



軟體開發套件第 2 版的開發人員指南

適用於 JavaScript 的 AWS SDK



適用於 JavaScript 的 AWS SDK: 軟體開發套件第 2 版的開發人員指南

Copyright © 2025 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon 的商標和商業外觀不得用於任何非 Amazon 的產品或服務，也不能以任何可能造成客戶混淆、任何貶低或使 Amazon 名譽受損的方式使用 Amazon 的商標和商業外觀。所有其他非 Amazon 擁有的商標均為其各自擁有者的財產，這些擁有者可能附屬於 Amazon，或與 Amazon 有合作關係，亦或受到 Amazon 贊助。

Table of Contents

.....	ix
什麼是 適用於 JavaScript 的 AWS SDK ?	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 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	2
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：建立 Amazon Cognito 身分集區	5
步驟 2：將政策新增至已建立的 IAM 角色	6
步驟 3：建立 HTML 頁面	6
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	10
Node.js 入門	11
使用案例	11
先決條件任務	11
步驟 1：安裝 SDK 和相依性	12
步驟 2：設定您的登入資料	12
步驟 3：建立專案的套件 JSON	13
步驟 4：編寫 Node.js 程式碼	14
步驟 5：執行範例	15
設定適用於 JavaScript 的 SDK	16
先決條件	16
設定 AWS Node.js 環境	16
支援的 Web 瀏覽器	17
安裝 開發套件	18
使用 Bower 進行安裝	19
載入軟體開發套件	19
從第 1 版進行升級	20

自動轉換輸入/輸出上的 Base64 和時間戳記類型	20
將 response.data.RequestId 遷移至 response.requestId	21
公開的包裝函數元素	21
捨棄的用戶端屬性	26
設定適用於 JavaScript 的 SDK	27
使用全域組態物件	27
設定全域組態	28
依服務設定組態	29
固定組態資料	30
設定 AWS 區域	30
在用戶端類別建構子	30
使用全域組態物件	30
使用環境變數	31
使用共用組態檔	31
設定區域的優先順序	31
指定自訂端點	32
端點字串格式	32
ap-northeast-3 區域的端點	32
MediaConvert 的端點	32
使用 進行 SDK 身分驗證 AWS	33
啟動 AWS 存取入口網站工作階段	34
更多身分驗證資訊	34
設定 登入資料	35
登入資料的最佳實務	35
在 Node.js 設定登入資料	36
在 Web 瀏覽器中設定登入資料	40
鎖定 API 版本	49
取得 API 版本	49
Node.js 的考量	50
使用內建 Node.js 模組	50
使用 NPM 套件	50
在 Node.js 中設定 maxSocket	51
在 Node.js 中重複使用 Keep-Alive 的連線	52
為 Node.js 設定代理	53
在 Node.js 中註冊憑證	53
瀏覽器指令碼考量	54

建立適用於瀏覽器的軟體開發套件	54
跨來源資源分享 (CORS)	57
使用 Webpack 進行綁定	61
安裝 Webpack	61
設定 Webpack	61
執行 Webpack	62
使用 Webpack 配套	63
匯入個別服務	64
綁定 Node.js	64
使用 服務	66
建立和呼叫服務物件	67
要求個別服務	67
建立服務物件	68
鎖定服務物件的 API 版本	69
指定服務物件參數	69
記錄 適用於 JavaScript 的 AWS SDK 通話	70
使用第三方記錄器	70
非同步呼叫服務	71
管理非同步呼叫	71
使用回呼函數	72
使用請求物件事件接聽程式	74
使用 async/await	79
使用 Promise	79
使用回應物件	81
在回應物件中存取傳回的資料	82
瀏覽傳回的資料分頁	83
存取來自回應物件的錯誤資訊	83
存取來源請求物件	83
使用 JSON	84
做為服務物件參數的 JSON	84
傳回 JSON 格式的資料	85
重試	86
指數退避型重試行為	86
適用於 JavaScript 的 SDK 程式碼範例	89
Amazon CloudWatch 範例	89
在 Amazon CloudWatch 中建立警示	90

在 Amazon CloudWatch 中使用警示動作	94
從 Amazon CloudWatch 取得指標	98
傳送事件到 Amazon CloudWatch Events	101
在 Amazon CloudWatch Logs 中使用訂閱篩選條件	106
Amazon DynamoDB 範例	110
在 DynamoDB 中建立和使用資料表	111
在 DynamoDB 中讀取和寫入單一項目	115
在 DynamoDB 中讀取和寫入批次中的項目	119
查詢和掃描 DynamoDB 資料表	122
使用 DynamoDB 文件用戶端	125
Amazon EC2 範例	131
建立 Amazon EC2 執行個體	132
管理 Amazon EC2 執行個體	134
使用 Amazon EC2 金鑰對	140
搭配 Amazon EC2 使用區域和可用區域	144
在 Amazon EC2 中使用安全群組	146
在 Amazon EC2 中使用彈性 IP 地址	150
MediaConvert 範例	154
取得區域特定的端點	155
建立和管理任務	157
使用任務範本	164
AWS IAM 範例	172
管理 IAM 使用者	173
處理 IAM 政策	178
管理 IAM 存取金鑰	183
處理 IAM 伺服器憑證	188
管理 IAM 帳戶別名	192
Amazon Kinesis 範例	195
使用 Amazon Kinesis 擷取網頁捲動進度	195
Amazon S3 範例	202
Amazon S3 瀏覽器範例	202
Amazon S3 Node.js 範例	230
Amazon SES 範例	250
管理實體	250
使用電子郵件範本	255
使用 Amazon SES 傳送電子郵件	261

使用 IP 地址篩選條件	267
使用接收規則	271
Amazon SNS 範例	276
管理主題	276
發佈訊息至主題	282
管理訂閱	283
傳送簡訊	289
Amazon SQS 範例	295
在 Amazon SQS 中使用佇列	296
在 Amazon SQS 中傳送和接收訊息	300
在 Amazon SQS 中管理可見性逾時	303
在 Amazon SQS 中啟用長輪詢	305
在 Amazon SQS 中使用無效字母佇列	309
教學課程	312
教學課程：在 Amazon EC2 執行個體上設定 Node.js	312
先決條件	312
程序	312
建立 Amazon Machine Image	314
相關資源	314
API 參考和變更記錄	315
GitHub 上的軟體開發套件變更記錄	315
遷移至 v3	316
安全	317
資料保護	317
身分和存取權管理	318
目標對象	318
使用身分驗證	319
使用政策管理存取權	321
AWS 服務 如何使用 IAM	323
對 AWS 身分和存取進行故障診斷	324
合規驗證	325
恢復能力	326
基礎設施安全性	326
強制執行 TLS 最低版本	327
在 Node.js 中驗證和強制執行 TLS	327
在瀏覽器指令碼中驗證並強制執行 TLS	330

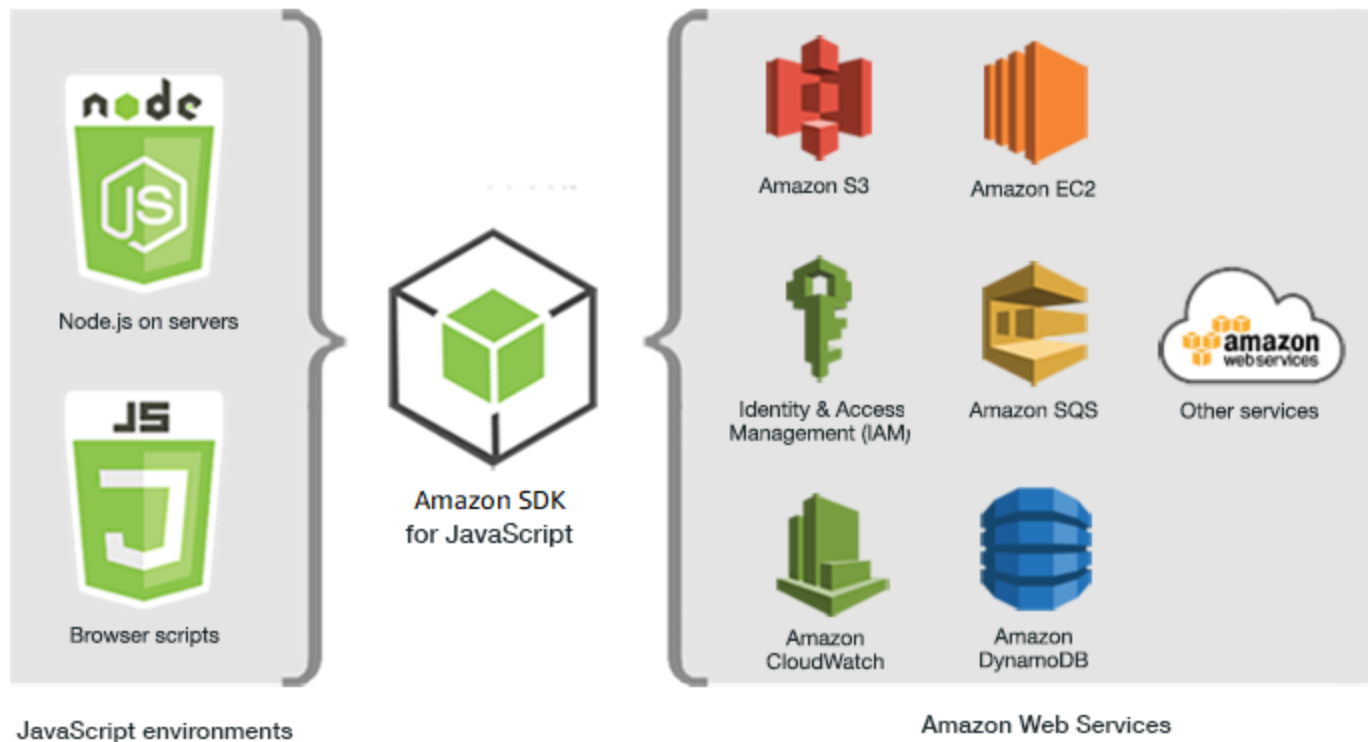
其他資源	332
AWS SDKs和工具參考指南	332
JavaScript 開發套件論壇	332
GitHub 上的 JavaScript 開發套件和開發人員指南	332
Gitter 上的 JavaScript SDK	332
文件歷史記錄	333
文件歷史記錄	333
舊版更新	334

我們已[宣布](#)即將end-of-support。適用於 JavaScript 的 AWS SDK 我們建議您遷移至 [適用於 JavaScript 的 AWS SDK v3](#)。如需日期、其他詳細資訊和如何遷移的資訊，請參閱連結的公告。

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 適用於 JavaScript 的 AWS SDK ？

為 AWS 服務適用於 JavaScript 的 AWS SDK提供 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。若要了解 目前支援哪些服務 適用於 JavaScript 的 AWS SDK，請參閱 <https://github.com/aws/aws-sdk-js/blob/master/SERVICES.md>。如需 GitHub 上適用於 JavaScript 的 SDK 的相關資訊，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 SDKs和工具維護政策](#)
- [AWS SDKs和工具版本支援矩陣](#)

使用軟體開發套件搭配 Node.js

Node.js 是一個跨平台執行時間環境，其可用來執行伺服器端 JavaScript 應用程式。您可以在 Amazon EC2 執行個體上設定 Node.js 以在伺服器上執行。您也可以使用 Node.js 撰寫隨需 AWS Lambda 函數。

使用 Node.js 的開發套件與您在 web 瀏覽器使用 JavaScript 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 AWS Amplify 使用 SDK

對於以瀏覽器為基礎的 Web、行動和混合式應用程式，您也可以使用 [AWS GitHub 上的 Amplify Library](#)，[該程式庫](#)擴展了適用於 JavaScript 的 SDK，提供宣告式界面。

Note

AWS Amplify 等架構可能無法提供與適用於 JavaScript 的 SDK 相同的瀏覽器支援。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 瀏覽器都支援執行 JavaScript。Web 瀏覽器中執行的 JavaScript 程式碼通常稱為用戶端 JavaScript。

在 Web 瀏覽器中使用適用於 JavaScript 的 SDK 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。

如需支援的瀏覽器清單 適用於 JavaScript 的 AWS SDK，請參閱 [支援的 Web 瀏覽器](#)。

常用案例

在瀏覽器指令碼中使用適用於 JavaScript 的 SDK 可讓您實現許多令人信服的使用案例。以下是您可以使用適用於 JavaScript 的 SDK 存取各種 Web 服務，在瀏覽器應用程式中建置的幾個想法。

- 將自訂主控台建置到 AWS 服務，您可以在其中跨區域和服務存取和結合功能，以最符合您的組織或專案需求。

- 使用 Amazon Cognito Identity 來啟用已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 和其他的第三方身分驗證。
- 使用 Amazon Kinesis 即時處理點擊串流或其他行銷資料。
- 使用 Amazon DynamoDB 進行無伺服器資料持久性，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 封裝您可以從瀏覽器指令碼叫用的專屬邏輯，而無需下載並向使用者公開您的智慧財產權。

關於範例

您可以在程式碼範例程式庫中瀏覽適用於 JavaScript 的 SDK 範例。 [AWS](#)

入門 適用於 JavaScript 的 AWS SDK

適用於 JavaScript 的 AWS SDK 提供瀏覽器指令碼或 Node.js 中 Web 服務的存取權。本節有兩個入門練習，說明如何在每一個 JavaScript 環境中使用適用於 JavaScript 的 SDK。

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 Identity 從瀏覽器指令碼存取 AWS 服務。
- 如何使用 Amazon Polly 將文字轉換為合成語音。
- 如何使用 presigner 物件建立預先簽章的 URL。

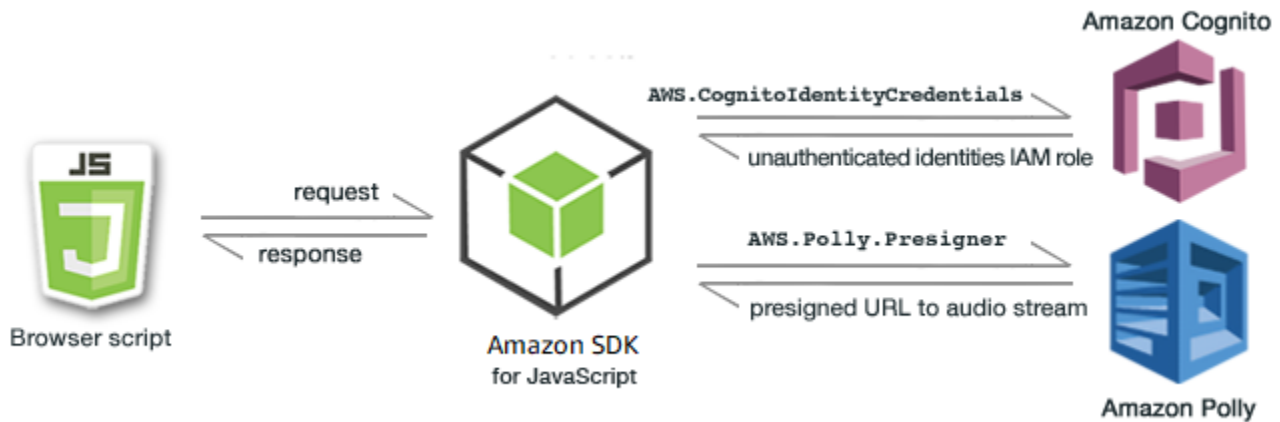
使用案例

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 來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](#)。

此範例示範如何設定和執行簡單的瀏覽器指令碼，該指令碼會接受您輸入的文字、將該文字傳送至 Amazon Polly，然後傳回文字的合成音訊 URL 供您播放。瀏覽器指令碼使用 Amazon Cognito Identity 來提供存取 AWS 服務所需的登入資料。您會在瀏覽器指令碼中看到載入和使用適用於 JavaScript 的 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。



瀏覽器指令碼使用適用於 JavaScript 的開發套件，使用這些 APIs 來合成文字：

- [AWS.CognitoIdentityCredentials](#) 建構函數
- [AWS.Polly.Presigner](#) 建構函數
- [getSynthesizeSpeechUrl](#)

步驟 1：建立 Amazon Cognito 身分集區

在本練習中，您會建立並使用 Amazon Cognito 身分集區，為 Amazon Polly 服務提供未經驗證的瀏覽器指令碼存取權。建立身分集區也會建立兩個 IAM 角色，一個用於支援身分提供者驗證的使用者，另一個用於支援未經驗證的訪客使用者。

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。如需新增 Amazon Cognito 身分集區的詳細資訊，請參閱《Amazon Cognito 開發人員指南》中的[教學課程：建立身分集區](#)。

建立 Amazon Cognito 身分集區

1. 登入 AWS Management Console，並在 <https://console.aws.amazon.com/cognito/> 主控台開啟 Cognito 主控台。
2. 在左側導覽窗格中，選擇身分集區。
3. 選擇 建立身分池。
4. 在設定身分集區信任中，選擇訪客存取以進行使用者身分驗證。
5. 在設定許可中，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. 在設定屬性中，在身分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. 請注意身分集區 ID 和新建立的 Amazon Cognito 身分集區的區域。您需要這些值來取代 中的 `IDENTITY_POOL_ID` 和 `REGION`[步驟 4：編寫瀏覽器指令碼](#)。

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

步驟 2：將政策新增至已建立的 IAM 角色

若要啟用瀏覽器指令碼存取 Amazon Polly 以進行語音合成，請使用為 Amazon Cognito 身分集區建立的未驗證 IAM 角色。這需要您將 IAM 政策新增至角色。如需修改 IAM 角色的詳細資訊，請參閱《IAM 使用者指南》中的[修改角色許可政策](#)。

將 Amazon 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，並在 <https://console.aws.amazon.com/iam/> : //www. 開啟 IAM 主控台。
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），然後選擇許可索引標籤。
4. 選擇新增許可，然後選擇連接政策。
5. 在此角色的新增許可頁面中，尋找並選取 AmazonPollyReadOnly 的核取方塊。

Note

您可以使用此程序來啟用對任何 AWS 服務的存取。

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，您就可以建置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

建立瀏覽器指令碼時，首先要做的是在頁面的 `<script>` 元素後面新增 `<audio>` 元素，以包含適用於 JavaScript 的 SDK。若要尋找目前的 `SDK_VERSION_NUMBER`，請參閱《API 參考指南》中適用於 JavaScript 的 SDK 的 API 參考。[適用於 JavaScript 的 AWS SDK](#)

```
<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。設定 SDK AWS 的區域和登入資料。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。

若要使用 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。將下列程式碼中的 `IDENTITY_POOL_ID` 和 `REGION` 取代為中記下的值[步驟 1：建立 Amazon Cognito 身分集區](#)。

```
<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 會將合成語音傳回為音訊串流。在瀏覽器中播放該音訊的最簡單方法是讓 Amazon Polly 在預先簽章的 URL 上提供音訊，然後您可以在網頁中將 src 設定為 <audio> 元素的屬性。

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。

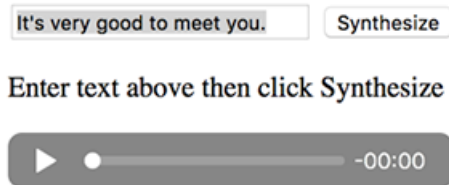
```
// Create the Polly service object and presigner object
var polly = new AWS.Polly({apiVersion: '2016-06-10'});
var signer = new AWS.Polly.Presigner(speechParams, polly)

// 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，您也能夠在 [GitHub 上](#)取得。

```
<!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

以下是此應用程式的變化，您可以用來進一步探索在瀏覽器指令碼中使用適用於 JavaScript 的 SDK。

- 實驗其他音訊輸出格式。
- 新增 選項，以選取 Amazon Polly 提供的任何各種語音。
- 整合 Facebook 或 Amazon 等身分提供者，以搭配已驗證的 IAM 角色使用。

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 Simple Storage Service (Amazon S3) 服務物件。
- 如何建立 Amazon S3 儲存貯體，並將物件上傳至該儲存貯體。

使用案例

此範例示範如何設定和執行簡單的 Node.js 模組，以建立 Amazon S3 儲存貯體，然後將文字物件新增至其中。

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，您可以將其納入儲存貯體名稱。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](https://nodejs.org/en/download/current/) 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

- [步驟 1：安裝 SDK 和相依性](#)
- [步驟 2：設定您的登入資料](#)
- [步驟 3：建立專案的套件 JSON](#)

- [步驟 4：編寫 Node.js 程式碼](#)
- [步驟 5：執行範例](#)

步驟 1：安裝 SDK 和相依性

您可以使用 npm (Node.js 套件管理員) 安裝適用於 JavaScript 的 SDK 套件。 <https://www.npmjs.com>

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中安裝適用於 JavaScript 的 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。

如需安裝 Node.js 套件的詳細資訊，請參閱 npm (Node.js 套件管理員) 網站上的 [在本機下載和安裝](#) 套件和建立 Node.js 模組。 <https://docs.npmjs.com/getting-started/creating-node-modules>
<https://www.npmjs.com> 如需下載和安裝的資訊 適用於 JavaScript 的 AWS SDK，請參閱 [安裝適用於 JavaScript 的 SDK](#)。

