

---

# Amazon API Gateway

## 開發人員指南



## Amazon API Gateway: 開發人員指南

Copyright © 2019 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon's trademarks and trade dress may not be used in connection with any product or service that is not Amazon's, in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits Amazon. All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Amazon.

## Table of Contents

什麼是 Amazon API Gateway ? .....	1
API Gateway 架構 .....	1
API Gateway 的功能 .....	2
API Gateway 使用案例 .....	2
使用 API Gateway 建立 REST API .....	3
使用 API Gateway 來建立 WebSocket API .....	3
誰使用 API Gateway ? .....	3
存取 API Gateway .....	4
AWS 無伺服器基礎設施的部分 .....	4
如何開始使用 Amazon API Gateway .....	5
API Gateway 概念 .....	5
API Gateway 定價 .....	7
API Gateway 合規 .....	8
PCI DSS .....	8
HIPAA .....	8
API Gateway 入門 .....	9
先決條件：準備在 API Gateway 中建置 API .....	9
註冊 AWS 帳戶 .....	9
建立 IAM 管理員使用者 .....	9
建立具有 Lambda 整合的 REST API .....	9
步驟 1：在 Lambda 主控台建立 Lambda 函數 .....	10
步驟 2：在 API Gateway 主控台建立 REST API .....	10
步驟 3：在 API Gateway 主控台部署您的 REST API .....	12
步驟 4：在 Lambda 主控台建立第二個 Lambda 函數 .....	13
步驟 5：在 API Gateway 主控台將資源、方法和參數新增到 REST API .....	13
後續步驟 .....	16
建立具有模擬整合的 REST API .....	16
步驟 1：建立 API .....	16
步驟 2：建立模擬整合 .....	18
步驟 3：定義成功回應 .....	19
步驟 4：新增 HTTP 500 狀態碼和錯誤訊息 .....	19
步驟 5：測試模擬整合 .....	20
後續步驟 .....	22
Amazon API Gateway 影片 .....	23
來自 Twitch 「在 AWS 上建置」影集的 Amazon API Gateway 影片 .....	23
其他 Amazon API Gateway 影片 .....	23
教學課程 .....	24
建置具有 Lambda 整合的 API .....	24
教學：具有 Lambda 代理整合的 Hello World API .....	24
教學：建置具有跨帳戶 Lambda 代理整合的 API .....	29
教學：建置具有 Lambda 非代理整合的 API .....	31
教學課程：匯入範例來建立 REST API .....	40
建置具有 HTTP 整合的 API .....	47
教學：建置具有 HTTP 代理整合的 API .....	47
教學：建置具有 HTTP 非代理整合的 API .....	52
教學：建置具有私有整合的 API .....	80
教學：建置具有 AWS 整合的 API .....	81
事前準備 .....	82
步驟 1：建立資源 .....	82
步驟 2：建立 GET 方法 .....	82
步驟 3：建立 AWS 服務代理執行角色 .....	83
步驟 4：指定方法設定並測試方法 .....	84
步驟 5：部署 API .....	85
步驟 6：測試 API .....	85

步驟 7：清除	85
教學課程：Calc API (含三個整合)	86
建立 AWS 帳戶	86
建立可擔任的 IAM 角色	87
建立 Calc Lambda 函數	88
測試 Calc Lambda 函數	88
建立 Calc API	90
整合 1：建立 GET 方法與查詢參數搭配來呼叫 Lambda 函數	91
整合 2：建立 POST 方法與 JSON 承載搭配來呼叫 Lambda 函數	93
整合 3：建立 GET 方法與路徑參數搭配來呼叫 Lambda 函數	95
Lambda 函數之範例 API 的 OpenAPI 定義	100
教學：在 API Gateway 中建立 REST API 做為 Amazon S3 代理	104
設定 API 呼叫 Amazon S3 動作的 IAM 許可	105
建立 API 資源以代表 Amazon S3 資源	106
公開 API 方法來列出發起人的 Amazon S3 儲存貯體	107
公開 API 方法來存取 Amazon S3 儲存貯體	112
公開 API 方法來存取儲存貯體中的 Amazon S3 物件	115
使用 REST API 用戶端呼叫 API	118
做為 Amazon S3 代理之範例 API 的 OpenAPI 定義	119
教學：建立 REST API 做為 Amazon Kinesis 代理	129
建立 API 的 IAM 角色和政策來存取 Kinesis	130
開始建立 API 做為 Kinesis 代理	131
列出 Kinesis 中的串流	131
在 Kinesis 中建立、說明和刪除串流	136
取得 Kinesis 中串流的記錄，並將記錄新增至其中	141
做為 Kinesis 代理之範例 API 的 OpenAPI 定義	151
建立、部署和呼叫 REST API	167
建立 REST API	167
選擇 API 端點類型	168
初始化 REST API 設定	168
設定 REST API 方法	188
設定 REST API 整合	199
設定閘道回應	237
設定資料對應	242
支援二進位承載	283
啟用承載壓縮功能	304
啟用請求驗證	307
匯入 REST API	319
控制和管理 REST API 的存取	324
使用 API Gateway 資源政策	324
使用 IAM 許可	333
使用標籤來控制對 REST API 的存取	348
使用 Lambda 授權方	348
針對 REST API 使用 Cognito 使用者集區做為授權方	363
為 REST API 資源啟用 CORS	371
使用用戶端 SSL 憑證	377
使用 AWS WAF 來保護您的 API 避免受到常見的網路攻擊	396
使用 API 金鑰的用量計劃	397
記錄 REST API	411
API Gateway 中的 API 文件表示	411
使用 API Gateway 主控台記錄 API	419
使用 API Gateway 主控台發佈 API 文件	427
使用 API Gateway REST API 記錄 API	427
使用 API Gateway REST API 發佈 API 文件	442
匯入 API 文件	448
控制 API 文件的存取	452
更新和維護 REST API	453

變更公有或私有 API 端點類型 .....	454
使用主控台維護 API .....	455
部署 REST API .....	457
部署 REST API .....	458
設定階段 .....	460
在 API Gateway 中針對 REST API 產生開發套件 .....	481
呼叫 REST API .....	497
在 API Gateway 主控台中取得 API 的呼叫 URL .....	497
使用主控台測試 REST API 方法 .....	498
使用 Postman 來呼叫 REST API .....	499
透過產生的開發套件呼叫 REST API .....	499
透過 AWS Amplify JavaScript 程式庫呼叫 REST API .....	517
如何呼叫私有 API .....	517
追蹤、記錄並監控 API .....	518
使用 AWS X-Ray 追蹤 API 執行 .....	518
透過 AWS CloudTrail 記錄向 Amazon API Gateway API 發出的呼叫 .....	527
使用 Amazon CloudWatch 監控 API 執行 .....	528
適用於 REST API 的 OpenAPI 延伸 .....	533
x-amazon-apigateway-any-method .....	533
x-amazon-apigateway-api-key-source .....	534
x-amazon-apigateway-authorizer .....	535
x-amazon-apigateway-authtype .....	537
x-amazon-apigateway-binary-media-type .....	538
x-amazon-apigateway-documentation .....	538
x-amazon-apigateway-gateway-responses .....	539
x-amazon-apigateway-gateway-responses.gatewayResponse .....	540
x-amazon-apigateway-gateway-responses.responseParameters .....	540
x-amazon-apigateway-gateway-responses.responseTemplates .....	541
x-amazon-apigateway-integration .....	542
x-amazon-apigateway-integration.requestTemplates .....	544
x-amazon-apigateway-integration.requestParameters .....	545
x-amazon-apigateway-integration.responses .....	545
x-amazon-apigateway-integration.response .....	546
x-amazon-apigateway-integration.responseTemplates .....	547
x-amazon-apigateway-integration.responseParameters .....	548
x-amazon-apigateway-request-validator .....	548
x-amazon-apigateway-requestValidators .....	549
x-amazon-apigateway-requestValidators.requestValidator .....	550
建立、部署和呼叫 WebSocket API .....	551
關於 WebSocket API .....	551
管理連線使用者和用戶端應用程式 .....	552
呼叫後端整合 .....	552
將資料從後端服務傳送到連線的用戶端 .....	555
建立 WebSocket API .....	555
使用 AWS CLI 命令建立 WebSocket API .....	555
使用 API Gateway 主控台建立 WebSocket API .....	555
設定 WebSocket API 路由 .....	556
設定路由 .....	556
指定路由請求設定 .....	557
設定 WebSocket API 整合 .....	557
設定 WebSocket API 整合請求 .....	558
設定 WebSocket API 整合回應 .....	560
設定 WebSocket API 路由回應 .....	561
使用 API Gateway 主控台設定路由回應 .....	562
使用 AWS CLI 設定路由回應 .....	562
部署 WebSocket API .....	562
使用 AWS CLI 建立 WebSocket API 部署 .....	563

使用 API Gateway 主控台建立 WebSocket API 部署 .....	564
呼叫 WebSocket API .....	564
使用 wscat 以連接到 WebSocket API 和將訊息傳送到其中 .....	564
在後端服務使用 @connections 命令 .....	565
控制 WebSocket API 的存取 .....	566
使用 IAM 授權 .....	566
建立 Lambda REQUEST 授權方函數 .....	567
使用 CloudWatch 監控 WebSocket API 執行 .....	568
WebSocket 選擇表達式 .....	570
.....	570
.....	571
.....	571
.....	572
.....	572
.....	572
.....	572
.....	572
.....	572
WebSocket 映射範本參考 .....	575
發佈您的 API .....	579
使用無伺服器開發人員入口網站將您的 API 編目 .....	579
建立開發人員入口網站 .....	580
開發人員入口網站設定 .....	581
為開發人員入口網站建立管理員使用者 .....	582
將 API Gateway 受管 API 發佈到您的開發人員入口網站 .....	583
更新或刪除 API Gateway 受管 API .....	584
移除非 API Gateway 受管 API .....	584
將非 API Gateway 受管 API 發佈到您的開發人員入口網站 .....	584
您的客戶如何使用您的開發人員入口網站 .....	585
開發人員入口網站的最佳實務 .....	586
將 API 當做 SaaS 銷售 .....	587
初始化 AWS Marketplace 與 API Gateway 的整合 .....	587
處理客戶的用量計劃訂閱 .....	588
設定 API 自訂網域名稱 .....	590
在 AWS Certificate Manager 中備妥憑證 .....	591
如何建立邊緣最佳化自訂網域名稱 .....	593
設定區域性自訂網域名稱 .....	599
遷移自訂網域名稱 .....	601
匯出 REST API .....	604
匯出 REST API 的請求 .....	604
下載 JSON 格式的 REST API OpenAPI 定義 .....	605
下載 YAML 格式的 REST API OpenAPI 定義 .....	605
下載 JSON 格式且具有 Postman 延伸的 REST API OpenAPI 定義 .....	606
下載 YAML 格式且具有 API Gateway 整合的 REST API OpenAPI 定義 .....	606
使用 API Gateway 主控台匯出 REST API .....	606
設定 Canary Release 部署 .....	607
API Gateway 中的 Canary Release 部署 .....	607
建立 Canary Release 部署 .....	608
更新 Canary Release .....	612
提升 Canary Release .....	614
停用 Canary Release .....	616
監控您的 API 組態 .....	618
支援的資源類型 .....	618
設定 AWS Config .....	619
設定 AWS Config 以記錄 API Gateway 資源 .....	619
在 AWS Config 主控台檢視 API Gateway 組態詳細資訊 .....	619
使用 AWS Config 規則評估 API Gateway 資源 .....	620
標記 API Gateway 資源 .....	621

可以標記的 API Gateway 資源 .....	621
標籤繼承 .....	622
標籤限制和使用慣例 .....	622
標記型存取控制 .....	623
範例 1：根據資源標籤限制動作 .....	623
範例 2：根據請求中的標籤限制動作 .....	624
範例 3：根據資源標籤拒絕動作 .....	624
範例 4：根據資源標籤允許動作 .....	625
API Gateway V1 和 V2 API 參考 .....	626
限制和重要說明 .....	627
API Gateway 限制 .....	627
API Gateway 帳戶等級限制 .....	627
設定和執行 WebSocket API 的 API Gateway 限制 .....	627
設定和執行 REST API 的 API Gateway 限制 .....	628
建立、部署和管理 API 的 API Gateway 限制 .....	629
重要說明 .....	630
適用於 REST 和 WebSocket API 的重要說明 .....	630
適用於 WebSocket API 的重要說明 .....	631
適用於 REST API 的重要說明 .....	631
文件歷史記錄 .....	634
舊版更新 .....	636
AWS Glossary .....	641

# 什麼是 Amazon API Gateway？

Amazon API Gateway 是一種 AWS 服務，可讓您建立、發佈、維護、監控和保護任何規模的 REST 和 WebSocket API。API 開發人員可以建立 API，以存取 AWS 或其他 Web 服務，以及 AWS 雲端中所存放的資料。身為 API Gateway API 開發人員，您可以建立要在自己用戶端應用程式 (app) 中使用的 API。或者，您可以讓 API 供第三方應用程式開發人員使用。如需詳細資訊，請參閱[the section called “誰使用 API Gateway？” \(p. 3\)](#)。

API Gateway 會建立 REST API：

- 以 HTTP 為基礎。
- 遵循 REST 通訊協定，用來啟用無狀態用戶端與伺服器通訊。
- 實作標準 HTTP 方法，例如 GET、POST、PUT、PATCH 和 DELETE。

如需 API Gateway REST API 的詳細資訊，請參閱[the section called “使用 API Gateway 建立 REST API” \(p. 3\)](#) 和[the section called “建立 REST API” \(p. 167\)](#)。

API Gateway 會建立 WebSocket API：

- 遵循 WebSocket 通訊協定，在用戶端與伺服器之間啟用狀態、全雙工通訊。
- 根據訊息內容路由傳入的訊息。

如需 API Gateway WebSocket API 的詳細資訊，請參閱[the section called “使用 API Gateway 來建立 WebSocket API” \(p. 3\)](#) 和[the section called “關於 WebSocket API” \(p. 551\)](#)。

## 主題

- [API Gateway 架構 \(p. 1\)](#)
- [API Gateway 的功能 \(p. 2\)](#)
- [API Gateway 使用案例 \(p. 2\)](#)
- [存取 API Gateway \(p. 4\)](#)
- [AWS 無伺服器基礎設施的部分 \(p. 4\)](#)
- [如何開始使用 Amazon API Gateway \(p. 5\)](#)
- [Amazon API Gateway 概念 \(p. 5\)](#)
- [API Gateway 定價 \(p. 7\)](#)
- [API Gateway 合規 \(p. 8\)](#)

## API Gateway 架構

下圖顯示 API Gateway 架構。



下圖說明您在 Amazon API Gateway 中建置的 API 如何讓您或開發人員客戶在建置 AWS 無伺服器應用程式時，享受整合又一致的開發人員體驗。API Gateway 負責處理有關接受和處理多達數十萬個並行 API 呼叫的所有工作，包括流量管理、授權和存取控制、監控和 API 版本管理。API Gateway 做為「門戶」，讓應用程式存取資料、商業邏輯，或後端服務的功能，例如在 Amazon Elastic Compute Cloud (Amazon EC2) 上執行的工作負載、在 AWS Lambda、任何 Web 應用程式或即時通訊應用程式上執行的程式碼。

