Bucket: 'amzn-s3-demo-bucket3', Key: 'keyOtherName'});
```

有關每種方法可用參數的詳細資訊，請參閱 API參考。

記錄 AWS SDK for JavaScript 通話

AWS SDK for JavaScript 配備內建記錄器，因此您可以記錄使用 SDK的 進行API呼叫 JavaScript。

若要在主控台中開啟記錄器並列印日誌項目，請將下列陳述式新增至程式碼。

```
AWS.config.logger = console;
```

以下為日誌輸出的範例。

```
[AWS s3 200 0.185s 0 retries] createMultipartUpload({ Bucket: 'amzn-s3-demo-logging-
bucket', Key: 'issues_1704' })
```

使用第三方記錄器

您也能使用第三方記錄器，前提是該記錄器需具備可寫入日誌檔案或伺服器的 `log()` 或 `write()` 操作。您必須先依指示安裝和設定自訂記錄器，才能搭配適用於 SDK的 使用 JavaScript。

logplease 正是能夠在瀏覽器指令碼或 Node.js 中使用的記錄器。在 Node.js 中，您能夠設定 logplease 以便將日誌項目寫入日誌檔案。同時，您還可以將其與 webpack 結合使用。

當您使用第三方記錄器時，所有選項都要設定完畢，才能將記錄器指派給 `AWS.Config.logger`。例如，以下會指定外部日誌檔案並設定 logplease 的日誌等級。

```
// Require AWS Node.js SDK
const AWS = require('aws-sdk')
```

```
// Require logplease
const logplease = require('logplease');
// Set external log file option
logplease.setLogfile('debug.log');
// Set log level
logplease.setLogLevel('DEBUG');
// Create logger
const logger = logplease.create('logger name');
// Assign logger to SDK
AWS.config.logger = logger;
```

如需有關 log please 的詳細資訊，請參閱 上的 logPlease Simple JavaScript Logger GitHub。

非同步呼叫服務

透過 提出的所有請求SDK都是非同步的。在撰寫瀏覽器指令碼時，請務必記住這一點。在 Web 瀏覽器中 JavaScript 執行 通常只有一個執行緒。對 AWS 服務進行非同步呼叫後，瀏覽器指令碼會繼續執行，程序中可以嘗試在傳回之前執行取決於該非同步結果的程式碼。

對 AWS 服務進行非同步呼叫包括管理這些呼叫，因此您的程式碼不會在資料可用之前嘗試使用資料。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。

主題

- 管理非同步呼叫
- 使用非同步回呼函數
- 使用請求物件事件接聽程式
- 使用 async/await
- 使用 JavaScript 承諾

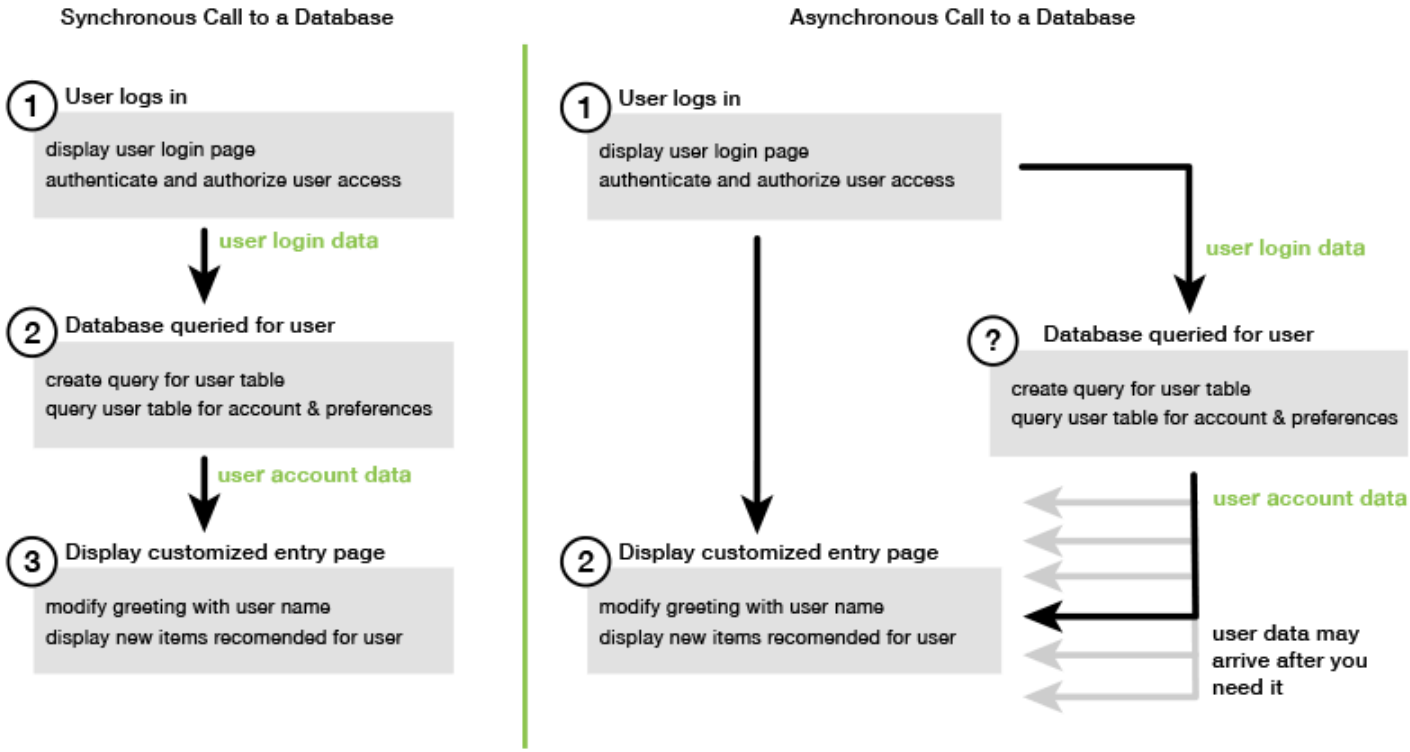
管理非同步呼叫

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. 客戶必須使用其登入憑證登入並驗證。
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。除非資料庫傳回客戶偏好設定，否則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。

請以非同步方式呼叫資料庫，避免網站因上述情況而停止運作。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。



使用非同步回呼函數

能建立 AWS.Request 物件的每個服務物件方法，都能接受將非同步回呼函數做為最後一個參數。這類回呼函數的簽章如下所示：

```
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。

```
new AWS.EC2({apiVersion: '2014-10-01'}).describeInstances(function(error, data) {
  if (error) {
    console.log(error); // an error occurred
  } else {
    console.log(data); // request succeeded
  }
});
```

存取請求和回應物件

在回呼函數中， JavaScript 關鍵字this是指大多數服務的基礎AWS.Response物件。在下方範例中，系統會在回呼函數內使用 AWS.Response 物件的 httpResponse 屬性來記錄原始回應資料和標頭，以協助偵錯。

```
new AWS.EC2({apiVersion: '2014-10-01'}).describeInstances(function(error, data) {
  if (error) {
    console.log(error); // an error occurred
    // Using this keyword to access AWS.Response object and properties
    console.log("Response data and headers: " + JSON.stringify(this.httpResponse));
  } else {
    console.log(data); // request succeeded
  }
});
```

除此之外，AWS.Response 物件具備 Request 屬性，該屬性包含由原始方法呼叫傳送的 AWS.Request，因此您也能存取所發出請求的詳細資訊。

使用請求物件事件接聽程式

在呼叫服務物件方法時，如果您沒有建立非同步回呼函數並將其做為參數傳遞，該方法便會產生 AWS.Request 物件，而此物件需要使用 send 方法手動傳送。

若要處理回應，您必須建立 AWS.Request 物件的事件接聽程式，藉此註冊回呼函數以進行方法呼叫。下方範例會說明如何建立用來呼叫服務物件方法的 AWS.Request 物件，以及成功回傳時所需的事件接聽程式。

```
// create the AWS.Request object
var request = new AWS.EC2({apiVersion: '2014-10-01'}).describeInstances();

// register a callback event handler
request.on('success', function(response) {
  // log the successful data response
  console.log(response.data);
});

// send the request
request.send();
```

呼叫 AWS.Request 物件上的 send 方法後，事件處理常式即會在服務物件收到 AWS.Response 物件時開始執行。

如需AWS.Request物件的詳細資訊，請參閱 API 參考Class: AWS.Request中的 。如需AWS.Response物件的詳細資訊，請參閱 API 參考Class: AWS.Response中的 使用回應物件或 。

變更多個回呼

您能夠在任何請求物件上註冊多個回呼，也可為不同事件或相同事件註冊多個回呼。而且，您還能鏈結回呼，如下方範例所示。

```
request.
  on('success', function(response) {
    console.log("Success!");
  }).
  on('error', function(response) {
    console.log("Error!");
  }).
```

```
  on('complete', function() {
    console.log("Always!");
  }).
  send();
```

請求物件完成事件

根據每個服務操作方法的回應，AWS.Request 物件會引發下述完成事件：

- success
- error
- complete

您能夠註冊回呼函數以回應任一事件。如需所有請求物件事件的完整清單，請參閱 API 參考Class: AWS.Request中的 。

success 事件

收到來自服務物件的成功回應時，系統就會引發 success 事件。以下是針對這個事件註冊回呼函數的方式。

```
request.on('success', function(response) {
  // event handler code
});
```

該回應會提供 data 屬性，其中包含來自服務的序列化回應資料。例如，下列呼叫 Amazon S3 服務物件listBuckets的方法

```
s3.listBuckets.on('success', function(response) {
  console.log(response.data);
}).send();
```

系統即會傳回回應，然後將下列 data 屬性內容列印至主控台。

```
{ Owner: { ID: '...', DisplayName: '...' },
  Buckets:
   [ { Name: 'someBucketName', CreationDate: someCreationDate },
     { Name: 'otherBucketName', CreationDate: otherCreationDate } ],
```

```
  RequestId: '...' }
```

error 事件

收到來自服務物件的錯誤回應時，系統就會引發 `error` 事件。以下是針對這個事件註冊回呼函數的方式。

```
request.on('error', function(error, response) {
  // event handling code
});
```

一旦引發 `error` 事件，回應的 `data` 屬性值將會是 `null`，而 `error` 屬性則會內含錯誤資料。系統會將相關聯的 `error` 物件做為第一個參數，並傳遞至註冊的回呼函數。以下方程式碼為例：

```
s3.config.credentials.accessKeyId = 'invalid';
s3.listBuckets().on('error', function(error, response) {
  console.log(error);
}).send();
```

系統即會傳回錯誤，然後將下列錯誤資料列印至主控台。

```
{ code: 'Forbidden', message: null }
```

complete 事件

系統會在服務物件呼叫完成時引發 `complete` 事件，無論呼叫成功或錯誤。以下是針對這個事件註冊回呼函數的方式。

```
request.on('complete', function(response) {
  // event handler code
});
```

不管成功與否，請使用 `complete` 事件回呼來處理任何需要執行的請求清除作業。如果您要在 `complete` 事件的回呼內部使用回應資料，請先檢查 `response.data` 或 `response.error` 屬性，再嘗試存取其中一個屬性，如下方範例所示。

```
request.on('complete', function(response) {
  if (response.error) {
```

```
    // an error occurred, handle it
  } else {
    // we can use response.data here
  }
}).send();
```

請求物件HTTP事件

AWS.Request 物件會根據每個服務操作方法的回應來提出這些HTTP事件：

- httpHeaders
- httpData
- httpUploadProgress
- httpDownloadProgress
- httpError
- httpDone

您能夠註冊回呼函數以回應任一事件。如需所有請求物件事件的完整清單，請參閱 API 參考Class: AWS.Request中的 。

httpHeaders 事件

當遠端伺服器傳送標頭時，系統就會引發 httpHeaders 事件。以下是針對這個事件註冊回呼函數的方式。

```
request.on('httpHeaders', function(statusCode, headers, response) {
  // event handling code
});
```

回呼函數的 statusCode 參數是HTTP狀態碼。headers 參數則包含回應標頭。

httpData 事件

系統會引發 httpData 事件，以便從服務串流回應資料封包。以下是針對這個事件註冊回呼函數的方式。

```
request.on('httpData', function(chunk, response) {
  // event handling code
```

```
});
```

將整個回應載入至無法實際使用的記憶體時，通常會使用這個事件來接收大型回應區塊。這個事件具有額外的 chunk 參數，其中包含一部分的伺服器實際資料。

若您針對 httpData 事件註冊回呼，回應的 data 屬性便會涵蓋請求的整個序列化輸出。如果您沒有內建處理常式所需的額外剖析和記憶體額外負荷，必須移除預設的 httpData 接聽程式。

httpUploadProgress 和 httpDownloadProgress 事件

當HTTP請求上傳更多資料時，就會引發httpUploadProgress事件。同樣地，當HTTP請求下載更多資料時，就會引發httpDownloadProgress事件。以下是針對這些事件註冊回呼函數的方式。

```
request.on('httpUploadProgress', function(progress, response) {
  // event handling code
})
.on('httpDownloadProgress', function(progress, response) {
  // event handling code
});
```

回呼函數的 progress 參數中，包含具備載入請求和總位元組數的物件。

httpError 事件

當HTTP請求失敗時，便會引發httpError事件。以下是針對這個事件註冊回呼函數的方式。

```
request.on('httpError', function(error, response) {
  // event handling code
});
```

回呼函數的 error 參數內含擲回的錯誤。

httpDone 事件

當伺服器完成資料傳送作業時，系統就會引發 httpDone 事件。以下是針對這個事件註冊回呼函數的方式。

```
request.on('httpDone', function(response) {
  // event handling code
});
```

使用 async/await

您可以在呼叫 時使用 `async/await` 模式 AWS SDK for JavaScript。接受回呼的大多數函數都不會傳回承諾。由於您只使用傳回承諾的`await`函數，若要使用將`.promise()`方法鏈結至呼叫結尾所需的`async/await`模式，並移除回呼。

下列範例使用 async/await 列出 中的所有 Amazon DynamoDB 資料表`us-west-2`。

```
var AWS = require("aws-sdk");
//Create an Amazon DynamoDB client service object.
dbClient = new AWS.DynamoDB({ region: "us-west-2" });
// Call DynamoDB to list existing tables
const run = async () => {
  try {
    const results = await dbClient.listTables({}).promise();
    console.log(results.TableNames.join("\n"));
  } catch (err) {
    console.error(err);
  }
};
run();
```

> **Note**
>
> 並非所有瀏覽器都支援非同步/等待。如需具有非同步/等待支援的瀏覽器清單，請參閱非同步函數。

使用 JavaScript 承諾

`AWS.Request.promise` 方法提供了一種能呼叫服務操作和管理非同步流程的方法，而非使用回呼。在 Node.js 與瀏覽器指令碼中，只要在沒有回呼函數的情況下呼叫服務操作，系統就會傳回 `AWS.Request` 物件。這時候，您能夠呼叫請求的 `send` 方法以進行服務呼叫。

然而，`AWS.Request.promise` 會立即開始呼叫服務，並傳回以回應 `data` 屬性履行的 promise，或是以回應 `error` 屬性拒絕的 promise。

```
var request = new AWS.EC2({apiVersion: '2014-10-01'}).describeInstances();

// create the promise object
```

```
var promise = request.promise();

// handle promise's fulfilled/rejected states
promise.then(
  function(data) {
    /* process the data */
  },
  function(error) {
    /* handle the error */
  }
);
```

接下來的範例會傳回以 data 物件履行，或是以 error 物件拒絕的 promise。使用 promise 時，單一回呼並不負責偵測錯誤。相反的，系統將根據請求成功或失敗來呼叫正確的回呼。

```
var s3 = new AWS.S3({apiVersion: '2006-03-01', region: 'us-west-2'});
var params = {
  Bucket: 'bucket',
  Key: 'example2.txt',
  Body: 'Uploaded text using the promise-based method!'
};
var putObjectPromise = s3.putObject(params).promise();
putObjectPromise.then(function(data) {
  console.log('Success');
}).catch(function(err) {
  console.log(err);
});
```

協調多個 promise

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 進行三次非同步呼叫，但只能在完成每次呼叫的承諾之後才能完成。

```
Promise.all([firstPromise, secondPromise, thirdPromise]).then(function(values) {

  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 the result to the caller of the Lambda function
  callback(null, values);
});
```

瀏覽器和 Node.js 支援 Promise

原生 JavaScript 承諾的支援（ECMAScript 2015）取決於程式碼執行的 JavaScript 引擎和版本。若要協助判斷程式碼需要執行的每個環境中對 JavaScript 承諾的支援，請參閱 上的ECMAScript相容性資料表 GitHub。

使用其他 Promise 實作

除了 ECMAScript2015 年的原生承諾實作之外，您也可以使用第三方承諾程式庫，包括：

- bluebird
- RSVP
- Q

如果您需要程式碼在不支援 5 ECMAScript 和 ECMAScript2015 年原生承諾實作的環境中執行，這些選用承諾程式庫會很有用。

若要使用第三方承諾程式庫，SDK請呼叫全域組態物件`setPromisesDependency`的方法，以設定對的承諾相依性。在瀏覽器指令碼中，請務必在載入 之前載入第三方承諾程式庫SDK。在下列範例中，SDK設定為在 bluebird 承諾程式庫中使用實作。

```
AWS.config.setPromisesDependency(require('bluebird'));
```

若要使用 JavaScript 引擎的原生承諾實作返回 ，請`setPromisesDependency`再次呼叫 ，`null`傳遞而非程式庫名稱。

使用回應物件

呼叫服務物件方法後，該方法即會將 `AWS.Response` 物件傳遞至回呼函數，以便傳回相關內容。您可以透過 `AWS.Response` 物件的屬性來存取回應內容。`AWS.Response` 物件有兩種屬性可供您存取回應內容：

- `data` 屬性

- `error` 屬性

在使用標準回呼機制的情況下，系統會提供這兩種屬性並將其做為非同步回呼函數上的參數，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);
    }
}
```

在回應物件中存取傳回的資料

`AWS.Response` 物件的 `data` 屬性包含服務請求所傳回的序列化資料。當請求成功時，`data` 屬性會包含一個物件，其中涵蓋所傳回資料的映射。如果發生錯誤，則 `data` 屬性可為 null。

以下是呼叫 DynamoDB 資料表`getItem`方法的範例，以擷取影像檔案名稱，作為遊戲的一部分。

```
// Initialize parameters needed to call DynamoDB
var slotParams = {
    Key : {'slotPosition' : {N: '0'}},
    TableName : 'slotWheels',
    ProjectionExpression: 'imageFile'
};

// prepare request object for call to DynamoDB
var request = new AWS.DynamoDB({region: 'us-west-2', apiVersion:
 '2012-08-10'}).getItem(slotParams);
// log the name of the image file to load in the slot machine
request.on('success', function(response) {
    // logs a value like "cherries.jpg" returned from DynamoDB
    console.log(response.data.Item.imageFile.S);
});
// submit DynamoDB request
request.send();
```

在此範例中，DynamoDB 資料表是影像的查詢，其顯示 中參數指定的槽機器提取結果`slotParams`。

成功呼叫getItem方法後，AWS.Response物件的 data 屬性會包含 DynamoDB 傳回的Item物件。您可以根據請求的 ProjectionExpression 參數來存取傳回的資料。在此案例中，該參數指的是 Item 物件的 imageFile 成員。由於 imageFile 成員會保留字串值，您將能透過 imageFile 的 S 子成員值來存取映像本身的檔案名稱。

瀏覽傳回的資料分頁

有時候，服務請求傳回的 data 屬性內容會橫跨多個頁面。若要存取下一頁資料，請呼叫 response.nextPage 方法。這個方法會傳送新的請求。您能夠利用回呼、成功和錯誤接聽程式來擷取該請求的回應。

您可以呼叫 response.hasNextPage 方法，檢查服務請求傳回的資料是否有其他資料頁面。這個方法會傳回布林值，指出呼叫 response.nextPage 時是否有傳回其他資料。

```
s3.listObjects({Bucket: 'bucket'}).on('success', function handlePage(response) {
    // do something with response.data
    if (response.hasNextPage()) {
        response.nextPage().on('success', handlePage).send();
    }
}).send();
```

存取來自回應物件的錯誤資訊

當服務錯誤或傳輸錯誤時，AWS.Response 物件的 error 屬性會包含可用的錯誤資料。所傳回錯誤的格式如下。

```
{ code: 'SHORT_UNIQUE_ERROR_CODE', message: 'a descriptive error message' }
```

若發生錯誤，data 屬性值即為 null。如果您要處理處於失敗狀態的事件，請務必在嘗試存取 data 屬性值前，隨時檢查該事件是否已設定 error 屬性。

存取來源請求物件

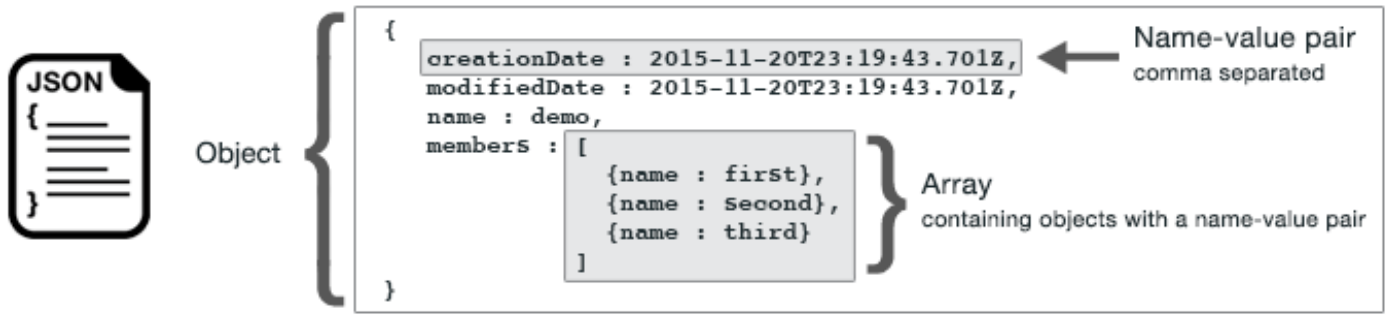
request 屬性可讓您存取來源 AWS.Request 物件。這可用來參照原始 AWS.Request 物件，以存取其所傳送的原始參數。在下方範例中，系統會使用 request 屬性來存取原始服務請求的 Key 參數。

```
s3.getObject({Bucket: 'bucket', Key: 'key'}).on('success', function(response) {
   console.log("Key was", response.request.params.Key);
}).send();
```

使用 JSON

JSON 是資料交換的格式，可人類和機器讀取。雖然 名稱JSON是JavaScript 物件記號 的縮寫，但 的格式JSON與任何程式設計語言無關。

SDK 的 JavaScript 用於JSON在提出請求時將資料傳送至服務物件，並從服務物件接收資料作為JSON。如需 的詳細資訊JSON，請參閱 https：//json.org。



JSON 以兩種方式代表資料：

- 物件，這是一種無順序的名稱/值對。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。
- 陣列是一種排序的值集合。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。

以下是包含物件陣列的JSON物件範例，其中物件代表卡片遊戲中的卡片。每個卡片都由兩個名稱值對定義，一個指定唯一值以識別該卡片，另一個指定指向對應卡片映像URL的 。

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

JSON as Service Object 參數

以下是JSON用於定義 Lambda 服務物件呼叫參數的簡單範例。

```
var pullParams = {
   FunctionName : 'slotPull',
```

```
   InvocationType : 'RequestResponse',
   LogType : 'None'
};
```

pullParams 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。叫用 Lambda 函數時，FunctionName、 InvocationType和 LogType是用來呼叫 Lambda 服務物件上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，如叫用 Lambda 函數的下列範例所示。

```
lambda = new AWS.Lambda({region: 'us-west-2', apiVersion: '2015-03-31'});
// create JSON object for service call parameters
var pullParams = {
   FunctionName : 'slotPull',
   InvocationType : 'RequestResponse',
   LogType : 'None'
};
// invoke Lambda function, passing JSON object
lambda.invoke(pullParams, function(err, data) {
   if (err) {
      console.log(err);
   } else {
      console.log(data);
   }
});
```

將資料傳回為 JSON

JSON 提供一種標準方法，在需要同時傳送多個值的應用程式部分之間傳遞資料。在傳遞給其回呼函數的 data 參數JSON中API，通常傳回的用戶端類別方法。例如，這是對 Amazon S3 用戶端類別getBucketCors方法的呼叫。

```
// call S3 to retrieve CORS configuration for selected bucket
s3.getBucketCors(bucketParams, function(err, data) {
  if (err) {
    console.log(err);
  } else if (data) {
    console.log(JSON.stringify(data));
  }
```

```
});
```

的值data是 JSON 物件，在此範例中JSON描述指定 Amazon S3 儲存貯體的目前CORS組態。

```
{
   "CORSRules": [
      {
          "AllowedHeaders":["*"],
          "AllowedMethods":["POST","GET","PUT","DELETE","HEAD"],
          "AllowedOrigins":["*"],
          "ExposeHeaders":[],
          "MaxAgeSeconds":3000
      }
   ]
}
```

SDK對於 JavaScript 代碼示例

本節中的主題包含如何 AWS SDK for JavaScript 搭配各種服務使用以執行一般工作APIs的範例。

在上的 AWS 文檔代碼示例存儲庫中找到這些示例和其他示例的源代碼 GitHub。若要提出 AWS 文件小組考慮產生的新程式碼範例，請建立新的要求。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，請參閱貢獻指導方針中的「撰寫程式碼」一節。

主題

- Amazon CloudWatch 示例
- Amazon DynamoDB 範例
- Amazon EC2 範例
- AWS Elemental MediaConvert 範例
- AWSIAM 範例
- Amazon Kinesis 示例
- Amazon S3 範例
- Amazon 簡單電子郵件服務
- Amazon 簡單通知服務示例
- Amazon SQS 範例

Amazon CloudWatch 示例

Amazon CloudWatch （CloudWatch）是一種 Web 服務，可以實時監控您的 Amazon Web Services 資源和應用程序。AWS您可以用 CloudWatch 來收集和追蹤指標，這些指標是您可以針對資源和應用程式測量的變數。 CloudWatch 警示會根據您定義的規則傳送通知或自動變更您正在監視的資源。

的 JavaScript API 會透過AWS.CloudWatch、AWS.CloudWatchEvents和AWS.CloudWatchLogs用戶端類別公開。CloudWatch 如需有關使用用 CloudWatch 戶端類別的詳細資訊 Class: AWS.CloudWatchClass: AWS.CloudWatchEvents，請參閱 API 參考資料Class: AWS.CloudWatchLogs中的、和。

主題

- 在 Amazon 中創建警報 CloudWatch
- 在 Amazon 中使用警報動作 CloudWatch
- 從 Amazon 獲取指標 CloudWatch
- 向 Amazon 活動發送 CloudWatch 活動
- 在 Amazon CloudWatch 日誌中使用訂閱篩選器

在 Amazon 中創建警報 CloudWatch

這個 Node.js 程式碼範例會說明：

- 如何擷取 CloudWatch 鬧鐘的基本資訊。
- 如何建立和刪除 CloudWatch 鬧鐘。

使用案例

警示會監看指定時段內的單一指標，並根據與多個時段內指定閾值相對的指標值來執行一或多個動作。

在此範例中，Node.js 模組系列會用於在 CloudWatch 中建立警示。Node.js 模組會使用 SDK JavaScript 來建立使用用AWS.CloudWatch戶端類別的下列方法警示：

- describeAlarms
- putMetricAlarm
- deleteAlarms

如需有關 CloudWatch 警示的詳細資訊，請參閱 Amazon CloudWatch 使用者指南中的建立 Amazon CloudWatch 警示。

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：

- 安裝 Node.js。如需安裝 Node.js 的詳細資訊，請參閱 Node.js 網站。
- 透過使用者登入資料建立共用組態檔。如需提供共用登入資料檔案的詳細資訊，請參閱 從共用登入資料檔案中在 Node.js 中載入登入資料。

說明警示

以檔名 cw_describealarms.js 建立一個 Node.js 模組。請務必依前述的內容來設定軟體開發套件。若要存取 CloudWatch，請建立AWS.CloudWatch服務物件。建立 JSON 物件來保留警示說明擷取所用的參數，限制返回到狀態為 INSUFFICIENT_DATA 的警示。然後呼叫 AWS.CloudWatch 服務物件的 describeAlarms 方法。

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

若要執行範例，請在命令列中輸入以下內容。

```
node cw_describealarms.js
```

您可以在這裡找到此範例程式碼 GitHub。

建立 CloudWatch 量度的警示

以檔名 `cw_putmetricalarm.js` 建立一個 Node.js 模組。請務必依前述的內容來設定軟體開發套件。若要存取 CloudWatch，請建立`AWS.CloudWatch`服務物件。為根據指標建立警示所需的參數建立 JSON 物件，在此情況下為 Amazon EC2 執行個體的 CPU 使用率。其餘參數都已設定，當指標超過 70% 的閾值時觸發警示。然後呼叫 `AWS.CloudWatch` 服務物件的 `describeAlarms` 方法。

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

若要執行範例，請在命令列中輸入以下內容。

```
node cw_putmetricalarm.js
```

您可以在這裡找到此範例程式碼 GitHub。

刪除警示

以檔名 cw_deletealarms.js 建立一個 Node.js 模組。請務必依前述的內容來設定軟體開發套件。若要存取 CloudWatch，請建立AWS.CloudWatch服務物件。建立 JSON 物件，來保留您要刪除的警示名稱。然後呼叫 AWS.CloudWatch 服務物件的 deleteAlarms 方法。

```
// 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);
  }
```

```
});
```

若要執行範例，請在命令列中輸入以下內容。

```
node cw_deletealarms.js
```

您可以在這裡找到此範例程式碼 GitHub。

在 Amazon 中使用警報動作 CloudWatch

這個 Node.js 程式碼範例會說明：

- 如何根據 CloudWatch 警示自動變更 Amazon EC2 執行個體的狀態。

使用案例

您可以使用警示動作建立自動停止、終止、重新開機或復原 Amazon EC2 執行個體的警示。當執行個體不再需要執行，您可以使用停止或終止動作。您可以使用重新啟動和恢復動作來自動重新啟動這些執行個體。

在此範例中，使用一系列 Node.js 模組來定義警示動作，以觸 CloudWatch 發 Amazon EC2 執行個體的重新開機。Node.js 模組會使用用`CloudWatch`戶端類別的下列方法 JavaScript 來管理 Amazon EC2 執行個體的開發套件：

- `enableAlarmActions`
- `disableAlarmActions`

如需 CloudWatch 警示動作的詳細資訊，請參閱 Amazon CloudWatch 使用者指南中的建立警示以停止、終止、重新開機或復原執行個體。

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：

- 安裝 Node.js。如需安裝 Node.js 的詳細資訊，請參閱 Node.js 網站。
- 透過使用者登入資料建立共用組態檔。如需提供共用登入資料檔案的詳細資訊，請參閱 從共用登入資料檔案中在 Node.js 中載入登入資料。
- 建立 IAM 角色，其政策授予描述、重新啟動、停止或終止 Amazon EC2 執行個體的權限。如需有關建立 IAM 角色的詳細資訊，請參閱 IAM 使用者指南中的建立角色以將許可委派給AWS服務。

您可以使用下列角色政策來建立 IAM 角色。

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
            "cloudwatch:Describe*",
            "ec2:Describe*",
            "ec2:RebootInstances",
            "ec2:StopInstances*",
            "ec2:TerminateInstances"
         ],
         "Resource": [
            "*"
         ]
      }
   ]
}
```

JavaScript 通過創建全局配置對象，然後為代碼設置區域來配置 SDK。在此範例中，區域會設為 us-west-2。

```
// Load the SDK for JavaScript
var AWS = require('aws-sdk');
// Set the Region
AWS.config.update({region: 'us-west-2'});
```

建立及啟用警示上的動作

以檔名 cw_enablealarmactions.js 建立一個 Node.js 模組。請務必依前述的內容來設定軟體開發套件。若要存取 CloudWatch，請建立AWS.CloudWatch服務物件。

建立 JSON 物件來保留用於建立警示的參數，指定 ActionsEnabled 為 true，和一系列將觸發警示的動作的 ARN。呼叫 AWS.CloudWatch 服務物件的 putMetricAlarm 方法，它會在警示不存在時建立警示，若警示存在則會更新。

在的回呼函數中putMetricAlarm，成功完成後，會建立包含 CloudWatch 警示名稱的 JSON 物件。呼叫 enableAlarmActions 方法，以啟用警示動作。

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

若要執行範例，請在命令列中輸入以下內容。

```
node cw_enablealarmactions.js
```

您可以在這裡找到此範例程式碼 GitHub。

停用警示上的動作

以檔名 `cw_disablealarmactions.js` 建立一個 Node.js 模組。請務必依前述的內容來設定軟體開發套件。若要存取 CloudWatch，請建立`AWS.CloudWatch`服務物件。建立包含 CloudWatch 警示名稱的 JSON 物件。呼叫 `disableAlarmActions` 方法以停用此警示的動作。

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

若要執行範例，請在命令列中輸入以下內容。

```
node cw_disablealarmactions.js
```

您可以在這裡找到此範例程式碼 GitHub。

從 Amazon 獲取指標 CloudWatch

這個 Node.js 程式碼範例會說明：

- 如何擷取已發佈 CloudWatch 量度清單。
- 如何將資料點發佈至 CloudWatch 指標。

使用案例

指標是有關您系統效能的資料。您可以啟用某些資源的詳細監控，例如 Amazon EC2 執行個體或您自己的應用程式指標。

在此範例中，使用一系列 Node.js 模組來取得指標，以 CloudWatch 及將事件傳送至 Amazon CloudWatch 事件。Node.js 模組會使用 SDK 來取得 JavaScript 使用 CloudWatch用`CloudWatch`戶端類別的下列方法的指標：

- `listMetrics`
- `putMetricData`

如需有關指 CloudWatch 標的詳細資訊，請參閱 Amazon 使用者指南中的使 CloudWatch 用 Amazon 指 CloudWatch 標。

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：

- 安裝 Node.js。如需安裝 Node.js 的詳細資訊，請參閱 Node.js 網站。

- 透過使用者登入資料建立共用組態檔。如需提供共用登入資料檔案的詳細資訊，請參閱 從共用登入資料檔案中在 Node.js 中載入登入資料。

列出指標

以檔名 cw_listmetrics.js 建立一個 Node.js 模組。請務必依前述的內容來設定軟體開發套件。若要存取 CloudWatch，請建立AWS.CloudWatch服務物件。建立 JSON 物件，其包含列出 AWS/Logs 命名空間內指標所需的參數。呼叫 listMetrics 方法來列出 IncomingLogEvents 指標。

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

若要執行範例，請在命令列中輸入以下內容。

```
node cw_listmetrics.js
```

您可以在這裡找到此範例程式碼 GitHub。

提交自訂指標

以檔名 cw_putmetricdata.js 建立一個 Node.js 模組。請務必依前述的內容來設定軟體開發套件。若要存取 CloudWatch，請建立AWS.CloudWatch服務物件。建立 JSON 物件，其中包含提交 PAGES_VISITED 自訂指標的資料點所需的參數。呼叫 putMetricData 方法。

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

若要執行範例，請在命令列中輸入以下內容。

```
node cw_putmetricdata.js
```

您可以在這裡找到此範例程式碼 GitHub。

向 Amazon 活動發送 CloudWatch 活動

這個 Node.js 程式碼範例會說明：

- 如何建立及更新觸發事件所用的規則。
- 如何定義一個或多個目標以回應事件。
- 如何傳送與目標相符的事件以進行處理。

使用案例

CloudWatch 事件提供近乎即時的系統事件串流，描述 Amazon Web Services 資源對任何不同目標的變更。使用簡單的規則，您可以比對事件，並將這些事件轉傳到一或多個目標函數或串流。

在此範例中，一系列 Node.js 模組用於將事件傳送至 CloudWatch 事件。Node.js 模組會使用 SDK JavaScript 來管理使用用`CloudWatchEvents`戶端類別的下列方法執行個體：