步驟 2：設定您的登入資料

您需要提供登入資料給 AWS，以便只有您的帳戶及其資源可以由 SDK 存取。如需取得帳戶登入資料的詳細資訊，請參閱 [使用進行 SDK 身分驗證 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：建立專案的套件 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。如需在 Node.js 專案package.json中使用的詳細資訊，請參閱[建立 package.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 [GitHub 的此處](#) 找到說明這些相依性的 JSON 範例檔案。

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，新增 `require` 函數呼叫以包含適用於 JavaScript 的 SDK 和 `uuid` 模組，以便供您使用。

建置唯一儲存貯體名稱，用於透過將唯一 ID 值附加至可辨識的字首來建立 Amazon S3 儲存貯體，在此情況下為 `'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 [GitHub 上](#) 找到這個範本程式碼。

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。

設定適用於 JavaScript 的 SDK

本節中的主題說明如何安裝適用於 JavaScript 的 SDK，以用於 Web 瀏覽器和 Node.js。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 原生開發人員應使用 AWS Amplify 來建立新專案 AWS。如需詳細資訊，請參閱 [aws-sdk-react-native](#) 存檔。

主題

- [先決條件](#)
- [安裝適用於 JavaScript 的 SDK](#)
- [載入適用於 JavaScript 的 SDK](#)
- [從第 1 版升級適用於 JavaScript 的 SDK](#)

先決條件

使用 之前 適用於 JavaScript 的 AWS SDK，請先判斷您的程式碼是否需要在 Node.js 或 Web 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 AWS Node.js 環境](#)
- [支援的 Web 瀏覽器](#)

設定 AWS Node.js 環境

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

- 選擇預先安裝 Node.js 的 Amazon Machine Image (AMI)，並使用該 AMI 建立 Amazon EC2 執行個體。建立 Amazon EC2 執行個體時，請從 [選擇您的 AMI AWS Marketplace](#)。搜尋 Node AWS Marketplace .js 並選擇 AMI 選項，其中包含預先安裝的 Node.js 版本 (32 位元或 64 位元)。
- 建立 Amazon EC2 執行個體，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](#)。
- 使用 建立無伺服器環境 AWS Lambda，以執行 Node.js 做為 Lambda 函數。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 應用程式部署至 AWS Elastic Beanstalk。如需搭配 Elastic Beanstalk 使用 Node.js 的詳細資訊，請參閱《AWS Elastic Beanstalk 開發人員指南》中的[將 Node.js 應用程式部署至 AWS Elastic Beanstalk](#)。

支援的 Web 瀏覽器

適用於 JavaScript 的 SDK 支援所有現代 Web 瀏覽器，包括這些最低版本：

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

AWS Amplify 等架構可能無法提供與適用於 JavaScript 的 SDK 相同的瀏覽器支援。如需詳細資訊，請參閱架構文件。

安裝適用於 JavaScript 的 SDK

是否及如何安裝 適用於 JavaScript 的 AWS SDK 取決於程式碼是在 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。若要了解 目前支援哪些服務 適用於 JavaScript 的 AWS SDK，請參閱 <https://github.com/aws/aws-sdk-js/blob/master/SERVICES.md>

Node

安裝 適用於 JavaScript 的 AWS SDK for Node.js 的偏好方法是使用 [Node.js 套件管理員 npm](#)。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 SDK 套件。託管 SDK 套件支援強制執行跨來源資源共用 (CORS) AWS 的服務子集。如需詳細資訊，請參閱[載入適用於 JavaScript 的 SDK](#)。

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 適用於 JavaScript 的 AWS SDK GitHub 下載目前的精簡和非精簡可分發版本，網址為：

<https://github.com/aws/aws-sdk-js/tree/master/dist>

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
```

載入適用於 JavaScript 的 SDK

載入適用於 JavaScript 的 SDK 的方式取決於您要載入它以在 Web 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。若要了解 目前支援哪些服務 適用於 JavaScript 的 AWS SDK，請參閱 <https://github.com/aws/aws-sdk-js/blob/master/SERVICES.md>

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 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_VERSION_NUMBER`，請參閱《API 參考指南》中適用於 JavaScript 的 SDK 的 API 參考。 [適用於 JavaScript 的 AWS SDK](#)

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。

從第 1 版升級適用於 JavaScript 的 SDK

以下備註可協助您將適用於 JavaScript 的 SDK 從第 1 版升級至第 2 版。

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"  
});
```

將 response.data.RequestId 遷移至 response.requestId

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	作業
AWS.ElastiCache	authorizeCacheSecurityGroupIngress
	createCacheCluster
	createCacheParameterGroup
	createCacheSecurityGroup
	createCacheSubnetGroup
	createReplicationGroup
	deleteCacheCluster
	deleteReplicationGroup
	describeEngineDefaultParameters
	modifyCacheCluster
	modifyCacheSubnetGroup
	modifyReplicationGroup
	purchaseReservedCacheNodesOffering
	rebootCacheCluster
	revokeCacheSecurityGroupIngress
AWS.RDS	addSourceIdentifierToSubscription
	authorizeDBSecurityGroupIngress
	copyDBSnapshot createDBInstance

用戶端類別	作業
	<code>createDBInstanceReadReplica</code>
	<code>createDBParameterGroup</code>
	<code>createDBSecurityGroup</code>
	<code>createDBSnapshot</code>
	<code>createDBSubnetGroup</code>
	<code>createEventSubscription</code>
	<code>createOptionGroup</code>
	<code>deleteDBInstance</code>
	<code>deleteDBSnapshot</code>
	<code>deleteEventSubscription</code>
	<code>describeEngineDefaultParameters</code>
	<code>modifyDBInstance</code>
	<code>modifyDBSubnetGroup</code>
	<code>modifyEventSubscription</code>
	<code>modifyOptionGroup</code>
	<code>promoteReadReplica</code>
	<code>purchaseReservedDBInstancesOffering</code>
	<code>rebootDBInstance</code>
	<code>removeSourceIdentifierFromSubscription</code>
	<code>restoreDBInstanceFromDBSnapshot</code>

用戶端類別	作業
	<code>restoreDBInstanceToPointInTime</code>
	<code>revokeDBSecurityGroupIngress</code>

用戶端類別	作業
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	作業
	<code>purchaseReservedNodeOffering</code>
	<code>rebootCluster</code>
	<code>restoreFromClusterSnapshot</code>
	<code>revokeClusterSecurityGroupIngress</code>
	<code>revokeSnapshotAccess</code>
	<code>rotateEncryptionKey</code>

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。

下列程式碼與適用於 JavaScript 的 SDK 第 1 版搭配使用：

```
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(...)
```

設定適用於 JavaScript 的 SDK

在使用適用於 JavaScript 的 SDK 來使用 API 叫用 Web 服務之前，您必須設定 SDK。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，您可能還必須設定資源的 AWS 許可。例如，您可以限制對 Amazon S3 儲存貯體的存取，或限制 Amazon DynamoDB 資料表進行唯讀存取。

[AWS SDKs和工具參考指南](#)也包含許多 AWS SDKs 中常見的設定、功能和其他基本概念。

本節中的主題說明各種設定適用於 Node.js 的 JavaScript 開發套件以及在 Web 瀏覽器中執行的 JavaScript 的方式。

主題

- [使用全域組態物件](#)
- [設定 AWS 區域](#)
- [指定自訂端點](#)
- [使用 進行 SDK 身分驗證 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
<code>credentials</code>	「必要」。指定判斷服務和資源存取所用的登入資料。
<code>region</code>	「必要」。指定為服務提出請求的區域。
<code>maxRetries</code>	選用。指定特定請求重試的次數上限。
<code>logger</code>	選用。指定要將偵錯資訊寫入其中的記錄器物件。
<code>update</code>	選用。使用新值更新目前組態。

如需組態物件的詳細資訊，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，如果您在 `config` 檔案中正確設定區域，則透過將 `AWS_SDK_LOAD_CONFIG` 環境變數設定為任何值並呼叫的靜態 `region` 屬性，即可取得該值 `AWS.config`：

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

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

依服務設定組態

您在適用於 JavaScript 的 SDK 中使用的每個服務，都是透過屬於該服務 API 的服務物件來存取。例如，若要存取 Amazon S3 服務，您可以建立 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。如需服務特定組態的詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK 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

區域是相同地理區域中的一組具名 AWS 資源。區域的範例為 `us-east-1`，即美國東部（維吉尼亞北部）區域。您可以在設定適用於 JavaScript 的 SDK 時指定區域，讓 SDK 存取該區域中的資源。某些服務僅在特定區域提供。

根據預設，適用於 JavaScript 的 SDK 不會選取區域。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 程式碼中設定區域，請如此處所示，更新 `AWS.Config` 全域組態物件。

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

如需每個區域中目前區域和可用服務的詳細資訊，請參閱 [AWS 區域和端點](#) AWS 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，適用於 JavaScript 的 SDK 會讀取並使用它。

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，則適用於 JavaScript 的 SDK 會在載入時自動搜尋 `config` 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
```

如需使用共用組態和登入資料檔案的詳細資訊，請參閱 AWS Command Line Interface 《使用者指南》中的 [從共用登入資料檔案中在 Node.js 中載入登入資料](#) 或 [組態和登入資料檔案](#)。

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，且共用的登入資料檔案 (`~/.aws/credentials` 或 指定的路徑 `AWS_SHARED_CREDENTIALS_FILE`) 包含已設定設定檔的區域，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，且組態檔案 (`~/.aws/config` 或 指示的路徑 `AWS_CONFIG_FILE`) 包含已設定設定檔的區域，則會使用該區域。

指定自訂端點

對適用於 JavaScript 的 SDK 中的 API 方法的呼叫會對服務端點 URIs 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。因此，此區域的 Amazon EC2 端點

`ec2.ap-northeast-3.amazonaws.com`

MediaConvert 的端點

您需要建立自訂端點以與 MediaConvert 搭配使用。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 MediaConvert 使用自訂端點的範例。

```
// 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 MediaConvert 的詳細資訊，請參閱《API 參考》或[AWS Elemental MediaConvert 《使用者指南》](#)中的 [AWS.MediaConvert](#) 類別。

使用 進行 SDK 身分驗證 AWS

使用 進行開發 AWS 時，您必須建立程式碼如何向 進行身分驗證 AWS 服務。您可以根據環境和您可用的存取權，以不同的方式設定 AWS 資源的程式設計 AWS 存取。

若要選擇您的身分驗證方法，並針對 SDK 進行設定，請參閱 AWS SDKs 和工具參考指南中的[身分驗證和存取](#)。

我們建議在本機開發且未獲得其雇主身分驗證方法的新使用者進行設定 AWS IAM Identity Center。此方法包括安裝 AWS CLI 以簡化組態，以及定期登入 AWS 存取入口網站。如果您選擇此方法，您的環境應該會在您完成 AWS SDKs 和工具參考指南中的 [IAM Identity Center 身分驗證](#) 程序後包含下列元素：

- 在執行應用程式之前，AWS CLI 您用來啟動 AWS 存取入口網站工作階段的。
- 共用 [AWSconfig 檔案](#)，其 [default] 設定檔具有一組可從 SDK 參考的組態值。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 檔案會設定 [region](#) 設定。這會設定軟體開發套件用於 AWS 請求 AWS 區域 的預設值。此區域用於未指定使用 區域的 SDK 服務請求。
- SDK 會使用設定檔的 [SSO 字符提供者組態](#)，在傳送請求至 之前取得憑證 AWS。sso_role_name 值是連接到 IAM Identity Center 許可集的 IAM 角色，允許存取應用程式中 AWS 服務 使用的。

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 sso_session 的設定是指具名 [sso-session 區段](#)。sso-session 區段包含啟動 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
```

適用於 JavaScript 的 SDK 不需要將其他套件（例如 SSO 和 SSO0IDC）新增至您的應用程式，即可使用 IAM Identity Center 身分驗證。

啟動 AWS 存取入口網站工作階段

在執行存取的應用程式之前 AWS 服務，您需要 SDK 的作用中 AWS 存取入口網站工作階段，才能使用 IAM Identity Center 身分驗證來解析登入資料。視您設定的工作階段長度而定，您的存取最終將會過期，開發套件將遇到身分驗證錯誤。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
```

如果您遵循指引並設定預設設定檔，則不需要使用 `--profile` 選項呼叫 命令。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經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 Identity Center 帳戶和許可集。

Note

如果您已有作用中的 AWS 存取入口網站工作階段並執行 `aws sso login`，則不需要提供登入資料。

登入程序可能會提示您允許 AWS CLI 存取您的資料。由於 AWS CLI 建置在適用於 Python 的 SDK 之上，因此許可訊息可能包含 `botocore` 名稱的變化。

更多身分驗證資訊

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。它們必須具有身分才能存取您的 AWS 環境和應用程式。屬於您組織成員的人類使用者 - 這表示您是開發人員 - 稱為人力身分。

存取時使用臨時憑證 AWS。您可以使用身分提供者，讓人類使用者擔任提供臨時登入資料的角色，以提供 AWS 帳戶的聯合存取。對於集中式存取管理，我們建議您使用 AWS IAM Identity Center (IAM Identity Center) 來管理對帳戶和這些帳戶中許可的存取。如需更多替代方案，請參閱下列內容：

- 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 [暫時安全登入](#) 資料。
- 若要了解其他適用於 JavaScript 憑證提供者的 SDK，請參閱 SDK AWS SDKs 和工具參考指南中的 [標準化憑證提供者](#)。

設定 登入資料

AWS 使用登入資料來識別誰正在呼叫 服務，以及是否允許存取請求的資源。

不論是在 web 瀏覽器或 Node.js 伺服器上執行，JavaScript 程式碼必須包含有效的登入資料，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。

有幾個方式可以來設定在 web 瀏覽器中 Node.js 和 JavaScript 之間不同的登入資料。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 資料表中的物件，否則請將這些許可設定為唯讀。

如需授予最低權限的詳細資訊，請參閱《IAM 使用者指南》中最佳實務主題的 [授予最低權限](#) 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。硬式編碼登入資料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《》中的 [管理 AWS 存取金鑰的最佳實務](#) 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. JavaScript 開發套件提供的其他登入資料提供者類別

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. 使用共用 AWS 組態檔案或共用登入資料檔案中指定的登入資料程序取得的登入資料。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 AWS Amazon EC2 執行個體的登入資料提供者從 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[Class: AWS.CredentialProviderChain](#)中的 [Class: AWS.Credentials](#)和。

Warning

雖然可以這樣做，但不建議硬式編碼應用程式中的 AWS 登入資料。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 Amazon EC2 的 IAM 角色自動提供登入資料給執行個體。如果您將執行個體設定為使用 IAM 角色，開發套件會自動選取應用程式的 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 SDKs 和工具參考指南中的 [為 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許可的特殊 IAM 角色。此角色稱為執行角色。設定 Lambda 函數時，您必須指定您建立的 IAM 角色做為對應的執行角色。

執行角色提供 Lambda 函數執行 和 呼叫其他 Web 服務所需的登入資料。因此，您不需要提供登入資料給在 Lambda 函數中寫入的 Node.js 程式碼。

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 登入資料保留在 SDKs 和命令列界面所使用的共用檔案中。載入適用於 JavaScript 的 SDK 時，會自動搜尋名為「憑證」的共用登入資料檔案。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 進行 SDK 身分驗證 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 (如有需要) 之後，您可以執行使用 SDK 的 JavaScript 檔案，例如名為 `script.js` 的檔案。

```
$ 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 會自動偵測您環境中設定為變數的 AWS 登入資料，並將其用於 SDK 請求，無需在應用程式中管理登入資料。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`

如需設定環境變數的詳細資訊，請參閱 AWS SDKs 和工具參考指南中的 [環境變數支援](#)。

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" }
```

然後使用下列程式碼：

```
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。若要這樣做，請在共用 AWS 組態檔案或共用登入資料檔案中指定登入資料程序。如果 `AWS_SDK_LOAD_CONFIG` 環境變數設定為任何值，則 SDK 會偏好組態檔案中指定的程序，而不是登入資料檔案中指定的程序（如果有的話）。

如需在共用 AWS 組態檔案或共用登入資料檔案中指定登入資料程序的詳細資訊，請參閱 AWS CLI 命令參考，特別是[從外部程序取得登入資料](#)的相關資訊。

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 Identity 驗證使用者並提供登入資料
2. 使用 Web 聯合身分
3. 在指令碼中將程式碼寫死

Warning

我們不建議硬式編碼指令碼中的 AWS 登入資料。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 Identity 驗證使用者](#)
- [使用 Web 聯合身分來驗證使用者](#)
- [Web 聯合身分範例](#)

使用 Amazon Cognito Identity 驗證使用者

為瀏覽器指令碼取得 AWS 登入資料的建議方法是使用 Amazon Cognito Identity 登入資料物件 `AWS.CognitoIdentityCredentials`。Amazon Cognito 透過第三方身分提供者啟用使用者身分驗證。

若要使用 Amazon Cognito Identity，您必須先在 Amazon Cognito 主控台中建立身分集區。身分集區代表應用程式提供給使用者的身分群組。提供給使用者的身分可唯一識別每個使用者帳戶。Amazon Cognito 身分不是憑證。在 AWS Security Token Service () 中使用 Web 聯合身分支援交換憑證 AWS STS。

Amazon Cognito 可協助您使用 `AWS.CognitoIdentityCredentials` 物件管理多個身分提供者的身分抽象。接著，會將載入的身分交換為在 AWS STS 中的登入資料。

設定 Amazon Cognito 身分登入資料物件

如果您尚未建立身分集區，請在設定之前，在 [Amazon Cognito 主控台](#) 中建立身分集區以與瀏覽器指令碼搭配使用 `AWS.CognitoIdentityCredentials`。建立身分集區的已驗證和未驗證 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提供者來登入應用程式。請務必適當地限制資源許可範圍，以避免從未經授權的使用者授與資源的存取權。

在您以連接的身分提供者設定身分集區之後，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` 屬性是身分提供者名稱與這些身分提供者的身分權杖的對應。您從身分提供者取得權杖的方式，取決於您使用的供應商。例如，如果 Facebook 是您的身分提供者之一，您可以使用 `FB.loginFacebook` 開發套件的 [FB.loginFacebook](#) 函數來取得身分提供者權杖：

```
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 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提供者登入，也能存取您的資源。這個程度的存取能在使用者登入前就顯示內容，非常有用。每個未驗證的使用者在 Amazon Cognito 中都有唯一的身分，即使他們尚未個別登入和驗證。

最初未驗證的使用者

使用者通常會以未經驗證的角色開始，您會為該角色設定組態物件的登入資料屬性，而不使用 Logins 屬性。在這種情況下，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提供者，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 `CognitoIdentityCredentials` 物件的詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK API 參考[AWS.CognitoIdentityCredentials](#)》中的。

使用 Web 聯合身分來驗證使用者

您可以直接設定個別身分提供者，以使用 Web 聯合身分存取 AWS 資源。AWS 目前支援透過數個身分提供者使用 Web 聯合身分驗證使用者：

- [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。接著，建立 IAM 角色並設定其許可。然後，您建立的 IAM 角色會用來授予您透過個別身分提供者為其設定的許可。例如，您可以設定角色，讓透過 Facebook 登入的使用者能夠讀取您控制的特定 Amazon S3 儲存貯體。