## API Gateway 的功能

Amazon API Gateway 提供如下功能：

- 支援狀態 ([WebSocket \(p. 551\)](#)) 和無狀態 ([REST \(p. 167\)](#)) API。
- 強大又有彈性的身分驗證 ([p. 324](#)) 機制，例如 AWS Identity and Access Management 政策、Lambda 授權方函數和 Amazon Cognito 使用者集區。
- [開發人員入口網站 \(p. 579\)](#)，用於發佈您的 API。
- [Canary 發行部署 \(p. 607\)](#)，可讓您安全地進行變更。
- API 使用情況和 API 變更的 [CloudTrail \(p. 527\)](#) 記錄和監控。
- CloudWatch 存取記錄和執行記錄，包括設定警報的能力。如需更多詳細資訊，請參閱 [the section called “使用 Amazon CloudWatch 監控 API 執行” \(p. 528\)](#) 及 [the section called “使用 CloudWatch 監控 WebSocket API 執行” \(p. 568\)](#)。
- 可以使用 AWS CloudFormation 範本來啟用 API 建立。如需詳細資訊，請參閱 [Amazon API Gateway 資源類型參考](#) 和 [Amazon API Gateway V2 資源類型參考](#)。
- 支援自訂網域名稱 ([p. 590](#))。
- 與 [AWS WAF \(p. 396\)](#) 的整合，用於保護 API 避免受到常見的網路攻擊。
- 與 [AWS X-Ray \(p. 518\)](#) 的整合，用於了解和分類效能延遲。

如需 API Gateway 功能發佈的完整清單，請參閱[文件歷史記錄 \(p. 634\)](#)。

## API Gateway 使用案例

### 主題

- [使用 API Gateway 建立 REST API \(p. 3\)](#)

- 使用 API Gateway 來建立 WebSocket API (p. 3)
- 誰使用 API Gateway ? (p. 3)

## 使用 API Gateway 建立 REST API

API Gateway REST API 是由資源和方法組成。資源是應用程式可透過資源路徑存取的邏輯實體。方法會對應至 API 使用者所提交的 REST API 請求，以及使用者所傳回的回應。

例如，/incomes 可以是代表應用程式使用者收入之資源的路徑。資源可以有適當的 HTTP 動詞 (例如 GET、POST、PUT、PATCH 和 DELETE) 所定義的一或多個操作。可識別 API 方法的資源路徑和操作組合。例如，POST /incomes 方法可新增發起人所獲得的收入，而 GET /expenses 方法可查詢發起人所產生的報告費用。

應用程式不必知道在後端存放和擷取所請求資料的位置。在 API Gateway REST API 中，前端的封裝方式是方法請求和方法回應。API 會使用整合請求和整合回應以與後端互動。

例如，如果使用 DynamoDB 做為後端，則 API 開發人員會設定整合請求，以將傳入的方法請求轉送至選擇的後端。設定包含適當 DynamoDB 動作的規格、所需的 IAM 角色和政策，以及所需的輸入資料轉換。後端會將結果傳回給 API Gateway 以做為整合回應。若要將特定 HTTP 狀態碼之適當方法回應的整合回應路由至用戶端，您可以設定整合回應以將所需的回應參數從整合對應至方法。您接著會將後端的輸出資料格式翻譯為前端的輸出資料格式 (必要時)。API Gateway 可讓您定義 [承載](#) 的結構描述或模型，以促進設定內文對應範本。

API Gateway 提供 REST API 管理功能，例如：

- 支援使用對 OpenAPI 的 API Gateway 延伸來產生軟體開發套件和建立 API 文件
- HTTP 請求的調節

## 使用 API Gateway 來建立 WebSocket API

在 WebSocket API 中，用戶端和伺服器都可以隨時將訊息傳送給彼此。後端伺服器可以輕鬆地將資料發送至連線的使用者和裝置，進而免除了實作複雜的輪詢機制。

例如，您可以使用 API Gateway WebSocket API 和 AWS Lambda 建置無伺服器應用程式，來在聊天室中從個別使用者或使用者群組接收訊息並將訊息傳送到其中。或者，您可以根據訊息內容叫用後端服務 (例如 AWS Lambda、Amazon Kinesis 或 HTTP 端點)。

您可以使用 API Gateway WebSocket API 來建構安全、即時通訊的應用程式，而無需佈建或管理任何伺服器來管理連線或大規模資料交換。針對性使用案例包含即時的應用程式，例如：

- 聊天應用程式
- 股票行情之類的即時儀表板
- 即時提醒和通知

API Gateway 提供 WebSocket API 管理功能，例如：

- 對連線和訊息進行監控和調節
- 使用 AWS X-Ray 來在訊息通過 API 進入後端服務時追蹤訊息
- 輕鬆整合 HTTP/HTTPS 端點

## 誰使用 API Gateway ?

使用 API Gateway 的開發人員有兩種：API 開發人員和應用程式開發人員。

API 開發人員會建立和部署 API，以啟用 API Gateway 中所需的功能。API 開發人員必須是擁有 API 之 AWS 帳戶中的 IAM 使用者。

應用程式開發人員建置的正常運作應用程式，可透過叫用由 API Gateway 中的 API 開發人員建立之 WebSocket 或 REST API 來呼叫 AWS 服務。

該應用程式開發人員是 API 開發人員的客戶。應用程式開發人員不需要擁有 AWS 帳戶，但前提是 API 不需要 IAM 許可或支援藉由 [Amazon Cognito 使用者集區聯合身分](#) 支援的第三方聯合身分提供者授權使用者。這類身分提供者包含 Amazon、Amazon Cognito 使用者集區、Facebook 和 Google。

## 建立和管理 API Gateway API

API 開發人員會使用名為 apigateway 的 API Gateway 服務元件 (用於 API 管理)，以建立、設定和部署 API。每個 API 都包含一組資源和方法。資源是應用程式可透過資源路徑存取的邏輯實體。

身為 API 開發人員，您可以使用 API Gateway 主控台 (如[Amazon API Gateway 入門 \(p. 9\)](#)中所述) 或透過呼叫 [API Gateway V1 和 V2 API 參考 \(p. 626\)](#) 來建立和管理 API。有數種方式可以呼叫此 API。它們包含使用 AWS 命令列界面 (CLI) 或使用 AWS 開發套件。此外，您還可以使用 [AWS CloudFormation 範本](#) 或 (在 REST API 的情形下) [OpenAPI 的 API Gateway 延伸 \(p. 533\)](#) 來建立 API。如需可用 API Gateway 區域的清單，以及相關聯的控制服務端點，請參閱[區域和端點](#)。

## 呼叫 API Gateway API

應用程式開發人員會將 API Gateway 服務元件用於 API 執行 (名稱為 execute-api)，以叫用在 API Gateway 中建立或部署的 API。基礎程式設計實體是透過所建立的 API 所公開。有數種方式可以呼叫這類 API。您可以使用 API Gateway 主控台來測試呼叫 API。針對 REST API，您可以使用 REST API 用戶端 (例如 [cURL](#) 或 [POSTMAN](#)) 或由 API Gateway 產生以讓 API 呼叫 API 的開發套件。

## 存取 API Gateway

您可透過以下方式存取 Amazon API Gateway：

- AWS Management Console – 本指南中的程序說明如何使用 AWS Management Console 執行任務。
- AWS 開發套件 – 如果您使用的是 AWS 有提供開發套件的程式設計語言，即可使用開發套件來存取 API Gateway。開發套件可簡化身份驗證、與您的開發環境輕鬆整合，並可存取 API Gateway 命令。如需詳細資訊，請參閱 [Amazon Web Services 適用工具](#)。
- API Gateway V1 與 V2 API – 如果您是使用未提供開發套件的程式設計語言，請參閱 [Amazon API Gateway 第 1 版 API 參考](#) 和 [Amazon API Gateway 第 2 版 API 參考](#)。
- AWS Command Line Interface – 如需詳細資訊，請參閱 [AWS Command Line Interface User Guide](#) 中的 [使用 AWS Command Line Interface 開始設定](#)。
- AWS Tools for Windows PowerShell – 如需詳細資訊，請參閱 [AWS Tools for Windows PowerShell User Guide](#) 中的 [設定 AWS Tools for Windows PowerShell](#)。

## AWS 無伺服器基礎設施的部分

與 [AWS Lambda](#) 一起使用，API Gateway 會形成 AWS 無伺服器基礎設施的應用程式面向部分。若要讓應用程式呼叫可公開使用的 AWS 服務，您可以使用 Lambda 與所需的服務互動，並透過 API Gateway 中的 API 方法公開 Lambda 函數。AWS Lambda 會在高度可用的運算基礎設施上執行程式碼。它會執行所需的運算資源執行和管理。若要啟用無伺服器應用程式，API Gateway 支援與 AWS Lambda 和 HTTP 端點的 [簡化代理整合 \(p. 202\)](#)。

# 如何開始使用 Amazon API Gateway

如需 Amazon API Gateway 的快速簡介，請參閱下列主題：

- [Amazon API Gateway 入門](#)，提供逐步解說和影片讓您了解如何使用 Amazon API Gateway 建立 REST API。
- [在 Amazon API Gateway 中發表 WebSocket API](#)，說明如何建立 WebSocket API。

## Amazon API Gateway 概念

### API Gateway

API Gateway 是支援下列項目的 AWS 服務：

- 建立、部署和管理 REST 應用程式設計界面 (API) 來接觸後端 HTTP 端點、AWS Lambda 函數或其他 AWS 服務。
- 建立、部署和管理 WebSocket API 來接觸 AWS Lambda 函數或其他 AWS 服務。
- 透過前端 HTTP 及 WebSocket 端點呼叫已接觸的 API 方法。

### API Gateway REST API

與後端 HTTP 端點、Lambda 函數或其他 AWS 服務整合的 HTTP 資源和方法集合。此集合可以部署在一個或多個階段。一般而言，會根據應用程式邏輯將 API 資源組織為資源樹狀結構。每個 API 資源都可接觸一個或多個 API 方法，而這些方法必須擁有 API Gateway 所支援的唯一 HTTP 動詞。

### API Gateway WebSocket API

與後端 HTTP 端點、Lambda 函數或其他 AWS 服務整合的 WebSocket 路由和路由金鑰集合。此集合可以部署在一或多個階段。API 方法可透過前端 WebSocket 連線呼叫，該連線須能夠與已註冊的自訂網域名稱建立關聯。

### API 部署

您的 API Gateway API 時間點快照。若要可供用戶端使用，部署必須與一或多個 API 階段相關聯。

### API 開發人員

擁有 API Gateway 部署的 AWS 帳戶 (例如，也支援程式設計存取的服務提供者)。

### API 端點

API Gateway 內已部署至特定區域之 API 的主機名稱。主機名稱的格式為 `{api-id}.execute-api.{region}.amazonaws.com`。支援下列類型的 API 端點：

- [邊緣最佳化的 API 端點 \(p. 6\)](#)
- [私有 API 端點 \(p. 7\)](#)
- [區域 API 端點 \(p. 7\)](#)

### API 金鑰

API Gateway 用來辨識使用 REST 或 WebSocket API 之應用程式開發人員的英數字元字串。API Gateway 可代您產生 API 金鑰，或從 CSV 檔案加以匯入。您可以同時使用 API 金鑰與 Lambda 授權方 (p. 348) 或用量計劃 (p. 397)，以控制對 API 的存取。

請參閱 [API 端點 \(p. 5\)](#)。

### API 擁有者

請參閱 [API 開發人員 \(p. 5\)](#)。

## API 階段

REST 或 WebSocket API 生命週期狀態的邏輯參考 (如 'dev'、'prod'、'beta'、'v2')。您可以 API ID 和階段名稱來識別 API 階段。

### 應用程式開發人員

應用程式建立者，不一定有 AWS 帳戶而且會與 API 開發人員所部署的 API 互動。應用程式開發人員是您的客戶。通常會透過 [API 金鑰 \(p. 5\)](#) 識別應用程式開發人員。

### 回呼 URL

當新的用戶端透過 WebSocket 進行連線，您可在 API Gateway 中呼叫整合來存放用戶端的回呼 URL。之後您即可使用該回呼 URL 從後端系統傳送訊息到連接的用戶端。

### 開發人員入口網站

此應用程式允許您的客戶註冊、探索及訂閱 API 產品 (API Gateway 用量方案)、管理其 API 金鑰，以及檢視 API 的用量指標。

### 邊緣最佳化的 API 端點

API Gateway API 的預設主機名稱，通常使用 CloudFront 分佈來強化跨 AWS 區域的用戶端存取時會部署至指定區域。API 請求會路由到最近的 CloudFront 出現點 (POP)，通常可改善分散各地之用戶端的連線時間。

請參閱 [API 端點 \(p. 5\)](#)。

### 整合請求

API Gateway 內 WebSocket API 路由或 REST API 方法的內部界面，您要將其中路由請求或參數的本文及方法請求的本文，映射至後端要求的格式。

### 整合回應

API Gateway 內 WebSocket API 路由或 REST API 方法的內部界面，您要將其中從後端接收的狀態碼、標頭和承載，映射至將傳回用戶端應用程式的回應格式。

### 對應範本

以 [Velocity 範本語言 \(VTL\)](#) 表示的程式碼，可將請求本文從前端資料格式轉換為後端資料格式，或是將回應本文從後端資料格式轉換為前端資料格式。映射範本可指定於整合請求或整合回應中。它們可以參考在執行時間提供為內容和階段變數的資料。映射可如同 [身分轉換](#) 一樣簡單，可透過整合的現狀依照請求將標頭或本文從用戶端傳遞至後端。回應也是如此，其中承載會從後端傳遞至用戶端。

### 方法請求

API Gateway 中 REST API 方法的公有界面，其定義的參數與本文必須在應用程式開發人員的請求中傳送，藉以透過 API 存取後端。

### 方法回應

REST API 的公有界面，其定義的狀態碼、標頭和本文模型應為應用程式開發人員預期自 API 接收的回應。

### 模擬整合

模擬整合中，API 回應由 API Gateway 直接產生，無須整合後端。您是 API 開發人員，可決定 API Gateway 回應如何模擬整合請求。因此，您設定方法的整合請求和整合回應，以將回應與特定狀態碼建立關聯。

### 模型

資料結構描述，其指定請求或回應承載的資料結構。必須使用模型才能產生 API 的強類型開發套件。模型也用來驗證承載。模型方便用於產生範例對應範本以啟動生產對應範本的建立。模型雖然實用，但不是建立對應範本的必要項目。

## 私有 API

請參閱[私有 API 端點 \(p. 7\)](#)。

### 私有 API 端點

可由界面 VPC 端點接觸的 API 端點，可允許用戶端在 VPC 內安全存取私有 API 資源。私有 API 與公有網際網路彼此隔離，而且只能經由具備存取權限的 API Gateway VPC 端點存取。

### 私有整合

一種 API Gateway 整合類型，可讓用戶端透過私有 REST API 端點存取客戶 VPC 內的資源，無須將資源公開在公有網際網路。

### 代理整合

REST 或 WebSocket API 的簡化 API Gateway 整合組態。您可將代理整合設定為 HTTP 代理整合或 Lambda 代理整合。針對 HTTP 代理整合，API Gateway 會在前端與 HTTP 後端之間傳遞整個請求和回應。針對 Lambda 代理整合，API Gateway 會將整個請求做為後端 Lambda 函數的輸入來傳送。API Gateway 接著會將 Lambda 函數輸出轉換為前端 HTTP 回應。在 REST API 中，代理整合通常會與代理資源搭配使用，而代理資源會結合全部截獲 ANY 方法，以 Greedy 路徑變數（如 {proxy+}）呈現。