- `putRule`
- `putTargets`
- `putEvents`

如需有關 CloudWatch 事件的詳細資訊，請參閱 Amazon 事件使用者指南 PutEvents中的使用新增 CloudWatch 事件。

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：

- 安裝 Node.js。如需安裝 Node.js 的詳細資訊，請參閱 Node.js 網站。

- 透過使用者登入資料建立共用組態檔。如需提供共用登入資料檔案的詳細資訊，請參閱 從共用登入資料檔案中在 Node.js 中載入登入資料。
- 使用地獄世界藍圖建立 Lambda 函數，做為事件的目標。要了解操作方法，請參閱 Amazon CloudWatch 事件使用者指南中的步驟 1：建立AWS Lambda函數。
- 建立 IAM 角色，其政策授予 E CloudWatch vents 權限，其中包括`events.amazonaws.com`作為受信任的實體。如需有關建立 IAM 角色的詳細資訊，請參閱 IAM 使用者指南中的建立角色以將許可委派給AWS服務。

您可以使用下列角色政策來建立 IAM 角色。

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Sid": "CloudWatchEventsFullAccess",
         "Effect": "Allow",
         "Action": "events:*",
         "Resource": "*"
      },
      {
         "Sid": "IAMPassRoleForCloudWatchEvents",
         "Effect": "Allow",
         "Action": "iam:PassRole",
         "Resource": "arn:aws:iam::*:role/AWS_Events_Invoke_Targets"
      }
   ]
}
```

您可以使用下列信任關係來建立 IAM 角色。

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Principal": {
            "Service": "events.amazonaws.com"
         },
         "Action": "sts:AssumeRole"
      }
   ]
```

```
}
```

建立排程的規則

以檔名 cwe_putrule.js 建立一個 Node.js 模組。請務必依前述的內容來設定軟體開發套件。若要存取 CloudWatch 事件，請建立AWS.CloudWatchEvents服務物件。建立 JSON 物件，其中包含指定新排程規則所需的參數，其中包含下列項目：

- 規則的名稱
- 您先前建立的 IAM 角色的 ARN
- 排定每 5 分鐘為觸發的規則表達式

呼叫 putRule 方法來建立規則。回呼會傳回新的或更新規則的 ARN。

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

若要執行範例，請在命令列中輸入以下內容。

```
node cwe_putrule.js
```

您可以在這裡找到此範例程式碼 GitHub。

新增AWS Lambda函數目標

以檔名 cwe_puttargets.js 建立一個 Node.js 模組。請務必依前述的內容來設定軟體開發套件。若要存取 CloudWatch 事件，請建立AWS.CloudWatchEvents服務物件。建立 JSON 物件，其中包含指定要附加目標之規則所需的參數，包括您建立的 Lambda 函數的 ARN。呼叫 AWS.CloudWatchEvents 服務物件的 putTargets 方法。

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

若要執行範例，請在命令列中輸入以下內容。

```
node cwe_puttargets.js
```

您可以在這裡找到此範例程式碼 GitHub。

傳送事件

以檔名 cwe_putevents.js 建立一個 Node.js 模組。請務必依前述的內容來設定軟體開發套件。若要存取 CloudWatch 事件，請建立AWS.CloudWatchEvents服務物件。建立 JSON 物件，其包含傳送事件所需的參數。對於每個事件，包括事件來源、受事件影響的任何資源的 ARN，以及事件的詳細資訊。呼叫 AWS.CloudWatchEvents 服務物件的 putEvents 方法。

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

若要執行範例，請在命令列中輸入以下內容。

```
node cwe_putevents.js
```

您可以在這裡找到此範例程式碼 GitHub。

在 Amazon CloudWatch 日誌中使用訂閱篩選器

這個 Node.js 程式碼範例會說明：

- 如何在記錄檔中建立及刪除 CloudWatch 記錄事件的篩選器。

使用案例

訂閱可從 CloudWatch 日誌存取日誌事件的即時摘要，並將該摘要傳遞至其他服務 (例如 Amazon Kinesis 串流)AWS Lambda，或用於自訂處理、分析或載入到其他系統。訂閱篩選器會定義用於篩選哪些記錄事件傳遞至您的AWS資源的模式。

在此範例中，會使用一系列 Node.js 模組來列出、建立和刪除 CloudWatch 記錄中的訂閱篩選器。記錄事件的目的地是 Lambda 函數。Node.js 模組會使用 SDK JavaScript 來管理使用用`CloudWatchLogs`戶端類別的下列方法的訂閱篩選器：

- `putSubscriptionFilters`
- `describeSubscriptionFilters`
- `deleteSubscriptionFilter`

如需有關 CloudWatch 日誌訂閱的詳細資訊，請參閱 Amazon 日誌使用者指南中的使用訂閱即時處理 CloudWatch 日誌資料。

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：

- 安裝 Node.js。如需安裝 Node.js 的詳細資訊，請參閱 Node.js 網站。
- 透過使用者登入資料建立共用組態檔。如需提供共用登入資料檔案的詳細資訊，請參閱 從共用登入資料檔案中在 Node.js 中載入登入資料。
- 建立 Lambda 函數做為日誌事件的目的地。您將需要用到此函數的 ARN。如需設定 Lambda 函數的詳細資訊，請參閱 Amazon CloudWatch 日誌使用者指南AWS Lambda中的訂閱篩選器。

- 建立 IAM 角色，其政策授予呼叫您建立的 Lambda 函數的權限，並授予對 CloudWatch 記錄的完整存取權，或將下列政策套用至您為 Lambda 函數建立的執行角色。如需有關建立 IAM 角色的詳細資訊，請參閱 IAM 使用者指南中的建立角色以將許可委派給AWS服務。

您可以使用下列角色政策來建立 IAM 角色。

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
            "logs:CreateLogGroup",
            "logs:CreateLogStream",
            "logs:PutLogEvents"
         ],
         "Resource": "arn:aws:logs:*:*:*"
      },
      {
         "Effect": "Allow",
         "Action": [
            "lambda:InvokeFunction"
         ],
         "Resource": [
            "*"
         ]
      }
   ]
}
```

說明現有的訂閱篩選條件

以檔名 `cwl_describesubscriptionfilters.js` 建立一個 Node.js 模組。請務必依前述的內容來設定軟體開發套件。若要存取 CloudWatch 記錄檔，請建立`AWS.CloudWatchLogs`服務物件。建立 JSON 物件，其包括說明現有篩選條件所需的參數，包括日誌群組名稱和您要說明的篩選條件數量上限。呼叫 `describeSubscriptionFilters` 方法。

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

若要執行範例，請在命令列中輸入以下內容。

```
node cwl_describesubscriptionfilters.js
```

您可以在這裡找到此範例程式碼 GitHub。

建立訂閱篩選條件

以檔名 `cwl_putsubscriptionfilter.js` 建立一個 Node.js 模組。請務必依前述的內容來設定軟體開發套件。若要存取 CloudWatch 記錄檔，請建立`AWS.CloudWatchLogs`服務物件。建立 JSON 物件，其中包含建立篩選器所需的參數，包括目的地 Lambda 函數的 ARN、篩選器名稱、用於篩選的字串模式，以及記錄群組的名稱。呼叫 `putSubscriptionFilters` 方法。

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

若要執行範例，請在命令列中輸入以下內容。

```
node cwl_putsubscriptionfilter.js
```

您可以在這裡找到此範例程式碼 GitHub。

刪除訂閱篩選條件

以檔名 `cwl_deletesubscriptionfilters.js` 建立一個 Node.js 模組。請務必依前述的內容來設定軟體開發套件。若要存取 CloudWatch 記錄檔，請建立`AWS.CloudWatchLogs`服務物件。建立 JSON 物件，其包括刪除篩選條件所需的參數，包括篩選名稱和日誌群組。呼叫 `deleteSubscriptionFilters` 方法。

```
// 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);
```

```
  }
});
```

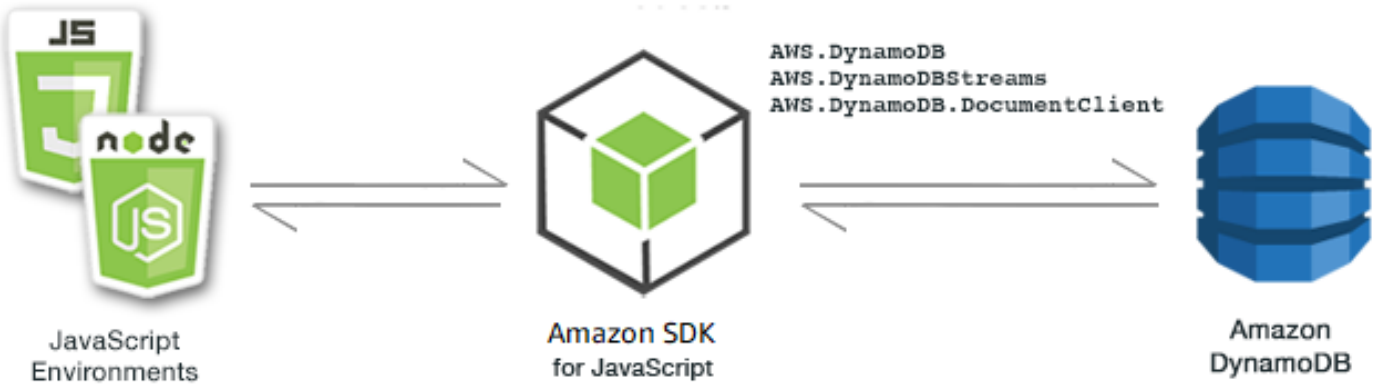
若要執行範例，請在命令列中輸入以下內容。

```
node cwl_deletesubscriptionfilter.js
```

您可以在這裡找到此範例程式碼 GitHub。

Amazon DynamoDB 範例

Amazon DynamoDB 是全受管的 NoSQL 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。



DynamoDB 的 JavaScript API 會透過AWS.DynamoDB、AWS.DynamoDBStreams和AWS.DynamoDB.DocumentClient用戶端類別公開。如需有關使用 DynamoDB 用戶端類別的詳細資訊 Class: AWS.DynamoDB，請參閱 API 參考資料Class: AWS.DynamoDB.DocumentClient中的Class: AWS.DynamoDBStreams、和。

主題

- 在 DynamoDB 中建立和使用表格
- 在 DynamoDB 中讀取和寫入單一項目
- 在 DynamoDB 中 Batch 讀取和寫入項目
- 查詢和掃描 DynamoDB 資料表
- 使用 DynamoDB 用戶端

在 DynamoDB 中建立和使用表格

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。

使用案例

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 JavaScript 來建立和使用用`AWS.DynamoDB`戶端類別的下列方法來處理資料表：

- createTable
- listTables
- describeTable
- deleteTable

先決條件任務

若要設定和執行此範例，請先完成這些任務：

- 安裝 Node.js。如需詳細資訊，請參閱 Node.js 網站。
- 透過使用者登入資料建立共用組態檔。如需提供共用登入資料檔案的詳細資訊，請參閱 從共用登入資料檔案中在 Node.js 中載入登入資料。

建立資料表

以檔名 `ddb_createtable.js` 建立一個 Node.js 模組。請務必依前述的內容來設定軟體開發套件。若要存取 DynamoDB，請建立`AWS.DynamoDB`服務物件。建立 JSON 物件，其中包含建立資料表所需的參數，在此範例中包含每個屬性的名稱和資料類型、主要結構描述、資料表名稱和要佈建的傳輸量單位。呼叫 DynamoDB 服務物件的`createTable`方法。

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

若要執行範例，請在命令列中輸入以下內容。

```
node ddb_createtable.js
```

您可以在這裡找到此範例程式碼 GitHub。

列出資料表

以檔名 ddb_listtables.js 建立一個 Node.js 模組。請務必依前述的內容來設定軟體開發套件。若要存取 DynamoDB，請建立AWS.DynamoDB服務物件。建立 JSON 物件，其中包含列出資料表所需的參數，在此範例中，列出的資料表數限制為 10。呼叫 DynamoDB 服務物件的listTables方法。

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

若要執行範例，請在命令列中輸入以下內容。

```
node ddb_listtables.js
```

您可以在這裡找到此範例程式碼 GitHub。

說明資料表

以檔名 ddb_describetable.js 建立一個 Node.js 模組。請務必依前述的內容來設定軟體開發套件。若要存取 DynamoDB，請建立AWS.DynamoDB服務物件。建立 JSON 物件，其中包含描述資料表所需的參數，在此範例中會包含要提供做為命令列參數的資料表名稱。呼叫 DynamoDB 服務物件的describeTable方法。

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

若要執行範例，請在命令列中輸入以下內容。

```
node ddb_describetable.js TABLE_NAME
```

您可以在這裡找到此範例程式碼 GitHub。

刪除資料表

以檔名 ddb_deletetable.js 建立一個 Node.js 模組。請務必依前述的內容來設定軟體開發套件。若要存取 DynamoDB，請建立AWS.DynamoDB服務物件。建立 JSON 物件，其中包含刪除資料表所需的參數，在此範例中會包含要提供做為命令列參數的資料表名稱。呼叫 DynamoDB 服務物件的deleteTable方法。

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

若要執行範例，請在命令列中輸入以下內容。

```
node ddb_deletetable.js TABLE_NAME
```

您可以在這裡找到此範例程式碼 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。

使用案例

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AWS.DynamoDB戶端類別的下列方法：

- putItem
- getItem
- deleteItem

先決條件任務

若要設定和執行此範例，請先完成這些任務：

- 安裝 Node.js。如需詳細資訊，請參閱 Node.js 網站。
- 透過使用者登入資料建立共用組態檔。如需提供共用登入資料檔案的詳細資訊，請參閱 從共用登入資料檔案中在 Node.js 中載入登入資料。
- 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 中建立和使用表格

寫入項目

以檔名 ddb_putitem.js 建立一個 Node.js 模組。請務必依前述的內容來設定軟體開發套件。若要存取 DynamoDB，請建立AWS.DynamoDB服務物件。建立 JSON 物件，其中包含新增項目所需的參數，在此範例中包含資料表名稱、定義要設定屬性的映射以及每個屬性的值。呼叫 DynamoDB 服務物件的putItem方法。

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

若要執行範例，請在命令列中輸入以下內容。

```
node ddb_putitem.js
```

您可以在這裡找到此範例程式碼 GitHub。

取得項目

以檔名 ddb_getitem.js 建立一個 Node.js 模組。請務必依前述的內容來設定軟體開發套件。若要存取 DynamoDB，請建立AWS.DynamoDB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，getItem 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 JSON 物件，其中包含取得項目所需的參數，在此範例中包含資料表名稱、您要取得之項目的索引鍵值以及會識別您要擷取之項目屬性的投射表達式。呼叫 DynamoDB 服務物件的getItem方法。

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

若要執行範例，請在命令列中輸入以下內容。

```
node ddb_getitem.js
```

您可以在這裡找到此範例程式碼 GitHub。

刪除項目

以檔名 ddb_deleteitem.js 建立一個 Node.js 模組。請務必依前述的內容來設定軟體開發套件。若要存取 DynamoDB，請建立AWS.DynamoDB服務物件。建立 JSON 物件，其包含刪除項目所需的參數，在此範例中包括資料表名稱、以及您在刪除之項目的索引鍵名稱和值。呼叫 DynamoDB 服務物件的deleteItem方法。

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

若要執行範例，請在命令列中輸入以下內容。

```
node ddb_deleteitem.js
```

您可以在這裡找到此範例程式碼 GitHub。

在 DynamoDB 中 Batch 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。

使用案例

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

- batchGetItem
- batchWriteItem

先決條件任務

若要設定和執行此範例，請先完成這些任務：

- 安裝 Node.js。如需詳細資訊，請參閱 Node.js 網站。
- 透過使用者登入資料建立共用組態檔。如需提供共用登入資料檔案的詳細資訊，請參閱 從共用登入資料檔案中在 Node.js 中載入登入資料。

- 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 中建立和使用表格

批次讀取項目

以檔名 ddb_batchgetitem.js 建立一個 Node.js 模組。請務必依前述的內容來設定軟體開發套件。若要存取 DynamoDB，請建立AWS.DynamoDB服務物件。建立 JSON 物件，其包含取得項目批次所需的參數，在此範例中包括要讀取的一或多個資料表、要在每個資料表中讀取的索引鍵值以及會指定要傳回屬性的投射表達式。呼叫 DynamoDB 服務物件的batchGetItem方法。

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

若要執行範例，請在命令列中輸入以下內容。

```
node ddb_batchgetitem.js
```

您可以在這裡找到此範例程式碼 GitHub。

批次寫入項目

以檔名 ddb_batchwriteitem.js 建立一個 Node.js 模組。請務必依前述的內容來設定軟體開發套件。若要存取 DynamoDB，請建立AWS.DynamoDB服務物件。建立 JSON 物件，其包含取得項目批次所需的參數，在此範例中包括要寫入項目的資料表、要為每個項目寫入的索引鍵以及含值的屬性。呼叫 DynamoDB 服務物件的batchWriteItem方法。

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

若要執行範例，請在命令列中輸入以下內容。

```
node ddb_batchwriteitem.js
```

您可以在這裡找到此範例程式碼 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。

使用案例

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 JavaScript 來查詢和掃描使用 DynamoDB 用戶端類別的下列方法的資料表：

- query
- scan

先決條件任務

若要設定和執行此範例，請先完成這些任務：

- 安裝 Node.js。如需詳細資訊，請參閱 Node.js 網站。
- 透過使用者登入資料建立共用組態檔。如需提供共用登入資料檔案的詳細資訊，請參閱 從共用登入資料檔案中在 Node.js 中載入登入資料。
- 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 中建立和使用表格

查詢資料表

此範例會查詢內含影片系列之相關集資訊的表格，會為超過 9 集且子標題中含有指定片語的第二季集傳回集標題和子標題。

以檔名 ddb_query.js 建立一個 Node.js 模組。請務必依前述的內容來設定軟體開發套件。若要存取 DynamoDB，請建立AWS.DynamoDB服務物件。建立 JSON 物件，其包含查詢資料表所需的參數，在此範例中包括資料表名稱、查詢所需的 ExpressionAttributeValues、使用這些值來定義查詢要傳回之項目的 KeyConditionExpression 以及要為每個項目傳回的屬性值名稱。呼叫 DynamoDB 服務物件的query方法。

```
// 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 = {
  ExpressionAttributeValues: {
    ":s": { N: "2" },
    ":e": { N: "09" },
    ":topic": { S: "PHRASE" },
  },
  KeyConditionExpression: "Season = :s and Episode > :e",
  ProjectionExpression: "Episode, Title, Subtitle",
  FilterExpression: "contains (Subtitle, :topic)",
  TableName: "EPISODES_TABLE",
};
```

```
ddb.query(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    //console.log("Success", data.Items);
    data.Items.forEach(function (element, index, array) {
      console.log(element.Title.S + " (" + element.Subtitle.S + ")");
    });
  }
});
```

若要執行範例，請在命令列中輸入以下內容。

```
node ddb_query.js
```

您可以在這裡找到此範例程式碼 GitHub。

掃描資料表

以檔名 ddb_scan.js 建立一個 Node.js 模組。請務必依前述的內容來設定軟體開發套件。若要存取 DynamoDB，請建立AWS.DynamoDB服務物件。建立 JSON 物件，其包含掃描資料表項目所需的參數，在此範例中包括資料表名稱、要為每個相符項目傳回的屬性值清單，以及要篩選結果組以尋找內含指定片語的表達式。呼叫 DynamoDB 服務物件的scan方法。

```
// 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 + ")"
      );
    });
  }
});
```

若要執行範例，請在命令列中輸入以下內容。

```
node ddb_scan.js
```

您可以在這裡找到此範例程式碼 GitHub。

使用 DynamoDB 用戶端

這個 Node.js 程式碼範例會說明：

- 如何使用文件用戶端存取 DynamoDB 表格。

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，以及將註釋的響應數據轉換為本地類型。 JavaScript

如需 DynamoDB 文件用戶端類別的詳細資訊，請參閱 API 參考`AWS.DynamoDB.DocumentClient`中的。如需使用 Amazon DynamoDB 進行程式設計的詳細資訊，請參閱 Amazon DynamoDB 開發人員指南中的使用 DynamoDB 進行程式設計。

在此範例中，您可以使用一系列 Node.js 模組，使用文件用戶端在 DynamoDB 表上執行基本操作。程式碼會使用 SDK 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：

- get
- put
- update
- query
- delete

先決條件任務

若要設定和執行此範例，請先完成這些任務：

- 安裝 Node.js。如需詳細資訊，請參閱 Node.js 網站。
- 透過使用者登入資料建立共用組態檔。如需提供共用登入資料檔案的詳細資訊，請參閱 從共用登入資料檔案中在 Node.js 中載入登入資料。
- 建立一個您可以存取其項目的 DynamoDB 表格。如需使用 SDK 建立 DynamoDB 資料表的詳細資訊 JavaScript，請參閱。在 DynamoDB 中建立和使用表格您也可以使用 DynamoDB 主控台來建立資料表。

從資料表取得項目

以檔名 `ddbdoc_get.js` 建立一個 Node.js 模組。請務必依前述的內容來設定軟體開發套件。若要存取 DynamoDB，請建立一個物件`AWS.DynamoDB.DocumentClient`。建立 JSON 物件，其包含從資料表取得項目所需的參數，在此範例中包括資料表名稱、在該資料表中雜湊索引鍵的名稱，以及您要取得之項目的雜湊索引鍵值。呼叫 DynamoDB 文件用戶端的`get`方法。

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

若要執行範例，請在命令列中輸入以下內容。

```
node ddbdoc_get.js
```

您可以在這裡找到此範例程式碼 GitHub。

將項目放置在資料表中

以檔名 ddbdoc_put.js 建立一個 Node.js 模組。請務必依前述的內容來設定軟體開發套件。若要存取 DynamoDB，請建立一個物件AWS.DynamoDB.DocumentClient。建立 JSON 物件，其包含將項目寫入資料表所需的參數，在此範例中包括資料表名稱和要新增或更新，且內含雜湊索引鍵和值之項目的描述，以及要在項目上設定的屬性名稱和值。呼叫 DynamoDB 文件用戶端的put方法。

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

若要執行範例，請在命令列中輸入以下內容。

```
node ddbdoc_put.js
```

您可以在這裡找到此範例程式碼 GitHub。

在資料表中更新項目

以檔名 ddbdoc_update.js 建立一個 Node.js 模組。請務必依前述的內容來設定軟體開發套件。若要存取 DynamoDB，請建立一個物件AWS.DynamoDB.DocumentClient。建立 JSON 物件，其包含將項目寫入資料表所需的參數，在此範例中包括資料表名稱、要更新的項目索引鍵，以及一組 UpdateExpressions，其會定義要使用您在 ExpressionAttributeValues 參數中將值指派至其中的符記所更新的項目屬性。呼叫 DynamoDB 文件用戶端的update方法。

```
// 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" });

// Create variables to hold numeric key values
var season = SEASON_NUMBER;
var episode = EPISODES_NUMBER;

var params = {
  TableName: "EPISODES_TABLE",
  Key: {
    Season: season,
```

```
    Episode: episode,
  },
  UpdateExpression: "set Title = :t, Subtitle = :s",
  ExpressionAttributeValues: {
    ":t": "NEW_TITLE",
    ":s": "NEW_SUBTITLE",
  },
};

docClient.update(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

若要執行範例，請在命令列中輸入以下內容。

```
node ddbdoc_update.js
```

您可以在這裡找到此範例程式碼 GitHub。

查詢資料表

此範例會查詢內含影片系列之相關集資訊的表格，會為超過 9 集且子標題中含有指定片語的第二季集傳回集標題和子標題。

以檔名 ddbdoc_query.js 建立一個 Node.js 模組。請務必依前述的內容來設定軟體開發套件。若要存取 DynamoDB，請建立一個物件AWS.DynamoDB.DocumentClient。建立 JSON 物件，其包含查詢資料表所需的參數，在此範例中包括資料表名稱、查詢所需的 ExpressionAttributeValues，以及使用這些值來定義查詢要傳回之項目的 KeyConditionExpression。呼叫 DynamoDB 文件用戶端的query方法。

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

若要執行範例，請在命令列中輸入以下內容。

```
node ddbdoc_query.js
```

您可以在這裡找到此範例程式碼 GitHub。

從資料表中刪除項目

以檔名 ddbdoc_delete.js 建立一個 Node.js 模組。請務必依前述的內容來設定軟體開發套件。若要存取 DynamoDB，請建立一個物件AWS.DynamoDB.DocumentClient。建立 JSON 物件，其包含要在資料表中刪除項目所需的參數，在此範例中包括資料表名稱，以及您要刪除之項目的雜湊索引鍵名稱和值。呼叫 DynamoDB 文件用戶端的delete方法。

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

若要執行範例，請在命令列中輸入以下內容。

```
node ddbdoc_delete.js
```

您可以在這裡找到此範例程式碼 GitHub。

Amazon EC2 範例

亞馬遜彈性計算雲（Amazon EC2）是一種提供在雲中託管虛擬服務器的 Web 服務。該服務透過提供可重新調整大小的運算容量來降低開發人員進行 web 規模雲端運算的難度。

Amazon EC2 的 JavaScript API 是透過AWS.EC2用戶端類別公開的。如需使用 Amazon EC2 用戶端類別的詳細資訊，請參閱 API 參考資料Class: AWS.EC2中的。

主題

- 創建一個 Amazon EC2 實例

- 管理 Amazon EC2 執行個體
- 使用 Amazon EC2 金鑰對
- 搭配 Amazon EC2 使用區域和可用區域
- 在 Amazon EC2 中使用安全群組
- 在 Amazon EC2 中使用彈性 IP 地址

創建一個 Amazon EC2 實例

這個 Node.js 程式碼範例會說明：

- 如何從公共 Amazon 機器映像（AMI）創建亞馬遜 EC2 實例。
- 如何建立和指派標籤給新的 Amazon EC2 執行個體。

關於範例

在此範例中，您可以使用 Node.js 模組建立 Amazon EC2 執行個體，並為其指派 key pair 和標籤。程式碼會使用 SDK JavaScript 來建立執行個體，並使用 Amazon EC2 用戶端類別的下列方法來標記執行個體：

- `runInstances`
- `createTags`

先決條件任務

若要設定和執行此範例，請先完成這些任務。

- 安裝 Node.js。如需詳細資訊，請參閱 Node.js 網站。
- 透過使用者登入資料建立共用組態檔。如需提供共用登入資料檔案的詳細資訊，請參閱 從共用登入資料檔案中在 Node.js 中載入登入資料。
- 建立金鑰對。如需詳細資訊，請參閱 使用 Amazon EC2 金鑰對。您在此範例中會使用金鑰對名稱。

建立和標記執行個體

以檔名 ec2_createinstances.js 建立一個 Node.js 模組。請務必依前述的內容來設定軟體開發套件。

建立物件來傳遞 AWS.EC2 用戶端類別之 runInstances 方法的參數，包含要指派的金鑰對名稱和要執行的 AMI ID。若要呼叫 runInstances 方法，請建立叫用 Amazon EC2 服務物件的 promise 來傳遞參數。接著在 promise 回呼中處理回應。

接下來的程式碼會在新的執行個體中新增Name標籤，Amazon EC2 主控台可辨識該標籤並顯示在執行個體清單的 [名稱] 欄位中。您可以對執行個體新增高達 50 個標籤，所有標籤都可在單一呼叫中新增至 createTags。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Load credentials and set region from JSON file
AWS.config.update({ region: "REGION" });

// Create EC2 service object
var ec2 = new AWS.EC2({ apiVersion: "2016-11-15" });

// AMI is amzn-ami-2011.09.1.x86_64-ebs
var instanceParams = {
  ImageId: "AMI_ID",
  InstanceType: "t2.micro",
  KeyName: "KEY_PAIR_NAME",
  MinCount: 1,
  MaxCount: 1,
};

// Create a promise on an EC2 service object
var instancePromise = new AWS.EC2({ apiVersion: "2016-11-15" })
  .runInstances(instanceParams)
  .promise();

// Handle promise's fulfilled/rejected states
instancePromise
  .then(function (data) {
    console.log(data);
    var instanceId = data.Instances[0].InstanceId;
    console.log("Created instance", instanceId);
    // Add tags to the instance
    tagParams = {
```

```
      Resources: [instanceId],
      Tags: [
        {
          Key: "Name",
          Value: "SDK Sample",
        },
      ],
    };
    // Create a promise on an EC2 service object
    var tagPromise = new AWS.EC2({ apiVersion: "2016-11-15" })
      .createTags(tagParams)
      .promise();
    // Handle promise's fulfilled/rejected states
    tagPromise
      .then(function (data) {
        console.log("Instance tagged");
      })
      .catch(function (err) {
        console.error(err, err.stack);
      });
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

若要執行範例，請在命令列中輸入以下內容。

```
node ec2_createinstances.js
```

您可以在這裡找到此範例程式碼 GitHub。

管理 Amazon EC2 執行個體

這個 Node.js 程式碼範例會說明：

- 如何擷取 Amazon EC2 執行個體的基本資訊。
- 如何啟動和停止對 Amazon EC2 執行個體的詳細監控。

- 如何啟動和停止亞 Amazon EC2 實例。
- 如何重新啟動 Amazon EC2 實例。

使用案例

在此範例中，您使用一系列的 Node.js 模組來執行多項基本執行個體管理操作。Node.js 模組使用開發套件 JavaScript 來管理執行個體，方法是使用下列 Amazon EC2 用戶端類別方法：

- `describeInstances`
- `monitorInstances`
- `unmonitorInstances`
- `startInstances`
- `stopInstances`
- `rebootInstances`

如需 Amazon EC2 執行個體生命週期的詳細資訊，請參閱 Amazon EC2 使用者指南中的執行個體生命週期。

先決條件任務

若要設定和執行此範例，請先完成這些任務：

- 安裝 Node.js。如需安裝 Node.js 的詳細資訊，請參閱 Node.js 網站。
- 透過使用者登入資料建立共用組態檔。如需提供共用登入資料檔案的詳細資訊，請參閱 從共用登入資料檔案中在 Node.js 中載入登入資料。
- 創建一個 Amazon EC2 實例。如需建立 Amazon EC2 執行個體的詳細資訊，請參閱 Amazon EC2 使用者指南中的 Amazon EC2 執行個體或 Amazon EC2 使用者指南中的 Amazon EC2 執行個體。

描述您的執行個體

以檔名 `ec2_describeinstances.js` 建立一個 Node.js 模組。請務必依前述的內容來設定軟體開發套件。要訪問 Amazon EC2，請創建一個`AWS.EC2`服務對象。呼叫 Amazon EC2 服務物件的`describeInstances`方法，以擷取執行個體的詳細說明。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
```

```
// Set the region
AWS.config.update({ region: "REGION" });

// Create EC2 service object
var ec2 = new AWS.EC2({ apiVersion: "2016-11-15" });

var params = {
  DryRun: false,
};

// Call EC2 to retrieve policy for selected bucket
ec2.describeInstances(params, function (err, data) {
  if (err) {
    console.log("Error", err.stack);
  } else {
    console.log("Success", JSON.stringify(data));
  }
});
```

若要執行範例，請在命令列中輸入以下內容。

```
node ec2_describeinstances.js
```

您可以在這裡找到此範例程式碼 GitHub。

管理執行個體監控

以檔名 `ec2_monitorinstances.js` 建立一個 Node.js 模組。請務必依前述的內容來設定軟體開發套件。要訪問 Amazon EC2，請創建一個`AWS.EC2`服務對象。新增您要控制監控的執行個體的執行個體 ID。

根據命令列引數 (`ON`或`OFF`) 的值，呼叫 Amazon EC2 服務物件的`monitorInstances`方法以開始對指定執行個體的詳細監控，或呼叫方`unmonitorInstances`法。在嘗試變更這些執行個體的監控前，使用 `DryRun` 參數來測試您是否有變更執行個體監控的許可。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create EC2 service object
var ec2 = new AWS.EC2({ apiVersion: "2016-11-15" });
```

```
var params = {
  InstanceIds: ["INSTANCE_ID"],
  DryRun: true,
};

if (process.argv[2].toUpperCase() === "ON") {
  // Call EC2 to start monitoring the selected instances
  ec2.monitorInstances(params, function (err, data) {
    if (err && err.code === "DryRunOperation") {
      params.DryRun = false;
      ec2.monitorInstances(params, function (err, data) {
        if (err) {
          console.log("Error", err);
        } else if (data) {
          console.log("Success", data.InstanceMonitorings);
        }
      });
    } else {
      console.log("You don't have permission to change instance monitoring.");
    }
  });
} else if (process.argv[2].toUpperCase() === "OFF") {
  // Call EC2 to stop monitoring the selected instances
  ec2.unmonitorInstances(params, function (err, data) {
    if (err && err.code === "DryRunOperation") {
      params.DryRun = false;
      ec2.unmonitorInstances(params, function (err, data) {
        if (err) {
          console.log("Error", err);
        } else if (data) {
          console.log("Success", data.InstanceMonitorings);
        }
      });
    } else {
      console.log("You don't have permission to change instance monitoring.");
    }
  });
}
```

若要執行範例，請在命令列中輸入以下內容，指定 ON 來開始詳細監控或 OFF 來停止監控。

```
node ec2_monitorinstances.js ON
```

您可以在這裡找到此範例程式碼 GitHub。

啟動和停用執行個體

以檔名 ec2_startstopinstances.js 建立一個 Node.js 模組。請務必依前述的內容來設定軟體開發套件。要訪問 Amazon EC2，請創建一個AWS.EC2服務對象。新增您要啟動或停止之執行個體的執行個體 ID。

根據命令列引數 (START或STOP) 的值，呼叫 Amazon EC2 服務物件的startInstances方法來啟動指定的執行個體，或呼叫停止執行個體的stopInstances方法。在實際嘗試啟動或停止所選執行個體前，使用 DryRun 參數來測試您是否有許可。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create EC2 service object
var ec2 = new AWS.EC2({ apiVersion: "2016-11-15" });

var params = {
  InstanceIds: [process.argv[3]],
  DryRun: true,
};

if (process.argv[2].toUpperCase() === "START") {
  // Call EC2 to start the selected instances
  ec2.startInstances(params, function (err, data) {
    if (err && err.code === "DryRunOperation") {
      params.DryRun = false;
      ec2.startInstances(params, function (err, data) {
        if (err) {
          console.log("Error", err);
        } else if (data) {
          console.log("Success", data.StartingInstances);
        }
      });
    } else {
      console.log("You don't have permission to start instances.");
    }
  });
} else if (process.argv[2].toUpperCase() === "STOP") {
  // Call EC2 to stop the selected instances
```

```
  ec2.stopInstances(params, function (err, data) {
    if (err && err.code === "DryRunOperation") {
      params.DryRun = false;
      ec2.stopInstances(params, function (err, data) {
        if (err) {
          console.log("Error", err);
        } else if (data) {
          console.log("Success", data.StoppingInstances);
        }
      });
    } else {
      console.log("You don't have permission to stop instances");
    }
  });
}
```

若要執行範例，請在命令列中輸入以下內容，指定 START 來開始執行個體，或 STOP 來加以停止。

```
node ec2_startstopinstances.js START INSTANCE_ID
```

您可以在這裡找到此範例程式碼 GitHub。

重新啟動執行個體

以檔名 ec2_rebootinstances.js 建立一個 Node.js 模組。請務必依前述的內容來設定軟體開發套件。要訪問 Amazon EC2，請創建一個 Amazon EC2 服務對象。新增您要重新啟動之執行個體的執行個體 ID。呼叫 AWS.EC2 服務物件的 rebootInstances 方法來重新啟動指定的執行個體。在實際嘗試重新啟動執行個體前，使用 DryRun 參數來測試您是否有重新啟動這些執行個體的許可。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create EC2 service object
var ec2 = new AWS.EC2({ apiVersion: "2016-11-15" });

var params = {
  InstanceIds: ["INSTANCE_ID"],
  DryRun: true,
};
```

```
// Call EC2 to reboot instances
ec2.rebootInstances(params, function (err, data) {
  if (err && err.code === "DryRunOperation") {
    params.DryRun = false;
    ec2.rebootInstances(params, function (err, data) {
      if (err) {
        console.log("Error", err);
      } else if (data) {
        console.log("Success", data);
      }
    });
  } else {
    console.log("You don't have permission to reboot instances.");
  }
});
```

若要執行範例，請在命令列中輸入以下內容。

```
node ec2_rebootinstances.js
```

您可以在這裡找到此範例程式碼 GitHub。

使用 Amazon EC2 金鑰對

這個 Node.js 程式碼範例會說明：

- 如何擷取金鑰對的相關資訊。
- 如何建立 key pair 以存取 Amazon EC2 執行個體。
- 如何刪除現有的金鑰對。

使用案例

Amazon EC2 使用公有金鑰加密法將登入資訊進行加密及解密。公開金鑰加密法會使用公開金鑰加密資料，隨後再由收件人透過私密金鑰來解密該資料。公有金鑰和私有金鑰稱為金鑰對。