同時擁有已設定權限的 IAM 角色，以及向您選擇的身分提供者註冊的應用程式之後，您可以設定 SDK，以使用協助程式程式碼取得 IAM 角色的登入資料，如下所示：

```
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提供者。`WebIdentityToken` 參數中的值是透過身分提供者成功登入時所擷取的存取字符。如需為每個身分提供者設定和擷取存取字符的詳細資訊，請參閱身分提供者的文件。

步驟 1：向身分提供者註冊

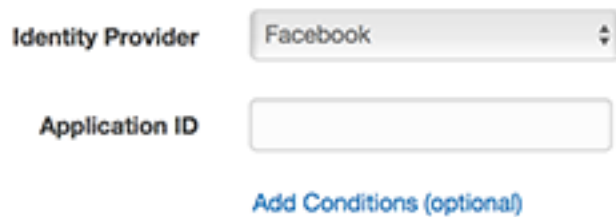
若要開始，請向您選擇要支援的身分提供者註冊應用程式。系統會要求您提供可辨識您應用程式和其作者的資訊。這可確保身分提供者了解誰在接收其使用者資訊。在每個案例中，身分供應商會發行您用來設定使用者角色的應用程式 ID。

步驟 2：為身分提供者建立 IAM 角色

從身分提供者取得應用程式 ID 之後，請前往 IAM 主控台，網址為 <https://console.aws.amazon.com/iam/> : //www.healthnet.com，以建立新的 IAM 角色。

為身分提供者建立 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 (身分提供者存取的角色)。
4. 針對 Grant access to web identity providers (將存取授予 web 身分提供者)，選擇 Select (選取)。
5. 從身分提供者清單中，選擇您要用於此 IAM 角色的身分提供者。



The screenshot shows the AWS IAM console interface for creating a new role. The 'Identity Provider' dropdown menu is set to 'Facebook'. Below it, the 'Application ID' input field is empty. A link labeled 'Add Conditions (optional)' is visible below the input field.

6. 輸入在 Application ID (應用程式 ID) 中由身分提供者提供的應用程式 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，請參閱《[IAM AWS 使用者指南](#)》中的 [IAM 許可概觀](#)。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)。如需關於 IAM 政策的詳細資訊，請參閱《[IAM 使用者指南](#)》中的 [IAM 政策概觀](#)。
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，請參閱《[IAM 使用者指南](#)》中的 [關於 Web 聯合身分](#)。

步驟 3：在登入後取得供應商存取字符

使用身分提供者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) 的身分提供者，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，請參閱身分提供者的軟體開發套件文件

- [Login with Amazon](#)
- [Facebook 登入](#)
- [Google 登入](#)

步驟 4：取得暫時登入資料

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。這些登入資料是透過 AWS Security Token Service 使用 Web 聯合身分的提供。使用者登入身分提供者，而傳回存取字符。針對您為此身分提供者建立的 IAM 角色，使用 ARN 設定 `AWS.WebIdentityCredentials` 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
  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提供者軟體開發套件，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，來確保身分提供者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 做為身分提供者。

```
<a href="#" id="login">
  
</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.');
```

```
};

(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 做為身分提供者：

```
<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 做為身分提供者。透過 Google 用於 web 聯合身分的存取字符合會存放在 `response.id_token`，而不是與其他身分提供者一樣放在 `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 版本號碼，可追蹤 API 相容性。AWS 服務中的 API YYYY-mm-dd 版本由格式化的日期字串識別。例如，Amazon S3 目前的 API 版本為 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 `DynamoDB` 和 `Amazon EC2` APIs 的特定版本以及目前的 `Amazon Redshift` API，請設定 `apiVersions`，如下所示：

```
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上的鎖定 API 版本一節，例如 <https://docs.aws.amazon.com/AWSJavaScriptSDK/latest/AWS/S3.html> for Amazon S3。

Node.js 的考量

雖然 Node.js 程式碼是 JavaScript，但在 Node.js 適用於 JavaScript 的 AWS SDK 中使用可能會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 GitHub 的此處](#)了解您可以使用的其他 Node.js 套件。

您可以搭配使用的 npm 套件範例適用於 JavaScript 的 AWS SDK 為 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 DynamoDB [靜態加密](#) 已與 [AWS KMS](#) 整合，因此您可能會遇到資料庫必須為每個操作重新建立新的 AWS KMS 快取項目的延遲。

設定適用於 JavaScript 的 SDK 以重複使用 TCP 連線的最簡單方法是將 AWS_NODEJS_CONNECTION_REUSE_ENABLED 環境變數設定為 1。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
  }
});
```

下列範例示範如何僅 keepAlive 針對 DynamoDB 用戶端設定：

```
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至網際網路，適用於 JavaScript 的 SDK 支援透過 `Proxy-agent` 等第三方 HTTP 代理程式使用 HTTP 或 HTTPS 代理。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含存取 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

下列主題說明在瀏覽器指令碼 適用於 JavaScript 的 AWS SDK 中使用 的特殊考量。

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

適用於 JavaScript 的 SDK 以 JavaScript 檔案提供，並支援預設的服務集。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，並且想要存取適用於 JavaScript 的 SDK 提供的所有服務，您可以複製儲存庫並執行建置軟體開發套件預設託管版本的相同建置工具，在本機建置軟體開發套件的自訂複本。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 Builder 建置適用於 JavaScript 的 SDK](#)
- [使用 CLI 建置適用於 JavaScript 的 SDK](#)
- [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 Builder 建置適用於 JavaScript 的 SDK

建立專屬建置的最簡單方法是使用 SDK 建置器 Web 應用程式，網址適用於 JavaScript 的 AWS SDK 為 <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 JavaScript 的 SDK

若要使用 建置適用於 JavaScript 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。

先從 GitHub 複製儲存庫，並將目錄變更為該目錄：

```
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

為了降低網路上的資料量，您可以透過名為壓縮的程序來壓縮 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，請使用下列命令：

```
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，若要建置 Amazon DynamoDB 的兩個 API 版本，請使用下列命令：

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

服務識別符和 API 版本可在服務特定的組態檔案中取得，網址為 <https://github.com/aws/aws-sdk-js/tree/master/apis>。

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。web 瀏覽器中執行的 JavaScript 並未原生支援此模組化方法。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 會在您可以與瀏覽器搭配使用的單一、可獨立自主運作的 JavaScript 檔案中，建立適用於瀏覽器指令碼的模組相依性。

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)，當您使用適用於 JavaScript 的 AWS SDK 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。

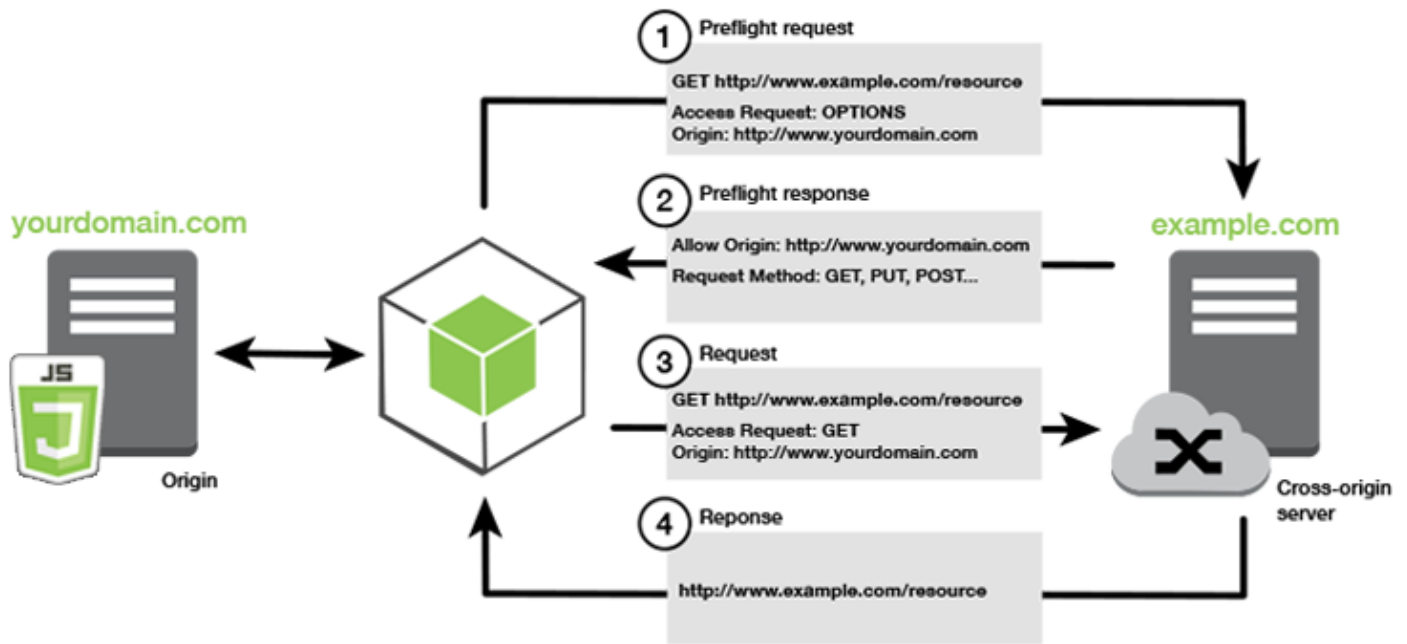
CORS 會根據以下要素判斷是否允許在跨來源請求中共享資源：

- 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。



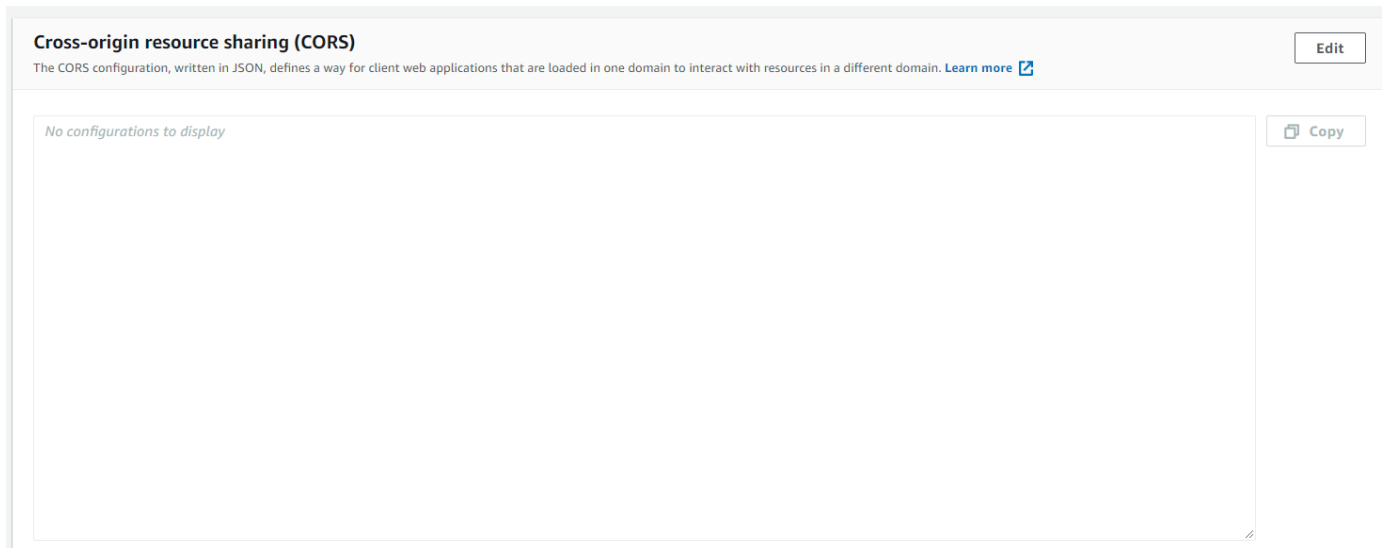
是否需要設定 CORS

Amazon S3 儲存貯體需要 CORS 組態，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，如果您從 Amazon S3 儲存貯體託管應用程式，並從 `*.s3.amazonaws.com` 或其他一些特定端點存取資源，您的請求將不會存取外部網域。因此，這個組態就不需要 CORS。在此情況下，CORS 仍會用於 Amazon S3 以外的服務。

設定 Amazon S3 儲存貯體的 CORS

您可以在 Amazon S3 主控台中設定 Amazon S3 儲存貯體以使用 CORS。

1. 在 Amazon S3 主控台中，選擇您要編輯的儲存貯體。
2. 選取許可索引標籤，然後向下捲動至跨來源資源共用 (CORS) 面板。



3. 選擇編輯，然後在 CORS 組態編輯器中輸入 CORS 組態，然後選擇儲存。

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

- <AllowedOrigin> 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。

如需範例組態，請參閱《Amazon Simple Storage Service 使用者指南》中的[如何在儲存貯體上設定 CORS？](#)。

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，它可讓瀏覽器的安全模型允許對 Amazon S3 提出請求。必須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 SDK 讀取從 Amazon S3 傳回的回應標頭。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。適用於 JavaScript 的 SDK 可以包含在 `webpack` 中，做為要包含在輸出套件中的其中一個相依性。

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 `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，例如適用於 JavaScript 的 SDK，則該程式碼會綁定，而不需要在組態中指定。

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。由於適用於 JavaScript 的 SDK 廣泛使用 JSON 檔案，如果 `json-loader` 未包含，Webpack 會在產生套件時擲出錯誤。

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。如果您使用適用於 JavaScript 的 SDK，則僅綁定應用程式實際使用的 SDK 部分可以大幅減少 Webpack 輸出的大小。

請考慮下列用來建立 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。使用此程式碼產生的 Webpack 套件會包含完整的 SDK，但當僅使用 Amazon S3 用戶端類別時，則不需要完整的 SDK。如果只包含 Amazon S3 服務所需的開發套件部分，則套件的大小會明顯較小。即使設定組態也不需要完整的 SDK，因為您可以在 Amazon S3 服務物件上設定組態資料。

以下是只包含 SDK 的 Amazon S3 部分時，相同程式碼的外觀。

```
// 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"
```

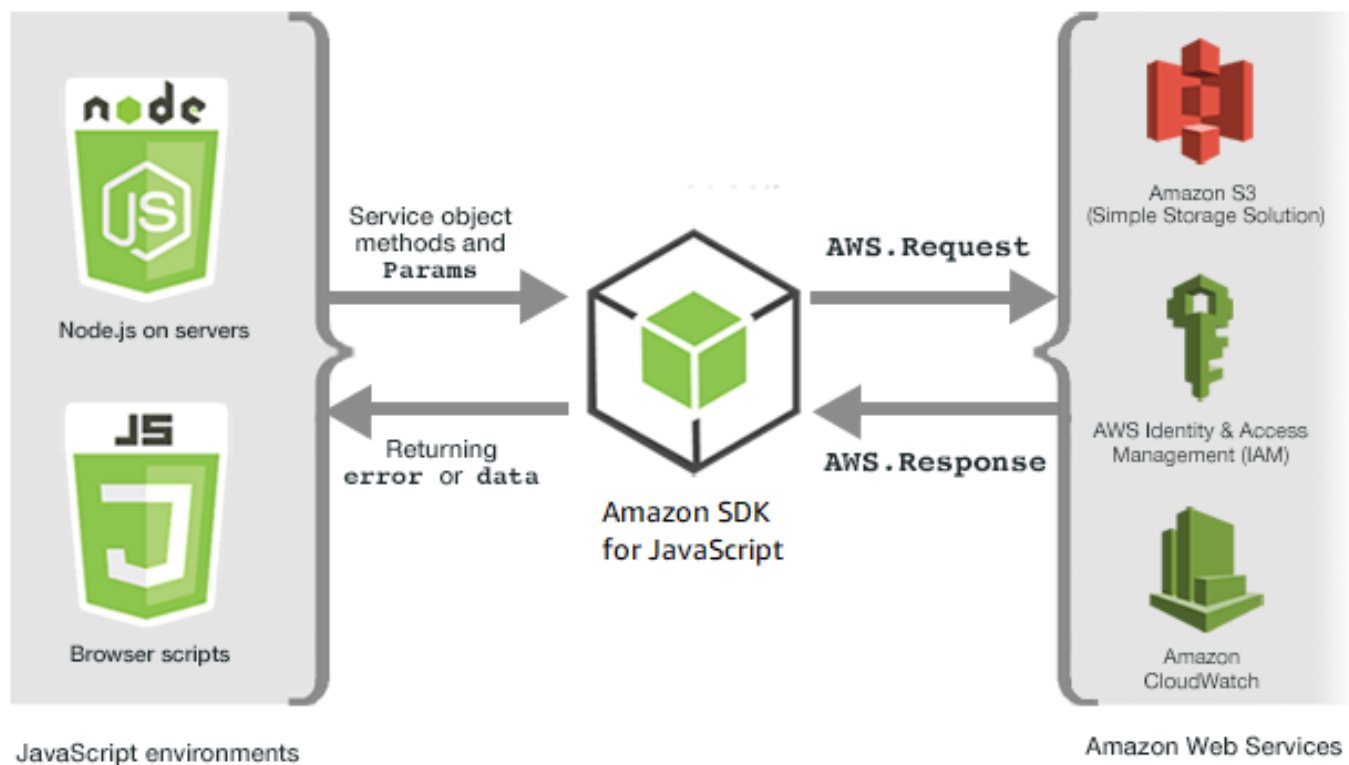
當您在磁碟空間有限的環境中執行 Node.js 應用程式時，這個做法十分實用。下方為 `webpack.config.js` 組態範例，而該組態會將 Node.js 指定為輸出目標。

```
// 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']
      }
    ]
  }
}
```

在適用於 JavaScript 的 SDK 中使用 服務

適用於 JavaScript 的 AWS SDK 可透過用戶端類別集合，提供其支援的 服務的存取權。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 以使用服務功能和資源。例如，Amazon DynamoDB APIs 可透過 `AWS.DynamoDB` 類別取得。

透過適用於 JavaScript 的 SDK 公開的服務會遵循請求回應模式，與呼叫應用程式交換訊息。在此模式中，負責叫用服務的程式碼會向服務端點提交 HTTP/HTTPS 請求。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。`AWS.Request` 物件會將請求封裝在軟體開發套件中，而 `AWS.Response` 物件則負責將回應封裝在軟體開發套件中。系統會透過多種技術 (如回呼函數或 JavaScript promise) 將該物件提供給請求程式。

主題

- [建立和呼叫服務物件](#)

- [記錄 適用於 JavaScript 的 AWS SDK 通話](#)
- [非同步呼叫服務](#)
- [使用回應物件](#)
- [使用 JSON](#)
- [適用於 JavaScript 的 AWS SDK v2 中的重試策略](#)

建立和呼叫服務物件

JavaScript API 支援大多數可用的 AWS 服務。JavaScript API 中的每個服務類別皆可讓您在服務內存取所有 API 呼叫。如需 JavaScript API 中服務類別、操作和參數的詳細資訊，請參閱 [API 參考](#)。

在 Node.js 中使用軟體開發套件時，您能夠利用 `require` 新增軟體開發套件封裝至應用程式，其可支援所有現有服務。

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

當您搭配使用軟體開發套件與瀏覽器 JavaScript 時，則可善用 AWS 託管的軟體開發套件封裝，將軟體開發套件封裝載入至瀏覽器指令碼。若要載入軟體開發套件封裝，請新增下列 `<script>` 標籤：

```
<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》中適用於 JavaScript 的 SDK 的 API 參考。 [適用於 JavaScript 的 AWS SDK](#)

預設託管 SDK 套件支援一部分的可用 AWS 服務。如需適用於瀏覽器的託管軟體開發套件封裝預設服務清單，請參閱 API 參考中的 [支援的服務](#)。如果 CORS 安全性檢查功能遭停用，您可以搭配其他服務使用軟體開發套件。在這種情況下，您可以建置自訂版本的軟體開發套件，並視需要加入其他服務。如需建置自訂版本軟體開發套件的詳細資訊，請參閱 [建立適用於瀏覽器的軟體開發套件](#)。

要求個別服務

如先前所示，需要適用於 JavaScript 的 SDK 將整個 SDK 納入您的程式碼。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'});
```

在上述範例中，`require` 函數會指定整個軟體開發套件。如果只包含 Amazon S3 服務所需的開發套件部分，則透過網路傳輸的程式碼數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，當它只包含 SDK 的 Amazon S3 部分時看起來的樣子。

```
// 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 Browserify 或 webpack 等綁定工具時，要求個別服務只會納入軟體開發套件完整大小的一部分。這有助於處理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 JavaScript API 存取服務功能，您必須先建立服務物件，其可讓您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'});
```

在預設情況下，系統會使用設定軟體開發套件的全域設定來配置服務物件。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的，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使用美國西部（奧勒岡）（`us-west-2`）區域，請將以下內容新增至 `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

您能夠在建立物件時指定 `apiVersion` 選項，藉此鎖定服務物件以使用特定的服務 API 版本。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

在呼叫服務物件的方法時，請依照 API 所需來傳遞 JSON 格式的參數。例如，在 Amazon S3 中，若要取得指定儲存貯體和金鑰的物件，請將下列參數傳遞給 `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 參考[Class: AWS.S3](#)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 服務物件的 `Bucket` 參數繫結至名為 `amzn-s3-demo-bucket` 的儲存貯體。

```
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'});
```

如需每個方法可用的參數詳細資訊，請參閱 API 參考。

記錄 適用於 JavaScript 的 AWS SDK 通話

使用內建記錄器進行適用於 JavaScript 的 AWS SDK 檢測，因此您可以記錄使用適用於 JavaScript 的 SDK 進行的 API 呼叫。

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。您必須依照指示安裝和設定自訂記錄器，才能搭配適用於 JavaScript 的 SDK 使用。

`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 `logplease` 的詳細資訊，請參閱 GitHub 上的 [logplease 簡易 JavaScript 記錄器](#)。

非同步呼叫服務

透過軟體開發套件提出的所有請求皆為非同步執行。編寫瀏覽器指令碼時，請務必注意這點。Web 瀏覽器中執行的 JavaScript 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，而且在過程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 Promise](#)