### 區域 API 端點

API 的主機名稱，其部署至特定區域且用於服務相同 AWS 區域的用戶端（如 EC2 執行個體）。API 請求的目標是直接設為區域特定 API Gateway，而不需要通過任何 CloudFront 分佈。針對區域中請求，區域端點會略過與 CloudFront 分佈的不必要往返。此外，您還可以在區域端點上套用延遲型路由，藉此使用相同的區域 API 端點組態將 API 部署至多個區域、針對每個已部署的 API 設定相同的自訂網域名稱，並在 Route 53 中設定延遲型 DNS 記錄，將用戶端請求路由至擁有最低延遲的區域。

請參閱[API 端點 \(p. 5\)](#)。

### 路由

API Gateway 內的 WebSocket 路由係用來將傳入訊息根據其內容導向特定整合（如 AWS Lambda 函數）。定義 WebSocket API 時，須指定路由金鑰和整合後端。路由金鑰是訊息本文中的一個屬性。若傳入訊息的路由金鑰相符，將呼叫整合後端。預設路由也可設定用於不相符的路由金鑰或用來指定代理模型，將訊息現狀傳遞至執行路由並處理請求的後端元件。

### 路由請求

API Gateway 中 WebSocket API 方法的公有界面，其定義的本文必須在應用程式開發人員的請求中傳送，藉以透過 API 存取後端。

### 路由回應

WebSocket API 的公有界面，其定義的狀態碼、標頭和本文模型應為應用程式開發人員預期自 API Gateway 接收的內容。

### 用量計畫

[用量計畫 \(p. 397\)](#)提供所選的 API 用戶端，可存取一個或多個已部署 REST 或 WebSocket API。您可以使用用量計畫設定調節和配額限制，這些是針對個別用戶端 API 金鑰所強制執行。

### WebSocket 連線

API Gateway 會持續維持 API Gateway 本身和用戶端之間的連線。API Gateway 與後端整合（如 Lambda 函數）之間沒有持續連線；後端服務會根據用戶端傳送的訊息內容視需要呼叫。

# API Gateway 定價

如需一般 API Gateway 區域特定定價資訊，請參閱[Amazon API Gateway 定價](#)。

以下列出一般定價方案的例外狀況：

- Amazon API Gateway 中的 API 快取不符合 AWS 免費方案資格。
- 針對授權和身份驗證失敗，不會收取呼叫[授權類型](#)為 AWS\_IAM、CUSTOM 和 COGNITO\_USER\_POOLS 的方法費用。
- API 金鑰遺失或無效時，不會收取呼叫需要 API 金鑰之方法的費用。
- 請求率或爆量率超過預先設定的限制時，不會收取 API Gateway 調節請求費用。
- 速率限制或配額超過預先設定的限制時，不會收取用量計畫調節請求費用。

## API Gateway 合規

如需有關 API Gateway 對各種安全合規法規和稽核標準的合規性，請參閱以下頁面：

- [AWS 雲端合規](#)
- [合規計劃的 AWS 服務範圍](#)

此外，請參閱以下有關 API Gateway 如何符合 PCI DSS 和 HIPAA 標準的詳細資訊。

### PCI DSS

API Gateway supports the processing, storage, and transmission of credit card data by a merchant or service provider, and has been validated as being compliant with Payment Card Industry (PCI) Data Security Standard (DSS). For more information about PCI DSS, including how to request a copy of the AWS PCI Compliance Package, see [PCI DSS Level 1](#).

### HIPAA

This is a HIPAA Eligible Service. For more information about AWS, U.S. Health Insurance Portability and Accountability Act of 1996 (HIPAA), and using AWS services to process, store, and transmit protected health information (PHI), see [HIPAA Overview](#).

# Amazon API Gateway 入門

為了協助您開始使用 Amazon API Gateway，請閱讀以下實作逐步解說和其他資源：

- [先決條件：準備在 API Gateway 中建立 API](#) 探討如何建立 AWS 帳戶和設定 IAM 管理員使用者。
- [建立具有 Lambda 整合的 REST API](#) 說明如何在後端建立具有 Lambda 函數的 REST API。同時包含代理和非代理整合技術。
- [建立具有模擬整合的 REST API](#) 說明如何在沒有實際後端的情況下建立 REST API。
- [教學課程：匯入範例來建立 REST API](#) 說明如何匯入 OpenAPI 2.0 檔案來建立範例 API。
- [建立 WebSocket API](#) 說明如何在 API Gateway 中建立 WebSocket API。
- [Amazon API Gateway 影片](#) 提供的影片連結討論如何開始使用 API Gateway。

## 先決條件：準備在 API Gateway 中建置 API

### 主題

- [註冊 AWS 帳戶 \(p. 9\)](#)
- [建立 IAM 管理員使用者 \(p. 9\)](#)

第一次使用 Amazon API Gateway 之前，您必須已有 AWS 帳戶。

在[Amazon API Gateway 入門 \(p. 9\)](#)和[教學課程 \(p. 24\)](#)的許多教學中，已為您建立必要的 IAM 政策。不過，當您開始建立自己的 API，您需要研究以下和the section called “[使用 IAM 許可](#)” (p. 333)中的資訊。

## 註冊 AWS 帳戶

若要使用 Amazon API Gateway、AWS Lambda 及其他 AWS 服務，您需要有 AWS 帳戶。如果您還沒有帳戶，請瀏覽 [aws.amazon.com](https://aws.amazon.com) 並選擇 Create an AWS Account。如需詳細說明，請參閱[建立和啟用 AWS 帳戶](#)。

## 建立 IAM 管理員使用者

根據最佳實務，您也應該建立具有管理員許可的 AWS Identity and Access Management (IAM) 使用者。對於不需要根登入資料的所有工作，請使用此使用者。為主控台存取建立密碼，以及使用命令列工具的存取金鑰。如需說明，請參閱IAM 使用者指南中的[建立您的第一個 IAM 管理員使用者和群組](#)。

## 在 Amazon API Gateway 中建立具有 Lambda 整合的 REST API

您可以使用此逐步解說，在 Amazon API Gateway 中建立和部署具有 Lambda 代理整合和 Lambda 非代理整合的 REST API。

在 Lambda 整合中，來自用戶端的 HTTP 方法請求會映射到後端 [Lambda 函數叫用](#)。

在 Lambda 代理整合中，整個用戶端請求會按現狀傳送到後端 Lambda 函數，但不保留請求參數的順序。API Gateway 會將整個用戶端請求映射到後端 Lambda 函數的輸入 event 參數。Lambda 函數的輸出（包括狀態碼、標頭和內文）會按現狀傳回給用戶端。對於許多使用案例，這是慣用的整合類型。如需詳細資訊，請參閱the section called “[設定 Lambda 代理整合](#)” (p. 206)。

在 Lambda 非代理整合 中 (也稱為「自訂整合」) , 您需要設定方法讓用戶端請求的參數、標頭和內文轉換成您的後端 Lambda 函數所需的格式。您還要設定方法讓 Lambda 函數輸出轉換回用戶端所需的格式。如需詳細資訊 , 請參閱the section called “[設定 Lambda 自訂整合](#)” (p. 217)。

根據您的使用案例 , 您可以在 API Gateway API 中選擇使用 Lambda 代理整合、Lambda 非代理整合 , 或兩者都使用。

#### 主題

- [步驟 1：在 Lambda 主控台建立 Lambda 函數 \(p. 10\)](#)
- [步驟 2：在 API Gateway 主控台建立 REST API \(p. 10\)](#)
- [步驟 3：在 API Gateway 主控台部署您的 REST API \(p. 12\)](#)
- [步驟 4：在 Lambda 主控台建立第二個 Lambda 函數 \(p. 13\)](#)
- [步驟 5：在 API Gateway 主控台將資源、方法和參數新增到 REST API \(p. 13\)](#)
- [後續步驟 \(p. 16\)](#)

## 步驟 1：在 Lambda 主控台建立 Lambda 函數

在此步驟中 , 您使用 AWS Lambda 主控台來建立簡單的 Lambda 函數。您將在下列步驟中使用此函數。

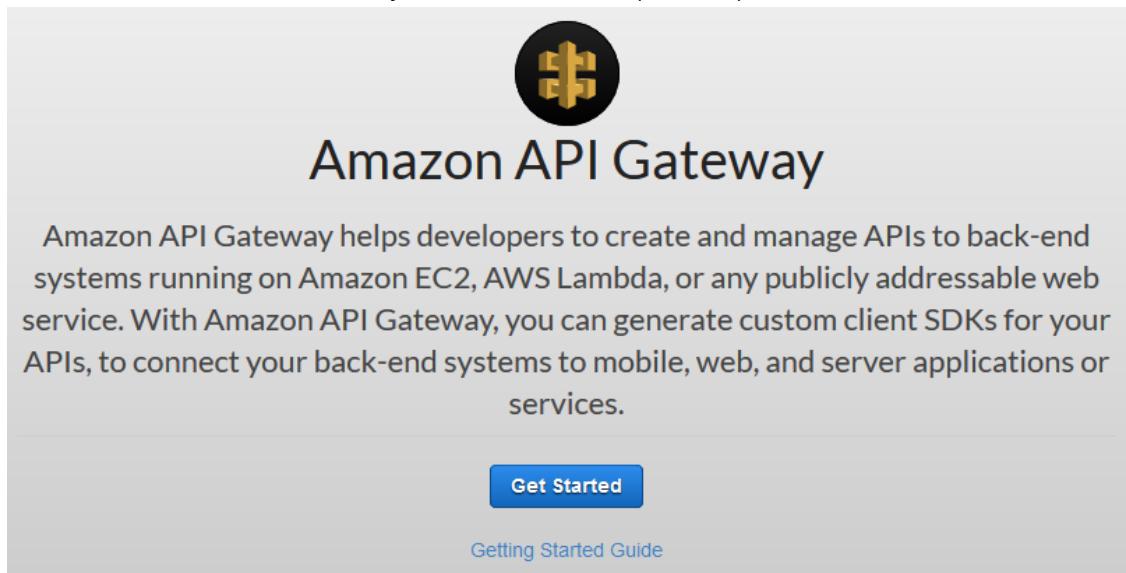
1. 完成[the section called “先決條件：準備在 API Gateway 中建置 API” \(p. 9\)](#) 中的步驟 (如果您尚未這麼做)。
2. 完成 AWS Lambda Developer Guide中的[建立 Lambda 函數](#)中的步驟。

## 步驟 2：在 API Gateway 主控台建立 REST API

在此步驟中 , 您在 API Gateway 主控台建立簡單的 REST API , 並將您的 Lambda 函數連接到此 API 當作後端。

1. 從 Services (服務) 功能表中 , 選擇 API Gateway 來前往 API Gateway 主控台。
2. 如果這是您第一次使用 API Gateway , 您會看到一個頁面向您介紹此服務的功能。選擇 Get Started (開始使用)。當 Create Example API (建立範例 API) 快顯出現時 , 選擇 OK (確定)。

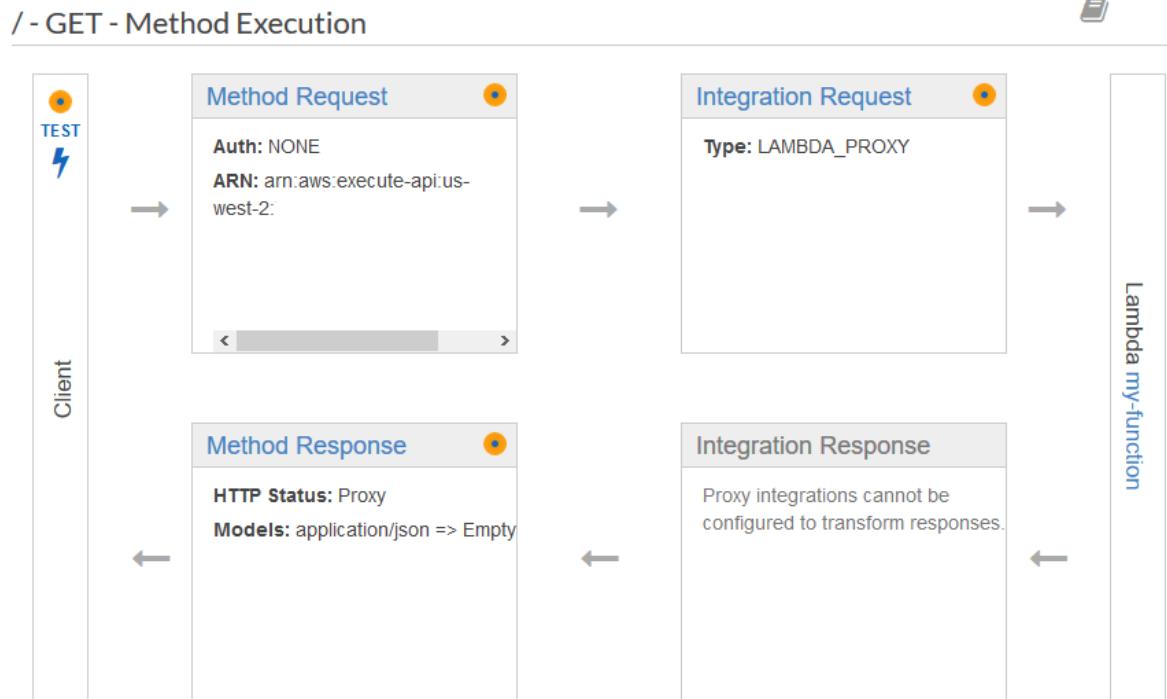
如果這不是第一次使用 API Gateway , 請選擇 Create API (建立 API)。



3. 在 Choose the protocol (選擇通訊協定) 下，選擇 REST。
4. 在 Create new API (建立新 API) 下，選擇 New API (新增 API)。
5. 在 Settings (設定) 下：
  - 針對 API name (API 名稱)，輸入 my-api。
  - 如有需要，在 Description (說明) 欄位中輸入說明，否則保留空白。
  - 將 Endpoint Type (端點類型) 設定保留為 Regional (區域)。
6. 選擇 Create API (建立 API)。
7. 在 Resources (資源) 下，什麼都沒有，您只會看到 /。這是根層級資源，對應於 API 的基本路徑 URL (<https://b123abcde4.execute-api.us-west-2.amazonaws.com/{stage-name}>)。

從 Actions (動作) 下拉式功能表中，選擇 Create Method (建立方法)。
8. 在資源名稱 (/) 下，您將會看到下拉式功能表。選擇 GET，然後選擇核取記號圖示以儲存您的選擇。
9. 在 / - GET – Setup (/ – GET – 設定) 窗格中，對於 Integration type (整合類型)，選擇 Lambda Function (Lambda 函數)。
10. 選擇 Use Lambda proxy integration (使用 Lambda 代理整合)。
11. 對於 Lambda Region (Lambda 區域)，選擇您建立 Lambda 函數的區域。
12. 在 Lambda Function (Lambda 函數) 欄位中，輸入任何字元，然後從下拉式功能表選擇 my-function (或您對上個步驟中建立的函數所給予的任何名稱)。(如果沒有出現下拉式功能表，請刪除您剛輸入的字元，讓下拉式功能表出現。) 將 Use Default Timeout (使用預設逾時) 保留勾選。選擇 Save (儲存) 以儲存您的選擇。
13. 當 Add Permission to Lambda Function (將許可新增至 Lambda 函數) 快顯出現時 (指出 "You are about to give API Gateway permission to invoke your Lambda function..." ('您將提供許可讓 API Gateway 叫用您的 Lambda 函數...'))，請選擇 OK (確定) 將該許可授與 API Gateway。