在此範例中，您使用一系列 Node.js 模組來執行多個 Amazon EC2 key pair 管理作業。Node.js 模組使用開發套件 JavaScript 來管理執行個體，方法是使用 Amazon EC2 用戶端類別的下列方法：

- createKeyPair
- deleteKeyPair
- describeKeyPairs

如需有關 Amazon EC2 金鑰配對的詳細資訊，請參閱 Amazon EC2 使用者指南中的 Amazon EC2 金鑰配對或 Amazon EC2 使用者指南中的 Amazon EC2 金鑰配對和視窗執行個體。

先決條件任務

若要設定和執行此範例，請先完成這些任務：

- 安裝 Node.js。如需安裝 Node.js 的詳細資訊，請參閱 Node.js 網站。
- 透過使用者登入資料建立共用組態檔。如需提供共用登入資料檔案的詳細資訊，請參閱 從共用登入資料檔案中在 Node.js 中載入登入資料。

描述您的金鑰對

以檔名 ec2_describekeypairs.js 建立一個 Node.js 模組。請務必依前述的內容來設定軟體開發套件。要訪問 Amazon EC2，請創建一個AWS.EC2服務對象。建立空白 JSON 物件，來保留 describeKeyPairs 方法傳回所有金鑰對描述所需的參數。您也可以將 JSON 檔案中參數的 KeyName 部分中的金鑰對名稱陣列，提供給 describeKeyPairs 方法。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create EC2 service object
var ec2 = new AWS.EC2({ apiVersion: "2016-11-15" });

// Retrieve key pair descriptions; no params needed
ec2.describeKeyPairs(function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", JSON.stringify(data.KeyPairs));
```

```
  }
});
```

若要執行範例，請在命令列中輸入以下內容。

```
node ec2_describekeypairs.js
```

您可以在這裡找到此範例程式碼 GitHub。

建立金鑰對

每個金鑰對都需要名稱。Amazon EC2 會將公有金鑰與您指定為金鑰名稱的名稱相關聯。以檔名 `ec2_createkeypair.js` 建立一個 Node.js 模組。請務必依前述的內容來設定軟體開發套件。要訪問 Amazon EC2，請創建一個`AWS.EC2`服務對象。建立 JSON 參數來指定金鑰對名稱，接著傳遞這些名稱來呼叫 `createKeyPair` 方法。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create EC2 service object
var ec2 = new AWS.EC2({ apiVersion: "2016-11-15" });

var params = {
  KeyName: "KEY_PAIR_NAME",
};

// Create the key pair
ec2.createKeyPair(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log(JSON.stringify(data));
  }
});
```

若要執行範例，請在命令列中輸入以下內容。

```
node ec2_createkeypair.js
```

您可以在這裡找到此範例程式碼 GitHub。

刪除金鑰對

以檔名 ec2_deletekeypair.js 建立一個 Node.js 模組。請務必依前述的內容來設定軟體開發套件。要訪問 Amazon EC2，請創建一個AWS.EC2服務對象。建立 JSON 參數來指定您要刪除的金鑰對名稱。然後呼叫 deleteKeyPair 方法。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create EC2 service object
var ec2 = new AWS.EC2({ apiVersion: "2016-11-15" });

var params = {
  KeyName: "KEY_PAIR_NAME",
};

// Delete the key pair
ec2.deleteKeyPair(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Key Pair Deleted");
  }
});
```

若要執行範例，請在命令列中輸入以下內容。

```
node ec2_deletekeypair.js
```

您可以在這裡找到此範例程式碼 GitHub。

搭配 Amazon EC2 使用區域和可用區域

這個 Node.js 程式碼範例會說明：

- 如何擷取區域與可用區域的描述。

使用案例

Amazon EC2 託管在全球多個地點。這些地點是由 區域及可用區域組成。各個 區域為獨立的地理區域。每個區域擁有多個隔離位置，稱為可用區域。Amazon EC2 提供將執行個體和資料放置在多個位置的功能。

在此範例中，您使用一系列的 Node.js 模組來擷取區域和可用區域的相關詳細資訊。Node.js 模組使用開發套件 JavaScript 來管理執行個體，方法是使用下列 Amazon EC2 用戶端類別的方法：

- `describeAvailabilityZones`
- `describeRegions`

如需有關區域和可用區域的詳細資訊，請參閱 Amazon EC2 使用者指南中的區域和可用區域或 Amazon EC2 使用者指南中的區域和可用區域。

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：

- 安裝 Node.js。如需安裝 Node.js 的詳細資訊，請參閱 Node.js 網站。
- 透過使用者登入資料建立共用組態檔。如需提供共用登入資料檔案的詳細資訊，請參閱 從共用登入資料檔案中在 Node.js 中載入登入資料。

說明區域與可用區域

以檔名 `ec2_describeregionsandzones.js` 建立一個 Node.js 模組。請務必依前述的內容來設定軟體開發套件。要訪問 Amazon EC2，請創建一個`AWS.EC2`服務對象。建立要以參數的形式傳遞的空白 JSON 物件，該物件會傳回所有可用描述。然後呼叫 `describeRegions` 和 `describeAvailabilityZones` 方法。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });
```

```
// Create EC2 service object
var ec2 = new AWS.EC2({ apiVersion: "2016-11-15" });

var params = {};

// Retrieves all regions/endpoints that work with EC2
ec2.describeRegions(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Regions: ", data.Regions);
  }
});

// Retrieves availability zones only for region of the ec2 service object
ec2.describeAvailabilityZones(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Availability Zones: ", data.AvailabilityZones);
  }
});
```

若要執行範例，請在命令列中輸入以下內容。

```
node ec2_describeregionsandzones.js
```

您可以在這裡找到此範例程式碼 GitHub。

在 Amazon EC2 中使用安全群組

這個 Node.js 程式碼範例會說明：

- 如何擷取安全群組的相關資訊。
- 如何建立安全群組以存取 Amazon EC2 執行個體。
- 如何刪除現有的安全群組。

使用案例

Amazon EC2 安全群組充當虛擬防火牆，可控制一個或多個執行個體的流量。您在各個安全群組新增規則，允許流量往返於建立關聯的執行個體。您可以隨時修改安全群組的規則；系統會自動將新規則套用至與安全群組相關聯的所有執行個體。

在此範例中，您使用一系列 Node.js 模組來執行多個涉及安全群組的 Amazon EC2 作業。Node.js 模組使用開發套件 JavaScript 來管理執行個體，方法是使用下列 Amazon EC2 用戶端類別的方法：

- `describeSecurityGroups`
- `authorizeSecurityGroupIngress`
- `createSecurityGroup`
- `describeVpcs`
- `deleteSecurityGroup`

如需有關 Amazon EC2 安全群組的詳細資訊，請參閱 Amazon EC2 使用者指南中的適用於 Linux 執行個體的 Amazon EC2 安全群組，或 Amazon EC2 使用者指南中的適用於 Windows 執行個體的 Amazon EC2 安全群組。

先決條件任務

若要設定和執行此範例，請先完成這些任務：

- 安裝 Node.js。如需安裝 Node.js 的詳細資訊，請參閱 Node.js 網站。
- 透過使用者登入資料建立共用組態檔。如需提供共用登入資料檔案的詳細資訊，請參閱 從共用登入資料檔案中在 Node.js 中載入登入資料。

說明您的安全群組

以檔名 `ec2_describesecuritygroups.js` 建立一個 Node.js 模組。請務必依前述的內容來設定軟體開發套件。要訪問 Amazon EC2，請創建一個`AWS.EC2`服務對象。建立要以參數的形式傳遞的 JSON 物件，包含您要描述的安全群組之群組 ID。然後調`describeSecurityGroups`用 Amazon EC2 服務對象的方法。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });
```

```
// Create EC2 service object
var ec2 = new AWS.EC2({ apiVersion: "2016-11-15" });

var params = {
  GroupIds: ["SECURITY_GROUP_ID"],
};

// Retrieve security group descriptions
ec2.describeSecurityGroups(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", JSON.stringify(data.SecurityGroups));
  }
});
```

若要執行範例，請在命令列中輸入以下內容。

```
node ec2_describesecuritygroups.js
```

您可以在這裡找到此範例程式碼 GitHub。

建立安全群組與規則

以檔名 `ec2_createsecuritygroup.js` 建立一個 Node.js 模組。請務必依前述的內容來設定軟體開發套件。要訪問 Amazon EC2，請創建一個`AWS.EC2`服務對象。為指定安全群組名稱、描述和 VPC 的 ID 的參數建立 JSON 物件。將參數傳遞至 `createSecurityGroup` 方法。

在順利建立安全群組後，您可以定義規則來允許傳入流量。為參數建立 JSON 物件，以指定 Amazon EC2 執行個體將在其上接收流量的 IP 通訊協定和輸入連接埠。將參數傳遞至 `authorizeSecurityGroupIngress` 方法。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Load credentials and set region from JSON file
AWS.config.update({ region: "REGION" });

// Create EC2 service object
var ec2 = new AWS.EC2({ apiVersion: "2016-11-15" });

// Variable to hold a ID of a VPC
```

```
var vpc = null;

// Retrieve the ID of a VPC
ec2.describeVpcs(function (err, data) {
  if (err) {
    console.log("Cannot retrieve a VPC", err);
  } else {
    vpc = data.Vpcs[0].VpcId;
    var paramsSecurityGroup = {
      Description: "DESCRIPTION",
      GroupName: "SECURITY_GROUP_NAME",
      VpcId: vpc,
    };
    // Create the instance
    ec2.createSecurityGroup(paramsSecurityGroup, function (err, data) {
      if (err) {
        console.log("Error", err);
      } else {
        var SecurityGroupId = data.GroupId;
        console.log("Success", SecurityGroupId);
        var paramsIngress = {
          GroupId: "SECURITY_GROUP_ID",
          IpPermissions: [
            {
              IpProtocol: "tcp",
              FromPort: 80,
              ToPort: 80,
              IpRanges: [{ CidrIp: "0.0.0.0/0" }],
            },
            {
              IpProtocol: "tcp",
              FromPort: 22,
              ToPort: 22,
              IpRanges: [{ CidrIp: "0.0.0.0/0" }],
            },
          ],
        };
        ec2.authorizeSecurityGroupIngress(paramsIngress, function (err, data) {
          if (err) {
            console.log("Error", err);
          } else {
            console.log("Ingress Successfully Set", data);
          }
        });
```

```
      }
    });
  }
});
```

若要執行範例，請在命令列中輸入以下內容。

```
node ec2_createsecuritygroup.js
```

您可以在這裡找到此範例程式碼 GitHub。

刪除安全群組

以檔名 ec2_deletesecuritygroup.js 建立一個 Node.js 模組。請務必依前述的內容來設定軟體開發套件。要訪問 Amazon EC2，請創建一個AWS.EC2服務對象。建立 JSON 參數來指定要刪除的安全群組名稱。然後呼叫 deleteSecurityGroup 方法。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create EC2 service object
var ec2 = new AWS.EC2({ apiVersion: "2016-11-15" });

var params = {
  GroupId: "SECURITY_GROUP_ID",
};

// Delete the security group
ec2.deleteSecurityGroup(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Security Group Deleted");
  }
});
```

若要執行範例，請在命令列中輸入以下內容。

```
node ec2_deletesecuritygroup.js
```

您可以在這裡找到此範例程式碼 GitHub。

在 Amazon EC2 中使用彈性 IP 地址

這個 Node.js 程式碼範例會說明：

- 如何擷取彈性 IP 地址的說明。
- 如何配置並發佈彈性 IP 地址。
- 如何將彈性 IP 地址與 Amazon EC2 執行個體建立關聯。

使用案例

彈性 IP 地址是針對動態雲端運算設計的靜態 IP 地址。彈性 IP 地址與您的 AWS 帳戶相關聯。這是一個可從網際網路存取的公有 IP 地址。如果您的執行個體沒有公有 IP 地址，則可將彈性 IP 地址與執行個體建立關聯，進而啟用與網際網路通訊的功能。

在此範例中，您可以使用一系列 Node.js 模組來執行數個涉及彈性 IP 地址的 Amazon EC2 作業。Node.js 模組使用開發套件 JavaScript 來管理彈性 IP 地址，方法是使用 Amazon EC2 用戶端類別的下列方法：

- `describeAddresses`
- `allocateAddress`
- `associateAddress`
- `releaseAddress`

如需 Amazon EC2 中彈性 IP 地址的詳細資訊，請參閱 Amazon EC2 使用者指南中的彈性 IP 地址或 Amazon EC2 使用者指南中的彈性 IP 地址。

先決條件任務

若要設定和執行此範例，請先完成這些任務：

- 安裝 Node.js。如需安裝 Node.js 的詳細資訊，請參閱 Node.js 網站。

- 透過使用者登入資料建立共用組態檔。如需提供共用登入資料檔案的詳細資訊，請參閱 從共用登入資料檔案中在 Node.js 中載入登入資料。
- 創建一個 Amazon EC2 實例。如需建立 Amazon EC2 執行個體的詳細資訊，請參閱 Amazon EC2 使用者指南中的 Amazon EC2 執行個體或 Amazon EC2 使用者指南中的 Amazon EC2 執行個體。

描述彈性 IP 地址

以檔名 `ec2_describeaddresses.js` 建立一個 Node.js 模組。請務必依前述的內容來設定軟體開發套件。要訪問 Amazon EC2，請創建一個`AWS.EC2`服務對象。建立要以參數形式傳遞的 JSON 物件，篩選您的 VPC 中所傳回的地址。若要擷取所有彈性 IP 地址的描述，可省略參數 JSON 的篩選條件。然後調`describeAddresses`用 Amazon EC2 服務對象的方法。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create EC2 service object
var ec2 = new AWS.EC2({ apiVersion: "2016-11-15" });

var params = {
  Filters: [{ Name: "domain", Values: ["vpc"] }],
};

// Retrieve Elastic IP address descriptions
ec2.describeAddresses(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", JSON.stringify(data.Addresses));
  }
});
```

若要執行範例，請在命令列中輸入以下內容。

```
node ec2_describeaddresses.js
```

您可以在這裡找到此範例程式碼 GitHub。

配置彈性 IP 地址並將其與 Amazon EC2 執行個體建立關聯

以檔名 ec2_allocateaddress.js 建立一個 Node.js 模組。請務必依前述的內容來設定軟體開發套件。要訪問 Amazon EC2，請創建一個AWS.EC2服務對象。針對用於配置彈性 IP 地址的參數建立 JSON 物件，其在這種情況下會指定 Domain 為 VPC。調allocateAddress用 Amazon EC2 服務對象的方法。

如果呼叫成功，傳給回呼函數的 data 參數便會包含 AllocationId 屬性，可識別配置的彈性 IP 地址。

為用於將彈性 IP 地址與 Amazon EC2 執行個體建立關聯的參數建立 JSON 物件，包括AllocationId來自新分配的地址和 Amazon EC2 執行個體InstanceId的地址。然後調associateAddresses用 Amazon EC2 服務對象的方法。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create EC2 service object
var ec2 = new AWS.EC2({ apiVersion: "2016-11-15" });

var paramsAllocateAddress = {
  Domain: "vpc",
};

// Allocate the Elastic IP address
ec2.allocateAddress(paramsAllocateAddress, function (err, data) {
  if (err) {
    console.log("Address Not Allocated", err);
  } else {
    console.log("Address allocated:", data.AllocationId);
    var paramsAssociateAddress = {
      AllocationId: data.AllocationId,
      InstanceId: "INSTANCE_ID",
    };
    // Associate the new Elastic IP address with an EC2 instance
    ec2.associateAddress(paramsAssociateAddress, function (err, data) {
      if (err) {
        console.log("Address Not Associated", err);
      } else {
        console.log("Address associated:", data.AssociationId);
      }
```

```
    });
  }
});
```

若要執行範例，請在命令列中輸入以下內容。

```
node ec2_allocateaddress.js
```

您可以在這裡找到此範例程式碼 GitHub。

釋放彈性 IP 地址

以檔名 ec2_releaseaddress.js 建立一個 Node.js 模組。請務必依前述的內容來設定軟體開發套件。要訪問 Amazon EC2，請創建一個AWS.EC2服務對象。為發佈彈性 IP 地址所用的參數建立 JSON 物件，其在這種情況下將 AllocationId 指定為彈性 IP 地址。釋放彈性 IP 地址也會將其與任何 Amazon EC2 執行個體斷開關聯。調releaseAddress用 Amazon EC2 服務對象的方法。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create EC2 service object
var ec2 = new AWS.EC2({ apiVersion: "2016-11-15" });

var paramsReleaseAddress = {
  AllocationId: "ALLOCATION_ID",
};

// Disassociate the Elastic IP address from EC2 instance
ec2.releaseAddress(paramsReleaseAddress, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Address released");
  }
});
```

若要執行範例，請在命令列中輸入以下內容。

```
node ec2_releaseaddress.js
```

您可以在這裡找到此範例程式碼 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AWS.MediaConvert用戶端類別公開。 MediaConvert 如需詳細資訊，請參閱 API 參考資料Class: AWS.MediaConvert中的。

主題

- 獲取您的區域特定端點 MediaConvert
- 建立和管理轉碼工作 MediaConvert
- 使用 Job 範本 MediaConvert

獲取您的區域特定端點 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

使用案例

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，方法是使用用 MediaConvert 戶端類別的這個方法：

- `describeEndpoints`

> **Important**
>
> 預設 Node.js HTTP/HTTPS 代理程式會為每個新的請求建立新的 TCP 連線。為了避免建立新連線的成本，AWS SDK for JavaScript會重複使用 TCP 連線。如需詳細資訊，請參閱在 Node.js 中重複使用 Keep-Alive 的連線。

先決條件任務

若要設定和執行此範例，請先完成這些任務：

- 安裝 Node.js。如需詳細資訊，請參閱 Node.js 網站。
- 透過使用者登入資料建立共用組態檔。如需提供共用登入資料檔案的詳細資訊，請參閱 從共用登入資料檔案中在 Node.js 中載入登入資料。
- 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。

取得端點 URL

以檔名 `emc_getendpoint.js` 建立一個 Node.js 模組。請務必依前述的內容來設定軟體開發套件。

創建一個對象，為`AWS.MediaConvert`客戶端類的`describeEndpoints`方法傳遞空的請求參數。若要呼叫 `describeEndpoints` 方法，請建立叫用 MediaConvert 服務物件的 promise 來傳遞參數。在 promise 回呼中處理回應。

```
// Load the SDK for JavaScript.
const aws = require("aws-sdk");

// Set the AWS Region.
aws.config.update({ region: "us-west-2" });

// Create the client.
const mediaConvert = new aws.MediaConvert({ apiVersion: "2017-08-29" });

exports.handler = async (event, context) => {
  // Create empty request parameters
  const params = {
    MaxResults: 0,
  };
```

```
  try {
    const { Endpoints } = await mediaConvert
      .describeEndpoints(params)
      .promise();
    console.log("Your MediaConvert endpoint is ", Endpoints);
  } catch (err) {
    console.log("MediaConvert Error", err);
  }
};
```

若要執行範例，請在命令列中輸入以下內容。

```
node emc_getendpoint.js
```

您可以在這裡找到此範例程式碼 GitHub。

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。

使用案例

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：

- `createJob`

- `cancelJob`
- `getJob`
- `listJobs`

先決條件任務

若要設定和執行此範例，請先完成這些任務：

- 安裝 Node.js。如需詳細資訊，請參閱 Node.js 網站。
- 透過使用者登入資料建立共用組態檔。如需提供共用登入資料檔案的詳細資訊，請參閱 從共用登入資料檔案中在 Node.js 中載入登入資料。
- 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」
- 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使用指南中的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。

設定軟體開發套件

JavaScript 透過建立全域設定物件，然後為程式碼設定 [區域]，以設定 SDK。在此範例中，區域會設為 `us-west-2`。由於每個帳戶都 MediaConvert 使用自訂端點，因此您還必須將用戶`AWS.MediaConvert`端類別設定為使用區域特定端點。若要這麼做，請在 `AWS.config.mediaconvert` 上設定 `endpoint` 參數。

```
// Load the SDK for JavaScript
var AWS = require("aws-sdk");
// Set the Region
AWS.config.update({ region: "us-west-2" });
// Set the custom endpoint for your account
AWS.config.mediaconvert = { endpoint: "ACCOUNT_ENDPOINT" };
```

定義簡單的轉碼任務

以檔名 `emc_createjob.js` 建立一個 Node.js 模組。請務必依前述的內容來設定軟體開發套件。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。

```
var params = {
  Queue: "JOB_QUEUE_ARN",
  UserMetadata: {
    Customer: "Amazon",
  },
  Role: "IAM_ROLE_ARN",
  Settings: {
    OutputGroups: [
      {
        Name: "File Group",
        OutputGroupSettings: {
          Type: "FILE_GROUP_SETTINGS",
          FileGroupSettings: {
            Destination: "s3://OUTPUT_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: "s3://INPUT_BUCKET_AND_FILE_NAME",
    },
  ],
  TimecodeConfig: {
    Source: "EMBEDDED",
  },
```

```
  },
};
```

建立轉碼任務

建立任務參數 JSON 後，透過為叫用 AWS.MediaConvert 服務物件建立 promise 來傳遞參數以呼叫 createJob 方法。接著在 promise 回呼中處理回應。回應 data 中會傳回所建立任務的 ID。

```
// Create a promise on a MediaConvert object
var endpointPromise = new AWS.MediaConvert({ apiVersion: "2017-08-29" })
  .createJob(params)
  .promise();

// Handle promise's fulfilled/rejected status
endpointPromise.then(
  function (data) {
    console.log("Job created! ", data);
  },
  function (err) {
    console.log("Error", err);
  }
);
```

若要執行範例，請在命令列中輸入以下內容。

```
node emc_createjob.js
```

您可以在這裡找到此範例程式碼 GitHub。

取消轉碼任務

以檔名 emc_canceljob.js 建立一個 Node.js 模組。請務必依前述的內容來設定軟體開發套件。建立包含要取消任務 ID 的 JSON。接著，若要呼叫 cancelJob 方法，請建立叫用 AWS.MediaConvert 服務物件的 promise 來傳遞參數。在 promise 回呼中處理回應。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the Region
AWS.config.update({ region: "us-west-2" });
// Set MediaConvert to customer endpoint
```

```
AWS.config.mediaconvert = { endpoint: "ACCOUNT_ENDPOINT" };

var params = {
  Id: "JOB_ID" /* required */,
};

// Create a promise on a MediaConvert object
var endpointPromise = new AWS.MediaConvert({ apiVersion: "2017-08-29" })
  .cancelJob(params)
  .promise();

// Handle promise's fulfilled/rejected status
endpointPromise.then(
  function (data) {
    console.log("Job  " + params.Id + " is canceled");
  },
  function (err) {
    console.log("Error", err);
  }
);
```

若要執行範例，請在命令列中輸入以下內容。

```
node ec2_canceljob.js
```

您可以在這裡找到此範例程式碼 GitHub。

列出最近的轉碼任務

以檔名 `emc_listjobs.js` 建立一個 Node.js 模組。請務必依前述的內容來設定軟體開發套件。

建立參數 JSON，包括指定以 `ASCENDING` 或 `DESCENDING` 順序排序清單的值、要檢查的任務佇列 ARN，以及要包含的任務狀態。接著，若要呼叫 `listJobs` 方法，請建立叫用 `AWS.MediaConvert` 服務物件的 promise 來傳遞參數。在 promise 回呼中處理回應。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the Region
AWS.config.update({ region: "us-west-2" });
// Set the customer endpoint
AWS.config.mediaconvert = { endpoint: "ACCOUNT_ENDPOINT" };
```

```
var params = {
  MaxResults: 10,
  Order: "ASCENDING",
  Queue: "QUEUE_ARN",
  Status: "SUBMITTED",
};

// Create a promise on a MediaConvert object
var endpointPromise = new AWS.MediaConvert({ apiVersion: "2017-08-29" })
  .listJobs(params)
  .promise();

// Handle promise's fulfilled/rejected status
endpointPromise.then(
  function (data) {
    console.log("Jobs: ", data);
  },
  function (err) {
    console.log("Error", err);
  }
);
```

若要執行範例，請在命令列中輸入以下內容。

```
node emc_listjobs.js
```

您可以在這裡找到此範例程式碼 GitHub。

使用 Job 範本 MediaConvert

這個 Node.js 程式碼範例會說明：

- 如何創建 MediaConvert 工作模板。
- 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。

使用案例

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：

- `createJobTemplate`
- `createJob`
- `deleteJobTemplate`
- `listJobTemplates`

先決條件任務

若要設定和執行此範例，請先完成這些任務：

- 安裝 Node.js。如需詳細資訊，請參閱 Node.js 網站。
- 透過使用者登入資料建立共用組態檔。如需提供共用登入資料檔案的詳細資訊，請參閱 從共用登入資料檔案中在 Node.js 中載入登入資料。
- 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。

建立任務範本

以檔名 `emc_create_jobtemplate.js` 建立一個 Node.js 模組。請務必依前述的內容來設定軟體開發套件。

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 建立和管理轉碼工作 MediaConvert的任務設定。

建立呼叫 `AWS.MediaConvert` 服務物件的 promise 並傳遞參數，來呼叫 `createJobTemplate` 方法。接著在 promise 回呼中處理回應。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the Region
AWS.config.update({ region: "us-west-2" });
// Set the custom endpoint for your account
```

```
AWS.config.mediaconvert = { endpoint: "ACCOUNT_ENDPOINT" };

var params = {
  Category: "YouTube Jobs",
  Description: "Final production transcode",
  Name: "DemoTemplate",
  Queue: "JOB_QUEUE_ARN",
  Settings: {
    OutputGroups: [
      {
        Name: "File Group",
        OutputGroupSettings: {
          Type: "FILE_GROUP_SETTINGS",
          FileGroupSettings: {
            Destination: "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",
    },
  },
};

// Create a promise on a MediaConvert object
var templatePromise = new AWS.MediaConvert({ apiVersion: "2017-08-29" })
```

```
  .createJobTemplate(params)
  .promise();

// Handle promise's fulfilled/rejected status
templatePromise.then(
  function (data) {
    console.log("Success!", data);
  },
  function (err) {
    console.log("Error", err);
  }
);
```

若要執行範例，請在命令列中輸入以下內容。

```
node emc_create_jobtemplate.js
```

您可以在這裡找到此範例程式碼 GitHub。

透過任務範本來建立轉譯任務

以檔名 `emc_template_createjob.js` 建立一個 Node.js 模組。請務必依前述的內容來設定軟體開發套件。

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`。接著，若要呼叫 `createJobs` 方法，請建立叫用 `AWS.MediaConvert` 服務物件的 promise 來傳遞參數。在 promise 回呼中處理回應。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the Region
AWS.config.update({ region: "us-west-2" });
// Set the custom endpoint for your account
AWS.config.mediaconvert = { endpoint: "ACCOUNT_ENDPOINT" };

var params = {
  Queue: "QUEUE_ARN",
  JobTemplate: "TEMPLATE_NAME",
  Role: "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: "s3://BUCKET_NAME/FILE_NAME",
      },
    ],
  },
};

// Create a promise on a MediaConvert object
var templateJobPromise = new AWS.MediaConvert({ apiVersion: "2017-08-29" })
  .createJob(params)
  .promise();

// Handle promise's fulfilled/rejected status
templateJobPromise.then(
  function (data) {
    console.log("Success! ", data);
  },
  function (err) {
    console.log("Error", err);
  }
);
```

若要執行範例，請在命令列中輸入以下內容。

```
node emc_template_createjob.js
```

您可以在這裡找到此範例程式碼 GitHub。

列出任務範本

以檔名 emc_listtemplates.js 建立一個 Node.js 模組。請務必依前述的內容來設定軟體開發套件。

建立物件，傳遞 AWS.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listTemplates方法，請創建一個用於調用 MediaConvert 服務對象的承諾，並傳遞參數。接著在 promise 回呼中處理回應。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the Region
AWS.config.update({ region: "us-west-2" });
// Set the customer endpoint
AWS.config.mediaconvert = { endpoint: "ACCOUNT_ENDPOINT" };

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

// Create a promise on a MediaConvert object
var listTemplatesPromise = new AWS.MediaConvert({ apiVersion: "2017-08-29" })
  .listJobTemplates(params)
  .promise();

// Handle promise's fulfilled/rejected status
listTemplatesPromise.then(
  function (data) {
    console.log("Success ", data);
  },
  function (err) {
    console.log("Error", err);
  }
);
```

若要執行範例，請在命令列中輸入以下內容。

```
node emc_listtemplates.js
```

您可以在這裡找到此範例程式碼 GitHub。

刪除任務範本

以檔名 emc_deletetemplate.js 建立一個 Node.js 模組。請務必依前述的內容來設定軟體開發套件。

建立物件，為 AWS.MediaConvert 用戶端類別的 deleteJobTemplate 方法傳遞您要刪除做為參數的任務範本名稱。要調用該deleteJobTemplate方法，請創建一個用於調用 MediaConvert 服務對象的承諾，並傳遞參數。在 promise 回呼中處理回應。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the Region
AWS.config.update({ region: "us-west-2" });
// Set the customer endpoint
AWS.config.mediaconvert = { endpoint: "ACCOUNT_ENDPOINT" };

var params = {
  Name: "TEMPLATE_NAME",
};

// Create a promise on a MediaConvert object
var deleteTemplatePromise = new AWS.MediaConvert({ apiVersion: "2017-08-29" })
  .deleteJobTemplate(params)
  .promise();

// Handle promise's fulfilled/rejected status
deleteTemplatePromise.then(
  function (data) {
    console.log("Success ", data);
  },
  function (err) {
    console.log("Error", err);
  }
);
```

若要執行範例，請在命令列中輸入以下內容。

```
node emc_deletetemplate.js
```

您可以在這裡找到此範例程式碼 GitHub。

AWSIAM 範例

AWS Identity and Access Management(IAM) 是一種網路服務，可讓 Amazon Web Services 客戶管理中的使用者和使用者許可AWS。此服務的目標是針對在雲端中擁有多個使用者或系統使用AWS產品的組織。透過 IAM，您可以集中管理使用者、存取金鑰等安全登入資料，以及控制使用者可存取哪些AWS資源的權限。

IAM 的 JavaScript API 是透過AWS.IAM用戶端類別公開的。如需使用 IAM 用戶端類別的詳細資訊，請參閱 API 參考資料Class: AWS.IAM中的。

主題

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

管理 IAM 使用者

這個 Node.js 程式碼範例會說明：

- 如何擷取 IAM 使用者清單。
- 如何建立和刪除使用者。

- 如何更新使用者名稱。

使用案例

在此範例中，會使用一系列 Node.js 模組來建立和管理 IAM 中的使用者。Node.js 模組會使用 SDK JavaScript 來建立、刪除及更新使用者使用用AWS.IAM戶端類別的下列方法：

- createUser
- listUsers
- updateUser
- getUser
- deleteUser

如需 IAM 使用者的詳細資訊，請參閱 IAM 使用者指南中的 IAM 使用者。

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：

- 安裝 Node.js。如需安裝 Node.js 的詳細資訊，請參閱 Node.js 網站。
- 透過使用者登入資料建立共用組態檔。如需提供共用登入資料檔案的詳細資訊，請參閱 從共用登入資料檔案中在 Node.js 中載入登入資料。

建立使用者

以檔名 iam_createuser.js 建立一個 Node.js 模組。請務必依前述的內容來設定軟體開發套件。若要存取 IAM，請建立AWS.IAM服務物件。建立包含所需參數的 JSON 物件，其應由要用於新使用者的使用者名稱所組成，以做為命令列參數。

呼叫 AWS.IAM 服務物件的 getUser 方法，查看該使用者名稱是否已存在。如果使用者名稱目前不存在，則請呼叫 createUser 方法來建立該名稱。如果名稱已存在，將該效果的訊息寫入主控台。

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

若要執行範例，請在命令列中輸入以下內容。

```
node iam_createuser.js USER_NAME
```

您可以在這裡找到此範例程式碼 GitHub。

列出帳戶中的使用者

以檔名 `iam_listusers.js` 建立一個 Node.js 模組。請務必依前述的內容來設定軟體開發套件。若要存取 IAM，請建立`AWS.IAM`服務物件。建立包含列出使用者所需參數的 JSON 物件，然後將 `MaxItems` 參數設為 10 來限制傳回的數量。呼叫 `AWS.IAM` 服務物件的 `listUsers` 方法。接下來，將第一位使用者的名稱和建立日期寫入主控台。

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

若要執行範例，請在命令列中輸入以下內容。

```
node iam_listusers.js
```

您可以在這裡找到此範例程式碼 GitHub。

更新使用者名稱

以檔名 `iam_updateuser.js` 建立一個 Node.js 模組。請務必依前述的內容來設定軟體開發套件。若要存取 IAM，請建立`AWS.IAM`服務物件。建立包含列出使用者所需參數的 JSON 物件，然後將目前和新的使用者名稱指定為命令列參數。呼叫 `AWS.IAM` 服務物件的 `updateUser` 方法。

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

若要執行範例，請在命令列中輸入以下指令來指定使用者目前的名稱，並在後面加上新的使用者名稱。

```
node iam_updateuser.js ORIGINAL_USERNAME NEW_USERNAME
```

您可以在這裡找到此範例程式碼 GitHub。

刪除使用者

以檔名 iam_deleteuser.js 建立一個 Node.js 模組。請務必依前述的內容來設定軟體開發套件。若要存取 IAM，請建立AWS.IAM服務物件。建立包含所需參數的 JSON 物件，其應由要刪除的使用者名稱所組成，以做為命令列參數。

呼叫 AWS.IAM 服務物件的 getUser 方法，查看該使用者名稱是否已存在。如果使用者名稱目前不存在，請將會產生該效果的訊息寫入主控台。如果使用者已存在，則請呼叫 deleteUser 方法將其刪除。

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

若要執行範例，請在命令列中輸入以下內容。

```
node iam_deleteuser.js USER_NAME
```

您可以在這裡找到此範例程式碼 GitHub。

處理 IAM 政策

這個 Node.js 程式碼範例會說明：

- 如何建立和刪除 IAM 政策。
- 如何從角色附加和卸離 IAM 政策。

使用案例

您能夠建立政策，進而將許可授與給使用者。該政策文件會列出使用者可執行的動作，以及會受到這些動作影響的資源。根據預設，未明確允許的任何動作或資源將被拒絕。可建立政策並連接至使用者、使用者群組、使用者擔任的角色以及資源。

在此範例中，會使用一系列 Node.js 模組來管理 IAM 中的政策。Node.js 模組使用 SDK JavaScript 來建立和刪除原則，以及使用用AWS.IAM戶端類別的下列方法附加和卸離角色原則：

- createPolicy
- getPolicy
- listAttachedRolePolicies
- attachRolePolicy
- detachRolePolicy

如需 IAM 使用者的詳細資訊，請參閱 IAM 使用者指南中的存取管理概觀：許可和政策。

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：

- 安裝 Node.js。如需安裝 Node.js 的詳細資訊，請參閱 Node.js 網站。
- 透過使用者登入資料建立共用組態檔。如需提供共用登入資料檔案的詳細資訊，請參閱 從共用登入資料檔案中在 Node.js 中載入登入資料。
- 建立可附加政策的 IAM 角色。如需建立角色的詳細資訊，請參閱 IAM 使用者指南中的建立 IAM 角色。

建立 IAM 政策

以檔名 `iam_createpolicy.js` 建立一個 Node.js 模組。請務必依前述的內容來設定軟體開發套件。若要存取 IAM，請建立`AWS.IAM`服務物件。建立兩個 JSON 物件，一個包含要建立的政策文件，另一個則包含建立政策所需的參數，且其中應包括要指定給該政策的政策 JSON 和名稱。請務必將參數中的政策 JSON 物件字串化。呼叫 `AWS.IAM` 服務物件的 `createPolicy` 方法。

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

若要執行範例，請在命令列中輸入以下內容。

```
node iam_createpolicy.js
```

您可以在這裡找到此範例程式碼 GitHub。

取得 IAM 政策

以檔名 `iam_getpolicy.js` 建立一個 Node.js 模組。請務必依前述的內容來設定軟體開發套件。若要存取 IAM，請建立`AWS.IAM`服務物件。建立一個 JSON 物件，其中包含擷取政策所需的參數，意即要取得的政策 ARN。呼叫 `AWS.IAM` 服務物件的 `getPolicy` 方法。接下來，將政策說明寫入主控台。