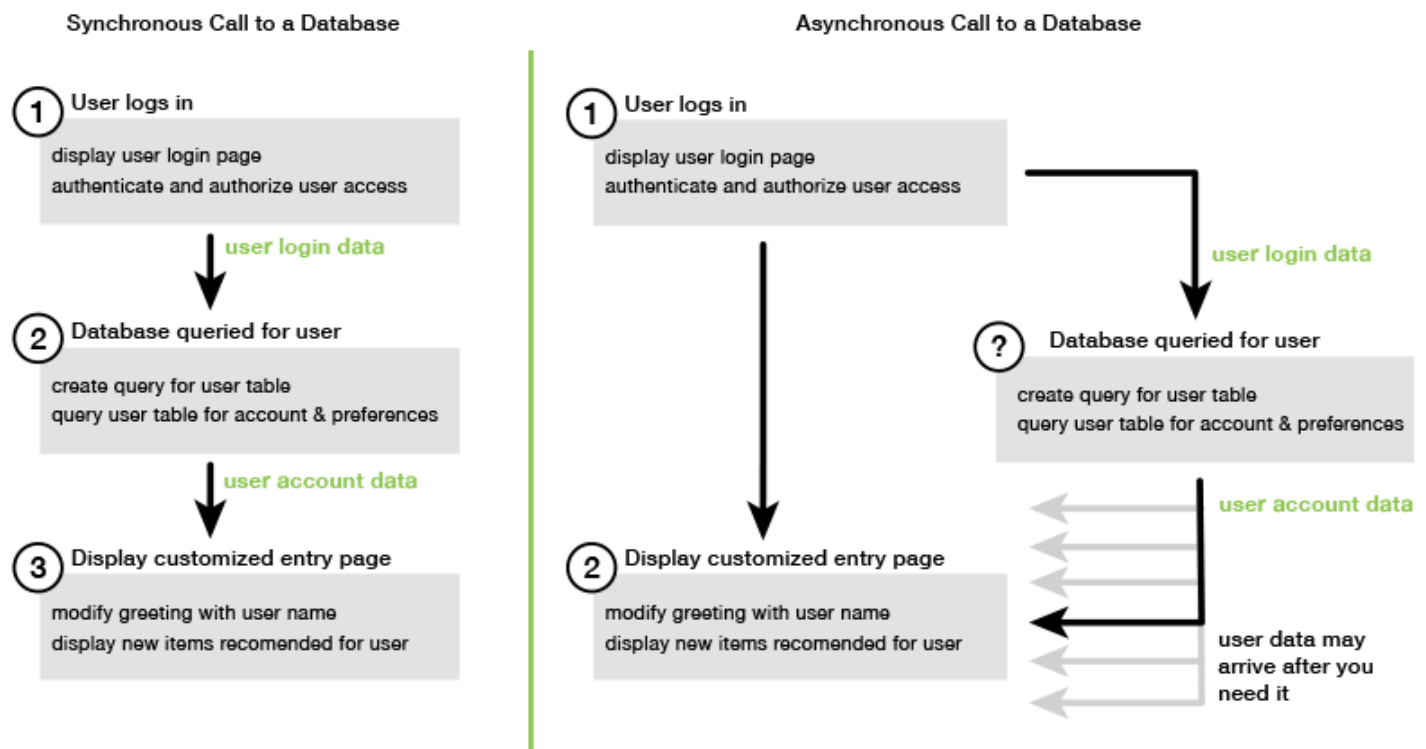
管理非同步呼叫

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

在回呼函數內部，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 [使用回呼物件](#)。如需 `AWS.Response` 物件的詳細資訊，請參閱《API 參考[Class: AWS.Response](#)》中的 [使用回呼物件](#)或 [使用回應物件](#)。

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分應，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分整清單，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，以下呼叫 Amazon S3 服務物件的 listBuckets 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

根據每個服務操作方法의 回應，AWS.Request 物件會引發下述 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，系統就會引發 `httpUploadProgress` 事件。同樣地，系統會在 HTTP 請求下載更多資料時引發 `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，系統就會引發 `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

您可以在對的呼叫中使用 async/await 模式適用於 JavaScript 的 AWS SDK。接受回呼的大多數函數都不會傳回 promise。由於您只使用傳回 promise 的 await 函數，若要使用 async/await 模式，您需要將 .promise() 方法鏈結至通話結尾，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 Promise

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 and Node.js 支援 Promise

對原生 JavaScript promise (ECMAScript 2015) 的支援將視程式碼執行的 JavaScript 引擎與版本而定。為協助您在需要執行程式碼的每個環境中，判斷其可支援的 JavaScript promise，請參閱 GitHub 上的 [ECMAScript 相容性資料表](#)。

使用其他 Promise 實作

除了 ECMAScript 2015 中的原生 promise 實作外，您也能運用第三方的 promise 程式庫，包括：

- [bluebird](#)
- [RSVP](#)
- [Q](#)

如果您需要在不支援 ECMAScript 5 和 ECMAScript 2015 原生 promise 實作的環境中執行程式碼，則這些選用 promise 程式庫會很有幫助。

請呼叫全域組態物件的 `setPromisesDependency` 方法，並在軟體開發套件上設定 promise 相依性，即可使用第三方 promise 程式庫。在瀏覽器指令碼中，請務必先載入第三方 promise 程式庫，再載入軟體開發套件。在下方範例中，軟體開發套件已設定為使用 bluebird promise 程式庫中的實作。

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

若要恢復使用 JavaScript 引擎的原生 promise 實作，您可以再次呼叫 `setPromisesDependency` 並傳遞 `null`，而不傳送程式庫名稱。

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，做為遊戲的一部分。

```
// 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

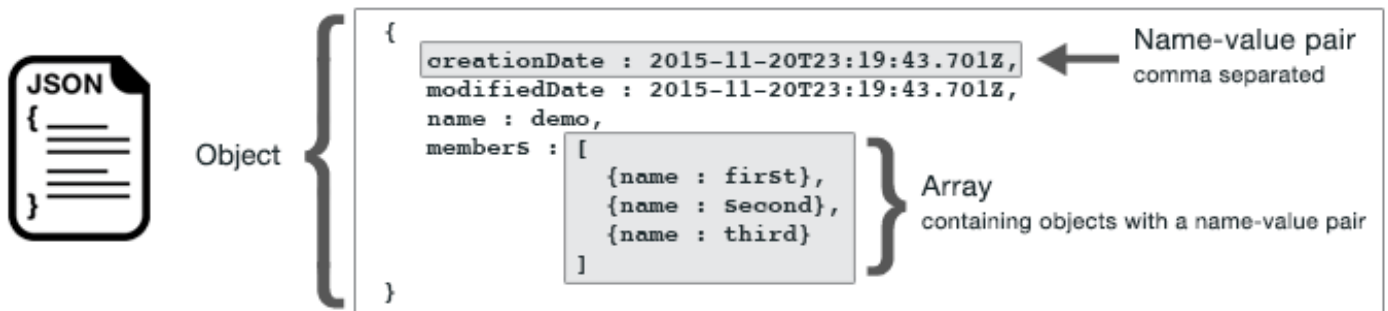
`request` 屬性可讓您存取來源 `AWS.Request` 物件。這可用來參照原始 `AWS.Request` 物件，以存取其所傳送的原始參數。在下方範例中，系統會使用 `request` 屬性來存取原始服務請求的 `Key` 參數。

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

使用 JSON

JSON 是一種人類和機器皆可讀取的資料交換格式。雖然 JSON 這個名稱是 JavaScript 物件標記法的縮寫，但 JSON 格式與任何程式設計語言互不相關。

適用於 JavaScript 的 SDK 會在提出請求時，使用 JSON 將資料傳送至服務物件，並以 JSON 的形式從服務物件接收資料。如需 JSON 的詳細資訊，請參閱 json.org。



JSON 代表資料的方式有兩種：

- 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。

下方 JSON 物件範例內含物件陣列，而這些物件代表卡片遊戲的卡片。每張卡片皆由兩個名稱/值對所定義；一個會指定可識別該卡片的唯一值，另一個則會指定可指向對應卡片映像的 URL。

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

做為服務物件參數的 JSON

以下是用於定義 Lambda 服務物件呼叫參數的簡單 JSON 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是用來在 Lambda 服務物件上呼叫 invoke 方法的參數。

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

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

傳回 JSON 格式的資料

JSON 提供了一種標準做法，能夠在需要同時傳送多個值的應用程式部分間傳遞資料。一般來說，API 中的用戶端類別方法會在傳遞至回呼函數的 data 參數內傳回 JSON。例如，以下是對 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
    }
  ]
}
```

適用於 JavaScript 的 AWS SDK v2 中的重試策略

網路上有許多元件 (例如 DNS 伺服器、交換器、負載平衡器和其他項目) 可以在指定請求之生命週期中的任何階段產生錯誤。一般在網路環境中處理這些錯誤回應的技術，就是在用戶端應用程式中實作重試。此技術可提高應用程式的可靠性，並降低 developer. AWS SDKs 的操作成本，為您的 AWS 請求實作自動重試邏輯。

指數退避型重試行為

適用於 JavaScript 的 AWS SDK v2 使用[指數退避搭配完整抖動](#)實作重試邏輯，以獲得更好的流程控制。指數退避的背後概念是，對於連續錯誤回應，讓重試之間的等待時間漸進拉長。抖動（隨機延遲）用於防止連續的碰撞。

v2 中的測試重試延遲

為了在 v2 中測試重試延遲，[node_modules/aws-sdk/lib/event_listeners.js](#) 中的程式碼已更新為變數延遲中存在console.log的值，如下所示：

```
// delay < 0 is a signal from customBackoff to skip retries
if (willRetry && delay >= 0) {
  resp.error = null;
  console.log('retry delay: ' + delay);
  setTimeout(done, delay);
} else {
```

```
done();  
}
```

使用預設組態重試延遲

您可以在 AWS 開發套件用戶端上測試任何操作的延遲。我們使用以下程式碼呼叫 DynamoDB 用戶端上的 `listTables` 操作：

```
import AWS from "aws-sdk";  
  
const region = "us-east-1";  
const client = new AWS.DynamoDB({ region });  
await client.listTables({}).promise();
```

為了測試重試，我們 `NetworkingError` 透過中斷網際網路與執行測試程式碼之裝置的連線來模擬。您也可以設定代理來傳回自訂錯誤。

執行程式碼時，您可以看到使用指數退避搭配抖動的重試延遲，如下所示：

```
retry delay: 7.39361151766359  
retry delay: 9.0672860785882  
retry delay: 134.89340825668168  
retry delay: 398.53559817403965  
retry delay: 523.8076165896343  
retry delay: 1323.8789643058465
```

當重試使用抖動時，您會在執行範例程式碼時取得不同的值。

使用自訂基礎重試延遲

適用於 JavaScript 的 AWS SDK v2 允許傳遞自訂基本毫秒數，以用於操作重試的指數退避。DynamoDB 以外的所有服務預設為 100 毫秒，其預設為 50 毫秒。

我們測試自訂基底為 1000 毫秒的重試，如下所示：

```
...  
const client = new AWS.DynamoDB({ region, retryDelayOptions: { base: 1000 } });  
...
```

我們 `NetworkingError` 透過中斷網際網路與執行測試程式碼之裝置的連線來模擬。您可以看到重試延遲的值高於先前的執行，預設值為 50 或 100 毫秒。

```
retry delay: 356.2841549924913
retry delay: 1183.5216495444615
retry delay: 2266.997988094194
retry delay: 1244.6948354966453
retry delay: 4200.323030066383
```

當重試使用抖動時，您會在執行範例程式碼時取得不同的值。

使用自訂退避演算法重試延遲

適用於 JavaScript 的 AWS SDK v2 也允許傳遞接受重試計數和錯誤的自訂退避函數，並傳回以毫秒為單位的延遲時間。如果結果是非零的負值，則不會再嘗試重試。

我們測試自訂退避函數，其使用基本值為 200 毫秒的線性退避，如下所示：

```
...
const client = new AWS.DynamoDB({
  region,
  retryDelayOptions: { customBackoff: (count, error) => (count + 1) * 200 },
});
...
```

我們 `NetworkingError` 透過中斷網際網路與執行測試程式碼之裝置的連線來模擬。您可以看到重試延遲的值是 200 的倍數。

```
retry delay: 200
retry delay: 400
retry delay: 600
retry delay: 800
retry delay: 1000
```

適用於 JavaScript 的 SDK 程式碼範例

本節中的主題包含如何使用 適用於 JavaScript 的 AWS SDK 搭配各種 服務的 APIs 來執行常見任務的範例。

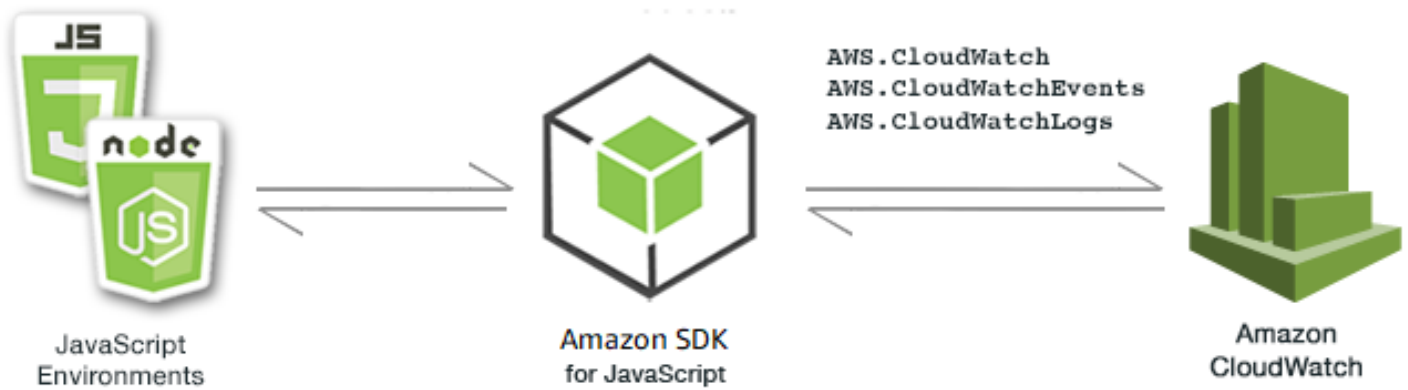
在 GitHub 上的 AWS 文件程式碼範例儲存庫中尋找這些範例和其他範例的原始程式碼。 [GitHub](#) 若要提議新的程式碼範例，讓 AWS 文件團隊考慮生產，請建立新的請求。該團隊想要產生比僅涵蓋個別 API 呼叫之簡易程式碼更為廣泛的程式碼範例，以涵蓋更為廣泛的案例和使用案例。如需說明，請參閱[貢獻準則](#)中的編寫程式碼一節。

主題

- [Amazon CloudWatch 範例](#)
- [Amazon DynamoDB 範例](#)
- [Amazon EC2 範例](#)
- [AWS Elemental MediaConvert 範例](#)
- [AWS IAM 範例](#)
- [Amazon Kinesis 範例](#)
- [Amazon S3 範例](#)
- [Amazon Simple Email Service 範例](#)
- [Amazon Simple Notification Service 範例](#)
- [Amazon SQS 範例](#)

Amazon CloudWatch 範例

Amazon CloudWatch (CloudWatch) 是一種 Web 服務，可 AWS 即時監控您在 上執行的 Amazon Web Services 資源和應用程式。您可以使用 CloudWatch 收集和追蹤指標，這些是您可以為您的資源和應用程式測量的變數。CloudWatch 警示會根據您定義的規則，傳送通知或自動變更您要監控的資源。



適用於 CloudWatch 的 JavaScript API 透過 `AWS.CloudWatch`、`AWS.CloudWatchEvents` 和 `AWS.CloudWatchLogs` 用戶端類別公開。如需使用 CloudWatch 用戶端類別的詳細資訊，請參閱 API 參考 [Class: `AWS.CloudWatchLogs`](#) 中的 [Class: `AWS.CloudWatchEvents`](#)、[Class: `AWS.CloudWatch`](#) 和 [Class: `AWS.CloudWatchLogs`](#)。

主題

- [在 Amazon CloudWatch 中建立警示](#)
- [在 Amazon CloudWatch 中使用警示動作](#)
- [從 Amazon CloudWatch 取得指標](#)
- [傳送事件到 Amazon CloudWatch Events](#)
- [在 Amazon CloudWatch Logs 中使用訂閱篩選條件](#)

在 Amazon CloudWatch 中建立警示



這個 Node.js 程式碼範例會說明：

- 如何擷取 CloudWatch 警示的基本資訊。
- 如何建立和刪除 CloudWatch 警示。

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，一系列 Node.js 模組用於在 CloudWatch 中建立警示。Node.js 模組使用適用於 JavaScript 的 SDK，以用戶端 `AWS.CloudWatch` 類別的這些方法建立警示：

- [describeAlarms](#)
- [putMetricAlarm](#)
- [deleteAlarms](#)

如需 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](#)。Amazon CloudWatch

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 [GitHub 上](#) 找到這個範本程式碼。

為 CloudWatch 指標建立警示

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，在此案例中為 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 [GitHub 上](#) 找到這個範本程式碼。

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 [GitHub 上](#) 找到這個範本程式碼。

在 Amazon CloudWatch 中使用警示動作



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

使用警示動作，您可以建立自動停止、終止、重新啟動或復原 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，一系列 Node.js 模組用於在 CloudWatch 中定義警示動作，以觸發 Amazon EC2 執行個體的重新啟動。Node.js 模組使用適用於 JavaScript 的 SDK，以下列 CloudWatch 用戶端類別的方法管理 Amazon EC2 執行個體：

- [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、重新啟動或復原執行個體](#)。

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，其政策會授予描述、重新啟動、停止或終止 Amazon EC2 執行個體的許可。如需建立 IAM 角色的詳細資訊，請參閱《IAM 使用者指南》中的 [建立角色以委派許可給 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": [
        "*"
      ]
    }
  ]
}
```

建立全域組態物件，然後設定程式碼的區域，以設定適用於 JavaScript 的 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，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 [GitHub 上](#) 找到這個範本程式碼。

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 [GitHub 上](#) 找到這個範本程式碼。

從 Amazon CloudWatch 取得指標



這個 Node.js 程式碼範例會說明：

- 如何擷取已發佈的 CloudWatch 指標清單。
- 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，一系列 Node.js 模組用於從 CloudWatch 取得指標，以及將事件傳送至 Amazon CloudWatch Events。Node.js 模組使用適用於 JavaScript 的 SDK，透過 CloudWatch 用戶端類別的這些方法來從 CloudWatch 取得指標：

- [listMetrics](#)
- [putMetricData](#)

如需 CloudWatch 指標的詳細資訊，請參閱《[Amazon CloudWatch 使用者指南](#)》中的使用 [Amazon CloudWatch 指標](#)。Amazon CloudWatch

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 [GitHub 上](#) 找到這個範本程式碼。

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 [GitHub 上](#) 找到這個範本程式碼。

傳送事件到 Amazon CloudWatch Events



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 Events 提供近乎即時的系統事件串流，將 Amazon Web Services 資源中的變更描述到任何各種目標。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，會使用一系列 Node.js 模組將事件傳送至 CloudWatch Events。Node.js 模組使用適用於 JavaScript 的 SDK，以下列 CloudWatchEvents 用戶端類別的方法管理執行個體：

- [putRule](#)
- [putTargets](#)
- [putEvents](#)

如需 CloudWatch Events 的詳細資訊，請參閱《Amazon CloudWatch Events 使用者指南》中的 [使用 PutEvents 新增事件](#)。Amazon CloudWatch

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 hello-world 藍圖建立 Lambda 函數，做為事件的目標。若要了解如何執行，請參閱《Amazon CloudWatch Events 使用者指南》中的 [步驟 1：建立 AWS Lambda 函數](#)。
- 建立 IAM 角色，其政策會授予 CloudWatch Events 的許可，並包含 `events.amazonaws.com` 做為信任的實體。如需建立 IAM 角色的詳細資訊，請參閱《IAM 使用者指南》中的 [建立角色以委派許可給 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 Events，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 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 [GitHub 上](#) 找到這個範本程式碼。

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 Events，請建立 `AWS.CloudWatchEvents` 服務物件。建立 JSON 物件，其中包含指定要連接目標之規則所需的參數，包括您建立之 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 [GitHub 上](#) 找到這個範本程式碼。

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 Events，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 [GitHub](#) 上找到這個範本程式碼。

在 Amazon CloudWatch Logs 中使用訂閱篩選條件



這個 Node.js 程式碼範例會說明：

- 如何在 CloudWatch Logs 中建立和刪除日誌事件的篩選條件。

使用案例

訂閱可讓您從 CloudWatch Logs 存取日誌事件的即時摘要，並將該摘要交付給其他服務，例如 Amazon Kinesis 串流 AWS Lambda，或用於自訂處理、分析或載入其他系統。訂閱篩選條件定義用於篩選哪些日誌事件交付至您的 AWS 資源的模式。

在此範例中，一系列 Node.js 模組用於列出、建立和刪除 CloudWatch Logs 中的訂閱篩選條件。日誌事件的目的地是 Lambda 函數。Node.js 模組使用適用於 JavaScript 的 SDK，以下列 CloudWatchLogs 用戶端類別方法管理訂閱篩選條件：

- [putSubscriptionFilters](#)
- [describeSubscriptionFilters](#)
- [deleteSubscriptionFilter](#)

如需 CloudWatch Logs 訂閱的詳細資訊，請參閱《Amazon CloudWatch Logs 使用者指南》中的[使用訂閱即時處理日誌資料](#)。

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 Logs 使用者指南》中的[使用訂閱篩選條件 AWS Lambda](#)。

- 建立 IAM 角色，其政策會授予許可，以叫用您建立的 Lambda 函數，並授予對 CloudWatch Logs 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[建立角色以委派許可給 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 Logs，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 cw1 = new AWS.CloudWatchLogs({ apiVersion: "2014-03-28" });

var params = {
  logGroupName: "GROUP_NAME",
  limit: 5,
};

cw1.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 cw1_describesubscriptionfilters.js
```