現在您會看到 / – GET – Method Execution (/ – GET – 方法執行) 窗格：



Method Execution (方法執行) 窗格依時鐘順序包含這些項目：

- Client (用戶端)：此方塊代表呼叫 API 的 GET 方法的用戶端 (瀏覽器或應用程式)。如果您選擇 Test (測試) 連結，然後選擇 Test (測試)，這樣會模擬來自用戶端的 GET 請求。
- Method Request (方法請求)：此方塊代表 API Gateway 收到的用戶端 GET 請求。如果您選擇 Method Request (方法請求)，您會看到一些事項的設定，例如授權，以及在方法請求當作整合請求傳遞到後端之前修改方法請求。在此步驟中，讓一切都設為預設值。
- Integration Request (整合請求)：此方塊代表傳送到後端的 GET 請求。以下設定取決於選取的 Integration Type (整合類型)。URL Path Parameters (URL 路徑參數)、URL Query String Parameters (URL 查詢字串參數)、HTTP Headers (HTTP 標頭) 和 Mapping Templates (映射範本) 設定適用於依特定後端所需來修改方法請求。讓 Integration Type (整合類型) 設為 Lambda function (Lambda 函數)，並讓其他設定設為預設值。
- Lambda **my-function**：此方塊代表您在步驟 1 建立的後端 Lambda 函數。如果您選擇 my-function，則會在 Lambda 主控台開啟 my-function Lambda 函數。
- Integration Response (整合回應)：此方塊代表後端的回應 (在當作方法回應傳送到用戶端之前)。若為 Lambda 代理整合，這整個方塊會呈現灰色，因為代理整合會按現狀傳回 Lambda 函數的輸出。若為 Lambda 非代理整合，您可以設定整合回應，將 Lambda 函數的輸出轉換成適合用戶端應用程式的形式。稍後在此演練中，您將了解怎麼做。

對於此入門程序，讓一切都設為預設值。

- Method Response (方法回應)：此方塊代表以 HTTP 狀態碼、選用的回應標頭及選用的回應內文形式，傳回給用戶端的方法回應。在預設情況下，Lambda 函數傳回的回應內文會按現狀以 JSON 文件形式傳遞，因此，回應內文的預設設定是 application/json，含有空的模型 (指出內文按現狀傳送)。這裡也讓一切都設為預設值。

## 步驟 3：在 API Gateway 主控台部署您的 REST API

一旦您完成步驟 2，您已建立 API，但還無法實際使用它。這是因為需要部署它。

1. 從 Actions (動作) 下拉式功能表中，選擇 Deploy API (部署 API)。
2. 從 Deployment stage (部署階段) 下拉式功能表中，選擇 [New Stage] ([新階段])。
3. 針對 Stage name (階段名稱)，輸入 prod。
4. 選擇 Deploy。
5. 在 prod Stage Editor (階段編輯器) 中，記下頂端的 Invoke URL (叫用 URL)。應該是這個格式：  
`(https://b123abcde4.execute-api.us-west-2.amazonaws.com/prod)`。如果您選擇 Invoke URL (叫用 URL)，則會以該 URL 開啟新的瀏覽器索引標籤。如果您重新整理新的瀏覽器索引標籤，您將會看到傳回的訊息內文 ("Hello from Lambda!")。

進行實際的 API 測試時，請使用 [cURL](#) 或 [POSTMAN](#) 等工具來測試您的 API。例如，如果您已在電腦上安裝 cURL，則可以如下所示測試您的 API：

1. 開啟終端機視窗。
2. 複製以下 cURL 命令，並將它貼到終端機視窗，同時將 `b123abcde4` 取代為您 API 的 API ID，以及將 `us-west-2` 取代為 API 的部署區域。

Linux or Macintosh

```
curl -X GET 'https://b123abcde4.execute-api.us-west-2.amazonaws.com/prod'
```

Windows

```
curl -X GET "https://b123abcde4.execute-api.us-west-2.amazonaws.com/prod"
```

此命令會傳回下列輸出："Hello from Lambda!"。

## 步驟 4：在 Lambda 主控台建立第二個 Lambda 函數

在此步驟中，您將建立第二個後端 Lambda 函數。這個函數接受輸入參數。在步驟 5，您將在 API 中建立子資源，此 API 有自己的 GET 方法，您會設定此方法將參數值傳給這個新的函數。

1. 從 Services (服務) 功能表中，選擇 Lambda 來前往 Lambda 主控台。
2. 選擇 Create function (建立函數)。
3. 選擇 Author from Scratch (從頭開始撰寫)。
4. 針對 Function name (函數名稱)，輸入 my-function2。
5. 在 Runtime (執行時間) 下，選擇 Node.js 8.10 或 Python 3.7。
6. 在 Permissions (許可) 下，展開 Choose or create an execution role (選擇或建立執行角色)。從 Execution role (執行角色) 下拉式清單中，選擇 Use an existing role (使用現有角色)。從 Existing role (現有角色) 中，從先前的 Lambda 函數選擇角色。角色的名稱是 service-role/my-function-a1b23c4d 格式。或者，選擇 Create a new role with basic Lambda permissions (建立具備基本 Lambda 許可的新角色)。
7. 選擇 Create function (建立函數)。
8. 在 Function code (函數程式碼) 窗格中，您將會看到預設的 "Hello from Lambda!" 函數。將整個函數換成下列程式碼：

Node.js 8.10

```
exports.handler = async (event) => {
  let myParam = event.myParam;
  const response = {
    statusCode: 200,
    body: JSON.stringify(myParam)
  };
  return response;
};
```

Python 3.7

```
import json
def lambda_handler(event, context):
    myParam = event['myParam']
    return {
        'statusCode': 200,
        'body': json.dumps(myParam)
    }
```

9. 選擇 Save (儲存)。

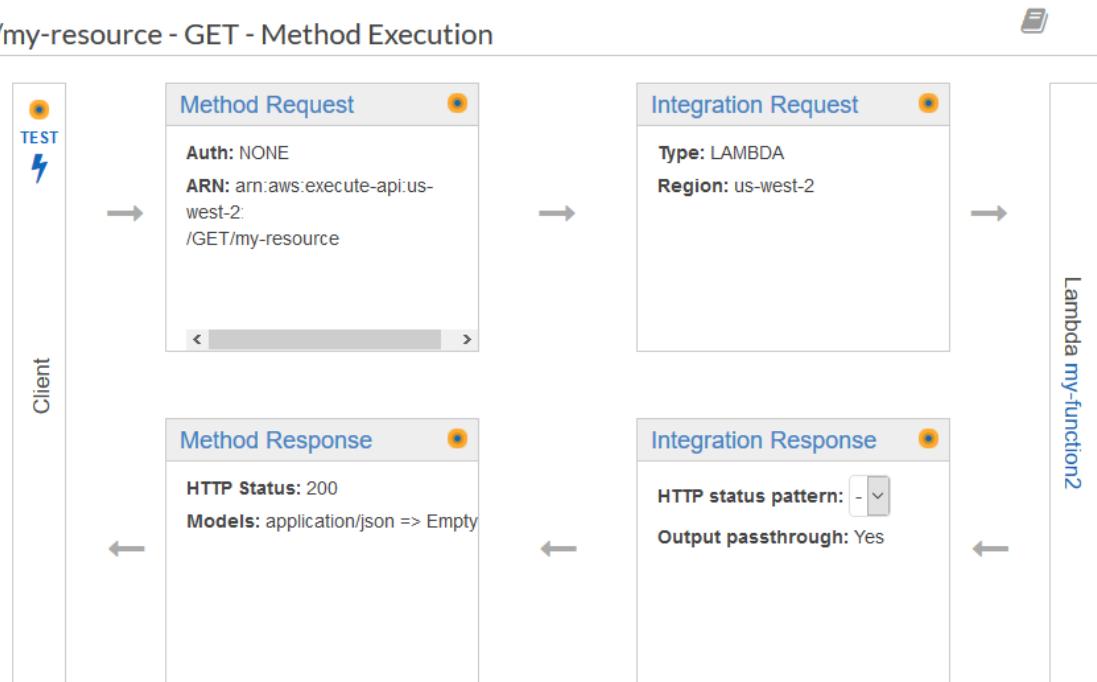
## 步驟 5：在 API Gateway 主控台將資源、方法和參數新增到 REST API

資源和方法是 REST API 的名詞和動詞。在此步驟中，您將為 API 建立子資源，並將 GET 方法新增到此資源。您會將查詢字串參數新增到新的方法，以符合您在步驟 4 建立的 Lambda 函數的輸入參數。您將整合新函數與此方法，以說明如何取得使用者輸入 (簡單的 "Hello from API Gateway!" 文字字串)，並映射到 Lambda 函數的輸入 (也是簡單的字串)。

1. 從 Services (服務) 功能表中，選擇 API Gateway 來前往 API Gateway 主控台。
2. 從 API 清單中選擇 my-api。
3. 在 Resources (資源) 下，選擇 /。
4. 從 Actions (動作) 下拉式功能表中，選擇 Create Resource (建立資源)。
5. 針對 Resource Name (資源名稱)，輸入 my-resource。請注意，Resource Path (資源路徑) 欄位中會自動填入資源名稱。
6. 選擇 Create Resource (建立資源)。
7. 從 Actions (動作) 下拉式功能表中，選擇 Create Method (建立方法)。
8. 在資源名稱 (my-resource) 下，您將會看到下拉式功能表。選擇 GET，然後選擇核取記號圖示以儲存您的選擇。
9. 在 /my-resource – GET – Setup (/my-resource – GET – 設定) 窗格中，對於 Integration type (整合類型)，選擇 Lambda Function (Lambda 函數)。
10. 對於 Lambda Region (Lambda 區域)，選擇您建立 Lambda 函數的區域。
11. 在 Lambda Function (Lambda 函數) 方塊中，輸入任何字元，然後從下拉式功能表選擇 my-function2。(如果沒有出現下拉式功能表，請刪除您剛輸入的字元，讓下拉式功能表出現。) 選擇 Save (儲存) 以儲存您的選擇。
12. 當 Add Permission to Lambda Function (將許可新增至 Lambda 函數) 快顯出現時 (指出 "You are about to give API Gateway permission to invoke your Lambda function..." ('您將提供許可讓 API Gateway 叫用您的 Lambda 函數...'))，請選擇 OK (確定) 將該許可授與 API Gateway。
13. 您將會看到 /my-resource – GET – Method Execution (/my-resource – GET – 方法執行) 窗格。請注意，這次 Integration Response (整合回應) 方塊不會呈現灰色。

選擇 Integration Request (整合請求)。

#### /my-resource - GET - Method Execution



14. 展開 Mapping Templates (映射範本)。
15. 將 Request body passthrough (請求內文傳遞) 設為 When there are no templates defined (未定義範本時)。
16. 選擇 Add mapping template (新增映射範本)。
17. 對於 Content-Type，輸入 application/json，並選擇核取記號圖示以儲存您的選擇。

18. 如果 Change passthrough behavior (變更傳遞行為) 快顯出現，請選擇 Yes, secure this integration。這可確保您的整合只允許的請求必須符合您剛定義的 Content-Type。
19. 在範本視窗中，輸入以下程式碼：

```
{  
    "myParam": "$input.params('myParam')"  
}
```

在輸入事件物件的主體中，此 JSON 文件將會傳送到您的 Lambda 函數。右手邊告訴 API Gateway 在您的輸入查詢字串中要尋找什麼。左手邊將它儲存到您將新增到 Lambda 函數的參數。

20. 選擇 Save。
21. 從 Actions (動作) 下拉式功能表中，選擇 Deploy API (部署 API)。
22. 從 Deployment stage (部署階段) 下拉式功能表中，選擇 prod。
23. 選擇 Deploy。

測試您的 API，如下所示：

1. 開啟終端機視窗。
2. 複製以下 cURL 命令，並將它貼到終端機視窗，同時將 **b123abcde4** 取代為您 API 的 API ID，以及將 **us-west-2** 取代為 API 的部署區域。

Linux or Macintosh

```
curl -X GET 'https://b123abcde4.execute-api.us-west-2.amazonaws.com/prod'
```

Windows

```
curl -X GET "https://b123abcde4.execute-api.us-west-2.amazonaws.com/prod"
```

這會叫用您的 API，您將會看到傳回 "Hello from Lambda!"。這是因為新的 Lambda 函數 my-function2 沒有連接到根資源 (/)。它會連接到您建立的新資源：my-resource。因此，您剛叫用的 Lambda 函數是您稍早建立的 "Hello from Lambda!" 函數：my-function。

3. 在 cURL 命令列中，在 prod 後面輸入 /my-resource?myParam=Hello%20from%20API%20Gateway!。完整命令應該如下所示：

Linux or Macintosh

```
curl -X GET 'https://b123abcde4.execute-api.us-west-2.amazonaws.com/prod/my-resource?myParam=Hello%20from%20API%20Gateway!'
```

Windows

```
curl -X GET "https://b123abcde4.execute-api.us-west-2.amazonaws.com/prod/my-resource?myParam=Hello%20from%20API%20Gateway!"
```

您現在應該會看到傳回正確的回應：{"statusCode": 200, "body": "\"Hello from API Gateway!\""}。

## 後續步驟

請探索下列任何或所有主題以繼續熟悉 Amazon API Gateway。

進一步了解	前往
將輸入傳送到後端 Lambda 函數	<a href="#">the section called “代理整合之 Lambda 函數的輸入格式” (p. 212)</a>
從後端 Lambda 函數傳回輸出	<a href="#">the section called “代理整合之 Lambda 函數的輸出格式” (p. 216)</a>
為您的 API 設定自訂網域名稱	<a href="#">the section called “設定 API 自訂網域名稱” (p. 590)</a>
將 Lambda 授權方函數新增到您的 API	<a href="#">the section called “使用 Lambda 授權方” (p. 348)</a>
將 Amazon Cognito 使用者集區授權方新增到您的 API	<a href="#">the section called “針對 REST API 使用 Cognito 使用者集區做為授權方” (p. 363)</a>
為您的 API 啟用 CORS	<a href="#">the section called “為 REST API 資源啟用 CORS” (p. 371)</a>

如需透過社群獲取 Amazon API Gateway 的相關協助，請參閱 [API Gateway 開發論壇](#)。(當您進入此論壇時，AWS 可能會要求您登入。)

如需直接從 AWS 獲取 API Gateway 的相關協助，請參閱 [AWS Support 頁面上的支援選項](#)。

另請參閱 [常見問答集 \(FAQ\)](#)，或[直接聯絡我們](#)。

## 在 Amazon API Gateway 中建立具有模擬整合的 REST API

您可以使用此逐步解說，在 Amazon API Gateway 中建立和部署具有模擬整合的 API。模擬整合可讓您的 API 直接傳回請求的回應，根本不需要將請求傳送到後端。這可讓您脫離後端以獨立開發 API。

### 主題

- [步驟 1：建立 API \(p. 16\)](#)
- [步驟 2：建立模擬整合 \(p. 18\)](#)
- [步驟 3：定義成功回應 \(p. 19\)](#)
- [步驟 4：新增 HTTP 500 狀態碼和錯誤訊息 \(p. 19\)](#)
- [步驟 5：測試模擬整合 \(p. 20\)](#)
- [後續步驟 \(p. 22\)](#)

## 步驟 1：建立 API

在此步驟中，您在 API Gateway 主控台建立具有 GET 方法和查詢參數的 API。

1. 完成[the section called “先決條件：準備在 API Gateway 中建置 API” \(p. 9\)](#) 中的步驟 (如果您尚未這麼做)。
2. 登入位於 <https://console.aws.amazon.com/apigateway> 的 API Gateway 主控台。

3. 如果這是您第一次使用 API Gateway，您會看到一個頁面向您介紹此服務的功能。選擇 Get Started (開始使用)。當 Create Example API (建立範例 API) 快顯出現時，選擇 OK (確定)。

如果不是第一次使用，請選擇 Create API (建立 API)。



Amazon API Gateway helps developers to create and manage APIs to back-end systems running on Amazon EC2, AWS Lambda, or any publicly addressable web service. With Amazon API Gateway, you can generate custom client SDKs for your APIs, to connect your back-end systems to mobile, web, and server applications or services.