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

若要執行範例，請在命令列中輸入以下內容。

```
node iam_getpolicy.js
```

您可以在這裡找到此範例程式碼 GitHub。

連接受管角色政策

以檔名 `iam_attachrolepolicy.js` 建立一個 Node.js 模組。請務必依前述的內容來設定軟體開發套件。若要存取 IAM，請建立`AWS.IAM`服務物件。建立 JSON 物件，其中包含取得附加至角色的受管 IAM 政策清單所需的參數，該清單由角色名稱組成。然後，提供角色名稱以做為命令列參數。呼叫 `AWS.IAM` 服務物件的 `listAttachedRolePolicies` 方法，其會將受管政策陣列回傳至回呼函數。

檢查陣列成員，確認要連接至角色的政策是否已連接。如果該政策尚未連接，則請呼叫 `attachRolePolicy` 方法來連接該政策。

```
// 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");
      }
    });
  }
});
```

若要執行範例，請在命令列中輸入以下內容。

```
node iam_attachrolepolicy.js IAM_ROLE_NAME
```

分離受管角色政策

以檔名 `iam_detachrolepolicy.js` 建立一個 Node.js 模組。請務必依前述的內容來設定軟體開發套件。若要存取 IAM，請建立`AWS.IAM`服務物件。建立 JSON 物件，其中包含取得附加至角色的受管 IAM 政策清單所需的參數，該清單由角色名稱組成。然後，提供角色名稱以做為命令列參數。呼叫 `AWS.IAM` 服務物件的 `listAttachedRolePolicies` 方法，其會在回呼函數中傳回受管政策陣列。

檢查陣列成員，確認要從角色分離的政策是否處於連接狀態。如果該政策處於連接狀態，則請呼叫 `detachRolePolicy` 方法來分離該政策。

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

若要執行範例，請在命令列中輸入以下內容。

```
node iam_detachrolepolicy.js IAM_ROLE_NAME
```

管理 IAM 存取金鑰

這個 Node.js 程式碼範例會說明：

- 如何管理使用者存取金鑰。

使用案例

使用者需要自己的存取金鑰，才能AWS從 SDK 進行程式設計呼叫。 JavaScript為了滿足這個需求，您可以為 IAM 使用者建立、修改、查看或輪換存取金鑰 (存取金鑰 ID 和私密存取金鑰)。在預設情況下，當您建立存取金鑰時，其狀態會是 `Active`，這表示使用者可以透過存取金鑰進行 API 呼叫。

在此範例中，IAM 中會使用一系列 Node.js 模組來管理存取金鑰。Node.js 模組使用的開發套件，使用用`AWS.IAM`戶端類別的下列方法 JavaScript 來管理 IAM 存取金鑰：

- createAccessKey
- listAccessKeys
- getAccessKeyLastUsed
- updateAccessKey
- deleteAccessKey

如需 IAM 存取金鑰的詳細資訊，請參閱 IAM 使用者指南中的存取金鑰。

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：

- 安裝 Node.js。如需安裝 Node.js 的詳細資訊，請參閱 Node.js 網站。
- 透過使用者登入資料建立共用組態檔。如需提供共用登入資料檔案的詳細資訊，請參閱 從共用登入資料檔案中在 Node.js 中載入登入資料。

建立使用者存取金鑰

以檔名 `iam_createaccesskeys.js` 建立一個 Node.js 模組。請務必依前述的內容來設定軟體開發套件。若要存取 IAM，請建立`AWS.IAM`服務物件。建立 JSON 物件，其中包含建立新存取金鑰所需的參數，其中包括 IAM 使用者的名稱。呼叫 `AWS.IAM` 服務物件的 `createAccessKey` 方法。

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

若要執行範例，請在命令列中輸入以下內容。請務必將傳回的檔案輸送至文字檔案，避免私密金鑰遺失；系統只會提供該金鑰一次。

```
node iam_createaccesskeys.js > newuserkeys.txt
```

您可以在這裡找到此範例程式碼 GitHub。

列出使用者存取金鑰

以檔名 `iam_listaccesskeys.js` 建立一個 Node.js 模組。請務必依前述的內容來設定軟體開發套件。若要存取 IAM，請建立`AWS.IAM`服務物件。建立 JSON 物件，其中包含擷取使用者存取金鑰所需的參數，其中包括 IAM 使用者的名稱，以及選擇性地列出您要列出的存取金鑰配對數目上限。呼叫 `AWS.IAM` 服務物件的 `listAccessKeys` 方法。

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

若要執行範例，請在命令列中輸入以下內容。

```
node iam_listaccesskeys.js
```

您可以在這裡找到此範例程式碼 GitHub。

取得上次使用存取金鑰的日期

以檔名 `iam_accesskeylastused.js` 建立一個 Node.js 模組。請務必依前述的內容來設定軟體開發套件。若要存取 IAM，請建立`AWS.IAM`服務物件。建立一個 JSON 物件，其中包含建立新存取金鑰所需的參數，意即要取得上次使用資訊的存取金鑰 ID。呼叫 `AWS.IAM` 服務物件的 `getAccessKeyLastUsed` 方法。

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

若要執行範例，請在命令列中輸入以下內容。

```
node iam_accesskeylastused.js
```

您可以在這裡找到此範例程式碼 GitHub。

更新存取金鑰狀態

以檔名 iam_updateaccesskey.js 建立一個 Node.js 模組。請務必依前述的內容來設定軟體開發套件。若要存取 IAM，請建立AWS.IAM服務物件。建立包含更新存取金鑰狀態所需參數的 JSON 物件，且其中應包括存取金鑰 ID 和更新的狀態；狀態可能是 Active 或 Inactive。呼叫 AWS.IAM 服務物件的 updateAccessKey 方法。

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

若要執行範例，請在命令列中輸入以下內容。

```
node iam_updateaccesskey.js
```

您可以在這裡找到此範例程式碼 GitHub。

刪除存取金鑰

以檔名 iam_deleteaccesskey.js 建立一個 Node.js 模組。請務必依前述的內容來設定軟體開發套件。若要存取 IAM，請建立AWS.IAM服務物件。建立包含刪除存取金鑰所需參數的 JSON 物件，且其中應包括存取金鑰 ID 和使用者名稱。呼叫 AWS.IAM 服務物件的 deleteAccessKey 方法。

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

若要執行範例，請在命令列中輸入以下內容。

```
node iam_deleteaccesskey.js
```

您可以在這裡找到此範例程式碼 GitHub。

處理 IAM 伺服器憑證

這個 Node.js 程式碼範例會說明：

- 如何執行管理 HTTPS 連線伺服器憑證的基本任務。

使用案例

若要在上啟用 HTTPS 連線到您的網站或應用程式AWS，您需要 SSL/TLS 伺服器憑證。若要將您從外部供應商取得的憑證與網站或應用程式搭配使用AWS，您必須將憑證上傳至 IAM，或將其匯入 Cer AWS tificate Manager。

在此範例中，一系列 Node.js 模組用來處理 IAM 中的伺服器憑證。Node.js 模組使用 SDK JavaScript 來管理使用用AWS.IAM戶端類別的下列方法的伺服器憑證：

- listServerCertificates
- getServerCertificate
- updateServerCertificate
- deleteServerCertificate

如需有關伺服器憑證的詳細資訊，請參閱 IAM 使用者指南中的使用伺服器憑證。

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：

- 安裝 Node.js。如需安裝 Node.js 的詳細資訊，請參閱 Node.js 網站。
- 透過使用者登入資料建立共用組態檔。如需提供共用登入資料檔案的詳細資訊，請參閱 從共用登入資料檔案中在 Node.js 中載入登入資料。

列出伺服器憑證

以檔名 iam_listservercerts.js 建立一個 Node.js 模組。請務必依前述的內容來設定軟體開發套件。若要存取 IAM，請建立AWS.IAM服務物件。呼叫 AWS.IAM 服務物件的 listServerCertificates 方法。

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

若要執行範例，請在命令列中輸入以下內容。

```
node iam_listservercerts.js
```

您可以在這裡找到此範例程式碼 GitHub。

取得伺服器憑證

以檔名 `iam_getservercert.js` 建立一個 Node.js 模組。請務必依前述的內容來設定軟體開發套件。若要存取 IAM，請建立`AWS.IAM`服務物件。建立包含取得憑證所需參數的 JSON 物件，其應由您要的伺服器憑證名稱所組成。呼叫 `AWS.IAM` 服務物件的 `getServerCertificates` 方法。

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

若要執行範例，請在命令列中輸入以下內容。

```
node iam_getservercert.js
```

您可以在這裡找到此範例程式碼 GitHub。

更新伺服器憑證

以檔名 iam_updateservercert.js 建立一個 Node.js 模組。請務必依前述的內容來設定軟體開發套件。若要存取 IAM，請建立AWS.IAM服務物件。建立包含更新憑證所需參數的 JSON 物件，其應由現有伺服器憑證名稱和新憑證名稱所組成。呼叫 AWS.IAM 服務物件的 updateServerCertificate 方法。

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

若要執行範例，請在命令列中輸入以下內容。

```
node iam_updateservercert.js
```

您可以在這裡找到此範例程式碼 GitHub。

刪除伺服器憑證

以檔名 iam_deleteservercert.js 建立一個 Node.js 模組。請務必依前述的內容來設定軟體開發套件。若要存取 IAM，請建立AWS.IAM服務物件。建立包含刪除伺服器憑證所需參數的 JSON 物件，其應由要刪除的憑證名稱所組成。呼叫 AWS.IAM 服務物件的 deleteServerCertificates 方法。

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

若要執行範例，請在命令列中輸入以下內容。

```
node iam_deleteservercert.js
```

您可以在這裡找到此範例程式碼 GitHub。

管理 IAM 帳戶別名

這個 Node.js 程式碼範例會說明：

- 如何管理AWS帳戶 ID 的別名。

使用案例

如果您希望登入頁面的 URL 包含您的公司名稱或其他易記識別碼，而非AWS帳戶 ID，您可以建立AWS帳戶 ID 的別名。如果您建立AWS帳戶別名，您的登入頁面 URL 會變更為合併別名。

在此範例中，會使用一系列 Node.js 模組來建立和管理 IAM 帳戶別名。Node.js 模組會使用 SDK JavaScript 來管理使用用`AWS.IAM`戶端類別的下列方法的別名：

- createAccountAlias
- listAccountAliases
- deleteAccountAlias

如需 IAM 帳戶別名的詳細資訊，請參閱 IAM 使用者指南中的您的AWS帳戶 ID 及其別名。

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：

- 安裝 Node.js。如需安裝 Node.js 的詳細資訊，請參閱 Node.js 網站。
- 透過使用者登入資料建立共用組態檔。如需提供共用登入資料檔案的詳細資訊，請參閱 從共用登入資料檔案中在 Node.js 中載入登入資料。

建立帳戶別名

以檔名 `iam_createaccountalias.js` 建立一個 Node.js 模組。請務必依前述的內容來設定軟體開發套件。若要存取 IAM，請建立`AWS.IAM`服務物件。建立包含所需參數的 JSON 物件以建立帳戶別名，且其中應包括要建立的別名。呼叫 `AWS.IAM` 服務物件的 `createAccountAlias` 方法。

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

若要執行範例，請在命令列中輸入以下內容。

```
node iam_createaccountalias.js ALIAS
```

您可以在這裡找到此範例程式碼 GitHub。

列出帳戶別名

以檔名 `iam_listaccountaliases.js` 建立一個 Node.js 模組。請務必依前述的內容來設定軟體開發套件。若要存取 IAM，請建立`AWS.IAM`服務物件。建立包含列出帳戶別名所需參數的 JSON 物件，且其中應包括要傳回的項目數量上限。呼叫 `AWS.IAM` 服務物件的 `listAccountAliases` 方法。

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

若要執行範例，請在命令列中輸入以下內容。

```
node iam_listaccountaliases.js
```

您可以在這裡找到此範例程式碼 GitHub。

刪除帳戶別名

以檔名 iam_deleteaccountalias.js 建立一個 Node.js 模組。請務必依前述的內容來設定軟體開發套件。若要存取 IAM，請建立AWS.IAM服務物件。建立包含刪除帳戶別名所需參數的 JSON 物件，且其中應包括要刪除的別名。呼叫 AWS.IAM 服務物件的 deleteAccountAlias 方法。

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

若要執行範例，請在命令列中輸入以下內容。

```
node iam_deleteaccountalias.js ALIAS
```

您可以在這裡找到此範例程式碼 GitHub。

Amazon Kinesis 示例

Amazon Kinesis 是串流資料的平台AWS，提供強大的服務來載入和分析串流資料，還可讓您針對特殊需求建置自訂串流資料應用程式。

Kinesis 的 JavaScript API 會透過`AWS.Kinesis`用戶端類別公開。如需有關使用 Kinesis 用戶端類別的詳細資訊，請參閱 API 參考資料`Class: AWS.Kinesis`中的。

主題

- 使用 Amazon Kinesis 擷取網頁捲動進度

使用 Amazon Kinesis 擷取網頁捲動進度

此瀏覽器指令碼範例顯示：

- 如何使用 Amazon Kinesis 擷取網頁中的捲動進度，做為串流頁面使用量指標的範例，以供日後分析。

使用案例

在此範例中，簡單 HTML 頁面會模擬部落格頁面的內容。當讀者捲動模擬的部落格文章時，瀏覽器指令碼會使用 SDK JavaScript 來記錄頁面向下捲動距離，並使用 Kinesis 用戶端類別的`putRecords`方法將該資料傳送至 Kinesis。然後，Amazon Kinesis 資料串流擷取的串流資料就可以由 Amazon EC2 執行個體處理，並存放在多個資料存放區中的任何一個，包括 Amazon DynamoDB 和 Amazon Redshift。

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：

- 建立 Kinesis 串流。您必須在瀏覽器指令碼中包含串流資源的 ARN。如需建立 Amazon Kinesis 資料串流的詳細資訊，請參閱 Amazon Kinesis 資料串流開發人員指南中的管理 Kinesis 串流。
- 建立具有針對未驗證身分啟用存取權的 Amazon Cognito 身分集區。您必須在程式碼中包含身分集區 ID，來取得瀏覽器指令碼的登入資料。如需 Amazon Cognito 身分識別集區的詳細資訊，請參閱 Amazon Cognito 開發人員指南中的身分集區。
- 建立 IAM 角色，其政策授與將資料提交至 Kinesis 串流的權限。如需有關建立 IAM 角色的詳細資訊，請參閱 IAM 使用者指南中的建立角色以將許可委派給AWS服務。

您可以使用下列角色政策來建立 IAM 角色。

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

部落格頁面

部落格頁面的 HTML 主要組成為一系列的段落，這些段落包含在 元素中。此 可捲動高度可用來協助計算讀取器在讀取過程中在內容捲動的距離。HTML 也包含一對 <script> 元素。其中一個元素會將 SDK 新增 JavaScript至頁面，另一個元素則新增瀏覽器指令碼，以擷取頁面上的捲動進度並將其報告給 Kinesis。

```
<!DOCTYPE html>
<html>
    <head>
        <title>AWS SDK for JavaScript - Amazon Kinesis Application</title>
    </head>
    <body>
        <div id="BlogContent" style="width: 60%; height: 800px; overflow: auto;margin:
 auto; text-align: center;">
            <div>
                <p>
                Lorem ipsum dolor sit amet, consectetur adipiscing elit. Vestibulum
 vitae nulla eget nisl bibendum feugiat. Fusce rhoncus felis at ultricies luctus.
 Vivamus fermentum cursus sem at interdum. Proin vel lobortis nulla. Aenean rutrum
 odio in tellus semper rhoncus. Nam eu felis ac augue dapibus laoreet vel in erat.
 Vivamus vitae mollis turpis. Integer sagittis dictum odio. Duis nec sapien diam.
 In imperdiet sem nec ante laoreet, vehicula facilisis sem placerat. Duis ut metus
 egestas, ullamcorper neque et, accumsan quam. Class aptent taciti sociosqu ad litora
 torquent per conubia nostra, per inceptos himenaeos.
                </p>
                <!-- Additional paragraphs in the blog page appear here -->
            </div>
        </div>
        <script src="https://sdk.amazonaws.com/js/aws-sdk-2.283.1.min.js"></script>
        <script src="kinesis-example.js"></script>
    </body>
</html>
```

設定軟體開發套件

透過呼叫提供 Amazon Cognito 身分識別集區識別碼的CognitoIdentityCredentials方法來取得設定 SDK 所需的登入資料。成功後，請在回呼函數中建立 Kinesis 服務物件。

下列程式碼片段說明此步驟。(如需完整範例，請參閱擷取 Web 頁面捲動進度程式碼。)

```
// Configure Credentials to use Cognito
```

```
AWS.config.credentials = new AWS.CognitoIdentityCredentials({
  IdentityPoolId: "IDENTITY_POOL_ID",
});

AWS.config.region = "REGION";
// We're going to partition Amazon Kinesis records based on an identity.
// We need to get credentials first, then attach our event listeners.
AWS.config.credentials.get(function (err) {
  // attach event listener
  if (err) {
    alert("Error retrieving credentials.");
    console.error(err);
    return;
  }
  // create Amazon Kinesis service object
  var kinesis = new AWS.Kinesis({
    apiVersion: "2013-12-02",
  });
```

建立捲動記錄

可使用包含部落格文章內容之 的 scrollHeight 和 scrollTop 屬性來計算捲動進度。每個捲動記錄都會在事件的事件偵聽程式函數中建立，然後新增至記錄陣列，以便定期提交給 Kinesis。scroll

下列程式碼片段說明此步驟。(如需完整範例，請參閱擷取 Web 頁面捲動進度程式碼。)

```
  // Get the ID of the Web page element.
  var blogContent = document.getElementById("BlogContent");

  // Get Scrollable height
  var scrollableHeight = blogContent.clientHeight;

  var recordData = [];
  var TID = null;
  blogContent.addEventListener("scroll", function (event) {
    clearTimeout(TID);
    // Prevent creating a record while a user is actively scrolling
    TID = setTimeout(function () {
      // calculate percentage
      var scrollableElement = event.target;
      var scrollHeight = scrollableElement.scrollHeight;
      var scrollTop = scrollableElement.scrollTop;
```

```
    var scrollTopPercentage = Math.round((scrollTop / scrollHeight) * 100);
    var scrollBottomPercentage = Math.round(
      ((scrollTop + scrollableHeight) / scrollHeight) * 100
    );

    // Create the Amazon Kinesis record
    var record = {
      Data: JSON.stringify({
        blog: window.location.href,
        scrollTopPercentage: scrollTopPercentage,
        scrollBottomPercentage: scrollBottomPercentage,
        time: new Date(),
      }),
      PartitionKey: "partition-" + AWS.config.credentials.identityId,
    };
    recordData.push(record);
  }, 100);
});
```

提交記錄至 Kinesis

每秒一次，如果陣列中有記錄，那些擱置的記錄會傳送至 Kinesis。

下列程式碼片段說明此步驟。(如需完整範例，請參閱擷取 Web 頁面捲動進度程式碼。)

```
// upload data to Amazon Kinesis every second if data exists
setInterval(function () {
  if (!recordData.length) {
    return;
  }
  // upload data to Amazon Kinesis
  kinesis.putRecords(
    {
      Records: recordData,
      StreamName: "NAME_OF_STREAM",
    },
    function (err, data) {
      if (err) {
        console.error(err);
      }
    }
  );
  // clear record data
```

```
    recordData = [];
  }, 1000);
});
```

擷取 Web 頁面捲動進度程式碼

以下是 Kinesis 擷取網頁捲動進度範例的瀏覽器指令碼。

```
// Configure Credentials to use Cognito
AWS.config.credentials = new AWS.CognitoIdentityCredentials({
  IdentityPoolId: "IDENTITY_POOL_ID",
});

AWS.config.region = "REGION";
// We're going to partition Amazon Kinesis records based on an identity.
// We need to get credentials first, then attach our event listeners.
AWS.config.credentials.get(function (err) {
  // attach event listener
  if (err) {
    alert("Error retrieving credentials.");
    console.error(err);
    return;
  }
  // create Amazon Kinesis service object
  var kinesis = new AWS.Kinesis({
    apiVersion: "2013-12-02",
  });

  // Get the ID of the Web page element.
  var blogContent = document.getElementById("BlogContent");

  // Get Scrollable height
  var scrollableHeight = blogContent.clientHeight;

  var recordData = [];
  var TID = null;
  blogContent.addEventListener("scroll", function (event) {
    clearTimeout(TID);
    // Prevent creating a record while a user is actively scrolling
    TID = setTimeout(function () {
      // calculate percentage
      var scrollableElement = event.target;
      var scrollHeight = scrollableElement.scrollHeight;
```

```
      var scrollTop = scrollableElement.scrollTop;

      var scrollTopPercentage = Math.round((scrollTop / scrollHeight) * 100);
      var scrollBottomPercentage = Math.round(
        ((scrollTop + scrollableHeight) / scrollHeight) * 100
      );

      // Create the Amazon Kinesis record
      var record = {
        Data: JSON.stringify({
          blog: window.location.href,
          scrollTopPercentage: scrollTopPercentage,
          scrollBottomPercentage: scrollBottomPercentage,
          time: new Date(),
        }),
        PartitionKey: "partition-" + AWS.config.credentials.identityId,
      };
      recordData.push(record);
    }, 100);
  });

  // upload data to Amazon Kinesis every second if data exists
  setInterval(function () {
    if (!recordData.length) {
      return;
    }
    // upload data to Amazon Kinesis
    kinesis.putRecords(
      {
        Records: recordData,
        StreamName: "NAME_OF_STREAM",
      },
      function (err, data) {
        if (err) {
          console.error(err);
        }
      }
    );
    // clear record data
    recordData = [];
  }, 1000);
});
```

Amazon S3 範例

Amazon Simple Storage Service (Amazon S3) 是一種提供可高度擴展雲端儲存的網路服務。Amazon S3 提供易於使用的物件儲存，具有簡單的 Web 服務界面，可從 Web 上的任何位置存放和擷取任意數量的資料。

Amazon S3 的 JavaScript API 是透過`AWS.S3`用戶端類別公開的。如需有關使用 Amazon S3 用戶端類別的詳細資訊，請參閱 API 參考資料`Class: AWS.S3`中的。

主題

- Amazon S3 瀏覽器示例
- Amazon S3 的例子 Node.js

Amazon S3 瀏覽器示例

下列主題顯示兩個範例，說明如 AWS SDK for JavaScript 何在瀏覽器中使用該儲存貯體與 Amazon S3 儲存貯體互動。

- 第一個顯示一個簡單的案例，其中任何 (未經驗證) 的使用者都可以檢視 Amazon S3 儲存貯體中的現有相片。
- 第二個顯示更複雜的情況，其中允許用戶對存儲桶中的照片執行操作，例如上傳，刪除等。

主題

- 從瀏覽器檢視 Amazon S3 儲存貯體中的相片
- 從瀏覽器將照片上傳到 Amazon S3

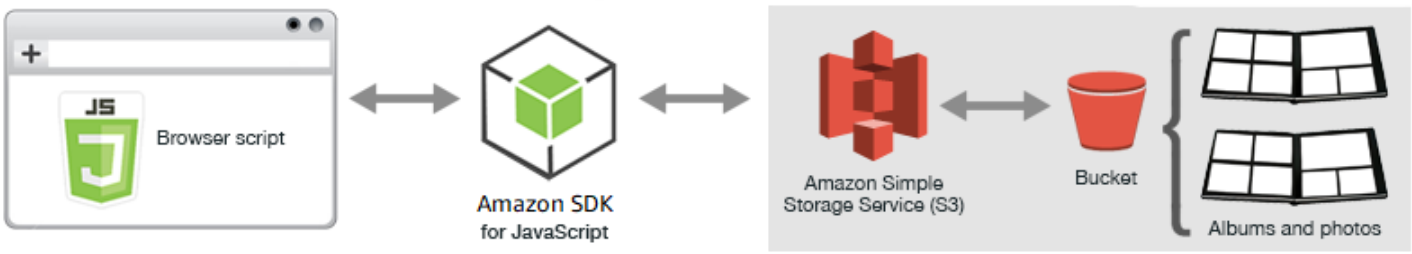
從瀏覽器檢視 Amazon S3 儲存貯體中的相片

此瀏覽器程式碼範例顯示：

- 如何在亞馬遜簡單存儲服務（Amazon S3）存儲桶中創建相冊，並允許未經身份驗證的用戶查看照片。

使用案例

在這個範例中，簡單的 HTML 頁面提供瀏覽器型應用程式，來檢視相簿中的相片。相冊位於 Amazon S3 存儲桶中，照片上傳到其中。



瀏覽器指令碼使用開發 JavaScript 套件與 Amazon S3 儲存貯體進行互動。此指令碼使用 Amazon S3 用戶端類別的`listObjects`方法，讓您能夠檢視相簿。

先決條件任務

若要設定和執行此範例，請先完成這些任務。

> **Note**
>
> 在此範例中，您必須對 Amazon S3 儲存貯體和 Amazon Cognito 身分識別集 AWS 區使用相同的區域。

建立儲存貯體

在 Amazon S3 主控台中，建立 Amazon S3 儲存貯體，您可以在其中存放相簿和相片。如需使用主控台建立 S3 儲存貯體的詳細資訊，請參閱 Amazon 簡單儲存服務使用者指南中的建立儲存貯體。

建立 S3 儲存貯體時，請務必執行下列動作：

- 記下儲存貯體名稱，這會在後續先決條件任務「設定角色許可」中用到。
- 選擇要在其中建立值 AWS 區的「區域」。這個區域必須與您在後續先決條件任務「建立身分集區」中用來建立 Amazon Cognito 身分集區的區域相同。
- 按照 Amazon 簡單儲存服務使用者指南中的設定網站存取權限來設定儲存貯體許可。

建立 身分集區

在 Amazon Cognito 主控台中，建立 Amazon Cognito 身分識別集區，如瀏覽器指令碼入門主題中所述。the section called “步驟 1：創建一個 Amazon Cognito 身份池”

建立識別集區時，請記下識別集區名稱以及未驗證身分的角色名稱。

設定角色許可

若要允許檢視相簿和相片，您必須將權限新增至剛建立之身分集區的 IAM 角色。首先，如下所示建立政策。

1. 開啟 IAM 主控台。
2. 在左邊的導覽窗格中，選擇 Policies (政策)，然後選擇 Create policy (建立政策) 按鈕。
3. 在 JSON 標籤上，輸入下列 JSON 定義，但將 BUCKET_NAME 更換為儲存貯體的名稱。

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
            "s3:ListBucket"
         ],
         "Resource": [
            "arn:aws:s3:::BUCKET_NAME"
         ]
      }
   ]
}
```

4. 選擇 Review Policy (檢閱政策) 按鈕、命名政策並提供描述 (如果需要)，然後選擇 Create policy (建立政策) 按鈕。

請務必記下名稱，以便稍後找到它並將其附加到 IAM 角色。

建立政策後，瀏覽回 IAM 主控台。在先前的先決條件任務「建立身分集區」中，尋找 Amazon Cognito 建立的未驗證身分的 IAM 角色。您使用剛建立的政策來新增許可至此身分。

雖然此工作的工作流程通常與瀏覽器指令碼中的入門主題相同，但有幾個不同the section called “步驟 2：將政策新增至建立的 IAM 角色”之處需要注意：

- 使用您剛建立的新政策，而不是 Amazon Polly 的政策。
- 在 Attach Permissions (連接許可) 頁面上，快速找到新政策、開啟 Filter policies (篩選政策) 清單，然後選擇 Customer managed (客戶受管)。

如需有關建立 IAM 角色的其他資訊，請參閱 IAM 使用者指南中的建立角色以將許可委派給 AWS 服務。

設定 CORS

在瀏覽器指令碼可以存取 Amazon S3 儲存貯體之前，您必須先設定其 CORS 組態，如下所示。

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

JSON

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

```
]
```

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<CORSConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
    <CORSRule>
        <AllowedOrigin>*</AllowedOrigin>
        <AllowedMethod>GET</AllowedMethod>
        <AllowedMethod>HEAD</AllowedMethod>
        <AllowedHeader>*</AllowedHeader>
    </CORSRule>
</CORSConfiguration>
```

建立相簿和上傳相片

由於這個範例只允許使用者檢視已經在儲存貯體中的相片，因此您需要在儲存貯體中建立相簿並上傳照片至相簿中。

> **Note**
>
> 在這個範例中，相片檔案的檔案名稱必須以單一底線 (「_」) 做為開頭。此字元在後續篩選程序中很重要。此外，請務必尊重相片擁有者的著作權。

1. 在 Amazon S3 主控台中，開啟您先前建立的儲存貯體。
2. 在 Overview (概觀) 標籤中，選擇 Create folder (建立資料夾) 按鈕來建立資料夾。在此範例中，將資料夾命名為「album1」、「album2」和「album3」。
3. 對於 album1 以及 album2，選取資料夾然後上傳相片到其中，如下所示：
   a. 選擇 Upload (上傳) 按鈕。
   b. 拖曳或選擇您要使用的相片檔案，然後選擇 Next (下一步)。
   c. 在 Manage public permissions (管理公有許可) 中，選擇 Grant public read access to this object(s) (授予對該物件的公有讀取許可)。
   d. 選擇 Upload (上傳) 按鈕 (位於左下角)。
4. 將 album3 保留為空。

定義網頁

相片檢視應用程式的 HTML 包含 元素，瀏覽器指令碼在其中建立檢視界面。第一個 <script> 元素會新增軟體開發套件至瀏覽器指令碼。第二個<script>元素會新增保存瀏覽器指令碼的外部 JavaScript 檔案。

在這個範例中，檔案名為 PhotoViewer.js，並位於 HTML 檔案的相同資料夾。若要尋找目前的 SDK_ 版本編號，請參閱 API 參考指南中適用於 SDK 的 API 參考。 JavaScript AWS SDK for JavaScript

```
<!DOCTYPE html>
<html>
  <head>
    <!-- **DO THIS**: -->
    <!--   Replace SDK_VERSION_NUMBER with the current SDK version number -->
    <script src="https://sdk.amazonaws.com/js/aws-sdk-SDK_VERSION_NUMBER.js"></script>
    <script src="./PhotoViewer.js"></script>
    <script>listAlbums();</script>
  </head>
  <body>
    <h1>Photo Album Viewer</h1>
    <div id="viewer" />
  </body>
</html>
```

設定軟體開發套件

呼叫 CognitoIdentityCredentials 方法，取得設定開發套件所需的登入資料。您需要提供 Amazon Cognito 可身份集區 ID。接著，建立一個 AWS.S3 服務物件。

```
// **DO THIS**:
//   Replace BUCKET_NAME with the bucket name.
//
var albumBucketName = "BUCKET_NAME";

// **DO THIS**:
//   Replace this block of code with the sample code located at:
//   Cognito -- Manage Identity Pools -- [identity_pool_name] -- Sample Code --
 JavaScript
//
// Initialize the Amazon Cognito credentials provider
AWS.config.region = "REGION"; // Region
```

```
AWS.config.credentials = new AWS.CognitoIdentityCredentials({
  IdentityPoolId: "IDENTITY_POOL_ID",
});

// Create a new service object
var s3 = new AWS.S3({
  apiVersion: "2006-03-01",
  params: { Bucket: albumBucketName },
});

// A utility function to create HTML.
function getHtml(template) {
  return template.join("\n");
}
```

此範例中的其他程式碼定義以下函數來收集和呈現儲存貯體中相簿和相片的相關資訊。

- `listAlbums`
- `viewAlbum`

於儲存貯體中列出相簿

為了列出儲存貯體中的所有現有相簿，應用程式的 `listAlbums` 函數會呼叫 `AWS.S3` 服務物件的 `listObjects` 方法。函數使用 `CommonPrefixes` 屬性，因此呼叫只會傳回做為相簿的物件 (也就是資料夾)。

該函數的其餘部分從 Amazon S3 存儲桶中獲取相冊列表，並生成在網頁上顯示相冊列表所需的 HTML。

```
// List the photo albums that exist in the bucket.
function listAlbums() {
  s3.listObjects({ Delimiter: "/" }, function (err, data) {
    if (err) {
      return alert("There was an error listing your albums: " + err.message);
    } else {
      var albums = data.CommonPrefixes.map(function (commonPrefix) {
        var prefix = commonPrefix.Prefix;
        var albumName = decodeURIComponent(prefix.replace("/", ""));
        return getHtml([
          "<li>",
          '<button style="margin:5px;" onclick="viewAlbum(\'' +
            albumName +
```

```
            "')\">",
          albumName,
          "</button>",
          "</li>",
        ]);
      });
      var message = albums.length
        ? getHtml(["<p>Click on an album name to view it.</p>"])
        : "<p>You do not have any albums. Please Create album.";
      var htmlTemplate = [
        "<h2>Albums</h2>",
        message,
        "<ul>",
        getHtml(albums),
        "</ul>",
      ];
      document.getElementById("viewer").innerHTML = getHtml(htmlTemplate);
    }
  });
}
```

檢視相簿

若要在 Amazon S3 儲存貯體中顯示相簿的內容，應用程式的`viewAlbum`函數會取一個相簿名稱，並為該相簿建立 Amazon S3 金鑰。然後該函數會呼叫 AWS.S3 服務物件的 `listObjects` 方法以取得相簿中的所有物件 (相片) 清單。

其餘函數會採用該相簿中的物件清單，並產生在網頁中顯示相片所需的 HTML。

```
// Show the photos that exist in an album.
function viewAlbum(albumName) {
  var albumPhotosKey = encodeURIComponent(albumName) + "/";
  s3.listObjects({ Prefix: albumPhotosKey }, function (err, data) {
    if (err) {
      return alert("There was an error viewing your album: " + err.message);
    }
    // 'this' references the AWS.Request instance that represents the response
    var href = this.request.httpRequest.endpoint.href;
    var bucketUrl = href + albumBucketName + "/";

    var photos = data.Contents.map(function (photo) {
      var photoKey = photo.Key;
      var photoUrl = bucketUrl + encodeURIComponent(photoKey);
```

```
      return getHtml([
        "<span>",
        "<div>",
        "<br/>",
        '<img style="width:128px;height:128px;" src="' + photoUrl + '"/>',
        "</div>",
        "<div>",
        "<span>",
        photoKey.replace(albumPhotosKey, ""),
        "</span>",
        "</div>",
        "</span>",
      ]);
    });
    var message = photos.length
      ? "<p>The following photos are present.</p>"
      : "<p>There are no photos in this album.</p>";
    var htmlTemplate = [
      "<div>",
      '<button onclick="listAlbums()">',
      "Back To Albums",
      "</button>",
      "</div>",
      "<h2>",
      "Album: " + albumName,
      "</h2>",
      message,
      "<div>",
      getHtml(photos),
      "</div>",
      "<h2>",
      "End of Album: " + albumName,
      "</h2>",
      "<div>",
      '<button onclick="listAlbums()">',
      "Back To Albums",
      "</button>",
      "</div>",
    ];
    document.getElementById("viewer").innerHTML = getHtml(htmlTemplate);
    document
      .getElementsByTagName("img")[0]
      .setAttribute("style", "display:none;");
  });
```

```
}
```

在 Amazon S3 儲存貯體中檢視相片：完整程式碼

本節包含可檢視 Amazon S3 儲存貯體中相片的範例完整 HTML 和 JavaScript 程式碼。如需詳細資訊和先決條件，請參閱上層區段。

HTML 的範例：

```
<!DOCTYPE html>
<html>
  <head>
    <!-- **DO THIS**: -->
    <!--   Replace SDK_VERSION_NUMBER with the current SDK version number -->
    <script src="https://sdk.amazonaws.com/js/aws-sdk-SDK_VERSION_NUMBER.js"></script>
    <script src="./PhotoViewer.js"></script>
    <script>listAlbums();</script>
  </head>
  <body>
    <h1>Photo Album Viewer</h1>
    <div id="viewer" />
  </body>
</html>
```