您可以在 [GitHub 上](#) 找到這個範本程式碼。

建立訂閱篩選條件

以檔名 `cw1_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 Logs，請建立 `AWS.CloudWatchLogs` 服務物件。建立包含建立篩選條件所需參數的 JSON 物件，包括目的地 Lambda 函數的 ARN、篩選條件的名稱、篩選的字串模式，以及日誌群組的名稱。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 cw1 = new AWS.CloudWatchLogs({ apiVersion: "2014-03-28" });

var params = {
  destinationArn: "LAMBDA_FUNCTION_ARN",
  filterName: "FILTER_NAME",
  filterPattern: "ERROR",
```

```
    logGroupName: "LOG_GROUP",
  };

  cwل.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 cwل_putsubscriptionfilter.js
```

您可以在 [GitHub 上](#) 找到這個範本程式碼。

刪除訂閱篩選條件

以檔名 `cwل_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 Logs，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 cwل = new AWS.CloudWatchLogs({ apiVersion: "2014-03-28" });

var params = {
  filterName: "FILTER",
  logGroupName: "LOG_GROUP",
};

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

```
}  
});
```

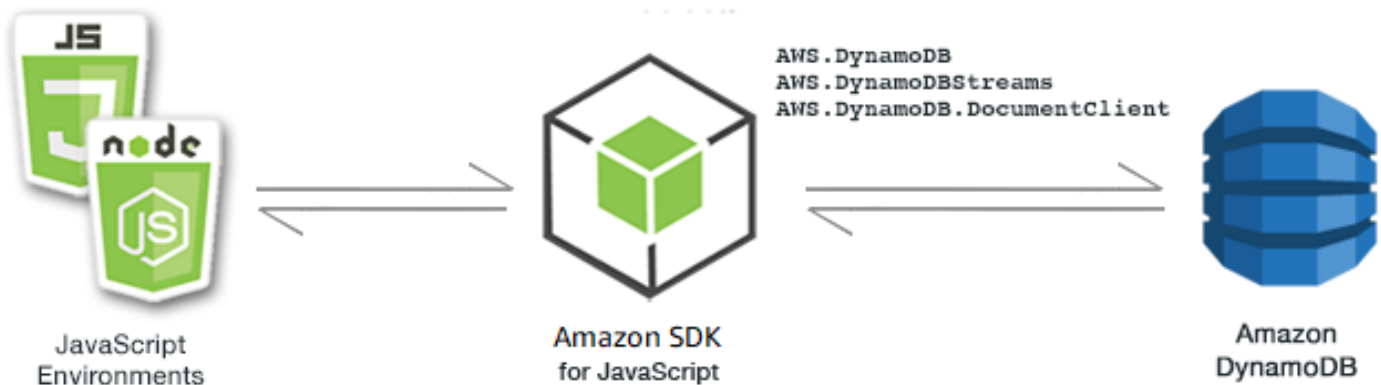
若要執行範例，請在命令列中輸入以下內容。

```
node cwl_deletesubscriptionfilter.js
```

您可以在 [GitHub 上](#) 找到這個範本程式碼。

Amazon DynamoDB 範例

Amazon DynamoDB 是全受管的 NoSQL 雲端資料庫，可同時支援文件和鍵值存放區模型。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 透過 `AWS.DynamoDB`、`AWS.DynamoDBStreams` 和 `AWS.DynamoDB.DocumentClient` 用戶端類別公開。如需使用 DynamoDB 用戶端類別的詳細資訊，請參閱 API 參考 [Class: AWS.DynamoDB.DocumentClient](#) 中的 [Class: AWS.DynamoDBStreams](#)、[Class: AWS.DynamoDB](#) 和。

主題

- [在 DynamoDB 中建立和使用資料表](#)
- [在 DynamoDB 中讀取和寫入單一項目](#)
- [在 DynamoDB 中讀取和寫入批次中的項目](#)
- [查詢和掃描 DynamoDB 資料表](#)
- [使用 DynamoDB 文件用戶端](#)

在 DynamoDB 中建立和使用資料表



這個 Node.js 程式碼範例會說明：

- 如何建立和管理用於存放和擷取 DynamoDB 資料的資料表。

使用案例

與其他資料庫系統類似，DynamoDB 會將資料存放在資料表中。DynamoDB 資料表是資料集合，會組織成類似於資料列的項目。若要在 DynamoDB 中存放或存取資料，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 模組來執行 DynamoDB 資料表的基本操作。此程式碼使用適用於 JavaScript 的 SDK，透過使用 `AWS.DynamoDB` 用戶端類別的這些方法來建立和使用資料表：

- [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 [GitHub 上](#) 找到這個範本程式碼。

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 [GitHub 上](#) 找到這個範本程式碼。

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 [GitHub 上](#) 找到這個範本程式碼。

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 [GitHub 上](#) 找到這個範本程式碼。

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，透過使用 AWS.DynamoDB 用戶端類別的下列方法來讀取和寫入 DynamoDB 資料表中的一個項目：

- [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](#)。
- 建立您可以存取其項目的 DynamoDB 資料表。如需建立 DynamoDB 資料表的詳細資訊，請參閱 [在 DynamoDB 中建立和使用資料表](#)。

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 [GitHub 上](#) 找到這個範本程式碼。

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 [GitHub 上](#) 找到這個範本程式碼。

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 [GitHub 上](#) 找到這個範本程式碼。

在 DynamoDB 中讀取和寫入批次中的項目



這個 Node.js 程式碼範例會說明：

- 如何讀取和寫入 DynamoDB 資料表中的項目批次。

使用案例

在此範例中，您會使用一系列 Node.js 模組，將一批項目放入 DynamoDB 資料表，並讀取一批項目。此程式碼使用適用於 JavaScript 的 SDK，以使用這些 DynamoDB 用戶端類別的方法執行批次讀取和寫入操作：

- [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](#)。

- 建立您可以存取其項目的 DynamoDB 資料表。如需建立 DynamoDB 資料表的詳細資訊，請參閱 [在 DynamoDB 中建立和使用資料表](#)。

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 [GitHub 上](#) 找到這個範本程式碼。

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 [GitHub 上](#) 找到這個範本程式碼。

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 資料表擷取的一或多個項目。此程式碼使用適用於 JavaScript 的 SDK，以使用這些 DynamoDB 用戶端類別的方法查詢和掃描資料表：

- [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](#)。
- 建立您可以存取其項目的 DynamoDB 資料表。如需建立 DynamoDB 資料表的詳細資訊，請參閱 [在 DynamoDB 中建立和使用資料表](#)。

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 [GitHub 上](#) 找到這個範本程式碼。

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 [GitHub 上](#) 找到這個範本程式碼。

使用 DynamoDB 文件用戶端



這個 Node.js 程式碼範例會說明：

- 如何使用文件用戶端存取 DynamoDB 資料表。

使用案例

DynamoDB 文件用戶端透過抽象化屬性值的概念，簡化了項目的使用。此摘錄會標註以輸入參數提供的原生 JavaScript 類型，並將標註的回應資料轉換為原生的 JavaScript 類型。

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 進程式設計](#)。DynamoDB

在此範例中，您會使用一系列 Node.js 模組，使用文件用戶端在 DynamoDB 資料表上執行基本操作。此程式碼使用適用於 JavaScript 的 SDK，以使用這些 DynamoDB 文件用戶端類別的方法查詢和掃描資料表：

- [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](#)。
- 建立您可以存取其項目的 DynamoDB 資料表。如需使用適用於 JavaScript 的 SDK 建立 DynamoDB 資料表的詳細資訊，請參閱 [在 DynamoDB 中建立和使用資料表](#)。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 `AWS.DynamoDB.DocumentClient` 物件。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 [GitHub 上](#) 找到這個範本程式碼。

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 `AWS.DynamoDB.DocumentClient` 物件。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 [GitHub](#) 上找到這個範本程式碼。

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 `AWS.DynamoDB.DocumentClient` 物件。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 [GitHub 上](#) 找到這個範本程式碼。

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 `AWS.DynamoDB.DocumentClient` 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`，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 [GitHub 上](#) 找到這個範本程式碼。

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 `AWS.DynamoDB.DocumentClient` 物件。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 [GitHub 上](#) 找到這個範本程式碼。

Amazon EC2 範例

Amazon Elastic Compute Cloud (Amazon EC2) 是一種 Web 服務，可在雲端提供虛擬伺服器託管。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 透過 `AWS.EC2` 用戶端類別公開。如需使用 Amazon EC2 用戶端類別的詳細資訊，請參閱 API 參考 [Class: AWS.EC2](#) 中的。

主題

- [建立 Amazon EC2 執行個體](#)
- [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](#)

建立 Amazon EC2 執行個體



這個 Node.js 程式碼範例會說明：

- 如何從公有 Amazon Machine Image (AMI) 建立 Amazon EC2 執行個體。
- 如何建立標籤並將其指派給新的 Amazon EC2 執行個體。

關於範例

在此範例中，您會使用 Node.js 模組來建立 Amazon EC2 執行個體，並將金鑰對和標籤指派給該執行個體。此程式碼使用適用於 JavaScript 的 SDK，透過使用這些 Amazon EC2 用戶端類別的方法建立和標記執行個體：

- [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。

接下來的程式碼會將 `Name` 標籤新增至新的執行個體，Amazon EC2 主控台會辨識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-eb
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 [GitHub 上](#) 找到這個範本程式碼。

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 Amazon EC2 執行個體的詳細監控。

- 如何啟動和停止 Amazon EC2 執行個體。
- 如何重新啟動 Amazon EC2 執行個體。

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 模組使用以下 Amazon EC2 用戶端類別方法，使用適用於 JavaScript 的 SDK 來管理執行個體：

- [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](#)。
- 建立 Amazon EC2 執行個體。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](#)。Amazon EC2

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。若要存取 Amazon EC2，請建立 `AWS.EC2` 服務物件。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 [GitHub 上](#) 找到這個範本程式碼。

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。若要存取 Amazon EC2，請建立 `AWS.EC2` 服務物件。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，開始對指定的執行個體進行詳細監控，或呼叫 `unmonitorInstances` 方法。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 [GitHub 上](#) 找到這個範本程式碼。

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。若要存取 Amazon EC2，請建立 `AWS.EC2` 服務物件。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 `stopInstances` 方法來停止它們。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 [GitHub 上](#) 找到這個範本程式碼。

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。若要存取 Amazon EC2，請建立 Amazon EC2 服務物件。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 [GitHub 上](#) 找到這個範本程式碼。

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。
- 如何建立金鑰對以存取 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，您會使用一系列 Node.js 模組來執行數個 Amazon EC2 金鑰對管理操作。Node.js 模組使用適用於 JavaScript 的 SDK，透過使用這些 Amazon EC2 用戶端類別的方法來管理執行個體：

- [createKeyPair](#)
- [deleteKeyPair](#)
- [describeKeyPairs](#)

如需 Amazon EC2 金鑰對的詳細資訊，請參閱[Amazon EC2 使用者指南](#)中的 [Amazon EC2 金鑰對](#)或 [Amazon EC2 使用者指南](#)中的 [Amazon EC2 金鑰對和 Windows 執行個體](#)。Amazon EC2

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。若要存取 Amazon EC2，請建立 `AWS.EC2` 服務物件。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 [GitHub 上](#) 找到這個範本程式碼。

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。若要存取 Amazon EC2，請建立 AWS.EC2 服務物件。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 [GitHub 上](#) 找到這個範本程式碼。

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。若要存取 Amazon EC2，請建立 `AWS.EC2` 服務物件。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 [GitHub 上](#) 找到這個範本程式碼。

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 託管於全球多個位置。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 可讓您將執行個體和資料放置在多個位置。

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 模組使用適用於 JavaScript 的 SDK，透過使用 Amazon EC2 用戶端類別的下列方法管理執行個體：

- [describeAvailabilityZones](#)
- [describeRegions](#)

如需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。若要存取 Amazon EC2，請建立 `AWS.EC2` 服務物件。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 [GitHub 上](#) 找到這個範本程式碼。

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，您會使用一系列 Node.js 模組來執行多個涉及安全群組的 Amazon EC2 操作。Node.js 模組使用適用於 JavaScript 的 SDK，透過使用 Amazon EC2 用戶端類別的下列方法管理執行個體：

- [describeSecurityGroups](#)
- [authorizeSecurityGroupIngress](#)
- [createSecurityGroup](#)
- [describeVpcs](#)
- [deleteSecurityGroup](#)

如需 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 Amazon 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](#)。Amazon EC2

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。若要存取 Amazon EC2，請建立 `AWS.EC2` 服務物件。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。然後呼叫 Amazon EC2 服務物件的 `describeSecurityGroups` 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 [GitHub](#) 上找到這個範本程式碼。

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。若要存取 Amazon EC2，請建立 `AWS.EC2` 服務物件。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 執行個體將接收流量的 IP 通訊協定和傳入連接埠。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 [GitHub 上](#) 找到這個範本程式碼。

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。若要存取 Amazon EC2，請建立 AWS 服務物件。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 [GitHub 上](#) 找到這個範本程式碼。

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 AWS 您的帳戶相關聯。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，您使用一系列 Node.js 模組來執行多個涉及彈性 IP 地址的 Amazon EC2 操作。Node.js 模組使用適用於 JavaScript 的 SDK，透過使用這些 Amazon EC2 用戶端類別的方法來管理彈性 IP 地址：

- [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](#)。
- 建立 Amazon EC2 執行個體。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](#)。Amazon EC2

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。若要存取 Amazon EC2，請建立 `AWS.EC2` 服務物件。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。然後呼叫 Amazon EC2 服務物件的 `describeAddresses` 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 [GitHub 上](#) 找到這個範本程式碼。

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。若要存取 Amazon EC2，請建立 `AWS.EC2` 服務物件。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`。呼叫 Amazon EC2 服務物件的 `allocateAddress` 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` 來自新配置地址的 和 Amazon EC2 執行個體 `InstanceId` 的。然後呼叫 Amazon EC2 服務物件的 `associateAddresses` 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 [GitHub 上](#) 找到這個範本程式碼。

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。若要存取 Amazon EC2，請建立 `AWS.EC2` 服務物件。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。釋出彈性 IP 地址也會將其與任何 Amazon EC2 執行個體取消關聯。呼叫 Amazon EC2 服務物件的 `releaseAddress` 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 [GitHub 上](#) 找到這個範本程式碼。

AWS Elemental MediaConvert 範例

AWS Elemental MediaConvert 是以檔案為基礎的影片轉碼服務，具有廣播級功能。您可以將其用於建立跨網際網路的廣播資產及隨選視訊 (VOD) 交付資產。如需詳細資訊，請參閱《[AWS Elemental MediaConvert 使用者指南](#)》。

MediaConvert 的 JavaScript API 透過 `AWS.MediaConvert` 用戶端類別公開。如需詳細資訊，請參閱 API 參考 [Class: AWS.MediaConvert](#) 中的。

主題

- [取得 MediaConvert 的區域特定端點](#)
- [在 MediaConvert 中建立和管理轉碼任務](#)
- [在 MediaConvert 中使用任務範本](#)

取得 MediaConvert 的區域特定端點



這個 Node.js 程式碼範例會說明：

- 如何從 MediaConvert 擷取區域特定的端點。

使用案例

在此範例中，您使用 Node.js 模組來呼叫 MediaConvert 並擷取區域特定的端點。您可以從服務預設端點擷取端點 URL，因此尚不需要您的區域特定端點。此程式碼使用適用於 JavaScript 的 SDK，透過使用 MediaConvert 用戶端類別的此方法擷取此端點：

- [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，會適用於 JavaScript 的 AWS SDK 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 IAM 角色，讓 MediaConvert 存取您的輸入檔案，以及存放輸出檔案的 Amazon S3 儲存貯體。如需詳細資訊，請參閱 AWS Elemental MediaConvert 《使用者指南》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。

建立物件以傳遞 `AWS.MediaConvert` 用戶端類別 `describeEndpoints` 方法的空白請求參數。若要呼叫 `describeEndpoints` 方法，請建立叫用 `MediaConvert` 服務物件並傳遞參數的 `promise`。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 [GitHub 上](#) 找到這個範本程式碼。

在 MediaConvert 中建立和管理轉碼任務



這個 Node.js 程式碼範例會說明：

- 如何指定要與 MediaConvert 搭配使用的區域特定端點。
- 如何在 MediaConvert 中建立轉碼任務。
- 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，您使用 Node.js 模組呼叫 MediaConvert 來建立和管理轉碼任務。此程式碼使用適用於 JavaScript 的 SDK，透過 MediaConvert 用戶端類別的這些方法來執行此操作：

- [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空間。如需詳細資訊，請參閱 AWS Elemental MediaConvert 《使用者指南》中的 [建立檔案的儲存體](#)。
- 將輸入影片上傳至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 《使用者指南》中的 [支援的輸入轉碼器和容器](#)。

- 建立 IAM 角色，讓 MediaConvert 存取您的輸入檔案，以及存放輸出檔案的 Amazon S3 儲存貯體。如需詳細資訊，請參閱AWS Elemental MediaConvert 《使用者指南》中的[設定 IAM 許可](#)。

設定軟體開發套件

建立全域組態物件，然後設定程式碼的區域，以設定適用於 JavaScript 的 SDK。在此範例中，區域會設為 us-west-2。由於 MediaConvert 會為每個帳戶使用自訂端點，因此您也必須將AWS.MediaConvert用戶端類別設定為使用區域特定的端點。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](#) 來產生 JSON 任務參數，方法是在主控台中選擇您的任務設定，然後選擇任務區段底部的顯示任務 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 [GitHub 上](#) 找到這個範本程式碼。

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 [GitHub 上](#) 找到這個範本程式碼。

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 [GitHub 上](#) 找到這個範本程式碼。

在 MediaConvert 中使用任務範本



這個 Node.js 程式碼範例會說明：

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

使用案例

在 MediaConvert 中建立轉碼任務所需的 JSON 很詳細，其中包含大量設定。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，您使用 Node.js 模組呼叫 MediaConvert 來建立、使用和管理任務範本。此程式碼使用適用於 JavaScript 的 SDK，透過 MediaConvert 用戶端類別的這些方法來執行此操作：

- [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 IAM 角色，讓 MediaConvert 存取您的輸入檔案，以及存放輸出檔案的 Amazon S3 儲存貯體。如需詳細資訊，請參閱 AWS Elemental MediaConvert 《使用者指南》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 [在 MediaConvert 中建立和管理轉碼任務](#) 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 [GitHub 上](#) 找到這個範本程式碼。

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 [GitHub 上](#) 找到這個範本程式碼。

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。若要呼叫 `listTemplates` 方法，請建立叫用 `MediaConvert` 服務物件並傳遞參數的 `promise`。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 [GitHub 上](#) 找到這個範本程式碼。

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。若要呼叫 `deleteJobTemplate` 方法，請建立叫用 `MediaConvert` 服務物件並傳遞參數的 `promise`。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 [GitHub 上](#) 找到這個範本程式碼。

AWS IAM 範例

AWS Identity and Access Management (IAM) 是一種 Web 服務，可讓 Amazon Web Services 客戶管理 中的使用者和使用者許可 AWS。此服務以雲端中具有多個使用者或系統的組織為目標，這些使用者或系統使用 AWS 產品。透過 IAM，您可以集中管理使用者、存取金鑰等安全登入資料，以及控制使用者可以存取哪些 AWS 資源的許可。



適用於 IAM 的 JavaScript API 透過 `AWS.IAM` 用戶端類別公開。如需使用 IAM 用戶端類別的詳細資訊，請參閱 API 參考 [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，一系列 Node.js 模組用於在 IAM 中建立和管理使用者。Node.js 模組使用適用於 JavaScript 的 SDK，以下列 `AWS.IAM` 用戶端類別方法建立、刪除和更新使用者：

- [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 [GitHub 上](#) 找到這個範本程式碼。

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 [GitHub 上](#) 找到這個範本程式碼。

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 [GitHub 上](#) 找到這個範本程式碼。

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 [GitHub](#) 上找到這個範本程式碼。

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。
- 如何從角色連接和分離 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，一系列 Node.js 模組用於管理 IAM 中的政策。Node.js 模組使用適用於 JavaScript 的 SDK 來建立和刪除政策，以及使用這些 AWS.IAM 用戶端類別的方法連接和分離角色政策：

- [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](#)。
- 建立您可以連接政策的 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 [GitHub 上](#) 找到這個範本程式碼。

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 [GitHub 上](#) 找到這個範本程式碼。

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，其中包含取得連接至角色的受管 IAM 政策清單所需的參數，該政策由角色名稱組成。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，其中包含取得連接至角色的受管 IAM 政策清單所需的參數，該政策由角色名稱組成。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 從適用於 JavaScript 的 SDK 對進程式設計呼叫。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，會使用一系列 Node.js 模組來管理 IAM 中的存取金鑰。Node.js 模組使用適用於 JavaScript 的 SDK，以下列 AWS.IAM 用戶端類別的方法管理 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。建立包含建立新存取金鑰所需參數的 JSON 物件，其中包含 IAM 使用者名稱。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 [GitHub 上](#) 找到這個範本程式碼。

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 使用者名稱，以及您想要列出的存取金鑰對數目上限。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 [GitHub 上](#) 找到這個範本程式碼。

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 [GitHub 上](#) 找到這個範本程式碼。

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 [GitHub 上](#) 找到這個範本程式碼。

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 [GitHub 上](#) 找到這個範本程式碼。

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

若要在上啟用網站或應用程式的 HTTPS 連線 AWS，您需要 SSL/TLS 伺服器憑證。若要將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 或匯入 AWS Certificate Manager。

在此範例中，一系列 Node.js 模組用於處理 IAM 中的伺服器憑證。Node.js 模組使用適用於 JavaScript 的 SDK，以下列 AWS.IAM 用戶端類別的方法管理伺服器憑證：

- [listServerCertificates](#)
- [getServerCertificate](#)

- [updateServerCertificate](#)
- [deleteServerCertificate](#)

如需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 [GitHub 上](#) 找到這個範本程式碼。

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 [GitHub 上](#) 找到這個範本程式碼。

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 [GitHub 上](#) 找到這個範本程式碼。

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 [GitHub 上](#) 找到這個範本程式碼。

管理 IAM 帳戶別名



這個 Node.js 程式碼範例會說明：

- 如何管理您 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符，而不是 AWS 您的帳戶 ID，您可以為 AWS 您的帳戶 ID 建立別名。如果您建立 AWS 帳戶別名，您的登入頁面 URL 會變更以納入別名。

在此範例中，一系列 Node.js 模組用於建立和管理 IAM 帳戶別名。Node.js 模組使用適用於 JavaScript 的 SDK，以下列 AWS.IAM 用戶端類別方法管理別名：

- [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 [AWS 您的帳戶 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 [GitHub 上](#) 找到這個範本程式碼。

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 [GitHub 上](#) 找到這個範本程式碼。

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 [GitHub](#) 上找到這個範本程式碼。

Amazon Kinesis 範例

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，也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 透過 `AWS.Kinesis` 用戶端類別公開。如需使用 Kinesis 用戶端類別的詳細資訊，請參閱 API 參考 [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 在網頁中擷取捲動進度，做為串流頁面用量指標的範例，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。當讀取器捲動模擬部落格文章時，瀏覽器指令碼會使用適用於 JavaScript 的 SDK 記錄向下捲動頁面的距離，並使用 Kinesis 用戶端類別的 `putRecords` 方法將資料傳送至 Kinesis。然後，Amazon Kinesis Data Streams 擷取的串流資料可以由 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 Data Streams 的詳細資訊，請參閱《Amazon [Kinesis Data Streams 開發人員指南](#)》中的[管理 Amazon Kinesis Streams](#)。
- 建立 Amazon Cognito 身分集區，並針對未驗證的身分啟用存取。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，其政策會授予將資料提交至 Kinesis 串流的許可。如需建立 IAM 角色的詳細資訊，請參閱《IAM 使用者指南》中的[建立角色以委派許可給 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 <div> 元素中。此 <div> 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。其中一個元素會將適用於 JavaScript 的 SDK 新增至頁面，另一個元素則新增瀏覽器指令碼，擷取頁面上的捲動進度，並將其回報給 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

呼叫 `CognitoIdentityCredentials` 方法，並提供 Amazon Cognito 身分集區 ID，以取得設定 SDK 所需的登入資料。成功後，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 `<div>` 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接聽程式函數中建立，`scroll` 然後新增至一系列記錄，以定期提交至 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](#)。)