[Get Started](#)

[Getting Started Guide](#)

4. 在 Choose the protocol (選擇通訊協定) 下，選擇 REST。
5. 在 Create new API (建立新 API) 下，選擇 New API (新增 API)。
6. 在 Settings (設定) 下：
  - 針對 API name (API 名稱)，輸入 my-api。
  - 如有需要，在 Description (說明) 欄位中輸入說明，否則保留空白。
  - 將 Endpoint Type (端點類型) 設定保留為 Regional (區域)。
7. 選擇 Create API。
8. 從 Actions (動作) 下拉式功能表中，選擇 Create Method (建立方法)。
9. 在資源名稱 (/) 下，您將會看到下拉式功能表。選擇 GET，然後選擇核取記號圖示以儲存您的選擇。
10. 您將會看到 / - GET - Setup (/ - GET - 設定) 窗格。針對 Integration type (整合類型)，選擇 Mock (模擬)。

## / - GET - Setup



Choose the integration point for your new method.

Integration type

Lambda Function [?](#)

HTTP [?](#)

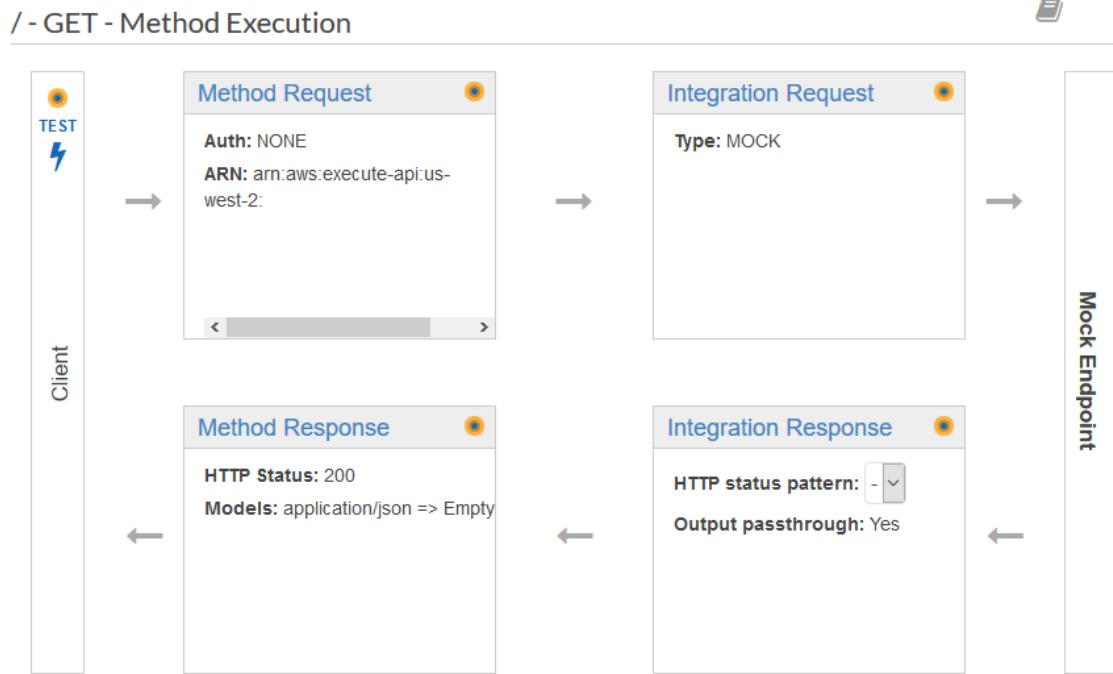
Mock [?](#)

AWS Service [?](#)

VPC Link [?](#)

[Save](#)

11. 選擇 Save (儲存)。現在您會看到 / - GET – Method Execution (/ - GET – 方法執行) 窗格：



Method Execution (方法執行) 窗格依時鐘順序包含這些項目：

- Client (用戶端)：此方塊代表呼叫 API 的 GET 方法的用戶端（瀏覽器或應用程式）。如果您選擇 Test (測試) 連結，然後選擇 Test (測試)，這樣會模擬來自用戶端的 GET 請求。
- Method Request (方法請求)：此方塊代表 API Gateway 收到的用戶端 GET 請求。如果您選擇 Method Request (方法請求)，您會看到一些事項的設定，例如授權，以及在方法請求當作整合請求傳遞到後端之前修改方法請求。
- Integration Request (整合請求)：此方塊代表傳送到後端的 GET 請求。若為模擬整合，Mapping Templates (映射範本) 可用來指定要傳回的回應 (HTTP 狀態碼和訊息)。
- Mock Endpoint (模擬端點)：此方塊代表空的後端。
- Integration Response (整合回應)：此方塊代表後端的回應 (在當作方法回應傳送到用戶端之前)。若為模擬整合，回應是在 Integration Request (整合請求) 方塊中定義 (因為沒有後端提供它)。您可以設定整合回應，將任意輸出轉換成適合用戶端應用程式的形式。稍後在此演練中，您將了解怎麼做。
- Method Response (方法回應)：此方塊代表可傳回給用戶端的 HTTP 狀態碼。若為模擬整合，針對每個狀態碼，回應的其餘部分是在 Integration Response (整合回應) 方塊中設定。

在此步驟中，讓一切都設為預設值。

## 步驟 2：建立模擬整合

在此步驟中，您將查詢字串參數新增到 GET 方法，然後指定 API 將傳回的回應代碼和訊息，而不是與後端整合。

首先，建立查詢字串參數：

1. 在 Method Execution (方法執行) 窗格中，選擇 Method Request (方法請求)。
2. 在 / - GET - Method Request (/ - GET - 方法請求) 窗格中，展開 URL Query String Parameters (URL 查詢字串參數)。

3. 選擇 Add query string (新增查詢字串)。
4. 對於 Name (名稱)，輸入 myParam，並選擇核取記號圖示以儲存您的選擇。

接著，建立映射範本，將查詢字串參數值映射到要傳回給用戶端的 HTTP 狀態碼值。

1. 選擇 Method Execution (方法執行)。
2. 選擇 Integration Request (整合請求)。
3. 展開 Mapping Templates (映射範本)。
4. 對於 Request body passthrough (請求內文傳遞)，選擇 When there are no templates defined (recommended) (未定義範本時 (建議))。
5. 在 Content-Type 下，選擇 application/json。
6. 將範本內容換成下列程式碼：

```
{  
    #if( $input.params('myParam') == "myValue" )  
        "statusCode": 200  
    #else  
        "statusCode": 500  
    #end  
}
```

7. 選擇 Save (儲存)。

## 步驟 3：定義成功回應

現在您將建立映射範本，將 HTTP 200 狀態碼值映射到要傳回給用戶端的成功訊息。

1. 選擇 Method Execution (方法執行) 和 Integration Response (整合回應)。
2. 展開 200 回應，然後展開 Mapping Templates (映射範本) 區段。
3. 在 Content-Type 下，選擇 application/json。
4. 對於範本內容，輸入以下程式碼：

```
{  
    "statusCode": 200,  
    "message": "Hello from API Gateway!"  
}
```

5. 選擇 Save (儲存)。

### Note

這個窗格中有兩個 Save (儲存) 按鈕。請務必選擇 Mapping Templates (映射範本) 區段中的那個按鈕。

## 步驟 4：新增 HTTP 500 狀態碼和錯誤訊息

在此步驟中，您新增前端可傳回給用戶端的 HTTP 500 狀態碼，然後將它映射到錯誤訊息。

首先，新增 500 狀態碼：

1. 選擇 Method Execution (方法執行) 和 Method Response (方法回應)。
2. 選擇 Add response (新增回應)。
3. 對於 HTTP status (HTTP 狀態)，輸入 500，並選擇核取記號圖示來儲存設定。

現在建立映射範本，將前端「傳回」給用戶端的 500 狀態碼值映射到錯誤訊息：

1. 選擇 Method Execution (方法執行) 和 Integration Response (整合回應)。
2. 選擇 Add integration response (新增整合回應)。
3. 對於 HTTP status regex (HTTP 狀態 regex)，輸入 `5\d{2}`。
4. 對於 Method response status (方法回應狀態)，從下拉式功能表選擇 500，然後選擇 Save (儲存)。
5. 展開 500 回應，然後展開 Mapping Templates (映射範本) 區段 (如果尚未展開的話)。
6. 在 Content-Type 下，選擇 Add mapping template (新增映射範本)。
7. 在 Content-Type 下的方塊中，輸入 `application/json`，並選擇核取記號圖示以儲存您的選擇。
8. 對於範本內容，輸入以下程式碼：

```
{  
    "statusCode": 500,  
    "message": "This is an error message."  
}
```

9. 選擇 Save (儲存)。

#### Note

這個窗格中有兩個 Save (儲存) 按鈕。請務必選擇 Mapping Templates (映射範本) 區段中的那個按鈕。

## 步驟 5：測試模擬整合

在此步驟中，您將測試模擬整合。

1. 選擇 Method Execution (方法執行)，然後選擇 Test (測試)。
2. 在 Query Strings (查詢字串) 下，輸入 `myParam=myValue`。
3. 選擇 Test (測試)。

您應該會看到類似以下的輸出：

Request: /?myParam=myValue



Status: 200

Latency: 5 ms

Response Body

```
{  
    "statusCode": 200,  
    "message": "Hello from API Gateway!"  
}
```

Response Headers

```
{"Content-Type": "application/json"}
```

Logs

```
Execution log for request 2083a122-7076-11e9-b3da-c5566e62847c  
Tue May 07 03:13:46 UTC 2019 : Starting execution for request: 2083a122-7076-11e9-b3da-c5566e62847c  
Tue May 07 03:13:46 UTC 2019 : HTTP Method: GET, Resource Pa
```

4. 在 Query Strings (查詢字串) 下，輸入 myParam=""。或者，完全刪除 Query Strings (查詢字串) 方塊中的任何內容。
5. 選擇 Test (測試)。

您應該會看到類似以下的輸出：

Request: /?myParam=""

Status: 500

Latency: 5 ms

Response Body



```
{  
    "statusCode": 500,  
    "message": "This is an error message."  
}
```

Response Headers

```
{"Content-Type": "application/json"}
```

Logs

```
Execution log for request b7132dc8-7076-11e9-897f-832e244f10ba  
Tue May 07 03:17:59 UTC 2019 : Starting execution for request b7132dc8-7076-11e9-897f-832e244f10ba  
Tue May 07 03:17:59 UTC 2019 : HTTP Method: GET, Resource Path: /
```

## 後續步驟

請探索下列任何或所有主題以繼續熟悉 Amazon API Gateway。

進一步了解	前往
定義方法請求和回應	<a href="#">the section called “設定 REST API 方法” (p. 188)</a>
定義整合回應	<a href="#">the section called “設定整合回應” (p. 205)</a>
為您的 API 設定自訂網域名稱	<a href="#">the section called “設定 API 自訂網域名稱” (p. 590)</a>
為您的 API 啟用 CORS	<a href="#">the section called “為 REST API 資源啟用 CORS” (p. 371)</a>

如需透過社群獲取 Amazon API Gateway 的相關協助，請參閱 [API Gateway 開發論壇](#)。(當您進入此論壇時，AWS 可能會要求您登入。)

如需直接從 AWS 獲取 API Gateway 的相關協助，請參閱 [AWS Support 頁面上的支援選項](#)。

另請參閱 [常見問答集 \(FAQ\)](#)，或[直接聯絡我們](#)。

## Amazon API Gateway 影片

本節提供關於 Amazon API Gateway 的幾部 AWS 教學影片，可做為額外的入門資源。

### 主題

- 來自 Twitch 「在 AWS 上建置」影集的 Amazon API Gateway 影片 (p. 23)
- 其他 Amazon API Gateway 影片 (p. 23)

## 來自 Twitch 「在 AWS 上建置」影集的 Amazon API Gateway 影片

以下影片來自 Twitch.tv 上的 [在 AWS 上建置影集](#)。

[Introduction to Building Happy Little APIs](#) (1 分鐘，YouTube 網站)

這部影片簡介 Building Happy Little APIs 影片系列。

[Building Happy Little APIs | Episode 1 – I Didn't Know Amazon API Gateway Did That](#) (60 分鐘，YouTube 網站)

這部影片簡介 Amazon API Gateway 及其可解決的問題。其中探討 API Gateway 的運作組件並提供可能使用案例的範例。此概觀旨在讓您充分了解為什麼應該使用 API Gateway 及其用途。

[Building Happy Little APIs | Episode 2 – No REST for the Weary](#) (55 分鐘，YouTube 網站)

這部影片說明如何針對後端建置簡單的應用程式，以搭配 AWS Lambda 函數來使用 Amazon API Gateway。您將了解如何使用 AWS Serverless Application Model (AWS SAM) 來塑造 API 及其幕後應用程式的模型。您也將了解整合和代理之間的差異，及這兩者各自的使用時機。

## 其他 Amazon API Gateway 影片

以下影片放在 YouTube 網站上的 AWS 線上技術會談頻道。

[Building APIs with Amazon API Gateway](#) (43 分鐘，YouTube 網站)

這部影片描述 Amazon API Gateway 的功能，並說明如何開始建置 API。

[Best Practices for Building Enterprise Grade APIs with Amazon API Gateway](#) (40 分鐘，YouTube 網站)

這部影片說明如何使用 Amazon API Gateway 與其他 AWS 服務，以設計和運用適合企業使用的 API。您將了解可協助您建立、維護和保護企業 API 的最佳實務。

# Amazon API Gateway 教學課程

下列教學提供實作練習以協助您了解 API Gateway。

## 主題

- [建置具有 Lambda 整合的 API Gateway API \(p. 24\)](#)
- [教學課程：匯入範例來建立 REST API \(p. 40\)](#)
- [建置具有 HTTP 整合的 API Gateway API \(p. 47\)](#)
- [教學：建置具有 API Gateway 私有整合的 API \(p. 80\)](#)
- [教學：建置具有 AWS 整合的 API Gateway API \(p. 81\)](#)
- [教學課程：使用兩個 AWS 服務整合和一個 Lambda 非代理整合建立 Calc REST API \(p. 86\)](#)
- [教學：在 API Gateway 中建立 REST API 做為 Amazon S3 代理 \(p. 104\)](#)
- [教學：在 API Gateway 中建立 REST API 做為 Amazon Kinesis 代理 \(p. 129\)](#)