您可以在這裡找到此範例程式碼 GitHub。

範例的瀏覽器指令碼程式碼：

```
//
// Data constructs and initialization.
//

// **DO THIS**:
//   Replace BUCKET_NAME with the bucket name.
//
var albumBucketName = "BUCKET_NAME";

// **DO THIS**:
//   Replace this block of code with the sample code located at:
//   Cognito -- Manage Identity Pools -- [identity_pool_name] -- Sample Code --
 JavaScript
//
// Initialize the Amazon Cognito credentials provider
```

```
AWS.config.region = "REGION"; // Region
AWS.config.credentials = new AWS.CognitoIdentityCredentials({
  IdentityPoolId: "IDENTITY_POOL_ID",
});

// Create a new service object
var s3 = new AWS.S3({
  apiVersion: "2006-03-01",
  params: { Bucket: albumBucketName },
});

// A utility function to create HTML.
function getHtml(template) {
  return template.join("\n");
}

//
// Functions
//

// List the photo albums that exist in the bucket.
function listAlbums() {
  s3.listObjects({ Delimiter: "/" }, function (err, data) {
    if (err) {
      return alert("There was an error listing your albums: " + err.message);
    } else {
      var albums = data.CommonPrefixes.map(function (commonPrefix) {
        var prefix = commonPrefix.Prefix;
        var albumName = decodeURIComponent(prefix.replace("/", ""));
        return getHtml([
          "<li>",
          '<button style="margin:5px;" onclick="viewAlbum(\'' +
            albumName +
            "')\">",
          albumName,
          "</button>",
          "</li>",
        ]);
      });
      var message = albums.length
        ? getHtml(["<p>Click on an album name to view it.</p>"])
        : "<p>You do not have any albums. Please Create album.";
      var htmlTemplate = [
        "<h2>Albums</h2>",
```

```
        message,
        "<ul>",
        getHtml(albums),
        "</ul>",
      ];
      document.getElementById("viewer").innerHTML = getHtml(htmlTemplate);
    }
  });
}

// Show the photos that exist in an album.
function viewAlbum(albumName) {
  var albumPhotosKey = encodeURIComponent(albumName) + "/";
  s3.listObjects({ Prefix: albumPhotosKey }, function (err, data) {
    if (err) {
      return alert("There was an error viewing your album: " + err.message);
    }
    // 'this' references the AWS.Request instance that represents the response
    var href = this.request.httpRequest.endpoint.href;
    var bucketUrl = href + albumBucketName + "/";

    var photos = data.Contents.map(function (photo) {
      var photoKey = photo.Key;
      var photoUrl = bucketUrl + encodeURIComponent(photoKey);
      return getHtml([
        "<span>",
        "<div>",
        "<br/>",
        '<img style="width:128px;height:128px;" src="' + photoUrl + '"/>',
        "</div>",
        "<div>",
        "<span>",
        photoKey.replace(albumPhotosKey, ""),
        "</span>",
        "</div>",
        "</span>",
      ]);
    });
    var message = photos.length
      ? "<p>The following photos are present.</p>"
      : "<p>There are no photos in this album.</p>";
    var htmlTemplate = [
      "<div>",
      '<button onclick="listAlbums()">',
```

```
      "Back To Albums",
      "</button>",
      "</div>",
      "<h2>",
      "Album: " + albumName,
      "</h2>",
      message,
      "<div>",
      getHtml(photos),
      "</div>",
      "<h2>",
      "End of Album: " + albumName,
      "</h2>",
      "<div>",
      '<button onclick="listAlbums()">',
      "Back To Albums",
      "</button>",
      "</div>",
    ];
    document.getElementById("viewer").innerHTML = getHtml(htmlTemplate);
    document
      .getElementsByTagName("img")[0]
      .setAttribute("style", "display:none;");
  });
}
```

您可以在這裡找到此範例程式碼 GitHub。

從瀏覽器將照片上傳到 Amazon S3

此瀏覽器程式碼範例顯示：

- 如何創建允許用戶在 Amazon S3 存儲桶中創建相冊並將照片上傳到相冊的瀏覽器應用程序。

使用案例

在此範例中，簡單的 HTML 頁面提供了一個以瀏覽器為基礎的應用程式，用於在 Amazon S3 儲存貯體中建立相簿，您可以在其中上傳相片。該應用程式可讓您刪除新增的相片與相簿。

瀏覽器指令碼使用開發 JavaScript 套件與 Amazon S3 儲存貯體進行互動。使用下列 Amazon S3 用戶端類別的方法來啟用相簿應用程式：

- `listObjects`
- `headObject`
- `putObject`
- `upload`
- `deleteObject`
- `deleteObjects`

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：

- 在 Amazon S3 主控台中，建立一個 Amazon S3 儲存貯體，您將用來將相片存放在相簿中。如需在主控台中建立值區的詳細資訊，請參閱 Amazon 簡單儲存服務使用者指南中的建立儲存貯體。請確定您同時擁有物件的讀取和寫入權限。如需有關設定值區權限的詳細資訊，請參閱設定網站存取權限。
- 在 Amazon Cognito 主控台中，使用聯合身分建立 Amazon Cognito 身分集區，並為與 Amazon S3 儲存貯體位於相同區域中的未驗證使用者啟用存取權限。您必須在程式碼中包含身分集區 ID，來取得瀏覽器指令碼的登入資料。如需 Amazon Cognito 聯合身分識別的詳細資訊，請參閱 Amazon Cognito 開發人員指南中的 Amazon Cognito 身分識別集區 (聯合身分識別)。
- 在 IAM 主控台中，尋找 Amazon Cognito 為未經驗證的使用者建立的 IAM 角色。新增下列政策以授與讀取和寫入權限給 Amazon S3 儲存貯體。如需有關建立 IAM 角色的詳細資訊，請參閱 IAM 使用者指南中的建立角色以將許可委派給AWS服務。

  針對 Amazon Cognito 為未經驗證的使用者建立的 IAM 角色使用此角色政策。

> **Warning**
>
> 如果您為未經授權使用者啟用存取權，您將授與該儲存貯體以及儲存貯體中所有物件的寫入權限給全世界的所有人。在此範例中，此安全狀態非常有用，可讓您專注於範例的主要目標。不過，在許多實際情況下，強烈建議採用更嚴格的安全性 (例如使用經驗證的使用者和物件擁有權)。

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
            "s3:DeleteObject",
            "s3:GetObject",
            "s3:ListBucket",
            "s3:PutObject",
            "s3:PutObjectAcl"
         ],
         "Resource": [
            "arn:aws:s3:::BUCKET_NAME",
            "arn:aws:s3:::BUCKET_NAME/*"
         ]
      }
   ]
}
```

設定 CORS

在瀏覽器指令碼可以存取 Amazon S3 儲存貯體之前，您必須先設定其 CORS 組態，如下所示。

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

JSON

```
[
    {
        "AllowedHeaders": [
            "*"
        ],
        "AllowedMethods": [
            "HEAD",
            "GET",
            "PUT",
            "POST",
            "DELETE"
        ],
        "AllowedOrigins": [
            "*"
        ],
        "ExposeHeaders": [
            "ETag"
        ]
    }
]
```

XML

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

網頁

上傳相片的應用程式 HTML 是由 元素所組成，該元素在建立上傳使用者界面之瀏覽器程式碼中。第一個 <script> 元素會新增軟體開發套件至瀏覽器指令碼。第二個<script>元素會新增保存瀏覽器指令碼的外部 JavaScript 檔案。

```
<!DOCTYPE html>
<html>
  <head>
     <!-- **DO THIS**: -->
    <!--   Replace SDK_VERSION_NUMBER with the current SDK version number -->
    <script src="https://sdk.amazonaws.com/js/aws-sdk-SDK_VERSION_NUMBER.js"></script>
    <script src="./s3_photoExample.js"></script>
    <script>
       function getHtml(template) {
          return template.join('\n');
       }
       listAlbums();
    </script>
  </head>
  <body>
    <h1>My Photo Albums App</h1>
    <div id="app"></div>
  </body>
</html>
```

設定軟體開發套件

透過呼叫提供 Amazon Cognito 身分識別集區識別碼的CognitoIdentityCredentials方法來取得設定 SDK 所需的登入資料。接下來，建立一個 AWS.S3 服務物件。

```
var albumBucketName = "BUCKET_NAME";
var bucketRegion = "REGION";
var IdentityPoolId = "IDENTITY_POOL_ID";

AWS.config.update({
  region: bucketRegion,
  credentials: new AWS.CognitoIdentityCredentials({
    IdentityPoolId: IdentityPoolId,
  }),
});
```

```
var s3 = new AWS.S3({
  apiVersion: "2006-03-01",
  params: { Bucket: albumBucketName },
});
```

此範例中幾乎所有其餘程式碼都組織為一系列的函數，這些函數收集並顯示有關儲存貯體中的相簿資訊，上傳和顯示上傳到相簿中的照片，並刪除照片和相簿。這些函數為：

- `listAlbums`
- `createAlbum`
- `viewAlbum`
- `addPhoto`
- `deleteAlbum`
- `deletePhoto`

於儲存貯體中列出相簿

應用程式會在 Amazon S3 儲存貯體中建立相簿，做為其金鑰以正斜線字元開頭的物件，表示物件以資料夾的方式運作。若要在儲存貯體列出所有現有相簿，該應用程式的 `listAlbums` 函數會在使用 `commonPrefix` 時同時呼叫 `AWS.S3` 服務物件的 `listObjects` 方法，以便呼叫僅傳回做為相簿使用的物件。

該函數的其餘部分從 Amazon S3 存儲桶中獲取相冊列表，並生成在網頁中顯示相冊列表所需的 HTML。它也會啟用刪除和開啟個別相簿。

```
function listAlbums() {
  s3.listObjects({ Delimiter: "/" }, function (err, data) {
    if (err) {
      return alert("There was an error listing your albums: " + err.message);
    } else {
      var albums = data.CommonPrefixes.map(function (commonPrefix) {
        var prefix = commonPrefix.Prefix;
        var albumName = decodeURIComponent(prefix.replace("/", ""));
        return getHtml([
          "<li>",
          "<span onclick=\"deleteAlbum('" + albumName + "')\">X</span>",
          "<span onclick=\"viewAlbum('" + albumName + "')\">",
          albumName,
          "</span>",
```

```
          "</li>",
        ]);
      });
      var message = albums.length
        ? getHtml([
            "<p>Click on an album name to view it.</p>",
            "<p>Click on the X to delete the album.</p>",
          ])
        : "<p>You do not have any albums. Please Create album.";
      var htmlTemplate = [
        "<h2>Albums</h2>",
        message,
        "<ul>",
        getHtml(albums),
        "</ul>",
        "<button onclick=\"createAlbum(prompt('Enter Album Name:'))\">",
        "Create New Album",
        "</button>",
      ];
      document.getElementById("app").innerHTML = getHtml(htmlTemplate);
    }
  });
}
```

在儲存貯體中建立相簿

若要在 Amazon S3 儲存貯體中建立相簿，應用程式的`createAlbum`函數會先驗證為新相簿指定的名稱，以確保其包含合適的字元。然後，函數會形成 Amazon S3 物件金鑰，並將其傳遞至 Amazon S3 服務物件的`headObject`方法。此方法會回傳特定金鑰的中繼資料，因此，如果它回傳資料時，那麼具有該金鑰的物件就已經存在了。

如果相簿尚未存在，則函數會呼叫 `AWS.S3` 服務物件的 `putObject` 方法以建立該相簿。然後它將呼叫 `viewAlbum` 函數以顯示新的空相簿。

```
function createAlbum(albumName) {
  albumName = albumName.trim();
  if (!albumName) {
    return alert("Album names must contain at least one non-space character.");
  }
  if (albumName.indexOf("/") !== -1) {
    return alert("Album names cannot contain slashes.");
  }
  var albumKey = encodeURIComponent(albumName);
```

```
  s3.headObject({ Key: albumKey }, function (err, data) {
    if (!err) {
      return alert("Album already exists.");
    }
    if (err.code !== "NotFound") {
      return alert("There was an error creating your album: " + err.message);
    }
    s3.putObject({ Key: albumKey }, function (err, data) {
      if (err) {
        return alert("There was an error creating your album: " + err.message);
      }
      alert("Successfully created album.");
      viewAlbum(albumName);
    });
  });
}
```

檢視相簿

若要在 Amazon S3 儲存貯體中顯示相簿的內容，應用程式的`viewAlbum`函數會取一個相簿名稱，並為該相簿建立 Amazon S3 金鑰。然後該函數會呼叫 `AWS.S3` 服務物件的 `listObjects` 方法以取得相簿中的所有物件 (相片) 清單。

其餘函數會從該相簿採用物件 (相片) 清單，並產生在網頁中顯示相片所需的 HTML。它也會啟用刪除個別照片並導覽回相簿清單的功能。

```
function viewAlbum(albumName) {
  var albumPhotosKey = encodeURIComponent(albumName) + "/";
  s3.listObjects({ Prefix: albumPhotosKey }, function (err, data) {
    if (err) {
      return alert("There was an error viewing your album: " + err.message);
    }
    // 'this' references the AWS.Response instance that represents the response
    var href = this.request.httpRequest.endpoint.href;
    var bucketUrl = href + albumBucketName + "/";

    var photos = data.Contents.map(function (photo) {
      var photoKey = photo.Key;
      var photoUrl = bucketUrl + encodeURIComponent(photoKey);
      return getHtml([
        "<span>",
        "<div>",
        '<img style="width:128px;height:128px;" src="' + photoUrl + '"/>',
```

```
        "</div>",
        "<div>",
        "<span onclick=\"deletePhoto('" +
          albumName +
          "','" +
          photoKey +
          "')\">",
        "X",
        "</span>",
        "<span>",
        photoKey.replace(albumPhotosKey, ""),
        "</span>",
        "</div>",
        "</span>",
      ]);
    });
    var message = photos.length
      ? "<p>Click on the X to delete the photo</p>"
      : "<p>You do not have any photos in this album. Please add photos.</p>";
    var htmlTemplate = [
      "<h2>",
      "Album: " + albumName,
      "</h2>",
      message,
      "<div>",
      getHtml(photos),
      "</div>",
      '<input id="photoupload" type="file" accept="image/*">',
      '<button id="addphoto" onclick="addPhoto(\'' + albumName + "')\">",
      "Add Photo",
      "</button>",
      '<button onclick="listAlbums()">',
      "Back To Albums",
      "</button>",
    ];
    document.getElementById("app").innerHTML = getHtml(htmlTemplate);
  });
}
```

新增相片至相簿

若要將相片上傳到 Amazon S3 儲存貯體中的相簿，應用程式的addPhoto功能會使用網頁中的檔案選擇器元素來識別要上傳的檔案。它接著會形成一個金鑰，以供現有相簿名稱和檔案名稱上傳相片。

該函數調用 Amazon S3 服務對象的upload方法來上傳照片。在上傳相片後，該函數會重新顯示該相簿以顯示上傳的照片。

```
function addPhoto(albumName) {
  var files = document.getElementById("photoupload").files;
  if (!files.length) {
    return alert("Please choose a file to upload first.");
  }
  var file = files[0];
  var fileName = file.name;
  var albumPhotosKey = encodeURIComponent(albumName) + "/";

  var photoKey = albumPhotosKey + fileName;

  // Use S3 ManagedUpload class as it supports multipart uploads
  var upload = new AWS.S3.ManagedUpload({
    params: {
      Bucket: albumBucketName,
      Key: photoKey,
      Body: file,
    },
  });

  var promise = upload.promise();

  promise.then(
    function (data) {
      alert("Successfully uploaded photo.");
      viewAlbum(albumName);
    },
    function (err) {
      return alert("There was an error uploading your photo: ", err.message);
    }
  );
}
```

刪除相片

若要從 Amazon S3 儲存貯體中的相簿刪除相片，應用程式的deletePhoto函數會呼叫 Amazon S3 服務物件的deleteObject方法。這會刪除由 photoKey 值傳遞給函數所指定的相片。

```
function deletePhoto(albumName, photoKey) {
  s3.deleteObject({ Key: photoKey }, function (err, data) {
```

```
    if (err) {
      return alert("There was an error deleting your photo: ", err.message);
    }
    alert("Successfully deleted photo.");
    viewAlbum(albumName);
  });
}
```

刪除相簿

若要刪除 Amazon S3 儲存貯體中的相簿，應用程式的`deleteAlbum`函數會呼叫 Amazon S3 服務物件的`deleteObjects`方法。

```
function deleteAlbum(albumName) {
  var albumKey = encodeURIComponent(albumName) + "/";
  s3.listObjects({ Prefix: albumKey }, function (err, data) {
    if (err) {
      return alert("There was an error deleting your album: ", err.message);
    }
    var objects = data.Contents.map(function (object) {
      return { Key: object.Key };
    });
    s3.deleteObjects(
      {
        Delete: { Objects: objects, Quiet: true },
      },
      function (err, data) {
        if (err) {
          return alert("There was an error deleting your album: ", err.message);
        }
        alert("Successfully deleted album.");
        listAlbums();
      }
    );
  });
}
```

將照片上傳到 Amazon S3：完整代碼

本節包含將相片上傳至 Amazon S3 相簿的範例完整 HTML 和 JavaScript 程式碼。如需詳細資訊和先決條件，請參閱上層區段。

HTML 的範例：

```
<!DOCTYPE html>
<html>
  <head>
     <!-- **DO THIS**: -->
    <!--   Replace SDK_VERSION_NUMBER with the current SDK version number -->
    <script src="https://sdk.amazonaws.com/js/aws-sdk-SDK_VERSION_NUMBER.js"></script>
    <script src="./s3_photoExample.js"></script>
    <script>
       function getHtml(template) {
          return template.join('\n');
       }
       listAlbums();
    </script>
  </head>
  <body>
    <h1>My Photo Albums App</h1>
    <div id="app"></div>
  </body>
</html>
```

您可以在這裡找到此範例程式碼 GitHub。

範例的瀏覽器指令碼程式碼：

```
var albumBucketName = "BUCKET_NAME";
var bucketRegion = "REGION";
var IdentityPoolId = "IDENTITY_POOL_ID";

AWS.config.update({
  region: bucketRegion,
  credentials: new AWS.CognitoIdentityCredentials({
    IdentityPoolId: IdentityPoolId,
  }),
});

var s3 = new AWS.S3({
  apiVersion: "2006-03-01",
  params: { Bucket: albumBucketName },
});

function listAlbums() {
  s3.listObjects({ Delimiter: "/" }, function (err, data) {
    if (err) {
```

```
      return alert("There was an error listing your albums: " + err.message);
    } else {
      var albums = data.CommonPrefixes.map(function (commonPrefix) {
        var prefix = commonPrefix.Prefix;
        var albumName = decodeURIComponent(prefix.replace("/", ""));
        return getHtml([
          "<li>",
          "<span onclick=\"deleteAlbum('" + albumName + "')\">X</span>",
          "<span onclick=\"viewAlbum('" + albumName + "')\">",
          albumName,
          "</span>",
          "</li>",
        ]);
      });
      var message = albums.length
        ? getHtml([
            "<p>Click on an album name to view it.</p>",
            "<p>Click on the X to delete the album.</p>",
          ])
        : "<p>You do not have any albums. Please Create album.";
      var htmlTemplate = [
        "<h2>Albums</h2>",
        message,
        "<ul>",
        getHtml(albums),
        "</ul>",
        "<button onclick=\"createAlbum(prompt('Enter Album Name:'))\">",
        "Create New Album",
        "</button>",
      ];
      document.getElementById("app").innerHTML = getHtml(htmlTemplate);
    }
  });
}

function createAlbum(albumName) {
  albumName = albumName.trim();
  if (!albumName) {
    return alert("Album names must contain at least one non-space character.");
  }
  if (albumName.indexOf("/") !== -1) {
    return alert("Album names cannot contain slashes.");
  }
  var albumKey = encodeURIComponent(albumName);
```

```
  s3.headObject({ Key: albumKey }, function (err, data) {
    if (!err) {
      return alert("Album already exists.");
    }
    if (err.code !== "NotFound") {
      return alert("There was an error creating your album: " + err.message);
    }
    s3.putObject({ Key: albumKey }, function (err, data) {
      if (err) {
        return alert("There was an error creating your album: " + err.message);
      }
      alert("Successfully created album.");
      viewAlbum(albumName);
    });
  });
}

function viewAlbum(albumName) {
  var albumPhotosKey = encodeURIComponent(albumName) + "/";
  s3.listObjects({ Prefix: albumPhotosKey }, function (err, data) {
    if (err) {
      return alert("There was an error viewing your album: " + err.message);
    }
    // 'this' references the AWS.Response instance that represents the response
    var href = this.request.httpRequest.endpoint.href;
    var bucketUrl = href + albumBucketName + "/";

    var photos = data.Contents.map(function (photo) {
      var photoKey = photo.Key;
      var photoUrl = bucketUrl + encodeURIComponent(photoKey);
      return getHtml([
        "<span>",
        "<div>",
        '<img style="width:128px;height:128px;" src="' + photoUrl + '"/>',
        "</div>",
        "<div>",
        "<span onclick=\"deletePhoto('" +
          albumName +
          "','" +
          photoKey +
          "')\">",
        "X",
        "</span>",
        "<span>",
```

```
        photoKey.replace(albumPhotosKey, ""),
        "</span>",
        "</div>",
        "</span>",
      ]);
    });
    var message = photos.length
      ? "<p>Click on the X to delete the photo</p>"
      : "<p>You do not have any photos in this album. Please add photos.</p>";
    var htmlTemplate = [
      "<h2>",
      "Album: " + albumName,
      "</h2>",
      message,
      "<div>",
      getHtml(photos),
      "</div>",
      '<input id="photoupload" type="file" accept="image/*">',
      '<button id="addphoto" onclick="addPhoto(\'' + albumName + "')\">",
      "Add Photo",
      "</button>",
      '<button onclick="listAlbums()">',
      "Back To Albums",
      "</button>",
    ];
    document.getElementById("app").innerHTML = getHtml(htmlTemplate);
  });
}

function addPhoto(albumName) {
  var files = document.getElementById("photoupload").files;
  if (!files.length) {
    return alert("Please choose a file to upload first.");
  }
  var file = files[0];
  var fileName = file.name;
  var albumPhotosKey = encodeURIComponent(albumName) + "/";

  var photoKey = albumPhotosKey + fileName;

  // Use S3 ManagedUpload class as it supports multipart uploads
  var upload = new AWS.S3.ManagedUpload({
    params: {
      Bucket: albumBucketName,
```

```
      Key: photoKey,
      Body: file,
    },
  });

  var promise = upload.promise();

  promise.then(
    function (data) {
      alert("Successfully uploaded photo.");
      viewAlbum(albumName);
    },
    function (err) {
      return alert("There was an error uploading your photo: ", err.message);
    }
  );
}

function deletePhoto(albumName, photoKey) {
  s3.deleteObject({ Key: photoKey }, function (err, data) {
    if (err) {
      return alert("There was an error deleting your photo: ", err.message);
    }
    alert("Successfully deleted photo.");
    viewAlbum(albumName);
  });
}

function deleteAlbum(albumName) {
  var albumKey = encodeURIComponent(albumName) + "/";
  s3.listObjects({ Prefix: albumKey }, function (err, data) {
    if (err) {
      return alert("There was an error deleting your album: ", err.message);
    }
    var objects = data.Contents.map(function (object) {
      return { Key: object.Key };
    });
    s3.deleteObjects(
      {
        Delete: { Objects: objects, Quiet: true },
      },
      function (err, data) {
        if (err) {
          return alert("There was an error deleting your album: ", err.message);
```

```
        }
        alert("Successfully deleted album.");
        listAlbums();
      }
    );
  });
}
```

您可以在這裡找到此範例程式碼 GitHub。

Amazon S3 的例子 Node.js

下列主題顯示如何使用 Node.js 與 Amazon S3 儲存貯體互動的範例。 AWS SDK for JavaScript

主題

- 建立與使用 Amazon S3 儲存貯體
- 設定 Amazon S3 儲存貯體
- 管理 Amazon S3 儲存貯體存取許可
- 處理 Amazon S3 儲存貯體政策
- 使用 Amazon S3 儲存貯體做為靜態 Web 主機

建立與使用 Amazon S3 儲存貯體

這個 Node.js 程式碼範例會說明：

- 如何在您的帳戶中取得並顯示 Amazon S3 儲存貯體的清單。
- 如何建立 Amazon S3 儲存貯體。
- 如何上傳物件至指定的儲存貯體。

使用案例

在此範例中，使用一系列 Node.js 模組來取得現有 Amazon S3 儲存貯體的清單、建立儲存貯體，以及將檔案上傳到指定的儲存貯體。這些 Node.js 模組使用開發套件，使用 JavaScript Amazon S3 用戶端類別的下列方法，從 Amazon S3 儲存貯體取得資訊，並將檔案上傳到 Amazon S3 儲存貯體：

- `listBuckets`
- `createBucket`
- `listObjects`
- `upload`
- `deleteBucket`

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：

- 安裝 Node.js。如需安裝 Node.js 的詳細資訊，請參閱 Node.js 網站。
- 透過使用者登入資料建立共用組態檔。如需提供共用登入資料檔案的詳細資訊，請參閱 從共用登入資料檔案中在 Node.js 中載入登入資料。

設定軟體開發套件

JavaScript 通過創建全局配置對象，然後為代碼設置區域來配置 SDK。在此範例中，區域會設為 `us-west-2`。

```
// Load the SDK for JavaScript
var AWS = require('aws-sdk');
// Set the Region
AWS.config.update({region: 'us-west-2'});
```

顯示 Amazon S3 存儲桶列表

以檔名 `s3_listbuckets.js` 建立一個 Node.js 模組。請務必依前述的內容來設定軟體開發套件。要訪問 Amazon 簡單存儲服務，請創建一個`AWS.S3`服務對象。呼叫 Amazon S3 服務物件的`listBuckets`方法以擷取儲存貯體清單。回呼函數的 `data` 參數具有一個包含映射陣列的 `Buckets` 屬性以代表儲存貯體。透過將其記錄至主控台以顯示儲存貯體清單。

```
// Load the AWS SDK for Node.js
```

```
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create S3 service object
s3 = new AWS.S3({ apiVersion: "2006-03-01" });

// Call S3 to list the buckets
s3.listBuckets(function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.Buckets);
  }
});
```

若要執行範例，請在命令列中輸入以下內容。

```
node s3_listbuckets.js
```

您可以在這裡找到此範例程式碼 GitHub。

建立 Amazon S3 儲存貯體

以檔名 `s3_createbucket.js` 建立一個 Node.js 模組。請務必依前述的內容來設定軟體開發套件。建立一個 `AWS.S3` 服務物件。該模組將採用單一命令行引數以指定新儲存貯體的名稱。

新增變數以保存用於呼叫 Amazon S3 服務物件`createBucket`方法的參數，包括新建立儲存貯體的名稱。Amazon S3 成功建立新儲存貯體後，回呼函數會將新儲存貯體的位置記錄到主控台。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create S3 service object
s3 = new AWS.S3({ apiVersion: "2006-03-01" });

// Create the parameters for calling createBucket
var bucketParams = {
  Bucket: process.argv[2],
};
```

```
// call S3 to create the bucket
s3.createBucket(bucketParams, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.Location);
  }
});
```

若要執行範例，請在命令列中輸入以下內容。

```
node s3_createbucket.js BUCKET_NAME
```

您可以在這裡找到此範例程式碼 GitHub。

將文件上傳到 Amazon S3 存儲桶

以檔名 s3_upload.js 建立一個 Node.js 模組。請務必依前述的內容來設定軟體開發套件。建立一個 AWS.S3 服務物件。該模組將採用兩條命令行引數，第一個用來指定目的地儲存貯體，第二個指定要上傳的檔案。

使用呼叫 Amazon S3 服務物件方upload法所需的參數建立變數。在 Bucket 參數中提供目標儲存貯體的名稱。Key 參數設為所選檔案的名稱，您可使用 Node.js path 模組來取得該名稱。Body 參數設為該檔案的內容，您可使用 Node.js fs 模組的 createReadStream 來取得該內容。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create S3 service object
var s3 = new AWS.S3({ apiVersion: "2006-03-01" });

// call S3 to retrieve upload file to specified bucket
var uploadParams = { Bucket: process.argv[2], Key: "", Body: "" };
var file = process.argv[3];

// Configure the file stream and obtain the upload parameters
var fs = require("fs");
var fileStream = fs.createReadStream(file);
fileStream.on("error", function (err) {
```

```
  console.log("File Error", err);
});
uploadParams.Body = fileStream;
var path = require("path");
uploadParams.Key = path.basename(file);

// call S3 to retrieve upload file to specified bucket
s3.upload(uploadParams, function (err, data) {
  if (err) {
    console.log("Error", err);
  }
  if (data) {
    console.log("Upload Success", data.Location);
  }
});
```

若要執行範例，請在命令列中輸入以下內容。

```
node s3_upload.js BUCKET_NAME FILE_NAME
```

您可以在這裡找到此範例程式碼 GitHub。

在 Amazon S3 存儲桶中列出對象

以檔名 `s3_listobjects.js` 建立一個 Node.js 模組。請務必依前述的內容來設定軟體開發套件。建立一個 `AWS.S3` 服務物件。

新增變數以保存用於呼叫 Amazon S3 服務物件`listObjects`方法的參數，包括要讀取的儲存貯體名稱。回呼函數會記錄物件 (檔案) 清單或故障訊息。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create S3 service object
s3 = new AWS.S3({ apiVersion: "2006-03-01" });

// Create the parameters for calling listObjects
var bucketParams = {
  Bucket: "BUCKET_NAME",
};
```

```
// Call S3 to obtain a list of the objects in the bucket
s3.listObjects(bucketParams, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

若要執行範例，請在命令列中輸入以下內容。

```
node s3_listobjects.js
```

您可以在這裡找到此範例程式碼 GitHub。

刪除 Amazon S3 存儲桶

以檔名 `s3_deletebucket.js` 建立一個 Node.js 模組。請務必依前述的內容來設定軟體開發套件。建立一個 `AWS.S3` 服務物件。

新增變數以保存用於呼叫 Amazon S3 服務物件`createBucket`方法的參數，包括要刪除的儲存貯體名稱。儲存貯體必須為空後始可將其刪除。回呼函數會記錄成功或故障訊息。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create S3 service object
s3 = new AWS.S3({ apiVersion: "2006-03-01" });

// Create params for S3.deleteBucket
var bucketParams = {
  Bucket: "BUCKET_NAME",
};

// Call S3 to delete the bucket
s3.deleteBucket(bucketParams, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
```

```
    console.log("Success", data);
  }
});
```

若要執行範例，請在命令列中輸入以下內容。

```
node s3_deletebucket.js
```

您可以在這裡找到此範例程式碼 GitHub。

設定 Amazon S3 儲存貯體

這個 Node.js 程式碼範例會說明：

- 如何設定儲存貯體的跨來源資源共享 (CORS) 許可。

使用案例

在此範例中使用了一系列的 Node.js 模組以列出您的 Amazon S3 儲存貯體，以及設定 CORS 和儲存貯體記錄。Node.js 模組使用開發套件，使用 Amazon S3 用戶端類別的下列方法 JavaScript 來設定選取的 Amazon S3 儲存貯體：

- `getBucketCors`
- `putBucketCors`

如需將 CORS 組態與 Amazon S3 儲存貯體搭配使用的詳細資訊，請參閱 Amazon 簡單儲存服務使用者指南中的跨來源資源共用 (CORS)。

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：

- 安裝 Node.js。如需安裝 Node.js 的詳細資訊，請參閱 Node.js 網站。
- 透過使用者登入資料建立共用組態檔。如需提供共用登入資料檔案的詳細資訊，請參閱 從共用登入資料檔案中在 Node.js 中載入登入資料。

設定軟體開發套件

JavaScript 通過創建全局配置對象，然後為代碼設置區域來配置 SDK。在此範例中，區域會設為 us-west-2。

```
// Load the SDK for JavaScript
var AWS = require('aws-sdk');
// Set the Region
AWS.config.update({region: 'us-west-2'});
```

擷取儲存貯體 CORS 組態

以檔名 s3_getcors.js 建立一個 Node.js 模組。該模組將採用單一命令行引數來指定您需要的 CORS 組態之儲存貯體。請務必依前述的內容來設定軟體開發套件。建立一個 AWS.S3 服務物件。

您唯一需要傳遞的參數，就是在呼叫 getBucketCors 方法時所選取儲存貯體的名稱。如果儲存貯體目前具有 CORS 組態，Amazon S3 會傳回該組態，做為傳遞至回呼函數的data參數CORSRules屬性。

如果所選的儲存貯體沒有 CORS 組態，則該資訊會在 error 參數中傳回至回呼函數。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create S3 service object
s3 = new AWS.S3({ apiVersion: "2006-03-01" });

// Set the parameters for S3.getBucketCors
var bucketParams = { Bucket: process.argv[2] };

// call S3 to retrieve CORS configuration for selected bucket
s3.getBucketCors(bucketParams, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else if (data) {
    console.log("Success", JSON.stringify(data.CORSRules));
  }
});
```

若要執行範例，請在命令列中輸入以下內容。

```
node s3_getcors.js BUCKET_NAME
```

您可以在這裡找到此範例程式碼 GitHub。

設定儲存貯體 CORS 組態

以檔名 s3_setcors.js 建立一個 Node.js 模組。該模組採用多個命令行引數，第一個指定您想設定的 CORS 組態之儲存貯體。其他引數列舉您要為儲存貯體允許的 HTTP 方法 (POST, GET, PUT, PATCH, DELETE, POST)。依前述內容設定軟體開發套件。

建立一個 AWS.S3 服務物件。接下來，視 AWS.S3 服務物件的 putBucketCors 方法需要，建立一個 JSON 物件以保存 CORS 組態的值。為 AllowedHeaders 值指定 "Authorization"，並為 AllowedOrigins 值指定 "*"。在初始時設 AllowedMethods 值為空陣列。

指定允許的方法為命令列參數至 Node.js 模組，新增符合其中一個參數的各種方法。新增產生的 CORS 組態至包含在 CORSRules 參數中的組態陣列。在 Bucket 參數中指定您要為 CORS 設定的儲存貯體。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create S3 service object
s3 = new AWS.S3({ apiVersion: "2006-03-01" });

// Create initial parameters JSON for putBucketCors
var thisConfig = {
  AllowedHeaders: ["Authorization"],
  AllowedMethods: [],
  AllowedOrigins: ["*"],
  ExposeHeaders: [],
  MaxAgeSeconds: 3000,
};

// Assemble the list of allowed methods based on command line parameters
var allowedMethods = [];
process.argv.forEach(function (val, index, array) {
  if (val.toUpperCase() === "POST") {
    allowedMethods.push("POST");
  }
  if (val.toUpperCase() === "GET") {
```

```
    allowedMethods.push("GET");
  }
  if (val.toUpperCase() === "PUT") {
    allowedMethods.push("PUT");
  }
  if (val.toUpperCase() === "PATCH") {
    allowedMethods.push("PATCH");
  }
  if (val.toUpperCase() === "DELETE") {
    allowedMethods.push("DELETE");
  }
  if (val.toUpperCase() === "HEAD") {
    allowedMethods.push("HEAD");
  }
});

// Copy the array of allowed methods into the config object
thisConfig.AllowedMethods = allowedMethods;
// Create array of configs then add the config object to it
var corsRules = new Array(thisConfig);

// Create CORS params
var corsParams = {
  Bucket: process.argv[2],
  CORSConfiguration: { CORSRules: corsRules },
};

// set the new CORS configuration on the selected bucket
s3.putBucketCors(corsParams, function (err, data) {
  if (err) {
    // display error message
    console.log("Error", err);
  } else {
    // update the displayed CORS config for the selected bucket
    console.log("Success", data);
  }
});
```

若要執行範例，請在命令列中輸入以下內容 (包含所示的一個或更多 HTTP 方法)。

```
node s3_setcors.js BUCKET_NAME get put
```

您可以在這裡找到此範例程式碼 GitHub。

管理 Amazon S3 儲存貯體存取許可

這個 Node.js 程式碼範例會說明：

- 如何擷取或設定 Amazon S3 儲存貯體的存取控制清單。

使用案例

在此範例中，Node.js 模組用於顯示所選儲存貯體的儲存貯體存取控制清單 (ACL)，並套用變更至所選儲存貯體的 ACL。Node.js 模組使用開發套件，使用 Amazon S3 用戶端類別的下列方法 JavaScript 來管理 Amazon S3 儲存貯體存取許可：

- `getBucketAcl`
- `putBucketAcl`

如需 Amazon S3 儲存貯體存取控制清單的詳細資訊，請參閱 Amazon 簡單儲存服務使用者指南中的使用 ACL 管理存取。

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：

- 安裝 Node.js。如需安裝 Node.js 的詳細資訊，請參閱 Node.js 網站。
- 透過使用者登入資料建立共用組態檔。如需提供共用登入資料檔案的詳細資訊，請參閱 從共用登入資料檔案中在 Node.js 中載入登入資料。