```
// 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 + scrollableElement.clientHeight) / 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);
});
```

將記錄提交至 Kinesis

每秒一次，如果陣列中有記錄，這些待定記錄會傳送至 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",  
    });  
  
    // 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 Web 服務，可提供高度可擴展的雲端儲存。Amazon S3 提供易於使用的物件儲存，具有簡單的 Web 服務介面，可從 Web 上的任何位置儲存和擷取任意數量的資料。



Amazon S3 的 JavaScript API 透過 `AWS.S3` 用戶端類別公開。如需使用 Amazon S3 用戶端類別的詳細資訊，請參閱 API 參考 [Class: AWS.S3](#) 中的。

主題

- [Amazon S3 瀏覽器範例](#)
- [Amazon S3 Node.js 範例](#)

Amazon S3 瀏覽器範例

下列主題顯示兩個範例，說明如何適用於 JavaScript 的 AWS SDK 在瀏覽器中使用與 Amazon S3 儲存貯體互動。

- 第一個顯示簡單的案例，其中任何（未驗證）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案例，允許使用者對儲存貯體中的相片執行操作，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](#)
- [從瀏覽器上傳相片至 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 Amazon Simple Storage Service (Amazon S3) 儲存貯體中建立相簿，並允許未經驗證的使用者檢視相片。

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。相簿位於上傳相片的 Amazon S3 儲存貯體中。



瀏覽器指令碼使用適用於 JavaScript 的 SDK 與 Amazon S3 儲存貯體互動。指令碼使用 Amazon S3 用戶端類別的 [listObjects](#) 方法，讓您檢視相簿。

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 身分集區使用相同的 AWS 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 Simple Storage Service 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。
- 選擇要在其中建立儲存貯 AWS 體的區域。這必須與您在後續先決條件任務建立 Amazon Cognito 身分集區時所使用的區域相同：建立身分集區。
- 遵循《Amazon Simple Storage Service 使用者指南》中的 [設定網站存取許可](#)，以設定儲存貯體許可。

建立 身分集區

在 [Amazon Cognito 主控台](#) 中，建立 Amazon Cognito 身分集區，如瀏覽器指令碼主題 [the section called “步驟 1：建立 Amazon Cognito 身分集區”](#) 入門中所述。

建立身分集區時，請記下身分集區名稱，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，您必須將許可新增至您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連接到 IAM 角色。

建立政策後，導覽回 [IAM 主控台](#)。尋找 Amazon Cognito 在先前先決條件任務建立的未驗證身分的 IAM 角色：建立身分集區。您使用剛建立的政策來新增許可至此身分。

雖然此任務的工作流程通常與瀏覽器指令碼主題中的 [the section called “步驟 2：將政策新增至已建立的 IAM 角色”](#) 入門相同，但有幾個差異需要注意：

- 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 [建立角色以委派許可給 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 <div> 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> 元素會新增存放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_VERSION_NUMBER，請參閱《API 參考指南》中適用於 JavaScript 的 SDK 的 API 參考。[適用於 JavaScript 的 AWS SDK](#)

```
<!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 身分集區 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）。

函數的其餘部分會從 Amazon S3 儲存貯體取得相簿清單，並產生在網頁上顯示相簿清單所需的 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` 函數會取得相簿名稱，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/>",
    '',
    "</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

本節包含範例的完整 HTML 和 JavaScript 程式碼，其中可檢視 Amazon S3 儲存貯體中的相片。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 [GitHub 上](#) 找到這個範本程式碼。

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/>",
                '',
                "</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
    .getElementsByName("img")[0]
    .setAttribute("style", "display:none;");
});
}

```

您可以在 [GitHub](#) 上找到這個範本程式碼。

從瀏覽器上傳相片至 Amazon S3



此瀏覽器程式碼範例顯示：

- 如何建立瀏覽器應用程式，允許使用者在 Amazon S3 儲存貯體中建立相簿，並將相片上傳至相簿。

使用案例

在此範例中，簡單的 HTML 頁面提供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，您可以將相片上傳至其中。該應用程式可讓您刪除新增的相片與相簿。



瀏覽器指令碼使用適用於 JavaScript 的 SDK 與 Amazon S3 儲存貯體互動。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，建立您將用來將相片存放在相簿中的 Amazon S3 儲存貯體。如需在 主控台中建立儲存貯體的詳細資訊，請參閱《Amazon Simple Storage Service 使用者指南》中的[建立儲存貯體](#)。請確定您在物件上同時擁有讀取和寫入許可。如需設定儲存貯體許可的詳細資訊，請參閱[設定網站存取的許可](#)。
- 在 [Amazon Cognito 主控台](#) 中，使用聯合身分建立 Amazon Cognito 身分集區，並在與 Amazon S3 儲存貯體相同的區域中為未驗證的使用者啟用存取。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 聯合身分的詳細資訊，請參閱《[Amazon Cognito 開發人員指南](#)》中的 [Amazon Cognito 身分集區（聯合身分）](#)。Amazon Cognito
- 在 [IAM 主控台](#) 中，尋找 Amazon Cognito 為未驗證使用者建立的 IAM 角色。新增下列政策，將讀取和寫入許可授予 Amazon S3 儲存貯體。如需建立 IAM 角色的詳細資訊，請參閱《IAM 使用者指南》中的[建立角色以委派許可給 AWS 服務](#)。

針對 Amazon Cognito 為未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 <div> 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> 元素會新增存放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

呼叫 `CognitoIdentityCredentials` 方法，並提供 Amazon Cognito 身分集區 ID，以取得設定 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，指出物件功能為資料夾。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。

函數的其餘部分會從 Amazon S3 儲存貯體取得相簿清單，並產生在網頁中顯示相簿清單所需的 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，以確保其中包含適當的字元。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` 函數會取得相簿名稱，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>",
        '',
      ]
    );
  });
}
```

```

        "</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

若要將相片上傳至 Amazon S3 儲存貯體中的相簿，應用程式的 `addPhoto` 函數會使用網頁中的檔案挑選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。

函數會呼叫 Amazon S3 服務物件的 `upload` 方法，以上傳相片。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();
      }
    );
  });
}
```

將相片上傳至 Amazon S3：完整程式碼

本節包含相片上傳到 Amazon S3 相簿之範例的完整 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 [GitHub 上](#) 找到這個範本程式碼。

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>",
        '',
        "</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 [GitHub 上](#) 找到這個範本程式碼。

Amazon S3 Node.js 範例

下列主題顯示如何使用 Node.js 適用於 JavaScript 的 AWS SDK 與 Amazon S3 儲存貯體互動的範例。

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：

- 如何取得和顯示您帳戶中的 Amazon S3 儲存貯體清單。
- 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，一系列 Node.js 模組用於取得現有 Amazon S3 儲存貯體的清單、建立儲存貯體，以及將檔案上傳至指定的儲存貯體。這些 Node.js 模組使用適用於 JavaScript 的開發套件，透過 Amazon S3 用戶端類別的這些方法，從 Amazon S3 儲存貯體取得資訊並上傳檔案至 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

建立全域組態物件，然後為您的程式碼設定區域，以設定適用於 JavaScript 的 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 儲存貯體的清單

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。若要存取 Amazon Simple Storage Service，請建立 AWS.S3 服務物件。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 [GitHub 上](#) 找到這個範本程式碼。

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。

新增變數以保留用來呼叫 Amazon S3 服務物件 `createBucket` 方法的參數，包括新建立儲存貯體的名稱。回呼函數會在 Amazon S3 成功建立儲存貯體後，將新儲存貯體的位置記錄至主控台。

```
// 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 [GitHub 上](#) 找到這個範本程式碼。

將檔案上傳至 Amazon S3 儲存貯體

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 `upload` 方法所需的參數來建立變數。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 [GitHub 上](#) 找到這個範本程式碼。

列出 Amazon S3 儲存貯體中的物件

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。

新增變數以保留用來呼叫 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 [GitHub 上](#) 找到這個範本程式碼。

刪除 Amazon S3 儲存貯體

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。

新增變數以保留用來呼叫 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 [GitHub 上](#) 找到這個範本程式碼。

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 模組使用適用於 JavaScript 的 SDK，使用以下 Amazon S3 用戶端類別的方法設定選取的 Amazon S3 儲存貯體：

- [getBucketCors](#)
- [putBucketCors](#)

如需搭配 Amazon S3 儲存貯體使用 CORS 組態的詳細資訊，請參閱《Amazon Simple Storage Service 使用者指南》中的[跨來源資源共享 \(CORS\)](#)。

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

建立全域組態物件，然後為您的程式碼設定 區域，以設定適用於 JavaScript 的 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，做為傳遞給回呼函數的 `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 [GitHub 上](#) 找到這個範本程式碼。

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 [GitHub](#) 上找到這個範本程式碼。

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 模組使用適用於 JavaScript 的 SDK，使用以下 Amazon S3 用戶端類別的方法管理 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 Simple Storage Service 使用者指南》中的[使用 ACLs 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

建立全域組態物件，然後為您的程式碼設定 區域，以設定適用於 JavaScript 的 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 會在傳遞給回呼函數的 `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 [GitHub 上](#) 找到這個範本程式碼。

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，一系列 Node.js 模組用於擷取、設定或刪除 Amazon S3 儲存貯體上的儲存貯體政策。Node.js 模組使用以下 Amazon S3 用戶端類別的方法，使用適用於 JavaScript 的 SDK 來設定所選 Amazon S3 儲存貯體的政策：

- [getBucketPolicy](#)
- [putBucketPolicy](#)
- [deleteBucketPolicy](#)

如需 Amazon S3 儲存貯體政策的詳細資訊，請參閱《Amazon Simple Storage Service 使用者指南》中的[使用儲存貯體政策和使用政策](#)。

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

建立全域組態物件，然後為您的程式碼設定 區域，以設定適用於 JavaScript 的 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 會在傳遞給回呼函數的 `data` 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 [GitHub 上](#) 找到這個範本程式碼。

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 [GitHub 上](#) 找到這個範本程式碼。

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 [GitHub 上](#) 找到這個範本程式碼。

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 模組使用適用於 JavaScript 的 SDK，使用以下 Amazon S3 用戶端類別的方法設定選取的 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 Simple Storage Service 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

建立全域組態物件，然後為您的程式碼設定 區域，以設定適用於 JavaScript 的 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 會在傳遞給回呼函數的 `data` 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 [GitHub](#) 上找到這個範本程式碼。

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 [GitHub 上](#) 找到這個範本程式碼。

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 [GitHub 上](#) 找到這個範本程式碼。

Amazon Simple Email Service 範例

Amazon Simple Email Service (Amazon SES) 是一種雲端電子郵件傳送服務，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 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 [Class: AWS.SES](#) 中的。

主題

- [管理 Amazon SES 身分](#)
- [在 Amazon SES 中使用電子郵件範本](#)
- [使用 Amazon SES 傳送電子郵件](#)
- [在 Amazon SES 中使用 IP 地址篩選條件來接收電子郵件](#)
- [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：

- 如何驗證與 Amazon SES 搭配使用的電子郵件地址和網域。
- 如何將 IAM 政策指派給您的 Amazon SES 身分。
- 如何列出 AWS 帳戶的所有 Amazon SES 身分。
- 如何刪除與 Amazon SES 搭配使用的身分。

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

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 Simple [Email Service](#) 開發人員指南》中的在 [Amazon SES 中驗證電子郵件地址和網域](#)。如需在 Amazon SES 中傳送授權的資訊，請參閱 [Amazon SES 傳送授權概觀](#)。

使用案例

在此範例中，您使用一系列 Node.js 模組來驗證和管理 Amazon SES 身分。Node.js 模組使用適用於 JavaScript 的 SDK，使用以下 AWS.SES 用戶端類別的方法驗證電子郵件地址和網域：

- [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

建立全域組態物件，然後設定程式碼的區域，以設定適用於 JavaScript 的 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 模組列出要與 Amazon SES 搭配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 `listIdentities` 方法，請建立叫用 Amazon SES 服務物件、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 [GitHub 上](#) 找到這個範本程式碼。

驗證電子郵件地址身分

在此範例中，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。網域會新增至要驗證的 Amazon SES。

```
node ses_verifyemailidentity.js
```

您可以在 [GitHub 上](#) 找到這個範本程式碼。

驗證網域身分

在此範例中，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 `verifyDomainIdentity` 方法，請建立叫用 Amazon SES 服務物件、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。網域會新增至要驗證的 Amazon SES。

```
node ses_verifydomainidentity.js
```

您可以在 [GitHub 上](#) 找到這個範本程式碼。

刪除身分

在此範例中，使用 Node.js 模組來刪除與 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。若要呼叫 `deleteIdentity` 方法，請建立 `request` 以叫用 Amazon SES 服務物件，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 [GitHub 上](#) 找到這個範本程式碼。

在 Amazon SES 中使用電子郵件範本



這個 Node.js 程式碼範例会說明：

- 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電子郵件訊息。如需如何在 Amazon Simple Email Service 中建立和使用電子郵件範本的詳細資訊，請參閱《Amazon Simple Email Service 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 模組使用適用於 JavaScript 的 SDK，以用戶端 `AWS.SES` 類別的這些方法建立和使用電子郵件範本：

- [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 模組來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 會傳回範本清單。

```
node ses_listtemplates.js
```

您可以在 [GitHub 上](#) 找到這個範本程式碼。

取得電子郵件範本

在此範例中，使用 Node.js 模組來取得電子郵件範本以搭配 Amazon SES 使用。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 會傳回範本詳細資訊。

```
node ses_gettemplate.js
```

您可以在 [GitHub 上](#) 找到這個範本程式碼。

建立電子郵件範本

在此範例中，使用 Node.js 模組來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。範本會新增至 Amazon SES。

```
node ses_createtemplate.js
```

您可以在 [GitHub 上](#) 找到這個範本程式碼。

更新電子郵件範本

在此範例中，使用 Node.js 模組來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 會傳回範本詳細資訊。

```
node ses_updatetemplate.js
```

您可以在 [GitHub 上](#) 找到這個範本程式碼。

刪除電子郵件範本

在此範例中，使用 Node.js 模組來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 會傳回範本詳細資訊。

```
node ses_deletetemplate.js
```

您可以在 [GitHub 上](#) 找到這個範本程式碼。

使用 Amazon SES 傳送電子郵件



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 為您提供兩種不同的傳送電子郵件方式，取決於您希望對電子郵件訊息的合成進行多少控制：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 模組使用適用於 JavaScript 的 SDK，以用戶端AWS.SES類別的這些方法建立和使用電子郵件範本：

- [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，並立即將訊息排入佇列以供傳送。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 Simple Email Service 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。電子郵件會排入佇列，以供 Amazon SES 傳送。

```
node ses_sendemail.js
```

您可以在 [GitHub 上](#) 找到這個範本程式碼。

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。電子郵件會排入佇列，以供 Amazon SES 傳送。

```
node ses_sendtemplatedemail.js
```

您可以在 [GitHub 上](#) 找到這個範本程式碼。

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。電子郵件會排入佇列，以供 Amazon SES 傳送。

```
node ses_sendbulktemplatedemail.js
```

您可以在 [GitHub](#) 上找到這個範本程式碼。

在 Amazon SES 中使用 IP 地址篩選條件來接收電子郵件



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，篩選條件是一種資料結構，其中包含名稱、IP 地址範圍，以及是否允許或封鎖來自其中的郵件。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 Email Service 開發人員指南》](#) 中的 [Amazon SES 電子郵件接收概念](#)。

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 模組使用適用於 JavaScript 的 SDK，以用戶端 AWS.SES 類別的這些方法建立和使用電子郵件範本：

- [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

建立全域組態物件，然後設定程式碼的區域，以設定適用於 JavaScript 的 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。篩選條件是在 Amazon SES 中建立。

```
node ses_createreceiptfilter.js
```

您可以在 [GitHub 上](#) 找到這個範本程式碼。

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 會傳回篩選條件清單。

```
node ses_listreceiptfilters.js
```

您可以在 [GitHub 上](#) 找到這個範本程式碼。

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。篩選條件會從 Amazon SES 中刪除。