## 建置具有 Lambda 整合的 API Gateway API

若要建置具有 Lambda 整合的 API，您可以使用 Lambda 代理整合或 Lambda 非代理整合。

在 Lambda 代理整合中，可以透過任意的請求標頭、路徑變數、查詢字串參數和內文組合來表示整合式 Lambda 函數的輸入。此外，Lambda 函數可以使用 API 組態設定來影響其執行邏輯。針對 API 開發人員，設定 Lambda 代理整合十分簡單。除了選擇所指定區域中的特定 Lambda 函數之外，您需要執行的操作很少。API Gateway 會設定整合請求和整合回應。設定之後，整合式 API 方法可以隨著後端演進，而不需要修改現有設定。原因可能是後端 Lambda 函數開發人員會剖析傳入請求資料，並在未發生任何錯誤時回應用戶端所需的結果，或在發生錯誤時回應錯誤訊息。

在 Lambda 非代理整合中，您必須確保將 Lambda 函數的輸入提供為整合請求承載。這表示，身為 API 開發人員，您必須將用戶端提供為請求參數的任何輸入資料對應至適當的整合請求內文。您也可能需要將用戶端提供的請求內文翻譯為 Lambda 函數可辨識的格式。

## 主題

- [教學：建置具有 Lambda 代理整合的 Hello World API \(p. 24\)](#)
- [教學：建置具有跨帳戶 Lambda 代理整合的 API Gateway API \(p. 29\)](#)
- [教學：建置具有 Lambda 非代理整合的 API Gateway API \(p. 31\)](#)

## 教學：建置具有 Lambda 代理整合的 Hello World API

Lambda 代理整合 (p. 206) 是一種輕量型彈性 API Gateway API 整合類型，可讓您整合 API 方法 (或整個 API) 與 Lambda 函數。Lambda 函數的編寫方式可以是 [Lambda 支援的任何語言](#)。因為它是代理整合，所以您可以隨時變更 Lambda 函數實作，無需重新部署您的 API。

在此教學中，您將執行下列操作：

- 建立 "Hello, World!" Lambda 函數，做為後端的 API。
- 建立和測試 具有 Lambda 代理整合的 "Hello, World!" API。

### 主題

- 建立 "Hello, World!" Lambda 函數 (p. 25)
- 建立 "Hello, World!" API (p. 26)
- 部署和測試 API (p. 27)

## 建立 "Hello, World!" Lambda 函數

此函數會將問候語傳回給發起人，做為下列格式的 JSON 物件：

```
{  
    "greeting": "Good time, name of city.[ Happy day! ]"  
}
```

在 Lambda 主控台中建立 "Hello, World!" Lambda 函數

1. 登入位於 <https://console.aws.amazon.com/lambda> 的 Lambda 主控台。
2. 在 AWS 導覽列上，選擇區域 (例如，US East (N. Virginia) (美國東部 (維吉尼亞北部)))。

#### Note

記下您建立 Lambda 函數的區域。在建立 API 時，您將需要此區域。

3. 在導覽窗格中，選擇 Functions (函數)。
4. 選擇 Create function (建立函數)。
5. 選擇 Author from scratch (從頭開始撰寫)。
6. 在 Basic information (基本資訊) 下，請執行下列動作：
  - a. 在 Function name (函數名稱) 中，輸入 **GetStartedLambdaProxyIntegration**。
  - b. 從 Runtime (執行時間) 下拉式清單中，選擇 Node.js 8.10。
  - c. 在 Permissions (許可) 下，展開 Choose or create an execution role (選擇或建立執行角色)。從 Execution role (執行角色) 下拉式清單中，選擇 Create new role from AWS policy templates (從 AWS 政策範本建立新角色)。
  - d. 在 Role name (角色名稱) 中，輸入 **GetStartedLambdaBasicExecutionRole**。
  - e. 將 Policy templates (政策範本) 欄位留白。
  - f. 選擇 Create function (建立函數)。
7. 在內嵌程式碼編輯器的 Function code (函數程式碼) 下，複製/貼上下列程式碼：

```
'use strict';  
console.log('Loading hello world function');  
  
exports.handler = async (event) => {  
    let name = "you";  
    let city = 'World';  
    let time = 'day';  
    let day = '';  
    let responseCode = 200;  
    console.log("request: " + JSON.stringify(event));  
  
    if (event.queryStringParameters && event.queryStringParameters.name) {  
        console.log("Received name: " + event.queryStringParameters.name);  
        name = event.queryStringParameters.name;  
    }  
  
    if (event.queryStringParameters && event.queryStringParameters.city) {  
        console.log("Received city: " + event.queryStringParameters.city);  
        city = event.queryStringParameters.city;  
    }  
};
```

```
}

if (event.headers && event.headers['day']) {
    console.log("Received day: " + event.headers.day);
    day = event.headers.day;
}

if (event.body) {
    let body = JSON.parse(event.body)
    if (body.time)
        time = body.time;
}

let greeting = `Good ${time}, ${name} of ${city}.`;
if (day) greeting += ` Happy ${day}!`;

let responseBody = {
    message: greeting,
    input: event
};

// The output from a Lambda proxy integration must be
// in the following JSON object. The 'headers' property
// is for custom response headers in addition to standard
// ones. The 'body' property must be a JSON string. For
// base64-encoded payload, you must also set the 'isBase64Encoded'
// property to 'true'.
let response = {
    statusCode: responseCode,
    headers: {
        "x-custom-header" : "my custom header value"
    },
    body: JSON.stringify(responseBody)
};
console.log("response: " + JSON.stringify(response))
return response;
};
```

8. 選擇 Save (儲存)。

## 建立 "Hello, World!" API

現在，使用 API Gateway 主控台為您的 "Hello, World!" Lambda 函數建立 API。

### 建置 "Hello, World!" API

1. 登入位於 <https://console.aws.amazon.com/apigateway> 的 API Gateway 主控台。
2. 如果這是您第一次使用 API Gateway，您會看到一個頁面向您介紹此服務的功能。選擇 Get Started (開始使用)。當 Create Example API (建立範例 API) 快顯出現時，選擇 OK (確定)。

如果這不是第一次使用 API Gateway，請選擇 Create API (建立 API)。

3. 建立空白 API，如下所示：
  - a. 在 Choose the protocol (選擇通訊協定) 下，選擇 REST。
  - b. 在 Create new API (建立新 API) 下，選擇 New API (新增 API)。
  - c. 在 Settings (設定) 下：
    - 針對 API name (API 名稱)，輸入 **LambdaSimpleProxy**。
    - 如有需要，在 Description (說明) 欄位中輸入說明，否則保留空白。
    - 將 Endpoint Type (端點類型) 設定保留為 Regional (區域)。

- d. 選擇 Create API (建立 API)。
4. 建立 helloworld 資源，如下所示：
  - a. 在 Resources (資源) 樹狀結構中選擇根資源 (/)。
  - b. 從 Actions (動作) 下拉式功能表中，選擇 Create Resource (建立資源)。
  - c. 將 Configure as proxy resource (設定為 Proxy 資源) 保留為未核取狀態。
  - d. 針對 Resource Name (資源名稱)，輸入 **helloworld**。
  - e. 將 Resource Path (資源路徑) 設定保留為 /helloworld。
  - f. 將 Enable API Gateway CORS (啟用 API 通道 CORS) 保留為未核取狀態。
  - g. 選擇 Create Resource (建立資源)。
5. 在代理整合中，透過一個代表任何 HTTP 方法的囊括型 ANY 方法，將整個請求按現狀傳送到後端 Lambda 函數。實際的 HTTP 方法由用戶端在執行時間指定。ANY 方法可讓使用針對所有支援的 HTTP 方法而設定的單一 API 方法：DELETE、GET、HEAD、OPTIONS、PATCH、POST 和 PUT。

若要設定 ANY 方法，請執行下列動作：

- a. 在 Resources (資源) 清單中，選擇 /helloworld。
- b. 從 Actions (動作) 功能表中，選擇 Create method (建立方法)。
- c. 從下拉式功能表中選擇 ANY，然後選擇核取記號圖示
- d. 將 Integration type (整合類型) 設定保留為 Lambda Function (Lambda 函數)。
- e. 選擇 Use Lambda Proxy integration (使用 Lambda 代理整合)。
- f. 從 Lambda Region (Lambda 區域) 下拉式功能表中，選擇您已在其中建立 GetStartedLambdaProxyIntegration Lambda 函數的區域。
- g. 在 Lambda Function (函數) 欄位中，輸入任何字元，並從下拉式功能表中選擇 GetStartedLambdaProxyIntegration。
- h. 將 Use Default Timeout (使用預設逾時) 保留勾選。
- i. 選擇 Save (儲存)。
- j. 出現 Add Permission to Lambda Function (將許可新增至 Lambda 函數) 的提示時，選擇 OK (確定)。

## 部署和測試 API

### 在 API Gateway 主控台中部署 API

1. 從 Actions (動作) 下拉式功能表中，選擇 Deploy API (部署 API)。
2. 針對 Deployment stage (部署階段)，選擇 [new stage] ([新增階段])。
3. 針對 Stage name (階段名稱)，輸入 **test**。
4. 如有需要，輸入 Stage description (階段說明)。
5. 如有需要，輸入 Deployment description (部署說明)。
6. 選擇 Deploy (部署)。
7. 記下 API 的 Invoke URL (叫用 URL)。

### 使用瀏覽器和 cURL 來測試具有 Lambda 代理整合的 API

您可以使用瀏覽器或 **cURL** 來測試您的 API。

若要僅使用查詢字串參數來測試 GET 請求，您可以將 API helloworld 資源的 URL 輸入至瀏覽器位址列。例如：<https://r275xc9bmd.execute-api.us-east-1.amazonaws.com/test/helloworld?name=John&city=Seattle>

針對其他方法，您必須使用更進階的 REST API 測試公用程式，例如 [POSTMAN](#) 或 [cURL](#)。本教學使用 cURL。以下 cURL 命令範例假設 cURL 已安裝在您的電腦上。

若要使用 cURL 測試已部署的 API：

1. 開啟終端機視窗。
2. 複製以下 cURL 命令，並將它貼到終端機視窗，同時將 `r275xc9bmd` 取代為您 API 的 API ID，以及將 `us-east-1` 取代為 API 的部署區域。

```
curl -v -X POST \
  'https://r275xc9bmd.execute-api.us-east-1.amazonaws.com/test/helloworld?
name=John&city=Seattle' \
-H 'content-type: application/json' \
-H 'day: Thursday' \
-d '{ "time": "evening" }'
```

#### Note

如果您是在 Windows 上執行命令，請改用此語法：

```
curl -v -X POST "https://r275xc9bmd.execute-api.us-east-1.amazonaws.com/test/
helloworld?name=John&city=Seattle" -H "content-type: application/json" -H "day:
Thursday" -d "{ \"time\": \"evening\" }"
```

您應該取得承載的成功回應，如下所示：

```
{
  "message": "Good evening, John of Seattle. Happy Thursday!",
  "input": {
    "resource": "/helloworld",
    "path": "/helloworld",
    "httpMethod": "POST",
    "headers": {"Accept": "*/*", "Content-Type": "application/json", "Day": "Thursday", "Host": "r275xc9bmd.execute-api.us-east-1.amazonaws.com", "User-Agent": "curl/7.64.0", "X-Amzn-Trace-Id": "Root=1-1a2b3c4d-a1b2c3d4e5f6a1b2c3d4e5f6", "X-Forwarded-For": "72.21.198.64", "X-Forwarded-Port": "443", "X-Forwarded-Proto": "https"}, "MultiValueHeaders": {"Accept": ["*/*"], "Content-Type": ["application/json"], "Day": ["Thursday"]}, "Content-Type": "application/json", "Day": "Thursday", "Host": ["r275xc9bmd.execute-api.us-east-1.amazonaws.com"], "User-Agent": ["curl/0.0.0"], "X-Amzn-Trace-Id": ["Root=1-1a2b3c4d-a1b2c3d4e5f6a1b2c3d4e5f6"], "X-Forwarded-For": ["11.22.33.44"], "X-Forwarded-Port": ["443"], "X-Forwarded-Proto": ["https"]}, "QueryStringParameters": {"city": "Seattle", "name": "John"}, "MultiValueQueryStringParameters": {"city": ["Seattle"], "name": ["John"]}, "PathParameters": null, "StageVariables": null, "RequestContext": {"ResourceId": "3htbry", "Stage": "prod", "RequestId": "12345678901234567890123456789012", "HttpMethod": "GET", "Path": "/helloworld", "HttpVersion": "HTTP/1.1", "ApiId": "r275xc9bmd", "Account": "12345678901234567890123456789012", "Identity": "{'principal_arn': 'arn:aws:sts::12345678901234567890123456789012:assumed-role/lambda-role/12345678901234567890123456789012', 'source_ip': '123.45.67.89', 'source_port': 443, 'user_agent': 'curl/7.64.0', 'account_id': '12345678901234567890123456789012', 'stage': 'prod'}"}}
```

```
"resourcePath":"/helloworld",
"htt* Connection #0 to host r275xc9bmd.execute-api.us-east-1.amazonaws.com left intact
pMethod":"POST",
"extendedRequestId":"a1b2c3d4e5f6g7h=",
"requestTime":"20/Mar/2019:20:38:30 +0000",
"path":"/test/helloworld",
"accountId":"123456789012",
"protocol":"HTTP/1.1",
"stage":"test",
"domainPrefix":"r275xc9bmd",
"requestTimeEpoch":1553114310423,
"requestId":"test-invoke-request",
"identity":{"cognitoIdentityPoolId":null,
"accountId":null,
"cognitoIdentityId":null,
"caller":null,
"sourceIp":"test-invoke-source-ip",
"accessKey":null,
"cognitoAuthenticationType":null,
"cognitoAuthenticationProvider":null,
"userArn":null,
"userAgent":"curl/0.0.0","user":null
},
"domainName":"r275xc9bmd.execute-api.us-east-1.amazonaws.com",
"apiId":"r275xc9bmd"
},
"body":"{ \"time\": \"evening\" }",
"isBase64Encoded":false
}
}
```

如果您在先前的方法請求中將 POST 變更為 PUT，則應該會取得相同的回應。

若要測試 GET 方法，請複製以下 cURL 命令，並貼到終端機視窗，需要將 **r275xc9bmd** 換成 API 的 API ID，將 **us-east-1** 換成部署 API 的區域。

```
curl -X GET \
'https://r275xc9bmd.execute-api.us-east-1.amazonaws.com/test/helloworld/Seattle?
name=John' \
-H 'content-type: application/json' \
-H 'day: Thursday'
```

您應該取得與先前 POST 請求結果類似的回應，除了 GET 請求沒有任何承載之外。因此，body 參數會是 null。

## 教學：建置具有跨帳戶 Lambda 代理整合的 API Gateway API

您現在可以從不同的 AWS 帳戶使用 AWS Lambda 函數做為 API 整合後端。每個帳戶可以位於 Amazon API Gateway 可用的任何區域。這可讓您輕鬆地集中管理和分享跨多個 API 的 Lambda 後端函數。

在本節中，我們示範如何使用 Amazon API Gateway 主控台設定跨帳戶 Lambda 代理整合。

首先，我們會從某帳戶的 [the section called “教學課程：匯入範例來建立 REST API” \(p. 40\)](#) 中建立範例 API。然後，我們會在另一個帳戶建立 Lambda 函數。最後，我們使用跨帳戶 Lambda 整合，以讓範例 API 使用我們在第二個帳戶中建立的 Lambda 函數。