設定軟體開發套件

JavaScript 通過創建全局配置對象，然後為代碼設置區域來配置 SDK。在此範例中，區域會設為 `us-west-2`。

```
// Load the SDK for JavaScript
var AWS = require('aws-sdk');
```

```
// Set the Region
AWS.config.update({region: 'us-west-2'});
```

擷取現有儲存貯體存取控制清單

以檔名 s3_getbucketacl.js 建立一個 Node.js 模組。該模組將採用單一命令行引數來指定您需要的 ACL 組態之儲存貯體。請務必依前述的內容來設定軟體開發套件。

建立一個 AWS.S3 服務物件。您唯一需要傳遞的參數，就是在呼叫 getBucketAcl 方法時所選取儲存貯體的名稱。Amazon S3 會在傳遞至回呼函數的data參數中傳回目前的存取控制清單組態。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create S3 service object
s3 = new AWS.S3({ apiVersion: "2006-03-01" });

var bucketParams = { Bucket: process.argv[2] };
// call S3 to retrieve policy for selected bucket
s3.getBucketAcl(bucketParams, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else if (data) {
    console.log("Success", data.Grants);
  }
});
```

若要執行範例，請在命令列中輸入以下內容。

```
node s3_getbucketacl.js BUCKET_NAME
```

您可以在這裡找到此範例程式碼 GitHub。

處理 Amazon S3 儲存貯體政策

這個 Node.js 程式碼範例會說明：

- 如何擷取 Amazon S3 儲存貯體的儲存貯體政策。
- 如何新增或更新 Amazon S3 儲存貯體的儲存貯體政策。
- 如何刪除 Amazon S3 儲存貯體的儲存貯體政策。

使用案例

在此範例中，使用一系列 Node.js 模組來擷取、設定或刪除 Amazon S3 儲存貯體上的儲存貯體政策。Node.js 模組使用的開發套件，使用 JavaScript Amazon S3 用戶端類別的下列方法，為選定的 Amazon S3 儲存貯體設定政策：

- `getBucketPolicy`
- `putBucketPolicy`
- `deleteBucketPolicy`

如需 Amazon S3 儲存貯體政策的詳細資訊，請參閱 Amazon 簡單儲存貯體服務使用者指南中的使用儲存貯體政策和使用者政策。

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：

- 安裝 Node.js。如需安裝 Node.js 的詳細資訊，請參閱 Node.js 網站。
- 透過使用者登入資料建立共用組態檔。如需提供共用登入資料檔案的詳細資訊，請參閱 從共用登入資料檔案中在 Node.js 中載入登入資料。

設定軟體開發套件

JavaScript 通過創建全局配置對象，然後為代碼設置區域來配置 SDK。在此範例中，區域會設為 `us-west-2`。

```
// Load the SDK for JavaScript
var AWS = require('aws-sdk');
// Set the Region
AWS.config.update({region: 'us-west-2'});
```

擷取目前儲存貯體政策

以檔名 s3_getbucketpolicy.js 建立一個 Node.js 模組。該模組採用指定您想要政策的儲存貯體之單一命令行引數。請務必依前述的內容來設定軟體開發套件。

建立一個 AWS.S3 服務物件。您唯一需要傳遞的參數，就是在呼叫 getBucketPolicy 方法時所選取儲存貯體的名稱。如果儲存貯體目前有政策，Amazon S3 會在傳遞至回呼函data數的參數中傳回該政策。

如果所選的儲存貯體沒有政策，則該資訊會在 error 參數中傳回至回呼函數。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create S3 service object
s3 = new AWS.S3({ apiVersion: "2006-03-01" });

var bucketParams = { Bucket: process.argv[2] };
// call S3 to retrieve policy for selected bucket
s3.getBucketPolicy(bucketParams, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else if (data) {
    console.log("Success", data.Policy);
  }
});
```

若要執行範例，請在命令列中輸入以下內容。

```
node s3_getbucketpolicy.js BUCKET_NAME
```

您可以在這裡找到此範例程式碼 GitHub。

設定簡易儲存貯體政策

以檔名 s3_setbucketpolicy.js 建立一個 Node.js 模組。該模組採用的單一命令行引數，指定了您想套用政策的儲存貯體。依前述內容設定軟體開發套件。

建立一個 AWS.S3 服務物件。儲存貯體政策是在 JSON 中指定。首先，建立包含所有值 (識別儲存貯體的 Resource 值除外) 的 JSON 物件，以指定政策。

格式化政策所需的 Resource 字串，整合該所選的儲存貯體名稱。將字串插入至 JSON 物件。準備該參數以供 putBucketPolicy 方法使用，包括儲存貯體名稱和轉換為字串值的 JSON 政策。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create S3 service object
s3 = new AWS.S3({ apiVersion: "2006-03-01" });

var readOnlyAnonUserPolicy = {
  Version: "2012-10-17",
  Statement: [
    {
      Sid: "AddPerm",
      Effect: "Allow",
      Principal: "*",
      Action: ["s3:GetObject"],
      Resource: [""],
    },
  ],
};

// create selected bucket resource string for bucket policy
var bucketResource = "arn:aws:s3:::" + process.argv[2] + "/*";
readOnlyAnonUserPolicy.Statement[0].Resource[0] = bucketResource;

// convert policy JSON into string and assign into params
var bucketPolicyParams = {
  Bucket: process.argv[2],
  Policy: JSON.stringify(readOnlyAnonUserPolicy),
};

// set the new policy on the selected bucket
s3.putBucketPolicy(bucketPolicyParams, function (err, data) {
  if (err) {
    // display error message
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

若要執行範例，請在命令列中輸入以下內容。

```
node s3_setbucketpolicy.js BUCKET_NAME
```

您可以在這裡找到此範例程式碼 GitHub。

刪除儲存貯體政策

以檔名 s3_deletebucketpolicy.js 建立一個 Node.js 模組。該模組採用的單一命令行引數，指定了您想刪除的政策之儲存貯體。依前述內容設定軟體開發套件。

建立一個 AWS.S3 服務物件。您在呼叫 deleteBucketPolicy 方法時唯一需要傳遞的參數，就是所選取儲存貯體的名稱。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create S3 service object
s3 = new AWS.S3({ apiVersion: "2006-03-01" });

var bucketParams = { Bucket: process.argv[2] };
// call S3 to delete policy for selected bucket
s3.deleteBucketPolicy(bucketParams, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else if (data) {
    console.log("Success", data);
  }
});
```

若要執行範例，請在命令列中輸入以下內容。

```
node s3_deletebucketpolicy.js BUCKET_NAME
```

您可以在這裡找到此範例程式碼 GitHub。

使用 Amazon S3 儲存貯體做為靜態 Web 主機

這個 Node.js 程式碼範例會說明：

- 如何將 Amazon S3 儲存貯體設定為靜態 Web 主機。

使用案例

在此範例中使用一系列的 Node.js 模組以設定任何儲存貯體，並該儲存貯體做為靜態 web 託管。Node.js 模組使用開發套件，使用 Amazon S3 用戶端類別的下列方法 JavaScript 來設定選取的 Amazon S3 儲存貯體：

- `getBucketWebsite`
- `putBucketWebsite`
- `deleteBucketWebsite`

如需使用 Amazon S3 儲存貯體做為靜態 Web 主機的詳細資訊，請參閱 Amazon 簡單儲存服務使用者指南中的在 Amazon S3 上託管靜態網站。

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：

- 安裝 Node.js。如需安裝 Node.js 的詳細資訊，請參閱 Node.js 網站。
- 透過使用者登入資料建立共用組態檔。如需提供共用登入資料檔案的詳細資訊，請參閱 從共用登入資料檔案中在 Node.js 中載入登入資料。

設定軟體開發套件

JavaScript 通過創建全局配置對象，然後為代碼設置區域來配置 SDK。在此範例中，區域會設為 `us-west-2`。

```
// Load the SDK for JavaScript
```

```
var AWS = require('aws-sdk');
// Set the Region
AWS.config.update({region: 'us-west-2'});
```

擷取目前的儲存貯體網站組態

以檔名 s3_getbucketwebsite.js 建立一個 Node.js 模組。該模組採用的單一命令行引數，指定了您要的網站組態之儲存貯體。依前述內容設定軟體開發套件。

建立一個 AWS.S3 服務物件。建立函數，對於儲存貯體清單中所選的儲存貯體，擷取其目前的儲存貯體網站組態。您唯一需要傳遞的參數，就是在呼叫 getBucketWebsite 方法時所選取儲存貯體的名稱。如果儲存貯體目前具有網站組態，Amazon S3 會在傳遞至回呼函data數的參數中傳回該組態。

如果所選的儲存貯體沒有網站組態，則該資訊會在 err 參數中傳回至回呼函數。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create S3 service object
s3 = new AWS.S3({ apiVersion: "2006-03-01" });

var bucketParams = { Bucket: process.argv[2] };

// call S3 to retrieve the website configuration for selected bucket
s3.getBucketWebsite(bucketParams, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else if (data) {
    console.log("Success", data);
  }
});
```

若要執行範例，請在命令列中輸入以下內容。

```
node s3_getbucketwebsite.js BUCKET_NAME
```

您可以在這裡找到此範例程式碼 GitHub。

設定儲存貯體網站組態

以檔名 s3_setbucketwebsite.js 建立一個 Node.js 模組。請務必依前述的內容來設定軟體開發套件。建立一個 AWS.S3 服務物件。

建立套用至儲存貯體網站組態的函數。該組態允許所選之儲存貯體用作靜態 web 託管。網站組態於 JSON 中指定。首先，建立包含所有值 (識別錯誤文件的 Key 值和識別索引文件的 Suffix 值除外) 的 JSON 物件以指定網站組態。

插入文字輸入元素的值至 JSON 物件中。準備該參數以供 putBucketWebsite 方法使用，包括儲存貯體名稱和 JSON 網站組態。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create S3 service object
s3 = new AWS.S3({ apiVersion: "2006-03-01" });

// Create JSON for putBucketWebsite parameters
var staticHostParams = {
  Bucket: "",
  WebsiteConfiguration: {
    ErrorDocument: {
      Key: "",
    },
    IndexDocument: {
      Suffix: "",
    },
  },
};

// Insert specified bucket name and index and error documents into params JSON
// from command line arguments
staticHostParams.Bucket = process.argv[2];
staticHostParams.WebsiteConfiguration.IndexDocument.Suffix = process.argv[3];
staticHostParams.WebsiteConfiguration.ErrorDocument.Key = process.argv[4];

// set the new website configuration on the selected bucket
s3.putBucketWebsite(staticHostParams, function (err, data) {
  if (err) {
    // display error message
```

```
    console.log("Error", err);
  } else {
    // update the displayed website configuration for the selected bucket
    console.log("Success", data);
  }
});
```

若要執行範例，請在命令列中輸入以下內容。

```
node s3_setbucketwebsite.js BUCKET_NAME INDEX_PAGE ERROR_PAGE
```

您可以在這裡找到此範例程式碼 GitHub。

刪除儲存貯體網站組態

以檔名 s3_deletebucketwebsite.js 建立一個 Node.js 模組。請務必依前述的內容來設定軟體開發套件。建立一個 AWS.S3 服務物件。

建立刪除所選儲存貯體網站組態的函數。您在呼叫 deleteBucketWebsite 方法時唯一需要傳遞的參數，就是所選取儲存貯體的名稱。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create S3 service object
s3 = new AWS.S3({ apiVersion: "2006-03-01" });

var bucketParams = { Bucket: process.argv[2] };

// call S3 to delete website configuration for selected bucket
s3.deleteBucketWebsite(bucketParams, function (error, data) {
  if (error) {
    console.log("Error", err);
  } else if (data) {
    console.log("Success", data);
  }
});
```

若要執行範例，請在命令列中輸入以下內容。

```
node s3_deletebucketwebsite.js BUCKET_NAME
```

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 Simple Email Service (Amazon SES) 是雲端型電子郵件傳送服務，旨在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。

Amazon SES 的 JavaScript API 是透過AWS.SES用戶端類別公開。如需使用 Amazon SES 用戶端類別的詳細資訊，請參閱 API 參考資料Class: AWS.SES中的。

主題

- 管理 Amazon SES 身分
- 在 Amazon SES 中使用電子郵件模板
- 發送電子郵件使用 Amazon SES
- 在 Amazon SES 中使用 IP 位址篩選器進行電子郵件收據
- 在 Amazon SES 中使用接收規則

管理 Amazon SES 身分

這個 Node.js 程式碼範例會說明：

- 如何驗證搭配 Amazon SES 使用的電子郵件地址和網域。
- 如何為您的 Amazon SES 身分指派 IAM 政策。
- 如何列出您AWS帳戶的所有 Amazon SES 身份。
- 如何刪除與 Amazon SES 一起使用的身份。

Amazon SES 身分識別是 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。

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

使用案例

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

- `listIdentities`
- `deleteIdentity`
- `verifyEmailIdentity`
- `verifyDomainIdentity`

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：

- 安裝 Node.js。如需安裝 Node.js 的詳細資訊，請參閱 Node.js 網站。
- 透過使用者登入資料建立共用組態檔。如需有關提供登入資料 JSON 檔案的詳細資訊，請參閱 從共用登入資料檔案中在 Node.js 中載入登入資料。

設定軟體開發套件

JavaScript 通過創建全局配置對象，然後為代碼設置區域來配置 SDK。在此範例中，區域會設為 `us-west-2`。

```
// Load the SDK for JavaScript
var AWS = require('aws-sdk');

// Set the Region
AWS.config.update({region: 'us-west-2'});
```

列出您的身分

在此範例中，使用 Node.js 模組列出要搭配 Amazon SES 使用的電子郵件地址和網域。以檔名 `ses_listidentities.js` 建立一個 Node.js 模組。依前述內容設定軟體開發套件。

建立物件，以傳遞 AWS.SES 用戶端類別的 `listIdentities` 方法之 IdentityType 和其他參數。若要呼叫方`listIdentities`法，請建立用於叫用 Amazon SES 服務物件並傳遞參數物件的承諾。

然後，在 promise 回呼中處理 `response`。由 promise 傳回的 `data` 包含了一組網域身分的陣列，如 `IdentityType` 參數中所指定。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set region
AWS.config.update({ region: "REGION" });

// Create listIdentities params
var params = {
  IdentityType: "Domain",
  MaxItems: 10,
};

// Create the promise and SES service object
var listIDsPromise = new AWS.SES({ apiVersion: "2010-12-01" })
  .listIdentities(params)
  .promise();

// Handle promise's fulfilled/rejected states
listIDsPromise
  .then(function (data) {
    console.log(data.Identities);
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

若要執行範例，請在命令列中輸入以下內容。

```
node ses_listidentities.js
```

您可以在這裡找到此範例程式碼 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 SES 搭配使用的電子郵件寄件者。以檔名 `ses_verifyemailidentity.js` 建立一個 Node.js 模組。依前述內容設定軟體開發套件。若要存取 Amazon SES，請建立`AWS.SES`服務物件。

建立物件以傳遞 `AWS.SES` 用戶端類別的 `verifyEmailIdentity` 方法之 `EmailAddress` 參數。若要呼叫 `verifyEmailIdentity` 方法，請建立叫用 Amazon SES 服務物件的 promise 來傳遞參數。然後，在 promise 回呼中處理 `response`。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set region
AWS.config.update({ region: "REGION" });

// Create promise and SES service object
var verifyEmailPromise = new AWS.SES({ apiVersion: "2010-12-01" })
  .verifyEmailIdentity({ EmailAddress: "ADDRESS@DOMAIN.EXT" })
  .promise();

// Handle promise's fulfilled/rejected states
verifyEmailPromise
  .then(function (data) {
    console.log("Email verification initiated");
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

若要執行範例，請在命令列中輸入以下內容。網域已新增至 Amazon SES 以進行驗證。

```
node ses_verifyemailidentity.js
```

您可以在這裡找到此範例程式碼 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 SES 搭配使用的電子郵件網域。以檔名 `ses_verifydomainidentity.js` 建立一個 Node.js 模組。依前述內容設定軟體開發套件。

建立物件以傳遞 `AWS.SES` 用戶端類別的 `verifyDomainIdentity` 方法之 `Domain` 參數。若要呼叫方`verifyDomainIdentity`法，請建立用於叫用 Amazon SES 服務物件並傳遞參數物件的承諾。然後，在 promise 回呼中處理 `response`。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set region
AWS.config.update({ region: "REGION" });

// Create the promise and SES service object
var verifyDomainPromise = new AWS.SES({ apiVersion: "2010-12-01" })
  .verifyDomainIdentity({ Domain: "DOMAIN_NAME" })
  .promise();

// Handle promise's fulfilled/rejected states
verifyDomainPromise
  .then(function (data) {
    console.log("Verification Token: " + data.VerificationToken);
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

若要執行範例，請在命令列中輸入以下內容。網域已新增至 Amazon SES 以進行驗證。

```
node ses_verifydomainidentity.js
```

您可以在這裡找到此範例程式碼 GitHub。

刪除身分

在此範例中，使用 Node.js 模組刪除與 Amazon SES 搭配使用的電子郵件地址或網域。以檔名 `ses_deleteidentity.js` 建立一個 Node.js 模組。依前述內容設定軟體開發套件。

建立物件以傳遞 `AWS.SES` 用戶端類別的 `deleteIdentity` 方法之 `Identity` 參數。要調用該`deleteIdentity`方法，請創建一個用`request`於調用 Amazon SES 服務對象，並傳遞參數。然後，在 promise 回呼中處理 `response`。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set region
AWS.config.update({ region: "REGION" });

// Create the promise and SES service object
var deletePromise = new AWS.SES({ apiVersion: "2010-12-01" })
  .deleteIdentity({ Identity: "DOMAIN_NAME" })
  .promise();

// Handle promise's fulfilled/rejected states
deletePromise
  .then(function (data) {
    console.log("Identity Deleted");
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

若要執行範例，請在命令列中輸入以下內容。

```
node ses_deleteidentity.js
```

您可以在這裡找到此範例程式碼 GitHub。

在 Amazon SES 中使用電子郵件模板

這個 Node.js 程式碼範例會說明：

- 取得您所有的電子郵件範本清單。
- 擷取並更新電子郵件範本。
- 建立並刪除電子郵件範本。

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

使用案例

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 模組使用 SDK JavaScript 來建立和使用用AWS.SES戶端類別的下列方法的電子郵件範本：

- listTemplates
- createTemplate
- getTemplate
- deleteTemplate
- updateTemplate

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：

- 安裝 Node.js。如需安裝 Node.js 的詳細資訊，請參閱 Node.js 網站。
- 透過使用者登入資料建立共用組態檔。如需建立登入資料檔案的詳細資訊，請參閱 從共用登入資料檔案中在 Node.js 中載入登入資料。

列出您的電子郵件範本

在此範例中，使用 Node.js 模組建立電子郵件範本以搭配 Amazon SES 使用。以檔名 `ses_listtemplates.js` 建立一個 Node.js 模組。依前述內容設定軟體開發套件。

建立物件以傳遞 `AWS.SES` 用戶端類別的 `listTemplates` 方法之參數。若要呼叫 `listTemplates` 方法，請建立叫用 Amazon SES 服務物件的 promise 來傳遞參數。然後，在 promise 回呼中處理 `response`。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the promise and SES service object
```

```
var templatePromise = new AWS.SES({ apiVersion: "2010-12-01" })
  .listTemplates({ MaxItems: ITEMS_COUNT })
  .promise();

// Handle promise's fulfilled/rejected states
templatePromise
  .then(function (data) {
    console.log(data);
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

若要執行範例，請在命令列中輸入以下內容。Amazon SES 返回模板列表。

```
node ses_listtemplates.js
```

您可以在這裡找到此範例程式碼 GitHub。

取得電子郵件範本

在此範例中，使用 Node.js 模組取得要搭配 Amazon SES 使用的電子郵件範本。以檔名 `ses_gettemplate.js` 建立一個 Node.js 模組。依前述內容設定軟體開發套件。

建立物件以傳遞 `AWS.SES` 用戶端類別的 `getTemplate` 方法之 `TemplateName` 參數。若要呼叫 `getTemplate` 方法，請建立叫用 Amazon SES 服務物件的 promise 來傳遞參數。然後，在 promise 回呼中處理 `response`。

```
// Load the AWS SDK for Node.js.
var AWS = require("aws-sdk");
// Set the AWS Region.
AWS.config.update({ region: "REGION" });

// Create the promise and Amazon Simple Email Service (Amazon SES) service object.
var templatePromise = new AWS.SES({ apiVersion: "2010-12-01" })
  .getTemplate({ TemplateName: "TEMPLATE_NAME" })
  .promise();

// Handle promise's fulfilled/rejected states
templatePromise
  .then(function (data) {
    console.log(data.Template.SubjectPart);
```

```
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

若要執行範例，請在命令列中輸入以下內容。Amazon SES 返回模板詳細信息。

```
node ses_gettemplate.js
```

您可以在這裡找到此範例程式碼 GitHub。

建立電子郵件範本

在此範例中，使用 Node.js 模組建立電子郵件範本以搭配 Amazon SES 使用。以檔名 `ses_createtemplate.js` 建立一個 Node.js 模組。依前述內容設定軟體開發套件。

建立物件以傳遞 `AWS.SES` 用戶端類別的 `createTemplate` 方法 (包括 `TemplateName`、`HtmlPart`、`SubjectPart` 和 `TextPart`) 之參數。若要呼叫 `createTemplate` 方法，請建立叫用 Amazon SES 服務物件的 promise 來傳遞參數。然後，在 promise 回呼中處理 `response`。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create createTemplate params
var params = {
  Template: {
    TemplateName: "TEMPLATE_NAME" /* required */,
    HtmlPart: "HTML_CONTENT",
    SubjectPart: "SUBJECT_LINE",
    TextPart: "TEXT_CONTENT",
  },
};

// Create the promise and SES service object
var templatePromise = new AWS.SES({ apiVersion: "2010-12-01" })
  .createTemplate(params)
  .promise();

// Handle promise's fulfilled/rejected states
```

```
templatePromise
  .then(function (data) {
    console.log(data);
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

若要執行範例，請在命令列中輸入以下內容。該模板被添加到 Amazon SES。

```
node ses_createtemplate.js
```

您可以在這裡找到此範例程式碼 GitHub。

更新電子郵件範本

在此範例中，使用 Node.js 模組建立電子郵件範本以搭配 Amazon SES 使用。以檔名 `ses_updatetemplate.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，並搭配需要的 `TemplateName` 參數 (傳遞至 `AWS.SES` 用戶端類別的 `updateTemplate` 方法之參數)，以傳遞您要在範本中更新的 `Template` 參數值。若要呼叫 `updateTemplate` 方法，請建立叫用 Amazon SES 服務物件的 promise 來傳遞參數。然後，在 promise 回呼中處理 `response`。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create updateTemplate parameters
var params = {
  Template: {
    TemplateName: "TEMPLATE_NAME" /* required */,
    HtmlPart: "HTML_CONTENT",
    SubjectPart: "SUBJECT_LINE",
    TextPart: "TEXT_CONTENT",
  },
};

// Create the promise and SES service object
var templatePromise = new AWS.SES({ apiVersion: "2010-12-01" })
  .updateTemplate(params)
```

```
  .promise();

// Handle promise's fulfilled/rejected states
templatePromise
  .then(function (data) {
    console.log("Template Updated");
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

若要執行範例，請在命令列中輸入以下內容。Amazon SES 返回模板詳細信息。

```
node ses_updatetemplate.js
```

您可以在這裡找到此範例程式碼 GitHub。

刪除電子郵件範本

在此範例中，使用 Node.js 模組建立電子郵件範本以搭配 Amazon SES 使用。以檔名 `ses_deletetemplate.js` 建立一個 Node.js 模組。依前述內容設定軟體開發套件。

建立物件以傳遞需要的 `TemplateName` 參數至 `AWS.SES` 用戶端類別的 `deleteTemplate` 方法。若要呼叫 `deleteTemplate` 方法，請建立叫用 Amazon SES 服務物件的 promise 來傳遞參數。然後，在 promise 回呼中處理 `response`。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the promise and SES service object
var templatePromise = new AWS.SES({ apiVersion: "2010-12-01" })
  .deleteTemplate({ TemplateName: "TEMPLATE_NAME" })
  .promise();

// Handle promise's fulfilled/rejected states
templatePromise
  .then(function (data) {
    console.log("Template Deleted");
  })
  .catch(function (err) {
```

```
    console.error(err, err.stack);
  });
```

若要執行範例，請在命令列中輸入以下內容。Amazon SES 返回模板詳細信息。

```
node ses_deletetemplate.js
```

您可以在這裡找到此範例程式碼 GitHub。

發送電子郵件使用 Amazon SES

這個 Node.js 程式碼範例會說明：

- 傳送文字或 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 提供兩種不同的方式供您傳送電子郵件，這取決於您要對電子郵件訊息組成的控制程度：格式化和原始。如需詳細資訊，請參閱使用 Amazon SES API 傳送格式化電子郵件和使用 Amazon SES API 傳送原始電子郵件。

使用案例

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 模組使用 SDK JavaScript 來建立和使用用`AWS.SES`戶端類別的下列方法的電子郵件範本：

- `sendEmail`
- `sendTemplatedEmail`
- `sendBulkTemplatedEmail`

先決條件任務

- 安裝 Node.js。如需安裝 Node.js 的詳細資訊，請參閱 Node.js 網站。

- 透過使用者登入資料建立共用組態檔。如需有關提供登入資料 JSON 檔案的詳細資訊，請參閱 從共用登入資料檔案中在 Node.js 中載入登入資料。

電子郵件訊息傳送要求

Amazon SES 會撰寫電子郵件訊息，並立即將其排入佇列以進行傳送。若要使用 `SES.sendEmail` 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 沙盒中，您只能傳送至驗證的地址或網域，或傳送至與 Amazon SES 信箱模擬器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。收件人地址可為 To (收件人)：地址、CC (副本)：地址或 BCC (密件副本)：地址。若收件人電子郵件地址無效 (即不為 `UserName@[SubDomain.]Domain.TopLevelDomain` 格式)，即使該訊息包含其他有效的收件人，則整個訊息都將被拒絕。
- 該訊息的 To: (收件人)、CC: (副本)、和 BCC: (密件副本) 的收件人總和不得多於 50 名。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。

傳送電子郵件

在此範例中，使用 Node.js 模組以搭配 Amazon SES 傳送電子郵件。以檔名 `ses_sendemail.js` 建立一個 Node.js 模組。依前述內容設定軟體開發套件。

建立一個物件以傳遞定義欲傳送電子郵件的參數值，包括寄件者和接收者地址、主旨、電子郵件本文 (純文字和 HTML 格式)，至 `AWS.SES` 用戶端類別的 `sendEmail` 方法。若要呼叫 `sendEmail` 方法，請建立叫用 Amazon SES 服務物件的 promise 來傳遞參數。然後，在 promise 回呼中處理 `response`。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });
```

```
// Create sendEmail params
var params = {
  Destination: {
    /* required */
    CcAddresses: [
      "EMAIL_ADDRESS",
      /* more items */
    ],
    ToAddresses: [
      "EMAIL_ADDRESS",
      /* more items */
    ],
  },
  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: "Test email",
    },
  },
  Source: "SENDER_EMAIL_ADDRESS" /* required */,
  ReplyToAddresses: [
    "EMAIL_ADDRESS",
    /* more items */
  ],
};

// Create the promise and SES service object
var sendPromise = new AWS.SES({ apiVersion: "2010-12-01" })
  .sendEmail(params)
  .promise();

// Handle promise's fulfilled/rejected states
```

```
sendPromise
  .then(function (data) {
    console.log(data.MessageId);
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

若要執行範例，請在命令列中輸入以下內容。電子郵件已排入佇列，以便由 Amazon SES 傳送。

```
node ses_sendemail.js
```

您可以在這裡找到此範例程式碼 GitHub。

使用範本傳送電子郵件

在此範例中，使用 Node.js 模組以搭配 Amazon SES 傳送電子郵件。以檔名 `ses_sendtemplatedemail.js` 建立一個 Node.js 模組。依前述內容設定軟體開發套件。

建立一個物件以傳遞定義欲傳送電子郵件的參數值，包括寄件者和接收者地址、主旨、電子郵件本文 (純文字和 HTML 格式)，至 `AWS.SES` 用戶端類別的 `sendTemplatedEmail` 方法。若要呼叫 `sendTemplatedEmail` 方法，請建立叫用 Amazon SES 服務物件的 promise 來傳遞參數。然後，在 promise 回呼中處理 `response`。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create sendTemplatedEmail params
var params = {
  Destination: {
    /* required */
    CcAddresses: [
      "EMAIL_ADDRESS",
      /* more CC email addresses */
    ],
    ToAddresses: [
      "EMAIL_ADDRESS",
      /* more To email addresses */
    ],
```

```
  },
  Source: "EMAIL_ADDRESS" /* required */,
  Template: "TEMPLATE_NAME" /* required */,
  TemplateData: '{ "REPLACEMENT_TAG_NAME":"REPLACEMENT_VALUE" }' /* required */,
  ReplyToAddresses: ["EMAIL_ADDRESS"],
};

// Create the promise and SES service object
var sendPromise = new AWS.SES({ apiVersion: "2010-12-01" })
  .sendTemplatedEmail(params)
  .promise();

// Handle promise's fulfilled/rejected states
sendPromise
  .then(function (data) {
    console.log(data);
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

若要執行範例，請在命令列中輸入以下內容。電子郵件已排入佇列，以便由 Amazon SES 傳送。

```
node ses_sendtemplatedemail.js
```

您可以在這裡找到此範例程式碼 GitHub。

使用範本傳送大量電子郵件

在此範例中，使用 Node.js 模組以搭配 Amazon SES 傳送電子郵件。以檔名 `ses_sendbulktemplatedemail.js` 建立一個 Node.js 模組。依前述內容設定軟體開發套件。

建立一個物件以傳遞定義欲傳送電子郵件的參數值，包括寄件者和接收者地址、主旨、電子郵件本文 (純文字和 HTML 格式)，至 `AWS.SES` 用戶端類別的 `sendBulkTemplatedEmail` 方法。若要呼叫 `sendBulkTemplatedEmail` 方法，請建立叫用 Amazon SES 服務物件的 promise 來傳遞參數。然後，在 promise 回呼中處理 `response`。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });
```

```
// Create sendBulkTemplatedEmail params
var params = {
  Destinations: [
    /* required */
    {
      Destination: {
        /* required */
        CcAddresses: [
          "EMAIL_ADDRESS",
          /* more items */
        ],
        ToAddresses: [
          "EMAIL_ADDRESS",
          "EMAIL_ADDRESS",
          /* more items */
        ],
      },
      ReplacementTemplateData: '{ "REPLACEMENT_TAG_NAME":"REPLACEMENT_VALUE" }',
    },
  ],
  Source: "EMAIL_ADDRESS" /* required */,
  Template: "TEMPLATE_NAME" /* required */,
  DefaultTemplateData: '{ "REPLACEMENT_TAG_NAME":"REPLACEMENT_VALUE" }',
  ReplyToAddresses: ["EMAIL_ADDRESS"],
};

// Create the promise and SES service object
var sendPromise = new AWS.SES({ apiVersion: "2010-12-01" })
  .sendBulkTemplatedEmail(params)
  .promise();

// Handle promise's fulfilled/rejected states
sendPromise
  .then(function (data) {
    console.log(data);
  })
  .catch(function (err) {
    console.log(err, err.stack);
  });
```

若要執行範例，請在命令列中輸入以下內容。電子郵件已排入佇列，以便由 Amazon SES 傳送。

```
node ses_sendbulktemplatedemail.js
```

您可以在這裡找到此範例程式碼 GitHub。

在 Amazon SES 中使用 IP 位址篩選器進行電子郵件收據

這個 Node.js 程式碼範例會說明：

- 建立 IP 地址篩選條件以接受或拒絕來自一個 IP 地址或一個 IP 地址範圍的郵件。
- 列出您的現有 IP 地址篩選條件。
- 刪除 IP 地址篩選條件。

在 Amazon SES 中，篩選器是一種資料結構，其中包含名稱、IP 位址範圍，以及是否允許或封鎖來自該篩選器的郵件。您想要封鎖或允許的單一 IP 地址或一組 IP 地址範圍，是以無類別網域間路由選擇 (CIDR) 表示法來指定。如需 Amazon SES 如何接收電子郵件的詳細資訊，請參閱 Amazon SES 電子郵件接收概念 (位於 Amazon 簡單電子郵件服務開發人員指南

使用案例

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

- `createReceiptFilter`
- `listReceiptFilters`
- `deleteReceiptFilter`

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：

- 安裝 Node.js。如需安裝 Node.js 的詳細資訊，請參閱 Node.js 網站。
- 透過使用者登入資料建立共用組態檔。如需提供共用登入資料檔案的詳細資訊，請參閱 從共用登入資料檔案中在 Node.js 中載入登入資料。

設定軟體開發套件

JavaScript 通過創建全局配置對象，然後為代碼設置區域來配置 SDK。在此範例中，區域會設為 us-west-2。

```
// Load the SDK for JavaScript
var AWS = require('aws-sdk');
// Set the Region
AWS.config.update({region: 'us-west-2'});
```

建立 IP 地址篩選條件

在此範例中，使用 Node.js 模組以搭配 Amazon SES 傳送電子郵件。以檔名 ses_createreceiptfilter.js 建立一個 Node.js 模組。依前述內容設定軟體開發套件。

建立一個物件以傳遞定義 IP 篩選條件的參數值，包括篩選條件名稱、一個 IP 地址或要篩選的地址範圍，以及是否從篩選的地址允許或封鎖電子郵件流量。若要呼叫 createReceiptFilter 方法，請建立叫用 Amazon SES 服務物件的 promise 來傳遞參數。然後，在 promise 回呼中處理 response。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create createReceiptFilter params
var params = {
  Filter: {
    IpFilter: {
      Cidr: "IP_ADDRESS_OR_RANGE",
      Policy: "Allow" | "Block",
    },
    Name: "NAME",
  },
};

// Create the promise and SES service object
var sendPromise = new AWS.SES({ apiVersion: "2010-12-01" })
  .createReceiptFilter(params)
  .promise();

// Handle promise's fulfilled/rejected states
sendPromise
```

```
  .then(function (data) {
    console.log(data);
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

若要執行範例，請在命令列中輸入以下內容。該過濾器是在 Amazon SES 中創建的。

```
node ses_createreceiptfilter.js
```

您可以在這裡找到此範例程式碼 GitHub。

列出您的 IP 地址篩選條件

在此範例中，使用 Node.js 模組以搭配 Amazon SES 傳送電子郵件。以檔名 `ses_listreceiptfilters.js` 建立一個 Node.js 模組。依前述內容設定軟體開發套件。

建立一個空參數物件。若要呼叫 `listReceiptFilters` 方法，請建立叫用 Amazon SES 服務物件的 promise 來傳遞參數。然後，在 promise 回呼中處理 `response`。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the promise and SES service object
var sendPromise = new AWS.SES({ apiVersion: "2010-12-01" })
  .listReceiptFilters({})
  .promise();

// Handle promise's fulfilled/rejected states
sendPromise
  .then(function (data) {
    console.log(data.Filters);
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

若要執行範例，請在命令列中輸入以下內容。Amazon SES 返回過濾器列表。

```
node ses_listreceiptfilters.js
```

您可以在這裡找到此範例程式碼 GitHub。

刪除 IP 地址篩選條件

在此範例中，使用 Node.js 模組以搭配 Amazon SES 傳送電子郵件。以檔名 `ses_deletereceiptfilter.js` 建立一個 Node.js 模組。依前述內容設定軟體開發套件。

建立物件以傳遞要刪除的 IP 篩選條件名稱。若要呼叫 `deleteReceiptFilter` 方法，請建立叫用 Amazon SES 服務物件的 promise 來傳遞參數。然後，在 promise 回呼中處理 `response`。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the promise and SES service object
var sendPromise = new AWS.SES({ apiVersion: "2010-12-01" })
  .deleteReceiptFilter({ FilterName: "NAME" })
  .promise();

// Handle promise's fulfilled/rejected states
sendPromise
  .then(function (data) {
    console.log("IP Filter deleted");
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

若要執行範例，請在命令列中輸入以下內容。該過濾器從 Amazon SES 中刪除。