```
node ses_deletereceiptfilter.js
```

您可以在 [GitHub 上](#) 找到這個範本程式碼。

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 中的接收規則指定如何處理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。如果傳入電子郵件的收件人符合接收規則條件中指定的收件人，Amazon SES 會執行接收規則指定的動作。

若要使用 Amazon SES 做為您的電子郵件接收者，您必須至少有一個作用中的接收規則集。接收規則集是有序的接收規則集合，指定 Amazon SES 應該如何處理其在已驗證網域中收到的郵件。如需詳細資訊，請參閱《[Amazon Simple Email Service 開發人員指南](#)》中的[建立 Amazon SES 電子郵件接收的接收規則](#)和[建立 Amazon SES 電子郵件接收的接收規則集](#)。

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 模組使用適用於 JavaScript 的 SDK，以用戶端 `AWS.SES` 類別的這些方法建立和使用電子郵件範本：

- [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 接收規則

Amazon SES 的每個接收規則都包含動作的排序清單。此範例會使用 Amazon S3 動作建立接收規則，將郵件訊息傳遞至 Amazon S3 儲存貯體。如需接收規則動作的詳細資訊，請參閱《Amazon Simple Email Service 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，請建立儲存貯體政策以授予 Amazon SES PutObject 許可。如需建立此儲存貯體政策的相關資訊，請參閱《[Amazon Simple Email Service 開發人員指南](#)》中的[授予 Amazon SES 寫入 Amazon S3 儲存貯體的許可](#)。

在此範例中，使用 Node.js 模組在 Amazon SES 中建立接收規則，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 會建立接收規則。

```
node ses_createreceiptrule.js
```

您可以在 [GitHub](#) 上找到這個範本程式碼。

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_deletereciptrule.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 [GitHub 上](#) 找到這個範本程式碼。

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 [GitHub 上](#) 找到這個範本程式碼。

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 `deleteReceiptRuleSet` 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 [GitHub](#) 上找到這個範本程式碼。

Amazon Simple Notification Service 範例

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

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



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

Amazon SNS 的 JavaScript API 透過 公開 [Class: AWS.SNS](#)。

主題

- [在 Amazon SNS 中管理主題](#)
- [在 Amazon SNS 中發佈訊息](#)
- [在 Amazon SNS 中管理訂閱](#)
- [使用 Amazon SNS 傳送簡訊](#)

在 Amazon SNS 中管理主題



這個 Node.js 程式碼範例會說明：

- 如何在 Amazon SNS 中建立可發佈通知的主題。
- 如何刪除在 Amazon SNS 中建立的主題。
- 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，您會使用一系列 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 模組使用適用於 JavaScript 的 SDK，以下列 AWS.SNS 用戶端類別方法管理主題：

- [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 模組來建立 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 `createTopic` 方法，請建立叫用 Amazon SNS 服務物件、傳遞參數物件的 promise。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 [GitHub 上](#) 找到這個範本程式碼。

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 `listTopics` 方法，請建立叫用 Amazon SNS 服務物件、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 [GitHub 上](#) 找到這個範本程式碼。

刪除主題

在此範例中，使用 Node.js 模組來刪除 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 `deleteTopic` 方法，請建立叫用 Amazon SNS 服務物件、傳遞參數物件的 promise。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 [GitHub 上](#) 找到這個範本程式碼。

取得主題屬性

在此範例中，使用 Node.js 模組來擷取 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 `getTopicAttributes` 方法，請建立叫用 Amazon SNS 服務物件、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 [GitHub 上](#) 找到這個範本程式碼。

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 `setTopicAttributes` 方法，請建立叫用 Amazon SNS 服務物件、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 [GitHub](#) 上找到這個範本程式碼。

在 Amazon SNS 中發佈訊息



這個 Node.js 程式碼範例會說明：

- 如何將訊息發佈至 Amazon SNS 主題。

使用案例

在此範例中，您會使用一系列 Node.js 模組，將訊息從 Amazon SNS 發佈至主題端點、電子郵件或電話號碼。Node.js 模組使用適用於 JavaScript 的 SDK，使用此 `AWS.SNS` 用戶端類別的 `publish` 方法傳送訊息：

- [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](#)。

將訊息發佈至 Amazon SNS 主題

在此範例中，使用 Node.js 模組將訊息發佈至 Amazon SNS 主題。以檔名 `sns_publish_to_topic.js` 建立一個 Node.js 模組。依前述內容設定軟體開發套件。

建立包含發佈訊息參數的物件，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。建立叫用 Amazon SNS 服務物件、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 [GitHub 上](#) 找到這個範本程式碼。

在 Amazon SNS 中管理訂閱



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 AWS Lambda 函數至 Amazon SNS 主題。

- 如何取消訂閱 Amazon SNS 主題。

使用案例

在此範例中，您會使用一系列 Node.js 模組，將通知訊息發佈至 Amazon SNS 主題。Node.js 模組使用適用於 JavaScript 的 SDK，以下列 AWS.SNS 用戶端類別方法管理主題：

- [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 `listSubscriptionsByTopic` 方法，請建立叫用 Amazon SNS 服務物件、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 sublistPromise = new AWS.SNS({ apiVersion: "2010-03-31" })
    .listSubscriptionsByTopic(params)
    .promise();

// Handle promise's fulfilled/rejected states
sublistPromise
    .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 [GitHub 上](#) 找到這個範本程式碼。

使用電子郵件地址訂閱主題

在此範例中，使用 Node.js 模組來訂閱電子郵件地址，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 `subscribe` 方法將數個不同的端點訂閱至 Amazon SNS 主題，取決於傳遞參數所使用的值，因為本主題中的其他範例會顯示。

若要呼叫 `subscribe` 方法，請建立叫用 Amazon SNS 服務物件、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 [GitHub 上](#) 找到這個範本程式碼。

使用應用程式端點訂閱主題

在此範例中，使用 Node.js 模組來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 `subscribe` 方法，請建立叫用 Amazon SNS 服務物件、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 [GitHub 上](#) 找到這個範本程式碼。

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 的，以及 AWS Lambda 函數的 ARN 做為 Endpoint 參數。將參數傳遞至 AWS.SNS 用戶端類別的 subscribe 方法。

若要呼叫 subscribe 方法，請建立叫用 Amazon SNS 服務物件、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 [GitHub 上](#) 找到這個範本程式碼。

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 `unsubscribe` 方法，請建立叫用 Amazon SNS 服務物件、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 [GitHub 上](#) 找到這個範本程式碼。

使用 Amazon SNS 傳送簡訊



這個 Node.js 程式碼範例會說明：

- 如何取得和設定 Amazon SNS 的簡訊偏好設定。
- 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，您會使用一系列 Node.js 模組，將 SMS 文字訊息從 Amazon SNS 發佈至啟用 SMS 的裝置。Node.js 模組使用適用於 JavaScript 的 SDK，以用戶端 `AWS.SNS` 類別的下列方法發佈 SMS 訊息：

- [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 來指定簡訊的偏好設定，例如如何最佳化交付（用於成本或可靠交付）、您的每月花費限制、如何記錄訊息交付，以及是否訂閱每日簡訊用量報告。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。如需可用 SMS 屬性的詳細資訊，請參閱《Amazon Simple Notification Service 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 `getSMSAttributes` 方法，請建立叫用 Amazon SNS 服務物件、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 [GitHub 上](#) 找到這個範本程式碼。

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。如需可用 SMS 屬性的詳細資訊，請參閱《Amazon Simple Notification Service 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 `getSMSAttributes` 方法，請建立叫用 Amazon SNS 服務物件、傳遞參數物件的 promise。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 [GitHub 上](#) 找到這個範本程式碼。

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

`checkIfPhoneNumberIsOptedOut` 方法，請建立叫用 Amazon SNS 服務物件、傳遞參數物件的 `promise`。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 [GitHub 上](#) 找到這個範本程式碼。

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 `listPhoneNumbersOptedOut` 方法，請建立叫用 Amazon SNS 服務物件、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 [GitHub 上](#) 找到這個範本程式碼。

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 `publish` 方法，請建立叫用 Amazon SNS 服務物件、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 [GitHub 上](#) 找到這個範本程式碼。

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 可讓您解耦雲端應用程式的元件。Amazon SQS 包含具有高輸送量且至少一次處理的標準佇列，以及提供 FIFO (先進先出) 交付並僅處理一次的 FIFO 佇列。



Amazon SQS 的 JavaScript API 透過 `AWS.SQS` 用戶端類別公開。如需使用 Amazon SQS 用戶端類別的詳細資訊，請參閱 API 參考 [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 模組使用適用於 JavaScript 的 SDK 來啟用佇列，以呼叫用戶端 `AWS.SQS` 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 Simple [Queue Service 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 [GitHub 上](#) 找到這個範本程式碼。

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 [GitHub 上](#) 找到這個範本程式碼。

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 [GitHub 上](#) 找到這個範本程式碼。

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 [GitHub 上](#) 找到這個範本程式碼。

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 模組使用適用於 JavaScript 的 SDK，透過使用 AWS.SQS 用戶端類別的這些方法來傳送和接收訊息：

- [sendMessage](#)
- [receiveMessage](#)
- [deleteMessage](#)

如需 Amazon SQS 訊息的詳細資訊，請參閱《Amazon Simple Queue Service [開發人員指南](#)》中的 [傳送訊息至 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 [GitHub 上](#) 找到這個範本程式碼。

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 [GitHub 上](#) 找到這個範本程式碼。

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 模組使用適用於 JavaScript 的 SDK，透過使用此 `AWS.SQS` 用戶端類別的方法來管理可見性逾時：

- [changeMessageVisibility](#)

如需 Amazon SQS 可見性逾時的詳細資訊，請參閱《Amazon Simple Queue Service 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 Simple Queue Service，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 [GitHub 上](#) 找到這個範本程式碼。

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

長輪詢允許 Amazon SQS 在傳送回應之前等待指定的時間，讓訊息在佇列中變成可用，以減少空回應的數量。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 模組使用適用於 JavaScript 的 SDK，以下列 `AWS.SQS` 用戶端類別的方法啟用長輪詢：

- [setQueueAttributes](#)
- [receiveMessage](#)
- [createQueue](#)

如需 Amazon SQS 長輪詢的詳細資訊，請參閱《Amazon Simple Queue Service 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 [GitHub 上](#) 找到這個範本程式碼。

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 Simple Queue Service，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 [GitHub 上](#) 找到這個範本程式碼。

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 Simple Queue Service，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 [GitHub 上](#) 找到這個範本程式碼。

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 模組使用適用於 JavaScript 的 SDK，使用此 `AWS.SQS` 用戶端類別的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 Simple Queue Service 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 佇列以做為無效字母佇列。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 [GitHub](#) 上找到這個範本程式碼。

教學課程

以下教學課程說明如何使用 適用於 JavaScript 的 AWS SDK 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

搭配適用於 JavaScript 的 SDK 使用 Node.js 的常見案例是在 Amazon Elastic Compute Cloud (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。

先決條件

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 Amazon EC2 Machine Image (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 Amazon EC2](#)。

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 不會控制下列程式碼。在您執行前，請務必驗證其真確性及完整性。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/v0.39.7/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. 在命令列 nvm 中輸入以下內容以載入。

```
source ~/.bashrc
```

4. 使用 nvm 在命令列中輸入以下內容，以安裝最新的 Node.js LTS 版本。

```
nvm install --lts
```

安裝 Node.js 也會安裝 Node Package Manager (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。如果執行個體已終止，您需要再次安裝節點。另一種方法是在您擁有要保留的組態後，建立 Amazon EC2 執行個體的 Amazon Machine Image (AMI)，如下列主題所述。

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 Machine Image (AMI)。建立 AMI 可讓您輕鬆地使用相同的 Node.js 安裝佈建多個 Amazon EC2 執行個體。如需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 支援的 Linux AMI](#)。Amazon EC2

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：

- 節點版本管理工具 (nvm)：請參閱 [GitHub 上的 nvm 儲存庫](#)。
- 節點套件管理工具 (npm)：請參閱 [npm 網站](#)。

JavaScript API 參考

適用於 JavaScript 的最新版本 SDK 的 API 參考主題位於：

[適用於 JavaScript 的 AWS SDK API 參考指南](#)。

GitHub 上的軟體開發套件變更記錄

如需 2.4.8 版以上版本的變更記錄，請前往：

[變更日誌](#)。

遷移至 的 v3 適用於 JavaScript 的 AWS SDK

適用於 JavaScript 的 AWS SDK 第 3 版是第 2 版的主要重寫。如需遷移至第 3 版的詳細資訊，請參閱《開發人員指南 v3》中的[從 2.x 版遷移至 3.x 適用於 JavaScript 的 AWS SDK 版](#)。適用於 JavaScript 的 AWS SDK

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 與您之間共同責任。[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。我們的安全責任是最高優先順序 AWS，而第三方稽核人員會在[AWS 合規計劃](#)中定期測試和驗證我們的安全有效性。

雲端的安全性 – 您的責任取決於您使用 AWS 的服務，以及其他因素，包括資料的敏感度、組織的需求，以及適用的法律和法規。

此 AWS 產品或服務會透過其支援的特定 Amazon Web Services (AWS) 服務，遵循[共同責任模型](#)。如需 AWS 服務安全資訊，請參閱[AWS 服務安全文件頁面](#)，以及[AWS 合規計劃在 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 產品或服務的彈性](#)
- [此 AWS 產品或服務的基礎設施安全](#)
- [強制執行 TLS 最低版本](#)

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。如此模型所述，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[資料隱私權常見問答集](#)。如需有關歐洲資料保護的相關資訊，請參閱 AWS 安全性部落格上的 [AWS 共同的責任模型和 GDPR](#) 部落格文章。

基於資料保護目的，我們建議您保護 AWS 帳戶登入資料，並使用 AWS IAM Identity Center 或 AWS Identity and Access Management (IAM) 設定個別使用者。如此一來，每個使用者都只會獲得授與完成其任務所必須的許可。我們也建議您採用下列方式保護資料：

- 每個帳戶均要使用多重要素驗證 (MFA)。
- 使用 SSL/TLS 與 AWS 資源通訊。我們需要 TLS 1.2 並建議使用 TLS 1.3。
- 使用 設定 API 和使用者活動記錄 AWS CloudTrail。如需有關使用 CloudTrail 追蹤擷取 AWS 活動的資訊，請參閱AWS CloudTrail 《使用者指南》中的[使用 CloudTrail 追蹤](#)。
- 使用 AWS 加密解決方案，以及其中的所有預設安全控制 AWS 服務。
- 使用進階的受管安全服務 (例如 Amazon Macie)，協助探索和保護儲存在 Amazon S3 的敏感資料。
- 如果您在 AWS 透過命令列界面或 API 存取 時需要 FIPS 140-3 驗證的密碼編譯模組，請使用 FIPS 端點。如需有關 FIPS 和 FIPS 端點的更多相關資訊，請參閱[聯邦資訊處理標準 \(FIPS\) 140-3](#)。

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 AWS SDKs的其他 AWS 服務 時。您在標籤或自由格式文字欄位中輸入的任何資料都可能用於計費或診斷日誌。如果您提供外部伺服器的 URL，我們強烈建議請勿在驗證您對該伺服器請求的 URL 中包含憑證資訊。

身分和存取權管理

AWS Identity and Access Management (IAM) 是一種 AWS 服務，可協助管理員安全地控制對 AWS 資源的存取。IAM 管理員可控制誰可以進行驗證（登入）和授權（具有許可）來使用 AWS 資源。IAM 是您可以免費使用 AWS 服務的。

主題

- [目標對象](#)
- [使用身分驗證](#)
- [使用政策管理存取權](#)
- [AWS 服務 如何使用 IAM](#)
- [對 AWS 身分和存取進行故障診斷](#)

目標對象

使用 AWS Identity and Access Management (IAM) 的方式會有所不同，取決於您在 中執行的工作 AWS。

服務使用者 – 如果您使用 AWS 服務 執行任務，管理員會為您提供所需的登入資料和許可。當您使用更多 AWS 功能來執行工作時，您可能需要額外的許可。了解存取許可的管理方式可協助您向管理員請

求正確的許可。如果您無法存取 中的功能 AWS，請參閱 [對 AWS 身分和存取進行故障診斷](#) 或 AWS 服務 您正在使用的 使用者指南。

服務管理員 – 如果您負責公司 AWS 的資源，您可能擁有 的完整存取權 AWS。您的任務是判斷服務使用者應該存取哪些 AWS 功能和資源。接著，您必須將請求提交給您的 IAM 管理員，來變更您服務使用者的許可。檢閱此頁面上的資訊，了解 IAM 的基本概念。若要進一步了解貴公司如何使用 IAM AWS，請參閱您正在使用的 使用者指南 AWS 服務。

IAM 管理員：如果您是 IAM 管理員，建議您掌握如何撰寫政策以管理 AWS 存取權的詳細資訊。若要檢視您可以在 IAM 中使用的以 AWS 身分為基礎的政策範例，請參閱 AWS 服務 您正在使用的 使用者指南。

使用身分驗證

身分驗證是您 AWS 使用身分憑證登入 的方式。您必須以 AWS 帳戶根使用者身分、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 登入資料，都是聯合身分的範例。您以聯合身分登入時，您的管理員先前已設定使用 IAM 角色的聯合身分。當您使用聯合 AWS 身分存取 時，您會間接擔任角色。

根據您的使用者類型，您可以登入 AWS Management Console 或 AWS 存取入口網站。如需登入的詳細資訊 AWS，請參閱 AWS 登入 《使用者指南》中的 [如何登入您的 AWS 帳戶](#)。

如果您以 AWS 程式設計方式存取，AWS 會提供軟體開發套件 (SDK) 和命令列界面 (CLI)，以使用您的登入資料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自行簽署請求的詳細資訊，請參閱《IAM 使用者指南》中的 [適用於 API 請求的 AWS Signature 第 4 版](#)。

無論您使用何種身分驗證方法，您可能都需要提供額外的安全性資訊。例如，AWS 建議您使用多重驗證 (MFA) 來提高帳戶的安全性。如需更多資訊，請參閱《AWS IAM Identity Center 使用者指南》中的 [多重要素驗證](#) 和《IAM 使用者指南》中的 [IAM 中的 AWS 多重要素驗證](#)。

AWS 帳戶 根使用者

當您建立 時 AWS 帳戶，您會從一個登入身分開始，該身分可完整存取帳戶中的所有 AWS 服務 和資源。此身分稱為 AWS 帳戶 Theroot 使用者，可透過使用您用來建立帳戶的電子郵件地址和密碼登入來存取。強烈建議您不要以根使用者處理日常任務。保護您的根使用者憑證，並將其用來執行只能由根使用者執行的任務。如需這些任務的完整清單，了解需以根使用者登入的任務，請參閱 IAM 使用者指南中的 [需要根使用者憑證的任務](#)。

聯合身分

根據最佳實務，要求人類使用者，包括需要管理員存取權的使用者，使用聯合身分提供者 AWS 服務來使用臨時憑證來存取。

聯合身分是您企業使用者目錄、Web 身分提供者、AWS Directory Service、Identity Center 目錄，或 AWS 服務是透過身分來源提供的登入資料存取的任何使用者。當聯合身分存取時 AWS 帳戶，它們會擔任角色，而角色會提供臨時登入資料。

對於集中式存取權管理，我們建議您使用 AWS IAM Identity Center。您可以在 IAM Identity Center 中建立使用者和群組，也可以連接並同步到您自己的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 的詳細資訊，請參閱 AWS IAM Identity Center 使用者指南中的[什麼是 IAM Identity Center？](#)。

IAM 使用者和群組

[IAM 使用者](#)是中的身分 AWS 帳戶，具有單一人員或應用程式的特定許可。建議您盡可能依賴臨時憑證，而不是擁有建立長期憑證 (例如密碼和存取金鑰) 的 IAM 使用者。但是如果特定使用案例需要擁有長期憑證的 IAM 使用者，建議您輪換存取金鑰。如需更多資訊，請參閱[IAM 使用者指南](#)中的為需要長期憑證的使用案例定期輪換存取金鑰。

[IAM 群組](#)是一種指定 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，您可以擁有一個名為 IAMAdmins 的群組，並給予該群組管理 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，但角色僅提供臨時憑證。如需更多資訊，請參閱《IAM 使用者指南》中的[IAM 使用者的使用案例](#)。

IAM 角色

[IAM 角色](#)是具有特定許可 AWS 帳戶的身分。它類似 IAM 使用者，但不與特定的人員相關聯。若要暫時在中擔任 IAM 角色 AWS Management Console，您可以從[使用者切換至 IAM 角色（主控台）](#)。您可以透過呼叫 AWS CLI 或 AWS API 操作或使用自訂 URL 來擔任角色。如需使用角色的方法詳細資訊，請參閱《IAM 使用者指南》中的[擔任角色的方法](#)。

使用臨時憑證的 IAM 角色在下列情況中非常有用：

- 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。如需有關聯合角色的相關資訊

訊，請參閱《[IAM 使用者指南](#)》中的為第三方身分提供者 (聯合) 建立角色。如果您使用 IAM Identity Center，則需要設定許可集。為控制身分驗證後可以存取的內容，IAM Identity Center 將許可集與 IAM 中的角色相關聯。如需有關許可集的資訊，請參閱 AWS IAM Identity Center 使用者指南中的[許可集](#)。

- 暫時 IAM 使用者許可 – IAM 使用者或角色可以擔任 IAM 角色來暫時針對特定任務採用不同的許可。
- 跨帳戶存取權：您可以使用 IAM 角色，允許不同帳戶中的某人 (信任的主體) 存取您帳戶的資源。角色是授予跨帳戶存取權的主要方式。不過，對於某些 AWS 服務，您可以直接將政策連接到資源 (而不是使用角色做為代理)。如需了解使用角色和資源型政策進行跨帳戶存取之間的差異，請參閱《IAM 使用者指南》中的[IAM 中的跨帳戶資源存取](#)。
- 跨服務存取 – 有些 AWS 服務 使用其他 中的功能 AWS 服務。例如，當您在服務中進行呼叫時，該服務通常會在 Amazon EC2 中執行應用程式或將物件儲存在 Amazon Simple Storage Service (Amazon S3) 中。服務可能會使用呼叫主體的許可、使用服務角色或使用服務連結角色來執行此作業。
- 轉送存取工作階段 (FAS) – 當您使用 IAM 使用者或角色在其中執行動作時 AWS，您會被視為委託人。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，結合 請求向下游服務 AWS 服務 提出請求。只有當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，才會提出 FAS 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 請求時的政策詳細資訊，請參閱《[轉發存取工作階段](#)》。
- 服務角色 – 服務角色是服務擔任的 [IAM 角色](#)，可代表您執行動作。IAM 管理員可以從 IAM 內建立、修改和刪除服務角色。如需詳細資訊，請參閱《IAM 使用者指南》中的[建立角色以委派許可權給 AWS 服務](#)。
- 服務連結角色 – 服務連結角色是一種連結至 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。服務連結角色會出現在您的 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 EC2 上執行的應用程式 – 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登入資料，以及提出 AWS CLI 或 AWS API 請求。這是在 EC2 執行個體內儲存存取金鑰的較好方式。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描述檔。執行個體設定檔包含該角色，並且可讓 EC2 執行個體上執行的程式取得臨時憑證。如需詳細資訊，請參閱《IAM 使用者指南》中的[使用 IAM 角色來授予許可權給 Amazon EC2 執行個體上執行的應用程式](#)。

使用政策管理存取權

您可以透過建立政策並將其連接到身分或資源 AWS 來控制 AWS 中的存取。政策是 中的物件，AWS 當與身分或資源建立關聯時，會定義其許可。當委託人 (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。大多數政策會以 JSON 文件 AWS 形式存放在中。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。若要授予使用者對其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，使用者便能擔任這些角色。

IAM 政策定義該動作的許可，無論您使用何種方法來執行操作。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 API 取得角色資訊。

身分型政策

身分型政策是可以附加到身分 (例如 IAM 使用者、使用者群組或角色) 的 JSON 許可政策文件。這些政策可控制身分在何種條件下能對哪些資源執行哪些動作。如需了解如何建立身分型政策，請參閱《IAM 使用者指南》中的 [透過客戶管理政策定義自訂 IAM 許可](#)。

身分型政策可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。如需了解如何在受管政策及內嵌政策之間選擇，請參閱《IAM 使用者指南》中的 [在受管政策和內嵌政策間選擇](#)。

資源型政策

資源型政策是連接到資源的 JSON 政策文件。資源型政策的最常見範例是 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](#)。委託人可以包含帳戶、使用者、角色、聯合身分使用者或 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中使用來自 IAM 的 AWS 受管政策。

存取控制清單 (ACL)

存取控制清單 (ACL) 可控制哪些主體 (帳戶成員、使用者或角色) 擁有存取某資源的許可。ACL 類似於資源型政策，但它們不使用 JSON 政策文件格式。

Amazon S3、AWS WAF 和 Amazon VPC 是支援 ACLs 的服務範例。如需進一步了解 ACL，請參閱 Amazon Simple Storage Service 開發人員指南中的 [存取控制清單 \(ACL\) 概觀](#)。

其他政策類型

AWS 支援其他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 – 許可範圍**是一種進階功能，可供您設定身分型政策能授予 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 JSON 政策，可指定中組織或組織單位 (OU) 的最大許可 AWS Organizations。AWS Organizations 是一種服務，用於分組和集中管理您企業擁有 AWS 帳戶的多個。若您啟用組織中的所有功能，您可以將服務控制政策 (SCP) 套用到任何或所有帳戶。SCP 會限制成員帳戶中實體的許可，包括每個實體 AWS 帳戶根使用者。如需 Organizations 和 SCP 的詳細資訊，請參閱《AWS Organizations 使用者指南》中的 [服務控制政策](#)。
- **資源控制政策 (RCP)** – RCP 是 JSON 政策，可用來設定您帳戶中資源的可用許可上限，採取這種方式就不需要更新附加至您所擁有的每個資源的 IAM 政策。RCP 會限制成員帳戶中資源的許可，並可能影響身分的有效許可，包括 AWS 帳戶根使用者，無論它們是否屬於您的組織。如需 Organizations 和 RCPs 的詳細資訊，包括 AWS 服務支援 RCPs 清單，請參閱 AWS Organizations 《使用者指南》中的 [資源控制政策 \(RCPs\)](#)。
- **工作階段政策** – 工作階段政策是一種進階政策，您可以在透過撰寫程式的方式建立角色或聯合使用者的暫時工作階段時，做為參數傳遞。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 在涉及多個政策類型時決定是否允許請求，請參閱《IAM 使用者指南》中的 [政策評估邏輯](#)。

AWS 服務 如何使用 IAM

若要深入了解 如何使用 AWS 服務 大多數 IAM 功能，請參閱《IAM 使用者指南》中的與 [AWS IAM 搭配使用的服務](#)。

若要了解如何 AWS 服務 搭配 IAM 使用特定，請參閱相關服務使用者指南的安全章節。

對 AWS 身分和存取進行故障診斷

使用以下資訊來協助您診斷和修正使用 AWS 和 IAM 時可能遇到的常見問題。

主題

- [我無權在 中執行動作 AWS](#)
- [我未獲得執行 iam:PassRole 的授權](#)
- [我想要允許 以外的人員 AWS 帳戶 存取我的 AWS 資源](#)

我無權在 中執行動作 AWS

如果您收到錯誤，告知您未獲授權執行動作，您的政策必須更新，允許您執行動作。

下列範例錯誤會在mateojackson IAM 使用者嘗試使用主控台檢視一個虛構 *my-example-widget* 資源的詳細資訊，但卻無虛構 *awes:GetWidget* 許可時發生。

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
awes:GetWidget on resource: my-example-widget
```