### 建立 API Gateway 跨帳戶 Lambda 整合的 API

首先，您將建立 [the section called “教學課程：匯入範例來建立 REST API” \(p. 40\)](#) 中所述的範例 API。

### 建立範例 API

1. 登入 API Gateway 主控台。
2. 從 API Gateway API 首頁選擇 Create API (建立 API)：
3. 在 Create new API (建立新的 API) 下，選擇 Examples API (範例 API)。
4. 對於 Endpoint Type (端點類型)，選擇 Edge optimized (邊緣最佳化)。
5. 選擇 Import (匯入) 來建立範例 API。

## 在另一個帳戶建立 Lambda 整合函數

現在您可以從與您在範例 API 中建立的不同帳戶建立 Lambda 函數。

### 在另一個帳戶中建立 Lambda 函數

1. 透過與您在 API Gateway API 中建立的不同的帳戶登入 Lambda 主控台。
2. 選擇 Create function (建立函式)。
3. 選擇 Author from scratch (從頭開始撰寫)。
4. 在 Author from scratch (從頭開始撰寫) 下，進行下列操作：
  - a. 在 Name (名稱) 輸入欄位中，輸入函數名稱。
  - b. 從 Runtime (執行時間) 下拉式清單中，選擇支援的執行時間。在這個範例中，我們使用 Node.js 8.10。
  - c. 從 Role (角色) 下拉式清單中，選擇 Choose an existing role (選擇現有角色)、Create new role from template(s) (從範本建立新角色) 或 Create a custom role (建立自訂角色)。然後，遵循所選擇的確認說明。
  - d. 選擇 Create function (建立函數) 繼續。

在這個範例中，我們將略過 Designer (設計工具) 區段，並接著移至 Function code (函數程式碼) 區段。

5. 向下捲動到 Function code (函數程式碼) 窗格中。
6. 從 [the section called “教學：具有 Lambda 代理整合的 Hello World API” \(p. 24\)](#) 複製並貼上 Node.js 函數實作。
7. 從 Runtime (執行時間) 下拉式功能表中，選擇 Node.js 8.10。
8. 選擇 Save (儲存)。
9. 請注意適用於您函數的完整 ARN (在 Lambda 函數窗格的右上角中)。在建立跨帳戶 Lambda 整合時會需要用到。

## 設定跨帳戶 Lambda 整合

一旦在不同的帳戶有了 Lambda 整合功能，您就可以使用 API Gateway 主控台，在您的第一個帳戶中將其新增至 API。

### Note

如果您設定的是跨區域、跨帳戶授權方，新增到目標函數的 sourceArn 應使用函數的區域，而非 API 的區域。

### 設定跨帳戶 Lambda 整合

1. 在 API Gateway 主控台中選擇您的 API。

2. 選擇 Resources (資源)。
3. 在 Resources (資源) 窗格中，選擇 top-level GET 方法。
4. 在 Method Execution (方法執行) 窗格中，選擇 Integration Request (整合請求)。
5. 針對 Integration type (整合類型)，選擇 Lambda Function (&LAM; 函數)。
6. 選擇 Use Lambda Proxy integration (使用 &LAM; 代理整合)。
7. 將 LambdaRegion (&LAM; 區域) 設定為您帳戶的區域。
8. 對於 Lambda Function (&LAM; 函數)，複製/貼上您在第二個帳戶中建立之 Lambda 函數的完整 ARN 並選擇核取方塊。
9. 您會看到一個快顯視窗，告知您將許可新增至 Lambda 函數：您已從另一個帳戶選取 Lambda 函數。請確保您對此函數有適當的函數政策。您可以從帳戶 **123456789012**：接著一個 aws lambda add-permission 命令字串來執行以下 AWS CLI 命令進行確認。
10. 將 aws lambda add-permission 命令字串複製貼上至為第二個帳戶所設定的 AWS CLI 視窗。這會授與您的第一個帳戶對第二個帳戶 Lambda 函數的存取權。
11. 在 Lambda 主控台先前步驟中的快顯視窗中，選擇 OK (確定)。
12. 若要在 Lambda 主控台查看適用您函數的更新政策，
  - a. 選擇整合函數。
  - b. 在 Designer (設計工具) 窗格中，選擇該金鑰圖示。

在 Function policy (函數政策) 窗格中，您現在應該會看到內含 Condition 子句的 Allow 政策，其中 AWS:SourceArn 為您 API GET 方法的 ARN。

## 教學：建置具有 Lambda 非代理整合的 API Gateway API

在此演練中，我們使用 API Gateway 主控台建置一個 API，讓用戶端可以透過 Lambda 非代理整合 (也稱為自訂整合) 呼叫 Lambda 函數。如需 AWS Lambda 與 Lambda 函數的詳細資訊，請參閱[AWS Lambda Developer Guide](#)。

為方便學習，我們選擇了一個具有最少 API 設定的簡單 Lambda 函數，以逐步引導您建置具有 Lambda 自訂整合的 API Gateway API。必要時，我們會說明一些邏輯。如需 Lambda 自訂整合的更詳細範例，請參閱[教學課程：使用兩個 AWS 服務整合和一個 Lambda 非代理整合建立 Calc REST API \(p. 86\)](#)。

建立 API 之前，請在 AWS Lambda 中建立 Lambda 函數來設定 Lambda 後端，以下將進行說明。

### 主題

- [為 Lambda 非代理整合建立 Lambda 函數 \(p. 31\)](#)
- [建立具有 Lambda 非代理整合的 API \(p. 34\)](#)
- [測試呼叫 API 方法 \(p. 36\)](#)
- [部署 API \(p. 38\)](#)
- [在部署階段測試 API \(p. 38\)](#)
- [清除 \(p. 39\)](#)

## 為 Lambda 非代理整合建立 Lambda 函數

### Note

建立 Lambda 函數可能會導致您的 AWS 帳戶產生費用。

在此步驟中，您會為 Lambda 自訂整合建立 "Hello, World!" 之類的 Lambda 函數。在此演練中，此函數稱為 GetStartedLambdaIntegration。

以下是此 GetStartedLambdaIntegration Lambda 函數的 Node.js 實作：

```
'use strict';
var days = ['Sunday', 'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday'];

var times = ['morning', 'afternoon', 'evening', 'night', 'day'];

console.log('Loading function');

exports.handler = function(event, context, callback) {
  // Parse the input for the name, city, time and day property values
  let name = event.name === undefined ? 'you' : event.name;
  let city = event.city === undefined ? 'World' : event.city;
  let time = times.indexOf(event.time)<0 ? 'day' : event.time;
  let day = days.indexOf(event.day)<0 ? null : event.day;

  // Generate a greeting
  let greeting = 'Good ' + time + ', ' + name + ' of ' + city + '. ';
  if (day) greeting += 'Happy ' + day + '!';

  // Log the greeting to CloudWatch
  console.log('Hello: ', greeting);

  // Return a greeting to the caller
  callback(null, {
    "greeting": greeting
  });
};
```

對於 Lambda 自訂整合，API Gateway 會從用戶端傳遞 Lambda 函數的輸入做為整合請求內文。此輸入是 Lambda 函數處理常式的 event 物件。

我們的 Lambda 函數很簡單。它會剖析 event、name、city 與 time 屬性的輸入 day 物件。然後，它會傳回問候語 (以 { "message":greeting} 的 JSON 物件形式) 紿發起人。該訊息的模式為 "Good [morning|afternoon|day], [name|you] in [city|World]. Happy **day**!"。它假設 Lambda 函數的輸入屬於下列 JSON 物件：

```
{  
    "city": "...",  
    "time": "...",  
    "day": "...",  
    "name" : "..."  
}
```

如需詳細資訊，請參閱 [AWS Lambda Developer Guide](#)。

此外，此函數會藉由呼叫 `console.log(...)` 將其執行記錄到 Amazon CloudWatch。這有助於在偵錯函數時追蹤呼叫。若要允許 `GetStartedLambdaIntegration` 函數記錄呼叫，請使用適當的政策來設定 IAM 角色，讓 Lambda 函數可以建立 CloudWatch 串流並將日誌項目新增至串流。Lambda 主控台會引導您建立必要的 IAM 角色與政策。

如果您未使用 API Gateway 主控台設定 API (例如從 OpenAPI 檔案匯入 API 時)，您必須明確建立 (如果需要) 並設定呼叫角色與政策，API Gateway 才能呼叫 Lambda 函數。如需如何為 API Gateway API 設定 Lambda 呼叫與執行角色的詳細資訊，請參閱 [使用 IAM 許可控制 API 的存取](#) (p. 333)。

相較於 `GetStartedLambdaProxyIntegration` (Lambda 代理整合的 Lambda 函數) , Lambda 自訂整合的 `GetStartedLambdaIntegration` Lambda 函數只會接受來自 API Gateway API 整合請求內文的輸入。該函數可以傳回任何 JSON 物件、字串、數值、布林值 . 或甚至是二進位 Blob 的輸出。相較之

下，Lambda 代理整合的 Lambda 函數可以接受來自任何請求資料的輸入，但必須傳回特定 JSON 物件的輸出。Lambda 自訂整合的 GetStartedLambdaIntegration 函數可以接受 API 請求參數做為輸入，但前提是 API Gateway 在將用戶端請求轉送到後端之前，已將必要的 API 請求參數對應到整合請求內文。若要這樣做，API 開發人員必須建立對應範本，並在建立 API 時於 API 方法上設定此範本。

現在，建立 GetStartedLambdaIntegration Lambda 函數。

### 建立 Lambda 自訂整合的 **GetStartedLambdaIntegration** Lambda 函數

1. Open the AWS Lambda console at <https://console.aws.amazon.com/lambda/>.
2. 請執行下列其中一項：
  - 出現歡迎頁面時，請選擇 Get Started Now (立即開始使用)，然後選擇 Create a function (建立函數)。
  - 出現 Lambda > Functions (&LAM; > 函數) 清單頁面時，請選擇 Create a function (建立函數)。
3. 選擇 Author from scratch (從頭開始撰寫)。
4. 在 Author from scratch (從頭開始撰寫) 窗格上，執行下列操作：
  - a. 在 Name (名稱) 欄位中，輸入 **GetStartedLambdaIntegration** 做為 Lambda 函數名稱。
  - b. 對於 Runtime (執行時間)，請選擇 Node.js 8.10。
  - c. 針對 Role (角色)，選擇 Create new role from template(s)。
  - d. 針對 Role name (角色名稱)，輸入您的角色名稱 (例如 **GetStartedLambdaIntegrationRole**)。
  - e. 針對 Policy templates (政策範本)，選擇 Simple Microservice permissions。
  - f. 選擇 Create function (建立函式)。
5. 在 Configure function (設定函數) 窗格的 Function code (函數程式碼) 中，執行下列操作：
  - a. 如果尚未顯示，請選擇 Content entry type (內容項目類型) 下的 Edit code inline (編輯內嵌程式碼)。
  - b. 保留 Handler (處理常式) 設定為 index.handler。
  - c. 將 Runtime (執行時間) 設定為 Node.js 8.10。
  - d. 將本節開頭列出的 Lambda 函數程式碼複製並貼到內嵌程式碼編輯器。
  - e. 對於此區段中的所有其他欄位，則保留預設選項。
  - f. 選擇 Save (儲存)。
6. 若要測試新建立的函數，請從 Select a test event... (選取測試事件...) 選取 Configure test events (設定測試事件)。
  - a. 針對 Create new event (建立新事件)，請以下列內容取代任何預設程式碼陳述式，輸入 HelloWorldTest 做為事件名稱，然後選擇 Create (建立)。

```
{  
    "name": "Jonny",  
    "city": "Seattle",  
    "time": "morning",  
    "day": "Wednesday"  
}
```

- b. 選擇 Test (測試) 以呼叫函數。Execution result: succeeded (執行結果：成功) 區段會隨即顯示。展開 Detail (詳細資訊)，您會看到下列輸出。

```
{  
    "greeting": "Good morning, Jonny of Seattle. Happy Wednesday!"  
}
```

該輸出也會寫入 CloudWatch 日誌。

做為附帶練習，您可以使用 IAM 主控台，來檢視 Lambda 函數建立過程中所建立的 IAM 角色 (GetStartedLambdaIntegrationRole)。此 IAM 角色會連接到兩個內嵌政策。其中一個規定 Lambda 執行的最基本許可。它允許在建立 Lambda 函數的區域中，對您帳戶的任何 CloudWatch 資源呼叫 CloudWatch CreateLogGroup。此政策也允許建立 HelloWorldForLambdaIntegration Lambda 函數的 CloudWatch 串流與記錄日誌事件。

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": "logs:CreateLogGroup",  
            "Resource": "arn:aws:logs:region:account-id:*"  
        },  
        {  
            "Effect": "Allow",  
            "Action": [  
                "logs:CreateLogStream",  
                "logs:PutLogEvents"  
            ],  
            "Resource": [  
                "arn:aws:logs:region:account-id:log-group:/aws/lambda/  
GetStartedLambdaIntegration:*"  
            ]  
        }  
    ]  
}
```

另一個政策文件適用於呼叫此範例中未使用的其他 AWS 服務。您目前可以略過。

IAM 角色與信任實體相關聯，也就是 lambda.amazonaws.com。以下是信任關係：

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

此信任關係與內嵌政策的組合可讓 Lambda 函數呼叫 console.log() 函數來將事件記錄到 CloudWatch Logs。

如果您未使用 AWS Management Console 建立 Lambda 函數，則需要遵循這些範例來建立必要的 IAM 角色與政策，再手動將角色連接到您的函數。

## 建立具有 Lambda 非代理整合的 API

建立並測試 Lambda 函數 (GetStartedLambdaIntegration) 之後，您就可以透過 API Gateway API 公開函數。為了方便說明，我們會公開使用泛型 HTTP 方法的 Lambda 函數。我們使用請求內文、URL 路徑變數、查詢字串與標頭，從用戶端接收必要的輸入資料。我們開啟 API 的 API Gateway 請求驗證程式，確保所有必要的資料都已正確定義與指定。我們設定對應範本，讓 API Gateway 可以將用戶端提供的請求資料轉換成後端 Lambda 函數所需的有效格式。

## 建立透過 Lambda 自訂整合使用 Lambda 函數的 API

1. 啟動 API Gateway 主控台。
2. 選擇 Create new API (建立新的 API)。
  - a. 針對 API name (API 名稱) 輸入 **GetStartedLambdaNonProxyIntegration**。
  - b. 針對 Description (說明) 輸入 API 的說明或保留空白。
  - c. 選擇 Create API (建立 API)。
3. 在 Resources (資源) 下選擇根資源 (/)。從 Actions (動作) 選單中，選擇 Create Resource (建立資源)。
  - a. 針對 Resource Name (資源名稱) 輸入 city。
  - b. 以 {city} 取代 Resource Path (資源路徑)。這是樣板化路徑變數範例，可用來接受來自用戶端的輸入。稍後，我們將示範如何使用對應範本，將此路徑變數對應到 Lambda 函數輸入。
  - c. 選取 Enable API Gateway Cors (啟用 &ABP; Cors) 選項。
  - d. 選擇 Create Resource (建立資源)。
4. 反白選取新建立的 /{city} 資源後，從 Actions (動作) 中，選擇 Create Method (建立方法)。
  - a. 從 HTTP method (HTTP 方法) 下拉式選單中，選擇 ANY。ANY HTTP 動詞是用戶端在執行時間提交之有效 HTTP 方法的預留位置。此範例顯示 ANY 方法可用於 Lambda 自訂整合以及 Lambda 代理整合。
  - b. 若要儲存設定，請選擇核取記號。
5. 在 Method Execution (方法執行) 中，針對 /{city} ANY 方法，執行下列操作：
  - a. 針對 Integration type，選擇 Lambda Function (Lambda 函數)。
  - b. 保持 Use Lambda Proxy integration (使用 Lambda 代理整合) 方塊不勾選。
  - c. 選擇您建立 Lambda 函數的區域，例如 us-west-2。
  - d. 在 Lambda Function (Lambda 函數) 中輸入您的 Lambda 函數名稱，例如 GetStartedLambdaIntegration。
  - e. 將 Use Default timeout (使用預設逾時) 方塊保留勾選。
  - f. 選擇 Save (儲存)。
  - g. 在 Add Permission to Lambda Function (新增許可至 Lambda 函數) 快顯視窗中，選擇 OK (確定)，讓 API Gateway 設定 API 呼叫整合的 Lambda 函數所需的存取許可。
6. 您將於本步驟中設定以下項目：
  - 查詢字串參數 (time)
  - 標頭參數 (day)
  - 承載屬性 (callerName)

在執行時間，用戶端可以使用這些請求參數與請求內文來提供一天中的時間、星期幾與發起人名稱。您已設定 /{city} 路徑變數。

- a. 在 Method Execution (方法執行) 窗格中，選擇 Method Request (方法請求)。
- b. 展開 URL Query String Parameters (URL 查詢字串參數) 區段。選擇 Add query string (新增查詢字串)。針對 Name (名稱) 輸入 time。選取 Required (必要) 選項，然後選擇核取記號圖示以儲存設定。保留 Caching (快取) 空白，以避免此練習的不必要費用。
- c. 展開 HTTP Request Headers (HTTP 請求標頭) 區段。選擇 Add header (新增標頭)。針對 Name (名稱) 輸入 day。選取 Required (必要) 選項，然後選擇核取記號圖示以儲存設定。保留 Caching (快取) 空白，以避免此練習的不必要費用。
- d. 若要定義方法請求承載，請執行下列操作：
  - i. 若要定義模型，請從 API Gateway 主導覽窗格中，選擇 API 下的 Models (模型)，然後選擇 Create (建立)。

- ii. 針對 Model name (模型名稱) 輸入 GetStartedLambdaIntegrationUserInput。
- iii. 針對 Content type (內容類型) 輸入 application/json。
- iv. 將 Model description (模型描述) 保留空白。
- v. 將下列結構描述定義複製到 Model schema (模型結構描述) 編輯器：