```
node ses_deletereceiptfilter.js
```

您可以在這裡找到此範例程式碼 GitHub。

在 Amazon SES 中使用接收規則

這個 Node.js 程式碼範例會說明：

- 建立並刪除接收規則。
- 將接收規則編入接收規則組。

Amazon SES 中的接收規則會指定如何處理針對您擁有的電子郵件地址或網域收到的電子郵件。接收規則包括條件與動作排序清單。如果內送電子郵件的收件者符合接收規則條件中指定的收件者，Amazon SES 會執行接收規則指定的動作。

若要使用 Amazon SES 做為電子郵件接收者，您必須至少有一個作用中的接收規則集。接收規則集是一組已排序的接收規則集合，可指定 Amazon SES 應如何處理透過驗證網域接收的郵件。如需詳細資訊，請參閱 Amazon SES 電子郵件接收建立接收規則和 Amazon SES 電子郵件接收建立 Amazon SES 電子郵件接收的接收規則集 (英文)。

使用案例

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

- `createReceiptRule`
- `deleteReceiptRule`
- `createReceiptRuleSet`
- `deleteReceiptRuleSet`

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：

- 安裝 Node.js。如需安裝 Node.js 的詳細資訊，請參閱 Node.js 網站。
- 透過使用者登入資料建立共用組態檔。如需有關提供登入資料 JSON 檔案的詳細資訊，請參閱 從共用登入資料檔案中在 Node.js 中載入登入資料。

建立 Amazon S3 收據規則

Amazon SES 的每個接收規則都包含一份已排序的動作清單。此範例使用 Amazon S3 動作建立接收規則，該動作可將郵件訊息傳送到 Amazon S3 儲存貯體。如需接收規則動作的詳細資訊，請參閱 Amazon 簡易電子郵件服務開發人員指南中的動作選項。

若要讓 Amazon SES 將電子郵件寫入 Amazon S3 儲存貯體，請建立`PutObject`可授予 Amazon SES 許可的儲存貯體政策。如需有關建立此儲存貯體政策的資訊，請參閱 Amazon 簡單電子郵件服務開發人員指南中的授予 Amazon SES 寫入 Amazon S3 儲存貯體的權限。

在此範例中，使用 Node.js 模組在 Amazon SES 中建立接收規則，以將接收的訊息儲存在 Amazon S3 儲存貯體中。以檔名 `ses_createreceiptrule.js` 建立一個 Node.js 模組。依前述內容設定軟體開發套件。

建立參數物件以傳遞建立接收規則集所需的值。若要呼叫 `createReceiptRuleSet` 方法，請建立叫用 Amazon SES 服務物件的 promise 來傳遞參數。然後，在 promise 回呼中處理 `response`。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create createReceiptRule params
var params = {
  Rule: {
    Actions: [
      {
        S3Action: {
          BucketName: "S3_BUCKET_NAME",
          ObjectKeyPrefix: "email",
        },
      },
    ],
    Recipients: [
      "DOMAIN | EMAIL_ADDRESS",
      /* more items */
    ],
    Enabled: true | false,
    Name: "RULE_NAME",
    ScanEnabled: true | false,
    TlsPolicy: "Optional",
  },
```

```
  RuleSetName: "RULE_SET_NAME",
};

// Create the promise and SES service object
var newRulePromise = new AWS.SES({ apiVersion: "2010-12-01" })
  .createReceiptRule(params)
  .promise();

// Handle promise's fulfilled/rejected states
newRulePromise
  .then(function (data) {
    console.log("Rule created");
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

若要執行範例，請在命令列中輸入以下內容。Amazon SES 創建接收規則。

```
node ses_createreceiptrule.js
```

您可以在這裡找到此範例程式碼 GitHub。

刪除接收規則

在此範例中，使用 Node.js 模組以搭配 Amazon SES 傳送電子郵件。以檔名 `ses_deletereceiptrule.js` 建立一個 Node.js 模組。依前述內容設定軟體開發套件。

建立參數物件以傳遞接收規則所要刪除的名稱。若要呼叫 `deleteReceiptRule` 方法，請建立叫用 Amazon SES 服務物件的 promise 來傳遞參數。然後，在 promise 回呼中處理 `response`。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create deleteReceiptRule params
var params = {
  RuleName: "RULE_NAME" /* required */,
  RuleSetName: "RULE_SET_NAME" /* required */,
};
```

```
// Create the promise and SES service object
var newRulePromise = new AWS.SES({ apiVersion: "2010-12-01" })
  .deleteReceiptRule(params)
  .promise();

// Handle promise's fulfilled/rejected states
newRulePromise
  .then(function (data) {
    console.log("Receipt Rule Deleted");
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

若要執行範例，請在命令列中輸入以下內容。Amazon SES 會建立接收規則集清單。

```
node ses_deletereceiptrule.js
```

您可以在這裡找到此範例程式碼 GitHub。

建立接收規則集

在此範例中，使用 Node.js 模組以搭配 Amazon SES 傳送電子郵件。以檔名 `ses_createreceiptruleset.js` 建立一個 Node.js 模組。依前述內容設定軟體開發套件。

建立參數物件以為新接收規則集傳遞名稱。若要呼叫 `createReceiptRuleSet` 方法，請建立叫用 Amazon SES 服務物件的 promise 來傳遞參數。然後，在 promise 回呼中處理 `response`。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the promise and SES service object
var newRulePromise = new AWS.SES({ apiVersion: "2010-12-01" })
  .createReceiptRuleSet({ RuleSetName: "NAME" })
  .promise();

// Handle promise's fulfilled/rejected states
newRulePromise
  .then(function (data) {
    console.log(data);
```

```
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

若要執行範例，請在命令列中輸入以下內容。Amazon SES 會建立接收規則集清單。

```
node ses_createreceiptruleset.js
```

您可以在這裡找到此範例程式碼 GitHub。

刪除接收規則集

在此範例中，使用 Node.js 模組以搭配 Amazon SES 傳送電子郵件。以檔名 `ses_deletereceiptruleset.js` 建立一個 Node.js 模組。依前述內容設定軟體開發套件。

建立物件以傳遞接收規則集所要刪除的名稱。若要呼叫 `deleeReceiptRuleSet` 方法，請建立叫用 Amazon SES 服務物件的 promise 來傳遞參數。然後，在 promise 回呼中處理 `response`。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the promise and SES service object
var newRulePromise = new AWS.SES({ apiVersion: "2010-12-01" })
  .deleteReceiptRuleSet({ RuleSetName: "NAME" })
  .promise();

// Handle promise's fulfilled/rejected states
newRulePromise
  .then(function (data) {
    console.log(data);
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

若要執行範例，請在命令列中輸入以下內容。Amazon SES 會建立接收規則集清單。

```
node ses_deletereceiptruleset.js
```

您可以在這裡找到此範例程式碼 GitHub。

Amazon 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 (網頁伺服器、電子郵件地址、Amazon SQS 佇列、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 是透過 `Class: AWS.SNS`.

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AWS.SNS戶端類別的下列方法的主題：

- createTopic
- listTopics
- deleteTopic
- getTopicAttributes
- setTopicAttributes

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：

- 安裝 Node.js。如需安裝 Node.js 的詳細資訊，請參閱 Node.js 網站。
- 透過使用者登入資料建立共用組態檔。如需有關提供登入資料 JSON 檔案的詳細資訊，請參閱 從共用登入資料檔案中在 Node.js 中載入登入資料。

建立主題

在此範例中，使用 Node.js 模組建立 Amazon SNS 主題。以檔名 sns_createtopic.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，藉此將新主題的 Name 傳遞至 AWS.SNS 用戶端類別的 createTopic 方法。若要呼叫方createTopic法，請建立用於叫用 Amazon SNS 服務物件並傳遞參數物件的承諾。然後，在 promise 回呼中處理 response。promise 所傳回的 data 會包含主題 ARN。

```
// 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 createTopicPromise = new AWS.SNS({ apiVersion: "2010-03-31" })
  .createTopic({ Name: "TOPIC_NAME" })
  .promise();

// Handle promise's fulfilled/rejected states
createTopicPromise
  .then(function (data) {
    console.log("Topic ARN is " + data.TopicArn);
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

若要執行範例，請在命令列中輸入以下內容。

```
node sns_createtopic.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。以檔名 `sns_listtopics.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，並將其傳遞至 `AWS.SNS` 用戶端類別的 `listTopics` 方法。若要呼叫方`listTopics`法，請建立用於叫用 Amazon SNS 服務物件並傳遞參數物件的承諾。然後，在 promise 回呼中處理 `response`。promise 所傳回的 `data` 會包含主題 ARN 陣列。

```
// 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 listTopicsPromise = new AWS.SNS({ apiVersion: "2010-03-31" })
  .listTopics({})
  .promise();
```

```
// Handle promise's fulfilled/rejected states
listTopicsPromise
  .then(function (data) {
    console.log(data.Topics);
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

若要執行範例，請在命令列中輸入以下內容。

```
node sns_listtopics.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。以檔名 `sns_deletetopic.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 `AWS.SNS` 用戶端類別的 `deleteTopic` 方法。若要呼叫方`deleteTopic`法，請建立用於叫用 Amazon SNS 服務物件並傳遞參數物件的承諾。然後，在 promise 回呼中處理 `response`。

```
// 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 deleteTopicPromise = new AWS.SNS({ apiVersion: "2010-03-31" })
  .deleteTopic({ TopicArn: "TOPIC_ARN" })
  .promise();

// Handle promise's fulfilled/rejected states
deleteTopicPromise
  .then(function (data) {
    console.log("Topic Deleted");
  })
  .catch(function (err) {
    console.error(err, err.stack);
```

```
  });
```

若要執行範例，請在命令列中輸入以下內容。

```
node sns_deletetopic.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。以檔名 `sns_gettopicattributes.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 `AWS.SNS` 用戶端類別的 `getTopicAttributes` 方法。若要呼叫方`getTopicAttributes`法，請建立用於叫用 Amazon SNS 服務物件並傳遞參數物件的承諾。然後，在 promise 回呼中處理 `response`。

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

若要執行範例，請在命令列中輸入以下內容。

```
node sns_gettopicattributes.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。以檔名 `sns_settopicattributes.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 `AWS.SNS` 用戶端類別的 `setTopicAttributes` 方法。若要呼叫方`setTopicAttributes`法，請建立用於叫用 Amazon SNS 服務物件並傳遞參數物件的承諾。然後，在 promise 回呼中處理 `response`。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set region
AWS.config.update({ region: "REGION" });

// Create setTopicAttributes parameters
var params = {
  AttributeName: "ATTRIBUTE_NAME" /* required */,
  TopicArn: "TOPIC_ARN" /* required */,
  AttributeValue: "NEW_ATTRIBUTE_VALUE",
};

// Create promise and SNS service object
var setTopicAttribsPromise = new AWS.SNS({ apiVersion: "2010-03-31" })
  .setTopicAttributes(params)
  .promise();

// Handle promise's fulfilled/rejected states
setTopicAttribsPromise
  .then(function (data) {
    console.log(data);
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

若要執行範例，請在命令列中輸入以下內容。

```
node sns_settopicattributes.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`AWS.SNS`戶端類別的這個方法 JavaScript 來傳送訊息：

- `publish`

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：

- 安裝 Node.js。如需安裝 Node.js 的詳細資訊，請參閱 Node.js 網站。
- 透過使用者登入資料建立共用組態檔。如需有關提供登入資料 JSON 檔案的詳細資訊，請參閱 從共用登入資料檔案中在 Node.js 中載入登入資料。

將訊息發佈到 Amazon SNS 主題

在此範例中，使用 Node.js 模組將訊息發佈到 Amazon SNS 主題。以檔名 `sns_publishtotopic.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 SNS 主題的 ARN。如需可用簡訊屬性的詳細資訊，請參閱 SetSMSAttributes。

將參數傳遞至 `AWS.SNS` 用戶端類別的 `publish` 方法。建立用於叫用 Amazon SNS 服務物件並傳遞參數物件的承諾。接著在 promise 回呼中處理回應。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set region
```

```
AWS.config.update({ region: "REGION" });

// Create publish parameters
var params = {
  Message: "MESSAGE_TEXT" /* required */,
  TopicArn: "TOPIC_ARN",
};

// Create promise and SNS service object
var publishTextPromise = new AWS.SNS({ apiVersion: "2010-03-31" })
  .publish(params)
  .promise();

// Handle promise's fulfilled/rejected states
publishTextPromise
  .then(function (data) {
    console.log(
      `Message ${params.Message} sent to the topic ${params.TopicArn}`
    );
    console.log("MessageID is " + data.MessageId);
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

若要執行範例，請在命令列中輸入以下內容。

```
node sns_publishtotopic.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AWS.SNS戶端類別的下列方法的主題：

- subscribe
- confirmSubscription
- listSubscriptionsByTopic
- unsubscribe

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：

- 安裝 Node.js。如需安裝 Node.js 的詳細資訊，請參閱 Node.js 網站。
- 透過使用者登入資料建立共用組態檔。如需有關提供登入資料 JSON 檔案的詳細資訊，請參閱 從共用登入資料檔案中在 Node.js 中載入登入資料。

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。以檔名 sns_listsubscriptions.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 AWS.SNS 用戶端類別的 listSubscriptionsByTopic 方法。若要呼叫方listSubscriptionsByTopic法，請建立用於叫用 Amazon SNS 服務物件並傳遞參數物件的承諾。然後，在 promise 回呼中處理 response。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set region
AWS.config.update({ region: "REGION" });

const params = {
  TopicArn: "TOPIC_ARN",
};
```

```
// Create promise and SNS service object
var subslistPromise = new AWS.SNS({ apiVersion: "2010-03-31" })
  .listSubscriptionsByTopic(params)
  .promise();

// Handle promise's fulfilled/rejected states
subslistPromise
  .then(function (data) {
    console.log(data);
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

若要執行範例，請在命令列中輸入以下內容。

```
node sns_listsubscriptions.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。以檔名 `sns_subscribeemail.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 `AWS.SNS` 用戶端類別的 `subscribe` 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的參數值，如本主題中的其他範例所示。

若要呼叫方`subscribe`法，請建立用於叫用 Amazon SNS 服務物件並傳遞參數物件的承諾。然後，在 promise 回呼中處理 `response`。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set region
AWS.config.update({ region: "REGION" });

// Create subscribe/email parameters
var params = {
  Protocol: "EMAIL" /* required */,
```

```
  TopicArn: "TOPIC_ARN" /* required */,
  Endpoint: "EMAIL_ADDRESS",
};

// Create promise and SNS service object
var subscribePromise = new AWS.SNS({ apiVersion: "2010-03-31" })
  .subscribe(params)
  .promise();

// Handle promise's fulfilled/rejected states
subscribePromise
  .then(function (data) {
    console.log("Subscription ARN is " + data.SubscriptionArn);
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

若要執行範例，請在命令列中輸入以下內容。

```
node sns_subscribeemail.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。以檔名 `sns_subscribeapp.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 `application` 通訊協定、要訂閱的主題 `TopicArn`，以及要做為 `Endpoint` 參數的行動應用程式端點 ARN。將參數傳遞至 `AWS.SNS` 用戶端類別的 `subscribe` 方法。

若要呼叫方subscribe法，請建立用於叫用 Amazon SNS 服務物件並傳遞參數物件的承諾。然後，在 promise 回呼中處理 `response`。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set region
AWS.config.update({ region: "REGION" });

// Create subscribe/email parameters
```

```
var params = {
  Protocol: "application" /* required */,
  TopicArn: "TOPIC_ARN" /* required */,
  Endpoint: "MOBILE_ENDPOINT_ARN",
};

// Create promise and SNS service object
var subscribePromise = new AWS.SNS({ apiVersion: "2010-03-31" })
  .subscribe(params)
  .promise();

// Handle promise's fulfilled/rejected states
subscribePromise
  .then(function (data) {
    console.log("Subscription ARN is " + data.SubscriptionArn);
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

若要執行範例，請在命令列中輸入以下內容。

```
node sns_subscribeapp.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。以檔名 `sns_subscribelambda.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、要訂閱的主題 `TopicArn`，以及要做為 `Endpoint` 參數的 AWS Lambda 函數 ARN。將參數傳遞至 `AWS.SNS` 用戶端類別的 `subscribe` 方法。

若要呼叫方subscribe法，請建立用於叫用 Amazon SNS 服務物件並傳遞參數物件的承諾。然後，在 promise 回呼中處理 `response`。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set region
AWS.config.update({ region: "REGION" });
```

```
// Create subscribe/email parameters
var params = {
  Protocol: "lambda" /* required */,
  TopicArn: "TOPIC_ARN" /* required */,
  Endpoint: "LAMBDA_FUNCTION_ARN",
};

// Create promise and SNS service object
var subscribePromise = new AWS.SNS({ apiVersion: "2010-03-31" })
  .subscribe(params)
  .promise();

// Handle promise's fulfilled/rejected states
subscribePromise
  .then(function (data) {
    console.log("Subscription ARN is " + data.SubscriptionArn);
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

若要執行範例，請在命令列中輸入以下內容。

```
node sns_subscribelambda.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。以檔名 `sns_unsubscribe.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 `SubscriptionArn` 參數的物件，並指定要取消訂閱的訂閱 ARN。將參數傳遞至 `AWS.SNS` 用戶端類別的 `unsubscribe` 方法。

若要呼叫方unsubscribe法，請建立用於叫用 Amazon SNS 服務物件並傳遞參數物件的承諾。然後，在 promise 回呼中處理 `response`。

```
// 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 subscribePromise = new AWS.SNS({ apiVersion: "2010-03-31" })
  .unsubscribe({ SubscriptionArn: TOPIC_SUBSCRIPTION_ARN })
  .promise();

// Handle promise's fulfilled/rejected states
subscribePromise
  .then(function (data) {
    console.log(data);
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

若要執行範例，請在命令列中輸入以下內容。

```
node sns_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AWS.SNS戶端類別的下列方法：

- getSMSAttributes
- setSMSAttributes
- checkIfPhoneNumberIsOptedOut
- listPhoneNumbersOptedOut
- publish

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：

- 安裝 Node.js。如需安裝 Node.js 的詳細資訊，請參閱 Node.js 網站。
- 透過使用者登入資料建立共用組態檔。如需有關提供登入資料 JSON 檔案的詳細資訊，請參閱 從共用登入資料檔案中在 Node.js 中載入登入資料。

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。以檔名 `sns_getsmstype.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 `AWS.SNS` 用戶端類別的 `setTopicAttributes` 方法。若要呼叫方getSMSAttributes法，請建立用於叫用 Amazon SNS 服務物件並傳遞參數物件的承諾。然後，在 promise 回呼中處理 `response`。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set region
```

```
AWS.config.update({ region: "REGION" });

// Create SMS Attribute parameter you want to get
var params = {
  attributes: [
    "DefaultSMSType",
    "ATTRIBUTE_NAME",
    /* more items */
  ],
};

// Create promise and SNS service object
var getSMSTypePromise = new AWS.SNS({ apiVersion: "2010-03-31" })
  .getSMSAttributes(params)
  .promise();

// Handle promise's fulfilled/rejected states
getSMSTypePromise
  .then(function (data) {
    console.log(data);
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

若要執行範例，請在命令列中輸入以下內容。

```
node sns_getsmstype.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。以檔名 `sns_setsmstype.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 `AWS.SNS` 用戶端類別的 `setTopicAttributes` 方法。若要呼叫方`getSMSAttributes`法，請建立用於叫用 Amazon SNS 服務物件並傳遞參數物件的承諾。然後，在 promise 回呼中處理 `response`。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set region
AWS.config.update({ region: "REGION" });

// Create SMS Attribute parameters
var params = {
  attributes: {
    /* required */
    DefaultSMSType: "Transactional" /* highest reliability */,
    //'DefaultSMSType': 'Promotional' /* lowest cost */
  },
};

// Create promise and SNS service object
var setSMSTypePromise = new AWS.SNS({ apiVersion: "2010-03-31" })
  .setSMSAttributes(params)
  .promise();

// Handle promise's fulfilled/rejected states
setSMSTypePromise
  .then(function (data) {
    console.log(data);
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

若要執行範例，請在命令列中輸入以下內容。

```
node sns_setsmstype.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。以檔名 `sns_checkphoneoptout.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 `AWS.SNS` 用戶端類別的 `checkIfPhoneNumberIsOptedOut` 方法。若要呼叫

方checkIfPhoneNumberIsOptedOut法，請建立用於叫用 Amazon SNS 服務物件並傳遞參數物件的承諾。然後，在 promise 回呼中處理 response。

```
// 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 phonenumPromise = new AWS.SNS({ apiVersion: "2010-03-31" })
  .checkIfPhoneNumberIsOptedOut({ phoneNumber: "PHONE_NUMBER" })
  .promise();

// Handle promise's fulfilled/rejected states
phonenumPromise
  .then(function (data) {
    console.log("Phone Opt Out is " + data.isOptedOut);
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

若要執行範例，請在命令列中輸入以下內容。

```
node sns_checkphoneoptout.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。以檔名 sns_listnumbersoptedout.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 AWS.SNS 用戶端類別的 listPhoneNumbersOptedOut 方法。若要呼叫方listPhoneNumbersOptedOut法，請建立用於叫用 Amazon SNS 服務物件並傳遞參數物件的承諾。然後，在 promise 回呼中處理 response。

```
// 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 phonelistPromise = new AWS.SNS({ apiVersion: "2010-03-31" })
  .listPhoneNumbersOptedOut({})
  .promise();

// Handle promise's fulfilled/rejected states
phonelistPromise
  .then(function (data) {
    console.log(data);
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

若要執行範例，請在命令列中輸入以下內容。

```
node sns_listnumbersoptedout.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。以檔名 `sns_publishsms.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 `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 `AWS.SNS` 用戶端類別的 `publish` 方法。若要呼叫方`publish`法，請建立用於叫用 Amazon SNS 服務物件並傳遞參數物件的承諾。然後，在 promise 回呼中處理 `response`。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set region
AWS.config.update({ region: "REGION" });

// Create publish parameters
var params = {
  Message: "TEXT_MESSAGE" /* required */,
```

```
  PhoneNumber: "E.164_PHONE_NUMBER",
};

// Create promise and SNS service object
var publishTextPromise = new AWS.SNS({ apiVersion: "2010-03-31" })
  .publish(params)
  .promise();

// Handle promise's fulfilled/rejected states
publishTextPromise
  .then(function (data) {
    console.log("MessageID is " + data.MessageId);
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

若要執行範例，請在命令列中輸入以下內容。

```
node sns_publishsms.js
```

您可以在這裡找到此範例程式碼 GitHub。

Amazon SQS 範例

Amazon Simple Queue Service (Amazon SQS) 是快速、可靠、可擴展、完全受管的訊息佇列服務。Amazon SQS 可讓您分離雲端應用程式的元件。Amazon SQS 包括具有高輸送量和 at-least-once 處理能力的標準佇列，以及提供 FIFO (先進先出) 交付和僅處理一次的 FIFO 佇列。

Amazon SQS 的 JavaScript API 是透過`AWS.SQS`用戶端類別公開的。如需使用 Amazon SQS 用戶端類別的詳細資訊，請參閱 API 參考資料`Class: AWS.SQS`中的。

主題

- 在 Amazon SQS 中使用佇列
- 在 Amazon SQS 中傳送和接收訊息
- 在 Amazon SQS 中管理可見性逾時
- 在 Amazon SQS 中啟用長輪詢
- 在 Amazon SQS 中使用無效字母佇列

在 Amazon SQS 中使用佇列

這個 Node.js 程式碼範例會說明：

- 如何取得所有訊息佇列的清單
- 如何取得特定佇列的 URL
- 如何建立和刪除佇列

關於範例

在此範例中，您可以搭配一系列的 Node.js 模組來使用佇列。Node.js 模組會使用 SDK JavaScript 來啟用佇列呼叫用`AWS.SQS`戶端類別的下列方法：

- `listQueues`
- `createQueue`
- `getQueueUrl`
- `deleteQueue`

如需 Amazon SQS 訊息的詳細資訊，請參閱 Amazon 簡單佇列服務開發人員指南中的佇列運作方式。

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：

- 安裝 Node.js。如需安裝 Node.js 的詳細資訊，請參閱 Node.js 網站。
- 透過使用者登入資料建立共用組態檔。如需提供共用登入資料檔案的詳細資訊，請參閱 從共用登入資料檔案中在 Node.js 中載入登入資料。

列出佇列

以檔名 `sqs_listqueues.js` 建立一個 Node.js 模組。請務必依前述的內容來設定軟體開發套件。若要存取 Amazon SQS，請建立AWS.SQS服務物件。建立包含列出佇列所需參數的 JSON 物件，其依預設為空白物件。呼叫 `listQueues` 方法以擷取佇列清單。接著，回呼函數會傳回所有佇列的 URL。

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

若要執行範例，請在命令列中輸入以下內容。

```
node sqs_listqueues.js
```

您可以在這裡找到此範例程式碼 GitHub。

建立佇列

以檔名 `sqs_createqueue.js` 建立一個 Node.js 模組。請務必依前述的內容來設定軟體開發套件。若要存取 Amazon SQS，請建立AWS.SQS服務物件。建立包含列出佇列所需參數的 JSON 物件，且其中應包括建立的佇列名稱。該參數也能包含佇列屬性，例如訊息交付延遲的秒數，或是保留接收訊息所需的秒數。呼叫 `createQueue` 方法。接著，回呼函數會傳回所建立佇列的 URL。

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

若要執行範例，請在命令列中輸入以下內容。

```
node sqs_createqueue.js
```

您可以在這裡找到此範例程式碼 GitHub。

取得佇列 URL

以檔名 `sqs_getqueueurl.js` 建立一個 Node.js 模組。請務必依前述的內容來設定軟體開發套件。若要存取 Amazon SQS，請建立`AWS.SQS`服務物件。建立包含列出佇列所需參數的 JSON 物件，且其中應包括要取得 URL 的佇列名稱。呼叫 `getQueueUrl` 方法。接著，回呼函數會傳回指定佇列的 URL。

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

若要執行範例，請在命令列中輸入以下內容。

```
node sqs_getqueueurl.js
```

您可以在這裡找到此範例程式碼 GitHub。

刪除佇列

以檔名 `sqs_deletequeue.js` 建立一個 Node.js 模組。請務必依前述的內容來設定軟體開發套件。若要存取 Amazon SQS，請建立`AWS.SQS`服務物件。建立包含刪除佇列所需參數的 JSON 物件，其應由要刪除的佇列 URL 所組成。呼叫 `deleteQueue` 方法。

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

若要執行範例，請在命令列中輸入以下內容。

```
node sqs_deletequeue.js
```

您可以在這裡找到此範例程式碼 GitHub。

在 Amazon SQS 中傳送和接收訊息

這個 Node.js 程式碼範例會說明：

- 如何在佇列中傳送訊息。
- 如何在佇列中接收訊息。
- 如何在佇列中刪除訊息。

使用案例

在此範例中，您可以使用一系列的 Node.js 模組來傳送和接收訊息。Node.js 模組會使用 SDK JavaScript 來傳送和接收訊息，使用用AWS.SQS戶端類別的下列方法：

- `sendMessage`
- `receiveMessage`
- `deleteMessage`

如需 Amazon SQS 訊息的詳細資訊，請參閱 Amazon 簡單佇列服務開發人員指南中的傳送訊息到 Amazon SQS 佇列以及從 Amazon SQS 佇列接收和刪除訊息。

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：

- 安裝 Node.js。如需安裝 Node.js 的詳細資訊，請參閱 Node.js 網站。
- 透過使用者登入資料建立共用組態檔。如需提供共用登入資料檔案的詳細資訊，請參閱 從共用登入資料檔案中在 Node.js 中載入登入資料。
- 建立 Amazon SQS 佇列。如需建立佇列的範例，請參閱在 Amazon SQS 中使用佇列。

傳送訊息至佇列

以檔名 sqs_sendmessage.js 建立一個 Node.js 模組。請務必依前述的內容來設定軟體開發套件。若要存取 Amazon SQS，請建立AWS.SQS服務物件。建立包含訊息所需參數的 JSON 物件，且其中應包括要傳送此訊息的目標佇列 URL。在此範例中，該訊息會提供暢銷小說排行榜中的書籍詳細資訊，包括書名、作者和上榜週數。

呼叫 sendMessage 方法。接著，回呼函數會傳回該訊息的唯一 ID。

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

若要執行範例，請在命令列中輸入以下內容。

```
node sqs_sendmessage.js
```

您可以在這裡找到此範例程式碼 GitHub。

從佇列接收並刪除訊息

以檔名 `sqs_receivemessage.js` 建立一個 Node.js 模組。請務必依前述的內容來設定軟體開發套件。若要存取 Amazon SQS，請建立`AWS.SQS`服務物件。建立包含訊息所需參數的 JSON 物件，且應包括要從中接收訊息的佇列 URL。在此範例中，該參數會指定接收所有訊息屬性，並將接收的訊息數量上限設定為 10 則。

呼叫 `receiveMessage` 方法。接著，回呼函數會傳回 `Message` 物件陣列，您能夠從中擷取每個訊息的 `ReceiptHandle`，稍後再使用此屬性刪除該訊息。建立另一個 JSON 物件，其中包含刪除訊息所需的參數，意即佇列 URL 和 `ReceiptHandle` 值。接著，呼叫 `deleteMessage` 方法以刪除收到的訊息。

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

若要執行範例，請在命令列中輸入以下內容。

```
node sqs_receivemessage.js
```

您可以在這裡找到此範例程式碼 GitHub。

在 Amazon SQS 中管理可見性逾時

這個 Node.js 程式碼範例會說明：

- 如何指定佇列收到不可見訊息的時間間隔。

使用案例

在此範例中，Node.js 模組可用來管理可見性逾時。Node.js 模組會使用 SDK JavaScript 來管理可見性逾時，使用用AWS.SQS戶端類別的這個方法：

- changeMessageVisibility

如需 Amazon SQS 可見性逾時的詳細資訊，請參閱 Amazon 簡單佇列服務開發人員指南中的可見性逾時。

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：

- 安裝 Node.js。如需安裝 Node.js 的詳細資訊，請參閱 Node.js 網站。
- 透過使用者登入資料建立共用組態檔。如需提供共用登入資料檔案的詳細資訊，請參閱 從共用登入資料檔案中在 Node.js 中載入登入資料。
- 建立 Amazon SQS 佇列。如需建立佇列的範例，請參閱在 Amazon SQS 中使用佇列。
- 傳送訊息至佇列。如需傳送訊息至佇列的範例，請參閱在 Amazon SQS 中傳送和接收訊息。

變更可見性逾時

以檔名 sqs_changingvisibility.js 建立一個 Node.js 模組。請務必依前述的內容來設定軟體開發套件。若要存取 Amazon 簡單佇列服務，請建立AWS.SQS服務物件。從佇列接收訊息。

收到來自佇列的訊息後，請建立包含設定逾時所需參數的 JSON 物件，其中應包括含有訊息的佇列 URL、收到訊息時所傳回的 ReceiptHandle，以及新的逾時期間 (以秒為單位)。呼叫 changeMessageVisibility 方法。

```
// 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");
    }
  }
});
```

若要執行範例，請在命令列中輸入以下內容。

```
node sqs_changingvisibility.js
```

您可以在這裡找到此範例程式碼 GitHub。

在 Amazon SQS 中啟用長輪詢

這個 Node.js 程式碼範例會說明：

- 如何為新建立的佇列啟用長輪詢
- 如何為現有佇列啟用長輪詢
- 如何在收到訊息時啟用長輪詢

使用案例

長輪詢可讓 Amazon SQS 在傳送回應之前等待訊息在佇列中可用的指定時間，藉此減少空白回應的數量。此外，長輪詢可查詢所有伺服器而非指查詢取樣的伺服器，來減少假的空白回應。若要啟用長輪詢，您必須指定針對接收的訊息指定非零的等待時間。您可以設定佇列的 `ReceiveMessageWaitTimeSeconds` 參數，或是在收到訊息時設定 `WaitTimeSeconds` 參數，藉此指定等待時間。

在此範例中，您可以使用一系列的 Node.js 模組來啟用長輪詢。Node.js 模組會使用 SDK JavaScript 來啟用使用用`AWS.SQS`戶端類別下列方法的長輪詢：

- `setQueueAttributes`
- `receiveMessage`
- `createQueue`

如需 Amazon SQS 長輪詢的詳細資訊，請參閱 Amazon 簡單佇列服務開發人員指南中的長輪詢。

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：

- 安裝 Node.js。如需安裝 Node.js 的詳細資訊，請參閱 Node.js 網站。
- 透過使用者登入資料建立共用組態檔。如需提供共用登入資料檔案的詳細資訊，請參閱 從共用登入資料檔案中在 Node.js 中載入登入資料。

在建立佇列時啟用長輪詢

以檔名 `sqs_longpolling_createqueue.js` 建立一個 Node.js 模組。請務必依前述的內容來設定軟體開發套件。若要存取 Amazon SQS，請建立`AWS.SQS`服務物件。建立包含所需參數的 JSON 物件以建立佇列，其中應包括 `ReceiveMessageWaitTimeSeconds` 參數的非零值。呼叫 `createQueue` 方法。然後為佇列啟用長輪詢。

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

若要執行範例，請在命令列中輸入以下內容。

```
node sqs_longpolling_createqueue.js
```

您可以在這裡找到此範例程式碼 GitHub。

在現有佇列上啟用長輪詢

以檔名 `sqs_longpolling_existingqueue.js` 建立一個 Node.js 模組。請務必依前述的內容來設定軟體開發套件。若要存取 Amazon 簡單佇列服務，請建立`AWS.SQS`服務物件。建立包含設定佇列屬性所需參數的 JSON 物件，其中應包括 `ReceiveMessageWaitTimeSeconds` 參數的非零值和佇列 URL。呼叫 `setQueueAttributes` 方法。然後為佇列啟用長輪詢。

```
// 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 = {
  Attributes: {
    ReceiveMessageWaitTimeSeconds: "20",
  },
  QueueUrl: "SQS_QUEUE_URL",
};

sqs.setQueueAttributes(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

若要執行範例，請在命令列中輸入以下內容。

```
node sqs_longpolling_existingqueue.js
```

您可以在這裡找到此範例程式碼 GitHub。

在收到訊息時啟用長輪詢

以檔名 `sqs_longpolling_receivemessage.js` 建立一個 Node.js 模組。請務必依前述的內容來設定軟體開發套件。若要存取 Amazon 簡單佇列服務，請建立`AWS.SQS`服務物件。建立包含接收訊息所需參數的 JSON 物件，其中應包括 `WaitTimeSeconds` 參數的非零值和佇列 URL。呼叫 `receiveMessage` 方法。

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

若要執行範例，請在命令列中輸入以下內容。

```
node sqs_longpolling_receivemessage.js
```

您可以在這裡找到此範例程式碼 GitHub。

在 Amazon SQS 中使用無效字母佇列

這個 Node.js 程式碼範例會說明：

- 如何使用佇列來接收和保存其他佇列無法處理的訊息。

使用案例

無效字母佇列即為其他 (來源) 佇列，可以將無法成功處理的訊息當成目標以進行作業。您可以在無效字母佇列中擱置並隔離這類訊息，以確定無法成功處理訊息的原因。而且，您必須針對每個傳送訊息至無效字母佇列的來源佇列，進行個別設定。多個佇列可以將目標設為同一個無效字母佇列。

在此範例中，Node.js 模組可用來將訊息路由至無效字母佇列。Node.js 模組會使用 SDK JavaScript 來使用用AWS.SQS戶端類別的這個方法來使用無效字母佇列：

- setQueueAttributes

如需 Amazon SQS 無效字母佇列的詳細資訊，請參閱 Amazon 簡單佇列服務開發人員指南中的使用 Amazon SQS 無效字母佇列。

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：

- 安裝 Node.js。如需安裝 Node.js 的詳細資訊，請參閱 Node.js 網站。
- 透過使用者登入資料建立共用組態檔。如需提供共用登入資料檔案的詳細資訊，請參閱 從共用登入資料檔案中在 Node.js 中載入登入資料。
- 建立 Amazon SQS 佇列作為無效字母佇列。如需建立佇列的範例，請參閱在 Amazon SQS 中使用佇列。

設定來源佇列

在您建立佇列做為無效字母佇列後，您必須設定其他佇列，用於將未處理的訊息路由到無效字母佇列。若要執行此操作，請指定一個再驅動政策，用以識別當做無效字母佇列的佇列，並指定個別訊息路由到無效字母佇列之前的最大接收量。