在此情況下，必須更新 mateojackson 使用者的政策，允許使用 *awes:GetWidget* 動作存取 *my-example-widget* 資源。

如果您需要協助，請聯絡您的 AWS 管理員。您的管理員提供您的簽署憑證。

我未獲得執行 iam:PassRole 的授權

如果您收到錯誤，告知您未獲授權執行 iam:PassRole 動作，您的政策必須更新，允許您將角色傳遞給 AWS。

有些 AWS 服務 可讓您將現有角色傳遞給該服務，而不是建立新的服務角色或服務連結角色。如需執行此作業，您必須擁有將角色傳遞至該服務的許可。

名為 marymajor 的 IAM 使用者嘗試使用主控台在 AWS 中執行動作時，發生下列範例錯誤。但是，動作要求服務具備服務角色授予的許可。Mary 沒有將角色傳遞至該服務的許可。

```
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。針對支援基於資源的政策或存取控制清單 (ACL) 的服務，您可以使用那些政策來授予人員存取您的資源的許可。

如需進一步了解，請參閱以下內容：

- 若要了解 是否 AWS 支援這些功能，請參閱 [AWS 服務 如何使用 IAM](#)。
- 若要了解如何 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[您擁有 AWS 帳戶 的另一個 中提供存取權給](#) IAM 使用者。
- 若要了解如何將資源的存取權提供給第三方 AWS 帳戶，請參閱《IAM 使用者指南》中的[將存取權提供給第三方 AWS 帳戶 擁有](#)。
- 如需了解如何透過聯合身分提供存取權，請參閱 IAM 使用者指南中的[將存取權提供給在外部進行身分驗證的使用者 \(聯合身分\)](#)。
- 如需了解使用角色和資源型政策進行跨帳戶存取之間的差異，請參閱《IAM 使用者指南》中的 [IAM 中的跨帳戶資源存取](#)。

此 AWS 產品或服務的合規驗證

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 合規計劃範圍內](#)然後選擇您感興趣的合規計劃。如需一般資訊，請參閱 [AWS 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 取決於資料的機密性、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：

- [安全合規與治理](#) - 這些解決方案實作指南內容討論了架構考量，並提供部署安全與合規功能的步驟。
- [HIPAA 合格服務參考](#) – 列出 HIPAA 合格服務。並非所有 AWS 服務 都符合 HIPAA 資格。
- [AWS 合規資源](#) – 此工作手冊和指南的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責任模型。本指南摘要說明跨多個架構（包括國家標準技術研究所 (NIST)、支付卡產業安全標準委員會 (PCI) 和國際標準化組織 (ISO)) 保護 AWS 服務 和映射指南至安全控制的最佳實務。

- 《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符合內部實務、產業準則和法規的程度。
- [AWS Security Hub](#) – 這 AWS 服務 可讓您全面檢視其中的安全狀態 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 控制參考](#)」。
- [Amazon GuardDuty](#) – 這會監控您的環境是否有可疑和惡意活動，以 AWS 服務 偵測對您 AWS 帳戶、工作負載、容器和資料的潛在威脅。GuardDuty 可滿足特定合規架構所規定的入侵偵測需求，以協助您因應 PCI DSS 等各種不同的合規需求。
- [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，以及符合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 服務安全資訊，請參閱[AWS 服務安全文件頁面AWS](#)，以及合規計劃在 [AWS 合規工作範圍內的服務](#)。

此 AWS 產品或服務的彈性

AWS 全球基礎設施是以 AWS 區域 和可用區域為基礎建置。

AWS 區域 提供多個實體隔離和隔離的可用區域，這些可用區域與低延遲、高輸送量和高備援聯網連接。

透過可用區域，您可以設計與操作的應用程式和資料庫，在可用區域之間自動容錯移轉而不會發生中斷。可用區域的可用性、容錯能力和擴展能力，均較單一或多個資料中心的傳統基礎設施還高。

如需 AWS 區域和可用區域的詳細資訊，請參閱 [AWS 全球基礎設施](#)。

此 AWS 產品或服務會透過其支援的特定 Amazon Web Services (AWS) 服務，遵循[共同責任模型](#)。如需 AWS 服務安全資訊，請參閱[AWS 服務安全文件頁面AWS](#)，以及合規計劃在 [AWS 合規工作範圍內的服務](#)。

此 AWS 產品或服務的基礎設施安全

此 AWS 產品或服務使用 受管服務，因此受到 全球網路安全的 AWS 保護。如需 AWS 安全服務以及如何 AWS 保護基礎設施的相關資訊，請參閱[AWS 雲端安全](#)。若要使用基礎設施安全的最佳實務設計您的 AWS 環境，請參閱安全支柱 AWS Well-Architected Framework 中的[基礎設施保護](#)。

您可以使用 AWS 已發佈的 API 呼叫，透過網路存取此 AWS 產品或服務。使用者端必須支援下列專案：

- Transport Layer Security (TLS)。我們需要 TLS 1.2 並建議使用 TLS 1.3。
- 具備完美轉送私密(PFS)的密碼套件，例如 DHE (Ephemeral Diffie-Hellman)或 ECDHE (Elliptic Curve Ephemeral Diffie-Hellman)。現代系統(如 Java 7 和更新版本)大多會支援這些模式。

此外，請求必須使用存取金鑰 ID 和與 IAM 主體相關聯的私密存取金鑰來簽署。或者，您可以透過 [AWS Security Token Service](#) (AWS STS) 來產生暫時安全憑證來簽署請求。

此 AWS 產品或服務會透過其支援的特定 Amazon Web Services (AWS) 服務，遵循[共同責任模型](#)。如需 AWS 服務安全性資訊，請參閱[AWS 服務安全性文件頁面AWS](#)，以及[合規計劃在 AWS 合規工作範圍內的服務](#)。

強制執行 TLS 最低版本

若要在與 AWS 服務通訊時增加安全性，請將適用於 JavaScript 的 AWS SDK 設定為使用 TLS 1.2 或更新版本。

傳輸層安全性 (TLS) 是網頁瀏覽器和其他應用程式使用的通訊協定，以確保透過網路交換資料的隱私和完整性。

Important

截至 2024 年 6 月 10 日，我們[宣布](#) TLS 1.3 可在每個 AWS 區域的 AWS 服務 API 端點上使用。適用於 JavaScript 的 AWS SDK v2 不會交涉 TLS 版本本身。而是使用 Node.js 決定的 TLS 版本，可透過設定 `https.Agent`。AWS 建議使用目前的作用中 LTS 版本的 Node.js。

在 Node.js 中驗證和強制執行 TLS

當您適用於 JavaScript 的 AWS SDK 搭配 Node.js 使用時，會使用基礎 Node.js 安全層來設定 TLS 版本。

Node.js 12.0.0 及更新版本使用支援 TLS 1.3 的 OpenSSL 1.1.1b 最低版本。適用於 JavaScript 的 AWS SDK v2 預設為在可用時使用 TLS 1.3，但視需要預設為較低的版本。

驗證 OpenSSL 和 TLS 的版本

若要取得 Node.js 在您電腦上使用的 OpenSSL 版本，請執行以下命令。

```
node -p process.versions
```

在清單中的 OpenSSL 版本是 Node.js 使用的版本，如以下範例所示。

```
openssl: '1.1.1b'
```

若要取得 Node.js 在您電腦上使用的 TLS 版本，請啟動節點 shell，並依序執行以下命令。

```
> 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 預設使用此版本的 TLS，如果呼叫不成功，會嘗試交涉另一個版本的 TLS。

檢查支援的 TLS 版本下限和上限

開發人員可以使用下列指令碼檢查 Node.js 中支援的 TLS 版本下限和上限：

```
var tls = require("tls");  
console.log("Supported TLS versions:", tls.DEFAULT_MIN_VERSION + " to " +  
  tls.DEFAULT_MAX_VERSION);
```

最後一個命令會輸出預設的最小和最大 TLS 版本，如下列範例所示。

```
Supported TLS versions: TLSv1.2 to TLSv1.3
```

強制執行 TLS 的最低版本

Node.js 會在呼叫失敗時，交涉 TLS 的版本。從命令列執行指令碼或執行 JavaScript 程式碼中的每個要求時，您可以在此交涉期間強制執行允許的最低 TLS 版本。

若要從命令列指定最低 TLS 版本，您必須使用 Node.js 11.4.0 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

若要強制讓 TLS 1.2 成為允許的最小版本，請在執行指令碼時，指定 `--tls-min-v1.2` 引數，如下範例所示。

```
node --tls-min-v1.2 yourScript.js
```

若要為 JavaScript 程式碼中的特定請求指定允許的最低 TLS 版本，請使用 `httpOptions` 參數來指定通訊協定，如下範例所示。

```
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允許的最低 TLS 版本，請使用 `httpOptions` 參數來指定通訊協定，如下範例所示。

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

在瀏覽器指令碼中驗證並強制執行 TLS

當您在瀏覽器指令碼中使用適用於 JavaScript 的 SDK 時，瀏覽器設定會控制使用的 TLS 版本。瀏覽器使用的 TLS 版本無法透過指令碼探索或設定，而且必須由使用者設定。若要驗證並強制執行瀏覽器指令碼中使用的 TLS 版本，請參閱特定瀏覽器的指示。

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 TLS 1.2 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#####。
2. 在最佳相符項目下，按一下網際網路選項。
3. 在網際網路屬性視窗的進階索引標籤上，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，然後選取設定。
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 TLS 1.2 的選項方塊。
7. 按一下 OK (確定)。
8. 關閉您的瀏覽器並重新啟動 Google Chrome。

Mozilla Firefox

1. 開啟 Firefox。
2. 在地址列中，輸入 `about:config`，然後按 Enter 鍵。
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 TLS 1.2 的通訊協定成為預設值。
5. 按一下 OK (確定)。
6. 關閉您的瀏覽器並重新啟動 Mozilla Firefox。

Apple Safari

沒有啟用 SSL 通訊協定的選項。如果您使用的是 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 [適用於 JavaScript 的 AWS SDK](#)，提供了您可以使用的其他資源。

AWS SDKs和工具參考指南

[AWS SDKs和工具參考指南](#)也包含許多 AWS SDKs 中常見的設定、功能和其他基本概念。

JavaScript 開發套件論壇

您可以在 JavaScript SDK [JavaScript 論壇](#)中找到適用於 JavaScript 的 SDK 使用者感興趣的問題和討論。

GitHub 上的 JavaScript 開發套件和開發人員指南

GitHub 上有數個適用於 JavaScript 的 SDK 儲存庫。

- 適用於 JavaScript 的目前 SDK 可在 SDK [儲存庫](#)中使用。
- 適用於 JavaScript 的 SDK 開發人員指南（本文件）在其自己的[文件儲存庫](#)中以 Markdown 格式提供。
- 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。

Gitter 上的 JavaScript SDK

您也可以 Gitter 的 [JavaScript SDK 社群](#)中找到有關適用於 JavaScript 的 SDK 的問題和討論。

的文件歷史記錄 適用於 JavaScript 的 AWS SDK

- SDK 版本：請參閱 [JavaScript API 參考](#)
- 最新主要文件更新時間：2022 年 3 月 31 日

文件歷史記錄

下表說明 2018 年 5 月 適用於 JavaScript 的 AWS SDK 之後每個版本的重要變更。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
所有 區域的所有 AWS 服務 API 端點現在都支援 TLS 1.3	更新記錄 TLS 版本的支援 TLS 版本和方法。	2025 年 4 月 10 日
強制執行最低版本的 TLS	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 中設定登入資料，新的登入資料載入選擇	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
使用已設定登入資料程序的登入資料	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
瀏覽器指令碼的新入門	瀏覽器指令碼入門已重新撰寫，以簡化範例並存取 Amazon Polly 服務，以傳送文字並傳回您可以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 程式碼範例	已新增四個使用 Amazon SNS 的新 Node.js 程式碼範例。	2018 年 6 月 29 日

如需範例程式碼，請參閱

[Amazon SNS 範例](#)。

[Node.js 的新入門](#)

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 2018 年 6 月 適用於 JavaScript 的 AWS SDK 之前每個版本的重要變更。

變更	描述	日期
新的 AWS Elemental MediaConvert 程式碼範例	AWS Elemental MediaConvert 已新增三個用於的新 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
在 GitHub 按鈕上新增編輯功能	現在，每個主題的頁首上都會提供一個按鈕，其可帶您前往 GitHub 上相同主題的 Markdown 版本，讓您進行編輯以改善該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
GitHub 上的適用於 JavaScript 的 SDK 開發人員指南	適用於 JavaScript 的 SDK 開發人員指南在其自己的 文件儲	2018 年 2 月 16 日

變更	描述	日期
	存庫 中以 Markdown 格式提供。您可以張貼希望該指南解決的問題，或是提交提取要求以提交建議的變更。	
新的 Amazon DynamoDB 程式碼範例	已新增使用文件用戶端更新 DynamoDB 資料表的新 Node.js 程式碼範例。如需範本程式碼，請參閱 使用 DynamoDB 文件用戶端 。	2018 年 2 月 14 日
新增軟體開發套件記錄功能相關主題	已新增主題，說明如何記錄使用適用於 JavaScript 的 SDK 進行的 API 呼叫，包括使用第三方記錄器的相關資訊。請參閱 記錄 適用於 JavaScript 的 AWS SDK 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 程式碼範例	已更新包含 SDK 程式碼範例的區段，以包含五個使用 Amazon SES 的新範例。如需這些程式碼範例的詳細資訊，請參閱 Amazon Simple Email Service 範例 。	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連結，以取得 AWS 登入資料的詳細資訊。 • 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
新的 DynamoDB 程式碼範例	已更新包含 SDK 程式碼範例的區段，以重寫先前兩個範例，並新增三個全新範例來使用 DynamoDB。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	已更新包含 SDK 程式碼範例的區段，以包含五個使用 IAM 的新範例。如需這些程式碼範例的詳細資訊，請參閱AWS IAM 範例。	2016 年 12 月 23 日

變更	描述	日期
新的 CloudWatch 和 Amazon SQS 程式碼範例	已更新包含 SDK 程式碼範例的區段，以包含使用 CloudWatch 和 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 範例 和 Amazon SQS 範例 。	2016 年 12 月 20 日
新的 Amazon EC2 程式碼範例	已更新包含 SDK 程式碼範例的區段，以包含五個使用 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	先前在先決條件主題中找到的適用於 JavaScript 的開發套件所支援的瀏覽器清單，已指定自己的主題，使其在目錄中更加可見。	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 使用案例出現特殊考量事項時，該指南會適當標識這些事項。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