以檔名 `sqs_deadletterqueue.js` 建立一個 Node.js 模組。請務必依前述的內容來設定軟體開發套件。若要存取 Amazon SQS，請建立AWS.SQS服務物件。建立包含更新佇列屬性所需參數的 JSON 物件，其中應包括可指定無效字母佇列 ARN 及 `maxReceiveCount` 值的 `RedrivePolicy` 參數。此外，也請指定您要設定的 URL 來源佇列。呼叫 `setQueueAttributes` 方法。

```
// 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 = {
  Attributes: {
    RedrivePolicy:
      '{"deadLetterTargetArn":"DEAD_LETTER_QUEUE_ARN","maxReceiveCount":"10"}',
  },
  QueueUrl: "SOURCE_QUEUE_URL",
};
```

```
sqs.setQueueAttributes(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

若要執行範例，請在命令列中輸入以下內容。

```
node sqs_deadletterqueue.js
```

您可以在這裡找到此範例程式碼 GitHub。

教學課程

以下教學課程說明如何使用 AWS SDK for JavaScript執行各種相關任務。

主題

- 教學課程：在 Amazon EC2 執行個體上設定 Node.js

教學課程：在 Amazon EC2 執行個體上設定 Node.js

將 Node.js 與開發套件搭配使用的常見案例 JavaScript 是在亞馬遜彈性運算雲端 (Amazon EC2) 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 SSH 與其連線，接著在該執行個體上安裝 Node.js 並予以執行。

必要條件

本教學課程會假設您已啟動一個 Linux 執行個體，其具備可透過網際網路存取的公有 DNS 名稱，而您能夠使用 SSH 連線至該執行個體。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**
>
> 啟動新的 Amazon EC2 執行個體時，請使用 Amazon Linux 2023 亞馬遜機器映像 (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 使用者指南中的使用 Amazon EC2 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. 以 `ec2-user` 的身分使用 SSH 連線至 Linux 執行個體。
2. 在命令列中輸入以下指令，藉此安裝節點版本管理工具 (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
```

nvm 可以安裝多種 Node.js 版本且允許您在各版本間進行切換，因此我們會使用 nvm 安裝 Node.js。

3. nvm通過在命令行中鍵入以下命令來加載。

```
source ~/.bashrc
```

4. 使用 nvm 來安裝最新的 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)，因此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，則需要使用 nvm 來啟用已安裝的節點版本。如果實例終止，則需要再次安裝 node。另一種方法是在擁有要保留的組態後，製作 Amazon EC2 執行個體的 Amazon 機器映像 (AMI)，如以下主題所述。

建立 Amazon Machine Image

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 可讓您以相同的 Node.js 安裝輕鬆佈建多個 Amazon EC2 執行個體。如需有關從現有執行個體建立 AMI 的詳細資訊，請參閱 Amazon EC2 使用者指南中的建立 Amazon EBS 支援 Linux AMI。

相關資源

如需本主題所使用的命令和軟體詳細資訊，請參閱下方網頁：

- 節點版本管理器（nvm）：請參閱上的 nvm 回購。 GitHub
- 節點套件管理工具 (npm)：請參閱 npm 網站。

JavaScript API 參考資料

最新版 SDK 的「API 參考」主題可 JavaScript 在下列網址找到：

AWS SDK for JavaScript API 參考指南。

開啟 SDK 變更記錄 GitHub

如需 2.4.8 版以上版本的變更記錄，請前往：

更改日誌。

此 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。安全性是 AWS 和 之間的共同責任。共同責任模型 將此描述為雲端本身的安全和雲端內部的安全。

雲端的安全性 – AWS 負責保護執行 AWS 雲端中提供的所有服務的基礎設施，並提供您可以安全使用的服務。我們的安全責任是 的最高優先順序 AWS，而第三方稽核人員會定期測試和驗證我們的安全有效性，作為AWS 合規計畫的一部分。

雲端的安全性 – 您的責任取決於您使用 AWS 的服務，以及其他因素，包括資料的敏感度、組織的需求，以及適用的法律和法規。

此 AWS 產品或服務會透過其支援的特定 Amazon Web Services （ AWS ） 服務遵循共同責任模型。如需 AWS 服務安全性資訊，請參閱AWS 服務安全性文件頁面，以及AWS 合規計畫在 AWS 合規工作範圍內的服務。

主題

- 此 AWS 產品或服務中的資料保護
- 身分和存取權管理
- 此 AWS 產品或服務的合規驗證
- 此 AWS 產品或服務的復原能力
- 此 AWS 產品或服務的基礎設施安全
- 強制執行 的最小版本 TLS

此 AWS 產品或服務中的資料保護

AWS 共同責任模型適用於此 AWS 產品或服務中的資料保護。如本模型所述， AWS 負責保護執行所有 的全域基礎設施 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 安全部落格 上的AWS 共同責任模型和GDPR部落格文章。

為了資料保護，我們建議您保護 AWS 帳戶 憑證，並使用 AWS IAM Identity Center 或 AWS Identity and Access Management （ ） 設定個別使用者IAM。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 資源通訊。我們需要 TLS 1.2 和建議 TLS 1.3。
- 使用 設定 API 和使用者活動日誌 AWS CloudTrail。如需使用 CloudTrail 線索擷取 AWS 活動的資訊，請參閱 AWS CloudTrail 使用者指南 中的使用 CloudTrail 線索。
- 使用 AWS 加密解決方案，以及 中的所有預設安全控制項 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。
- 如果您在 AWS 透過命令列介面或 FIPS 存取 時需要 140-3 個經過驗證的密碼編譯模組API，請使用FIPS端點。如需可用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。這包括當您使用此 AWS 產品或服務，或使用主控台、 API AWS CLI或 的其他 AWS 服務 時 AWS SDKs。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。如果您將 URL提供給外部伺服器，強烈建議您在 中不要包含憑證資訊，URL以驗證您對該伺服器的請求。

身分和存取權管理

AWS Identity and Access Management （IAM） 是一種 AWS 服務 ，可協助管理員安全地控制對 AWS 資源的存取。IAM 管理員會控制誰可以驗證 （登入） 和授權 （具有許可） 使用 AWS 資源。IAM 是 AWS 服務 您可以免費使用的 。

主題

- 物件
- 使用身分驗證
- 使用政策管理存取權
- AWS 服務 如何使用 IAM
- 對 AWS 身分和存取權進行故障診斷

物件

使用 AWS Identity and Access Management （IAM） 的方式會有所不同，具體取決於您在 中執行的工作 AWS。

服務使用者 – 如果您使用 AWS 服務 執行任務，則管理員會為您提供所需的憑證和許可。當您使用更多 AWS 功能來執行工作時，您可能需要額外的許可。了解存取許可的管理方式可協助您向管理員請求

正確的許可。如果您無法存取 中的功能 AWS，請參閱 對 AWS 身分和存取權進行故障診斷或 AWS 服務 您正在使用的 使用者指南。

服務管理員 – 如果您負責公司 AWS 的資源，您可能擁有 的完整存取權 AWS。您的任務是判斷您的服務使用者應該存取哪些 AWS 功能和資源。然後，您必須向IAM管理員提交請求，以變更服務使用者的許可。請檢閱此頁面上的資訊，以了解 的基本概念IAM。若要進一步了解貴公司如何IAM搭配 使用 AWS，請參閱您正在使用的 使用者指南 AWS 服務 。

IAM 管理員 – 如果您是IAM管理員，您可能想要了解如何撰寫政策以管理 存取權的詳細資訊 AWS。若要檢視您可以在 中使用的以 AWS 身分為基礎的政策範例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 使用身分憑證登入 的方式。您必須以 AWS 帳戶根使用者身分、IAM使用者身分或擔任IAM角色來驗證 （登入 AWS）。

您可以使用透過身分來源提供的憑證，以聯合身分 AWS 身分登入 。 AWS IAM Identity Center （IAM Identity Center） 使用者、您公司的單一登入身分驗證，以及您的 Google 或 Facebook 憑證，都是聯合身分的範例。當您以聯合身分登入時，您的管理員先前會使用 IAM角色設定身分聯合。當您 AWS 使用聯合來存取 時，您會間接擔任 角色。

根據您身分的使用者類型，您可以登入 AWS Management Console 或 AWS 存取入口網站。如需登入的詳細資訊 AWS，請參閱 使用者指南 中的如何登入 AWS 帳戶您的 。 AWS 登入

如果您以 AWS 程式設計方式存取 ， AWS 會提供軟體開發套件 （SDK） 和命令列介面 （CLI），以使用您的憑證以密碼編譯方式簽署您的請求。如果您不使用 AWS 工具，則必須自行簽署請求。如需使用建議的方法來自行簽署請求的詳細資訊，請參閱 IAM 使用者指南 中的簽署 AWS API請求。

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 建議您使用多因素身分驗證 （MFA） 來提高帳戶的安全性。若要進一步了解，請參閱AWS IAM Identity Center 使用者指南中的多重要素驗證，以及使用者指南中的使用多重要素驗證 （MFA） AWS。 IAM

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，該身分可完全存取 帳戶中的所有 AWS 服務 和資源。此身分稱為 AWS 帳戶 根使用者，透過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。如需需要您以根使用者身分登入的任務完整清單，請參閱 IAM 使用者指南 中的需要根使用者憑證的任務。

聯合身分

作為最佳實務， 會要求人類使用者，包括需要管理員存取權的使用者，使用 AWS 服務 臨時憑證來與身分提供者使用聯合來存取 。

聯合身分是來自您的企業使用者目錄、Web 身分提供者、 AWS Directory Service、身分中心目錄，或使用透過身分來源提供的 AWS 服務 憑證存取的任何使用者。當聯合身分存取 時 AWS 帳戶，它們會擔任 角色，而角色會提供臨時憑證。

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，也可以將 連線和同步到您身分來源中的一組使用者 AWS 帳戶 和群組，以便在所有 和應用程式中使用。如需 IAM Identity Center 的相關資訊，請參閱 AWS IAM Identity Center 使用者指南 中的什麼是 IAM Identity Center？。

IAM 使用者和群組

IAM 使用者是 中具有單一個人或應用程式特定許可 AWS 帳戶 的身分。在可能的情況下，我們建議依賴臨時憑證，而不是建立具有密碼和存取金鑰等長期憑證IAM的使用者。不過，如果您有特定的使用案例需要IAM使用者長期憑證，建議您輪換存取金鑰。如需詳細資訊，請參閱 IAM 使用者指南 中的定期輪換需要長期憑證的使用案例存取金鑰。

IAM 群組是指定IAM使用者集合的身分。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，您可以擁有名為 的群組IAMAdmins，並授予該群組管理IAM資源的許可。

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。若要進一步了解，請參閱 IAM 使用者指南 中的何時建立IAM使用者 （ 而非角色 ）。

IAM 角色

IAM 角色是 中具有特定許可 AWS 帳戶 的身分。它類似於IAM使用者，但與特定人員無關。您可以透過 AWS Management Console 切換IAM角色 暫時在 中擔任角色。 https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles_use_switch-role-console.html您可以呼叫 或 AWS API AWS CLI 操作，或使用自訂 來擔任角色URL。如需使用角色方法的詳細資訊，請參閱 IAM 使用者指南 中的使用IAM角色。

IAM 具有臨時憑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 IAM 使用者指南 中的為第三方身分提供者建立角色。如果您使用 IAM Identity Center，您可以設定許可集。若要控制身分在身分驗證後可以存取的內容，IAMIdentity Center 會將許可集與 中的角色相關聯IAM。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。

- 臨時IAM使用者許可 – IAM使用者或角色可以擔任IAM角色，暫時接受特定任務的不同許可。
- 跨帳戶存取 – 您可以使用 IAM角色，允許不同帳戶中的某人（受信任的委託人）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。不過，使用部分 AWS 服務，您可以將政策直接連接至資源（而不是使用角色作為代理）。若要了解跨帳戶存取的角色和資源型政策之間的差異，請參閱 IAM 使用者指南 中的跨帳戶資源存取IAM。
- 跨服務存取 – 有些 AWS 服務 使用其他 中的功能 AWS 服務。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或在 Amazon S3 中儲存物件。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。
  - 轉送存取工作階段（FAS）– 當您使用IAM使用者或角色在 中執行動作時 AWS，您會被視為主體。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 使用呼叫 的委託人許可 AWS 服務，並結合請求向下游服務 AWS 服務 提出請求。FAS 只有在服務收到需要與其他 AWS 服務 或 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。如需提出FAS請求的政策詳細資訊，請參閱轉送存取工作階段 。
  - 服務角色 – 服務角色是服務代表您執行動作時擔任IAM的角色。IAM 管理員可以從 內部建立、修改和刪除服務角色IAM。如需詳細資訊，請參閱 使用者指南 中的建立角色以將許可委派給 AWS 服務 。 IAM
  - 服務連結角色 – 服務連結角色是連結至 的服務角色類型 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 ，並由 服務擁有。IAM 管理員可以檢視，但不能編輯服務連結角色的許可。
- 在 Amazon 上執行的應用程式 EC2 – 您可以使用 IAM角色來管理在EC2執行個體上執行之應用程式的臨時憑證，以及提出 AWS CLI 或 AWS API請求。最好將存取金鑰存放在EC2執行個體中。若要將 AWS 角色指派給EC2執行個體並將其提供給其所有應用程式，您可以建立連接至執行個體的執行個體設定檔。執行個體設定檔包含 角色，並啟用在EC2執行個體上執行的程式，以取得臨時憑證。如需詳細資訊，請參閱 IAM 使用者指南 中的使用 IAM角色將許可授予在 Amazon EC2執行個體上執行的應用程式。

若要了解如何使用IAM角色或IAM使用者，請參閱 IAM 使用者指南 中的建立IAM角色（而非使用者）的時機。

使用政策管理存取權

您可以透過建立政策並將其連接至 AWS 身分或資源 AWS 來控制 中的存取。政策是 AWS 其中的物件，當與身分或資源建立關聯時， 會定義其許可。當主體（使用者、根使用者或角色工作階段）發出請求時， 會 AWS 評估這些政策。政策中的許可決定是否允許或拒絕請求。大多數政策都以JSON文件 AWS 形式儲存在 中。如需JSON政策文件結構和內容的詳細資訊，請參閱 IAM 使用者指南 中的JSON政策概觀。

管理員可以使用 AWS JSON政策來指定誰可以存取內容。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。若要授予使用者對所需資源執行動作的許可，IAM管理員可以建立IAM政策。然後，管理員可以將IAM政策新增至角色，使用者可以擔任角色。

IAM 無論您用來執行操作的方法為何，政策都會定義動作的許可。例如，假設您有一個允許 `iam:GetRole` 動作的政策。具有該政策的使用者可以從 AWS Management Console、 AWS CLI或 AWS 取得角色資訊API。

身分型政策

身分型政策是您可以連接到身分的JSON許可政策文件，例如IAM使用者、使用者群組或角色。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。若要了解如何建立身分型政策，請參閱 IAM 使用者指南 中的建立IAM政策。

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。受管政策是獨立的政策，您可以連接到 中的多個使用者、群組和角色 AWS 帳戶。受管政策包括 AWS 受管政策和客戶受管政策。若要了解如何在受管政策或內嵌政策之間進行選擇，請參閱 IAM 使用者指南 中的在受管政策與內嵌政策之間進行選擇。

資源型政策

資源型政策是您連接至資源JSON的政策文件。資源型政策的範例包括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。主體可以包括帳戶、使用者、角色、聯合使用者或 AWS 服務。

資源型政策是位於該服務中的內嵌政策。您無法在資源型政策IAM中使用來自 的 AWS 受管政策。

存取控制清單（ACLs）

存取控制清單（ACLs）控制哪些主體（帳戶成員、使用者或角色）具有存取 資源的許可。ACLs 類似於資源型政策，雖然它們不使用JSON政策文件格式。

Amazon S3 AWS WAF和 Amazon VPC是支援 的服務範例ACLs。若要進一步了解 ACLs，請參閱 Amazon Simple Storage Service 開發人員指南 中的存取控制清單（ACL）概觀。

其他政策類型

AWS 支援其他較不常見的政策類型。這些政策類型可設定較常見政策類型授予您的最大許可。

- 許可界限 – 許可界限是一項進階功能，您可以在其中設定身分型政策可授予IAM實體（IAM使用者或角色）的最大許可。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。如需許可界限的詳細資訊，請參閱 IAM 使用者指南 中的IAM實體許可界限。
- 服務控制政策（SCPs）– SCPs是在 中指定組織或組織單位（OU）最大許可JSON的政策 AWS Organizations。 AWS Organizations 是一項用於分組和集中管理您企業 AWS 帳戶 所擁有多個項目的服務。如果您啟用組織中的所有功能，則可以將服務控制政策（SCPs）套用至任何或所有帳戶。SCP 限制成員帳戶中實體的許可，包括每個 AWS 帳戶根使用者。如需 Organizations 和 的詳細資訊SCPs，請參閱 AWS Organizations 使用者指南 中的服務控制政策。
- 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 使用者指南 中的工作階段政策。

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。若要了解如何 AWS 在涉及多種政策類型時決定是否允許請求，請參閱 IAM 使用者指南 中的政策評估邏輯。

AWS 服務 如何使用 IAM

若要取得如何使用 AWS 服務 大多數 IAM 功能的高階檢視，請參閱 IAM 使用者指南 中的 AWS 使用的服務IAM。

若要了解如何將特定 AWS 服務 與 搭配使用IAM，請參閱相關服務使用者指南中的安全區段。

對 AWS 身分和存取權進行故障診斷

使用下列資訊來協助您診斷和修正使用 AWS 和 時可能遇到的常見問題IAM。

主題

- 我無權在 中執行動作 AWS
- 我無權執行 iam：PassRole
- 我想要允許 以外的人員 AWS 帳戶 存取我的 AWS 資源

我無權在 中執行動作 AWS

如果您收到錯誤，告知您未獲授權執行動作，您的政策必須更新，允許您執行動作。

當mateojacksonIAM使用者嘗試使用主控台檢視虛構*my-example-widget*資源的詳細資訊，但沒有虛構awes:*GetWidget*許可時，會發生下列錯誤範例。

```
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。

我無權執行 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 marymajor IAM的使用者嘗試使用主控台在 中執行動作時，會發生下列錯誤範例 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。

我想要允許 以外的人員 AWS 帳戶 存取我的 AWS 資源

您可以建立一個角色，讓其他帳戶中的使用者或您組織外部的人員存取您的資源。您可以指定要允許哪些信任物件取得該角色。對於支援資源型政策或存取控制清單 （ACLs） 的服務，您可以使用這些政策來授予人員對資源的存取權。

如需進一步了解，請參閱以下內容：

- 若要了解 是否 AWS 支援這些功能，請參閱 AWS 服務 如何使用 IAM。
- 若要了解如何 AWS 帳戶 在您擁有的 資源之間提供存取權，請參閱 IAM 使用者指南 中的在您 AWS 帳戶 擁有的另一個資源中為IAM使用者提供存取權。
- 若要了解如何將資源的存取權提供給第三方 AWS 帳戶，請參閱 使用者指南 中的提供存取權給第三方 AWS 帳戶 擁有。 IAM
- 若要了解如何透過身分聯合提供存取權，請參閱 IAM 使用者指南 中的為外部驗證的使用者提供存取權 （ 身分聯合 ）。
- 若要了解使用角色和資源型政策進行跨帳戶存取之間的差異，請參閱 IAM 使用者指南 中的跨帳戶資源存取IAM。

此 AWS 產品或服務的合規驗證

若要了解 是否 AWS 服務 在特定合規計劃的範圍內，請參閱AWS 服務 依合規計劃範圍然後選擇您感興趣的合規計劃。如需一般資訊，請參閱 AWS Compliance Programs 。

您可以使用 下載第三方稽核報告 AWS Artifact。如需詳細資訊，請參閱在 中下載報告 AWS Artifact。

您在使用 時的合規責任 AWS 服務 取決於資料的敏感度、您公司的合規目標，以及適用的法律和法規。 AWS 提供下列資源以協助合規：

- 安全與合規快速入門指南 – 這些部署指南討論架構考量，並提供以 AWS 安全與合規為重心的基準環境部署步驟。
- Amazon Web Services 上HIPAA安全和合規的架構 – 本白皮書描述了公司如何使用 AWS 來建立HIPAA合格的應用程式。

> **Note**
>
> 並非所有 AWS 服務 都HIPAA符合資格。如需詳細資訊，請參閱HIPAA合格服務參考 。

- AWS 合規資源 – 此工作手冊和指南集合可能適用於您的產業和位置。
- AWS 客戶合規指南 – 透過合規的角度了解共同的責任模型。本指南摘要說明保護 AWS 服務 指南並映射到跨多個架構的安全控制項的最佳實務（包括國家標準和技術研究所（）NIST、支付卡產業安全標準委員會（PCI）和國際標準化組織（ISO））。
- AWS Config 開發人員指南中的使用規則評估資源 – AWS Config 服務會評估資源組態是否符合內部實務、產業指導方針和法規。
- AWS Security Hub – 這 AWS 服務 提供 內安全狀態的全面檢視 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 – 這會監控環境是否有可疑和惡意活動，藉此 AWS 服務 偵測 AWS 帳戶、工作負載、容器和資料的潛在威脅。 GuardDuty 可以協助您滿足特定合規架構強制要求的入侵偵測需求 DSS，以解決各種合規要求，例如 PCI 。
- AWS Audit Manager – 這 AWS 服務 可協助您持續稽核 AWS 用量，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。

此 AWS 產品或服務會透過其支援的特定 Amazon Web Services（AWS）服務遵循共同責任模型。如需 AWS 服務安全性資訊，請參閱AWS 服務安全性文件頁面，以及AWS 合規計畫在 AWS 合規工作範圍內的服務。

此 AWS 產品或服務的復原能力

AWS 全域基礎設施是以 AWS 區域 和可用區域為基礎。

AWS 區域 提供多個實體隔離和隔離的可用區域，這些區域與低延遲、高輸送量和高冗餘聯網連接。

透過可用區域，您可以設計與操作的應用程式和資料庫，在可用區域之間自動容錯移轉而不會發生中斷。可用區域的可用性、容錯能力和擴展能力，均較單一或多個資料中心的傳統基礎設施還高。

如需 AWS 區域和可用區域的詳細資訊，請參閱AWS 全域基礎設施。

此 AWS 產品或服務會透過其支援的特定 Amazon Web Services（AWS）服務遵循共同責任模型。如需 AWS 服務安全性資訊，請參閱AWS 服務安全性文件頁面，以及AWS 合規計畫在 AWS 合規工作範圍內的服務。

此 AWS 產品或服務的基礎設施安全

此 AWS 產品或服務使用受管服務，因此受到全球網路安全的 AWS 保護。如需有關 AWS 安全服務以及如何 AWS 保護基礎設施的資訊，請參閱 AWS Cloud Security 。若要使用基礎設施安全性的最佳實務來設計您的 AWS 環境，請參閱 Security Pillar AWS Well-Architected Framework 中的基礎設施保護。

您可以使用 AWS 已發佈的API呼叫，透過網路存取此 AWS 產品或服務。使用者端必須支援下列專案：

- Transport Layer Security （ TLS ）。我們需要 TLS 1.2 和 建議 TLS 1.3。
- 具有完美前向秘密 （ PFS ） 的加密套件，例如 DHE （ Ephemeral Diffie-Hellman ） 或 ECDHE （ Elliptic Curve Ephemeral Diffie-Hellman ）。現代系統(如 Java 7 和更新版本)大多會支援這些模式。

此外，必須使用與IAM委託人相關聯的存取金鑰 ID 和秘密存取金鑰來簽署請求。或者，您可以透過 AWS Security Token Service (AWS STS) 來產生暫時安全憑證來簽署請求。

此 AWS 產品或服務會透過其支援的特定 Amazon Web Services （ AWS ） 服務遵循共同責任模型。如需 AWS 服務安全性資訊，請參閱AWS 服務安全性文件頁面，以及AWS 合規計畫在 AWS 合規工作範圍內的服務。

強制執行 的最小版本 TLS

> **Important**
>
> AWS SDK for JavaScript v2 會自動交涉指定 AWS 服務端點支援的最高層級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。

若要在與服務通訊時增加安全性 AWS ， AWS SDK for JavaScript 請將 設定為使用 TLS 1.2 或更新版本。

Transport Layer Security （ TLS ） 是 Web 瀏覽器和其他應用程式所使用的通訊協定，可確保透過網路交換資料的隱私權和完整性。

在 Node.js TLS中驗證和強制執行

當您 AWS SDK for JavaScript 搭配 Node.js 使用 時，基礎 Node.js 安全層會用來設定 TLS版本。

Node.js 12.0.0 及更新版本使用支援 1.3 的 OpenSSL 1.1.1b TLS 最低版本。 AWS SDK for JavaScript v3 預設為使用 TLS1.3，但視需要預設為較低的版本。

驗證 OpenSSL 和 的版本 TLS

若要在電腦上取得 Node.js 使用的 OpenSSL 版本，請執行下列命令。

```
node -p process.versions
```

清單中的 OpenSSL 版本是 Node.js 使用的版本，如下列範例所示。

```
openssl: '1.1.1b'
```

若要在電腦上取得 Node.js TLS使用的 版本，請啟動 Node shell 並依序執行下列命令。

```
> var tls = require("tls");
> var tlsSocket = new tls.TLSSocket();
> tlsSocket.getProtocol();
```

最後一個命令會輸出 TLS版本，如下列範例所示。

```
'TLSv1.3'
```

Node.js 預設為使用此版本的 TLS，並嘗試在呼叫失敗TLS時交涉另一個版本的 。

強制執行 的最小版本 TLS

Node.js 會在呼叫失敗TLS時交涉 的版本。您可以在此交涉期間強制執行最低允許TLS版本，無論是從命令列執行指令碼，還是在 JavaScript 程式碼中按請求執行指令碼。

若要從命令列指定最低TLS版本，您必須使用 Node.js 11.0.0 版或更新版本。若要安裝特定的 Node.js 版本，請先使用 Node Version Manager 安裝和更新中找到的步驟，安裝 Node Version Manager (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
```

Enforcing TLS 1.2

若要強制執行 TLS 1.2 是允許的最小版本，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 程式碼中的特定請求指定最低允許TLS版本，請使用 `httpOptions` 參數來指定通訊協定，如下列範例所示。

```
const https = require("https");
const {NodeHttpHandler} = require("@aws-sdk/node-http-handler");
const {DynamoDBClient} = require("@aws-sdk/client-dynamodb");

const client = new DynamoDBClient({
    region: "us-west-2",
    requestHandler: new NodeHttpHandler({
        httpsAgent: new https.Agent(
            {
                secureProtocol: 'TLSv1_2_method'
            }
        )
    })
});
```

Enforcing TLS 1.3

若要強制執行 TLS 1.3 是允許的最小版本，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 程式碼中的特定請求指定最低允許TLS版本，請使用 `httpOptions` 參數來指定通訊協定，如下列範例所示。

```
const https = require("https");
const {NodeHttpHandler} = require("@aws-sdk/node-http-handler");
const {DynamoDBClient} = require("@aws-sdk/client-dynamodb");
```

```
const client = new DynamoDBClient({
    region: "us-west-2",
    requestHandler: new NodeHttpHandler({
        httpsAgent: new https.Agent(
            {
                secureProtocol: 'TLSv1_3_method'
            }
        )
    })
});
```

在瀏覽器指令碼TLS中驗證和強制執行

當您在瀏覽器指令碼 JavaScript 中使用 SDK 的 時，瀏覽器設定會控制所使用的 TLS 版本。瀏覽器TLS使用的 版本無法由指令碼探索或設定，且必須由使用者設定。若要驗證和強制執行瀏覽器指令碼TLS中使用的 版本，請參閱特定瀏覽器的說明。

Microsoft Internet Explorer

1. 開啟 Internet Explorer 。
2. 從選單列中，選擇工具 - 網際網路選項 - 進階索引標籤。
3. 向下捲動至安全類別，手動勾選使用 1.2 TLS 的選項方塊。
4. 按一下 OK (確定)。
5. 關閉瀏覽器並重新啟動 Internet Explorer。

Microsoft Edge

1. 在 Windows 選單搜尋方塊中，輸入 *Internet options*.
2. 在最佳相符項 下，按一下網際網路選項 。
3. 在網際網路屬性視窗中的進階索引標籤上，向下捲動至安全區段。
4. 勾選使用者 TLS 1.2 核取方塊。
5. 按一下 OK (確定)。

Google Chrome

1. 開啟 Google Chrome 。

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. 向下捲動至系統區段，然後按一下開啟代理設定...。
5. 選取進階索引標籤。
6. 向下捲動至安全類別，手動勾選使用 1.2 TLS 的選項方塊。
7. 按一下 OK (確定)。
8. 關閉瀏覽器並重新啟動 Google Chrome。

Mozilla Firefox

1. 開啟 Firefox 。
2. 在地址列中，輸入 about：config，然後按 Enter。
3. 在搜尋欄位中，輸入 tls 。尋找並按兩下 security.tls.version.min 的項目。
4. 將整數值設定為 3，強制將通訊協定 1.2 TLS 設為預設值。
5. 按一下 OK (確定)。
6. 關閉瀏覽器並重新啟動 Mozilla Firefox。

Apple Safari

沒有啟用SSL通訊協定的選項。如果您使用 Safari 第 7 版或更新版本，則會自動啟用 TLS 1.2。

其他資源

下列的連結針對 AWS SDK for JavaScript，提供了您可以使用的其他資源。

AWS軟體開發套件和工具參考指南

所以此AWS軟體開發套件和工具參考指南還包含許多設置，功能和其他常見的基礎概念AWS軟體開發套件。

JavaScript 軟體開發套件

您可以就適用於的開發套件使用者關注的事項找到相關的問題和討論 JavaScript 中的JavaScript 軟體開發套件。

JavaScript 上的開發套件和開發人員指南 GitHub

上有多個儲存庫 GitHub 適用於的開發套件 JavaScript。

- 目前的 SDK，適用於 JavaScript 可於使用軟體開發套件。
- 適用於的開發套件 JavaScript 在開發人員指南 (本文件) 專屬的 Markdown 格式版本文件儲存庫。
- 本指南隨附的部分範本程式碼可從軟體開發套件範本程式碼儲存庫取得。

JavaScript Gitter 上的開發套件

您也可以針對適用於的開發套件，找到相關的問題和討論 JavaScript 中的JavaScript 軟體開發套件Gitter 上。

AWS SDK for JavaScript 的文件歷史記錄

- SDK 版本：請參閱 JavaScript API 參考資料
- 最新的主要文件更新：2022 年 3 月 31 日

文件歷史記錄

下表說明 2018 年 5 月後每個 AWS SDK for JavaScript 版本的重要變更。如需獲得此文件更新的通知，您可以訂閱 RSS 摘要。

變更	描述	日期
強制執行 TLS 的最低版本	已新增 TLS 1.3 的相關資訊。	2022 年 3 月 31 日
從瀏覽器檢視 Amazon S3 儲存貯體中的相片	新增僅限檢視現有相簿中相片的範例。	2019 年 5 月 13 日
在 Node.js 中設定認證，新的認證載入選項	新增從 ECS 登入資料提供者或所設定的登入資料程序載入登入資料的相關資訊。	2019 年 4 月 25 日
使用已設定認證程序的認證	新增從所設定登入資料程序載入登入資料的相關資訊。	2019 年 4 月 25 日
瀏覽器指令碼中的新入門	已重寫瀏覽器指令碼入門，以簡化範例並存取 Amazon Polly 服務以傳送文字和傳回您可在瀏覽器中播放的合成語音。若要了解新增內容，請參閱瀏覽器指令碼入門。	2018年7月14日
新的 Amazon SNS 代碼示例	已新增四個使用 Amazon SNS 的新 Node.js 程式碼範例。如需範例程式碼，請參閱 Amazon SNS 範例。	2018 年 6 月 29 日

Node.js 中開始使用的新功能	重寫後的 Node.js 入門能夠使用更新過的範本程式碼，且會更加詳細說明如何建立 `package.json` 檔案與 Node.js 程式碼本身。若要了解新增內容，請參閱 Node.js 入門。	2018 年 4 月 6 日

舊版更新

下表說明在 2018 年 6 月之前，每個 AWS SDK for JavaScript 版本的重要變更。

變更	描述	日期
新增 AWS Elemental MediaConvert 程式碼範例	已新增三個適用於 AWS Elemental MediaConvert 的新 Node.js 程式碼範例。如需範本程式碼，請參閱 AWS Elemental MediaConvert 範例。	2018 年 5 月 21 日
「新增編輯」 GitHub 按鈕	現在，每個主題的標題都提供了一個按鈕，可將您帶到相同主題的降價版本，以 GitHub 便您進行編輯以提高指南的準確性和完整性。	2018 年 2 月 21 日
新增自訂端點的相關主題	已新增自訂端點格式及用法的相關資訊，該端點可用來提出 API 呼叫。請參閱 指定自訂端點。	2018 年 2 月 20 日
適用於 JavaScript 開發人員的 SDK 指南 GitHub	適用於 JavaScript 開發人員的 SDK 指南在其自己的文件存放庫中提供降價格式。您可以張貼希望該指南解決的問題，或	2018 年 2 月 16 日

變更	描述	日期
	是提交提取要求以提交建議的變更。	
新的 Amazon DynamoDB 代碼示例	已新增一個用於使用文件用戶端更新 DynamoDB 表格的新 Node.js 程式碼範例。如需範本程式碼，請參閱 使用 DynamoDB 用戶端。	2018 年 2 月 14 日
新增 AWS Cloud9 相關主題	已新增主題，其會說明如何使用 AWS Cloud9 開發、除錯瀏覽器與 Node.js 程式碼。請參閱 搭配使用 AWS Cloud9 與 AWS SDK for JavaScript。	2018 年 2 月 5 日
新增軟體開發套件記錄功能相關主題	已新增說明如何記錄使用 SDK 進行的 API 呼叫的主題，包括使用協力廠商記錄器的相關資訊。 JavaScript 請參閱 記錄 AWS SDK for JavaScript 通話。	2018 年 2 月 5 日
更新區域設定相關主題	已更新並擴充主題，其會說明如何設定用於軟體開發套件的區域，包括該區域設定優先順序的相關資訊。請參閱 設定 AWS 區域。	2017 年 12 月 12 日
新的 Amazon SES 代碼示例	包含 SDK 程式碼範例的區段已更新，其中包含五個使用 Amazon SES 的新範例。如需這些程式碼範例的詳細資訊，請參閱 Amazon 簡單電子郵件服務。	2017 年 11 月 9 日

變更	描述	日期
改善可用性	已根據最近的可用性測試進行多項變更，以便改善文件可用性。 • 更清楚地標識程式碼範例，指出其是以瀏覽器或 Node.js 執行程序為目標。 • TOC 連結不會再立即跳轉至其他 Web 內容，包括 API 參考。 • 在 [入門] 區段中包含更多關於取得AWS認證的詳細資訊的連結。 • 提供更多有關使用軟體開發套件所需常見 Node.js 功能的資訊。如需詳細資訊，請參閱Node.js 的考量。	2017 年 8 月 9 日
新 DynamoDB 程式碼範例	包含 SDK 程式碼範例的區段已更新為重新撰寫前兩個範例，並新增三個使用 DynamoDB 的全新範例。如需這些程式碼範例的詳細資訊，請參閱 Amazon DynamoDB 範例。	2017 年 6 月 21 日
新的 IAM 程式碼範例	包含 SDK 程式碼範例的區段已更新，其中包含五個使用 IAM 的新範例。如需這些程式碼範例的詳細資訊，請參閱 AWSIAM 範例。	2016 年 12 月 23 日

變更	描述	日期
新的 CloudWatch 和 Amazon SQS 代碼示例	包含 SDK 程式碼範例的區段已更新，其中包含使用 Amazon SQS CloudWatch 和使用 Amazon SQS 的新範例。如需這些程式碼範例的詳細資訊，請參閱 Amazon CloudWatch 示例 和 Amazon SQS 範例。	2016 年 12 月 20 日
新的 Amazon EC2 代碼示例	包含 SDK 程式碼範例的區段已更新，其中包含五個使用 Amazon EC2 的新範例。如需這些程式碼範例的詳細資訊，請參閱 Amazon EC2 範例。	2016 年 12 月 15 日
更容易看見受支援瀏覽器的清單	SDK 支援的瀏覽器清單 (先前在「必要條件」主題中找到) 已指定其自己的主題，以使其在目錄中更加明顯。 JavaScript	2016 年 11 月 16 日
新開發人員指南的初版	先前的開發人員指南現在已棄用。新的開發人員指南經過重新整理，因此使用者能更容易找到所需資訊。當 Node.js 或瀏覽器 JavaScript 案例呈現特殊考量時，會將這些考量識別為適當。此外，該指南還會提供額外的程式碼範例，而這些範例皆經過組織整理，使用者可更快速簡單地找到所需內容。	2016 年 10 月 28 日