```
{  
  "$schema": "http://json-schema.org/draft-04/schema#",  
  "title": "GetStartedLambdaIntegrationInputModel",  
  "type": "object",  
  "properties": {  
    "callerName": { "type": "string" }  
  }  
}
```

- vi. 選擇 Create model (建立模型) 完成定義輸入模型。
  - vii. 選擇 Resources (資源)、選擇 /{city} ANY 方法、選擇 Method Request (方法請求)，然後展開 Request body (請求內文)。選擇 Add model (新增模型)。針對 Content type (內容類型) 輸入 application/json。針對 Model name (模型名稱) 選擇 GetStartedLambdaIntegrationInput。選擇核取記號圖示來儲存設定。
7. 選擇 /{city} ANY 方法並選擇 Integration Request (整合請求) 以設定內文對應範本。此步驟會將之前設定的方法請求參數 nameQuery 或 nameHeader 對應到後端 Lambda 函數所需的 JSON 承載：
- a. 展開 Mapping Templates (對應範本) 區段。選擇 Add mapping template (新增對應範本)。針對 Content-Type 輸入 application/json。選擇核取記號圖示來儲存設定。
  - b. 在顯示的快顯視窗中，選擇 Yes, secure this integration (是，保護此整合)。
  - c. 針對 Request body passthrough (請求內文傳遞) 核取建議的 When there are no templates defined。
  - d. 從 Generate template (產生範本) 中選擇 GetStartedLambdaIntegrationUserInput，以產生初始對應範本。由於您已定義模型結構描述，因此可以使用此選項；若未定義，則需要從頭撰寫對應範本。
  - e. 在對應範本編輯器中使用下列內容取代所產生的對應指令碼：

```
#set($inputRoot = $input.path('$'))  
{  
  "city": "$input.params('city')",  
  "time": "$input.params('time')",  
  "day": " $input.params('day')",  
  "name": "$inputRoot.callerName"  
}
```

- f. 選擇 Save (儲存)。

## 測試呼叫 API 方法

API Gateway 主控台提供測試功能，可讓您測試呼叫 API，再進行部署。您可以使用主控台的 Test (測試) 功能，透過提交下列請求來測試 API：

```
POST /Seattle?time=morning  
day:Wednesday  
  
{  
  "callerName": "John"  
}
```

在此測試請求中，您會將 ANY 設定為 POST、將 {city} 設定為 Seattle、將 Wednesday 設定為 day 標頭值，並將 "John" 指派為 callerName 值。

### 測試呼叫 ANY /{city} 方法

1. 在 Method Execution (方法執行) 中，選擇 Test (測試)。
2. 從 Method (方法) 下拉式清單中，選擇 POST。
3. 在 Path (路徑) 中，輸入 **Seattle**。
4. 在 Query Strings (查詢字串) 中，輸入 **time=morning**。
5. 在 Headers (標頭) 中，輸入 **day=Wednesday**。
6. 在 Request Body (請求內文) 中，輸入 **{ "callerName": "John" }**。
7. 選擇 Test (測試)。
8. 遵循下列方式驗證所傳回的回應承載：

```
{  
    "greeting": "Good morning, John of Seattle. Happy Wednesday!"  
}
```

9. 您也可以檢視日誌，查看 API Gateway 如何處理請求與回應。

```
Execution log for request test-request  
Thu Aug 31 01:07:25 UTC 2017 : Starting execution for request: test-invoke-request  
Thu Aug 31 01:07:25 UTC 2017 : HTTP Method: POST, Resource Path: /Seattle  
Thu Aug 31 01:07:25 UTC 2017 : Method request path: {city=Seattle}  
Thu Aug 31 01:07:25 UTC 2017 : Method request query string: {time=morning}  
Thu Aug 31 01:07:25 UTC 2017 : Method request headers: {day=Wednesday}  
Thu Aug 31 01:07:25 UTC 2017 : Method request body before transformations:  
    { "callerName": "John" }  
Thu Aug 31 01:07:25 UTC 2017 : Request validation succeeded for content type  
    application/json  
Thu Aug 31 01:07:25 UTC 2017 : Endpoint request URI: https://  
lambda.us-west-2.amazonaws.com/2015-03-31/functions/arn:aws:lambda:us-  
west-2:123456789012:function:GetStartedLambdaIntegration/invocations  
Thu Aug 31 01:07:25 UTC 2017 : Endpoint request  
    headers: {x-amzn-lambda-integration-tag=test-request,  
    Authorization=*****  
    X-Amz-Date=20170831T010725Z, x-amzn-apigateway-api-id=beags1mmid, X-Amz-  
    Source-Arn=arn:aws:execute-api:us-west-2:123456789012:beags1mmid/null/POST/  
    {city}, Accept=application/json, User-Agent=AmazonAPIGateway_beags1mmid,  
    X-Amz-Security-Token=FOoDYXdzELL//////////wEaDMHGzEdEOT/VvGhabiK3AzgKrJw  
    +3zLqJZG4PhOq12K6W21+QotY2rrZy0zghLoiuRg3CAYNQ2eqgL5D54+63ey9bIdtwHGoyBdq8ecWxJK/  
    YUnT2RaU0L9HCG5p7FC05h3IvvlfFfcidQNxeYvsKJTLXI05/  
    yEnY3ttIAnpNYLOezD9Es8rBfyruHfJfOqextKlsC8DymCcqlGkig8qLKcZ0hWJWWwiPJiFgL7laabXs  
    ++ZhCa4hdZo4iqLG729DE4gaV1mJVdoAagiUwLMo+y4NxFDu0r710/  
    EO5nYCrppGVVBYiGk7H4T6sXuhTkbNNqVmXtV3ch5bOlh7 [TRUNCATED]  
Thu Aug 31 01:07:25 UTC 2017 : Endpoint request body after transformations: {  
    "city": "Seattle",  
    "time": "morning",  
    "day": "Wednesday",  
    "name" : "John"  
}  
Thu Aug 31 01:07:25 UTC 2017 : Sending request to https://lambda.us-  
west-2.amazonaws.com/2015-03-31/functions/arn:aws:lambda:us-  
west-2:123456789012:function:GetStartedLambdaIntegration/invocations  
Thu Aug 31 01:07:25 UTC 2017 : Received response. Integration latency: 328 ms  
Thu Aug 31 01:07:25 UTC 2017 : Endpoint response body before transformations:  
    {"greeting": "Good morning, John of Seattle. Happy Wednesday!"}  
Thu Aug 31 01:07:25 UTC 2017 : Endpoint response headers: {x-amzn-Remapped-Content-  
Length=0, x-amzn-RequestId=c0475a28-8de8-11e7-8d3f-4183da788f0f, Connection=keep-  
alive, Content-Length=62, Date=Thu, 31 Aug 2017 01:07:25 GMT, X-Amzn-Trace-  
Id=root-1-59a7614d-373151b01b0713127e646635;sampled=0, Content-Type=application/json}  
Thu Aug 31 01:07:25 UTC 2017 : Method response body after transformations:  
    {"greeting": "Good morning, John of Seattle. Happy Wednesday!"}
```

```
Thu Aug 31 01:07:25 UTC 2017 : Method response headers: {X-Amzn-Trace-Id=sampled=0;root=1-59a7614d-373151b01b0713127e646635, Content-Type=application/json}
Thu Aug 31 01:07:25 UTC 2017 : Successfully completed execution
Thu Aug 31 01:07:25 UTC 2017 : Method completed with status: 200
```

日誌會顯示對應前的傳入請求，以及對應後的整合請求。測試失敗時，日誌可用於評估原始輸入是否正確或對應範本是否正常運作。

## 部署 API

測試呼叫是一種模擬並有所限制。例如，它會略過 API 上所制定的任何授權機制。若要即時測試 API 執行，您必須先部署 API。若要部署 API，請建立一個階段來建立 API 在當時的快照。階段名稱也會定義 API 預設主機名稱後面的基底路徑。API 的根資源會附加在階段名稱後面。當您修改 API 時，您必須將它重新部署到新的或現有的階段，變更才會生效。

### 將 API 部署到階段

- 從 APIs 窗格中選擇 API，或從 Resources (資源) 窗格中選擇資源或方法。從 Actions (動作) 下拉式選單中，選擇 Deploy API (部署 API)。
- 針對 Deployment stage (部署階段)，選擇 New Stage (新增階段)。
- 針對 Stage name (階段名稱)，輸入名稱，例如 **test**。

#### Note

輸入必須為 UTF-8 編碼 (例如未本地化的) 文字。

- 針對 Stage description (階段說明)，輸入說明或保留空白。
- 針對 Deployment description (部署描述)，輸入說明或保留空白。
- 選擇 Deploy (部署)。成功部署 API 之後，您會看到 API 的基底 URL (預設主機名稱加上階段名稱) 顯示為 Stage Editor (階段編輯器) 頂端的 Invoke URL (呼叫 URL)。此基底 URL 的一般模式為 <https://api-id.region.amazonaws.com/stageName>。例如，在 `beags1mnid` 區域中建立並部署到 `us-west-2` 階段之 API (`test`) 的基底 URL 為 <https://beags1mnid.execute-api.us-west-2.amazonaws.com/test>。

## 在部署階段測試 API

您有數種方式可以測試已部署的 API。針對僅使用 URL 路徑變數或查詢字串參數的 GET 請求，您可以在瀏覽器中輸入 API 資源 URL。針對其他方法，您必須使用更進階的 REST API 測試公用程式，例如 [POSTMAN](#) 或 [cURL](#)。

### 使用 cURL 測試 API

- 在連線至網際網路的本機電腦上，開啟終端機視窗。
- 若要測試 POST /Seattle?time=evening：

複製下列 cURL 命令並將其貼入終端機視窗。

```
curl -v -X POST \
  'https://beags1mnid.execute-api.us-west-2.amazonaws.com/test/Seattle?time=evening' \
  -H 'content-type: application/json' \
  -H 'day: Thursday' \
  -H 'x-amz-docs-region: us-west-2' \
  -d '{
    "callerName": "John"
}'
```

您應該取得具有下列承載的成功回應：

```
{"greeting": "Good evening, John of Seattle. Happy Thursday!"}
```

如果您在此方法請求中將 POST 變更為 PUT，您會取得相同的回應。

3. 若要測試 GET /Boston?time=morning：

複製下列 cURL 命令並將其貼入終端機視窗。

```
curl -X GET \
  'https://beags1mnid.execute-api.us-west-2.amazonaws.com/test/Boston?time=morning' \
  -H 'Content-Type: application/json' \
  -H 'Day: Thursday' \
  -H 'x-amz-docs-region: us-west-2' \
  -d '{
    "callerName": "John"
}'
```

您會取得 400 Bad Request 回應與下列錯誤訊息：

```
{"message": "Invalid request body"}
```

這是因為您提交的 GET 請求無法接受承載，因此請求驗證失敗。

## 清除

若您不再需要因為此演練所建立的 Lambda 函數，可立即將其刪除。您也可以刪除隨附的 IAM 資源。

### Warning

如果您打算完成本系列中的其他演練，則不要刪除 Lambda 執行角色或 Lambda 呼叫角色。如果刪除您 API 相依的 Lambda 函數，這些 API 將無法再運作。而刪除 Lambda 函數無法復原。所以若要再次使用該 Lambda 函數，必須加以重新建立。

如果刪除 Lambda 函數相依的 IAM 資源，Lambda 函數將無法再運作，而且所有相依於該函數的 API 將無法再運作。而刪除 IAM 資源無法復原。所以若要再次使用該 IAM 資源，必須加以重新建立。

### 刪除 Lambda 函數

1. Sign in to the AWS Management Console and open the AWS Lambda console at <https://console.aws.amazon.com/lambda/>.
2. 從函數清單中，依序選擇 GetHelloWorld、Actions (動作) 與 Delete function (刪除函數)。出現提示時，再次選擇 Delete (刪除)。
3. 從函數清單中，依序選擇 GetHelloWithName、Actions (動作) 與 Delete function (刪除函數)。出現提示時，再次選擇 Delete (刪除)。

### 刪除相關聯的 IAM 資源

1. Open the IAM console at <https://console.aws.amazon.com/iam/>.
2. 從 Details (詳細資訊) 中，選擇 Roles (角色)。
3. 從角色清單中，依序選擇 APIGatewayLambdaExecRole、Role Actions (角色動作) 與 Delete Role (刪除角色)。出現提示時，選擇 Yes, Delete (是，刪除)。
4. 從 Details (詳細資訊) 中，選擇 Policies (政策)。

5. 從政策清單中，依序選擇 APIGatewayLambdaExecPolicy、Policy Actions (政策動作) 與 Delete (刪除)。出現提示時，選擇 Delete (刪除)。

這份演練到此結束。

## 教學課程：匯入範例來建立 REST API

您可以使用 Amazon API Gateway 主控台，利用 HTTP 整合，針對 PetStore 網站建立和測試簡單的 REST API。API 定義會預先設定為 OpenAPI 2.0 檔案。將 API 定義載入至 API Gateway 之後，您可以使用 API Gateway 主控台來檢查 API 的基本結構，或僅部署和測試 API。

PetStore 範例 API 支援下列方法，讓用戶端存取 `http://petstore-demo-endpoint.execute-api.com/petstore/pets` 的 HTTP 後端網站。

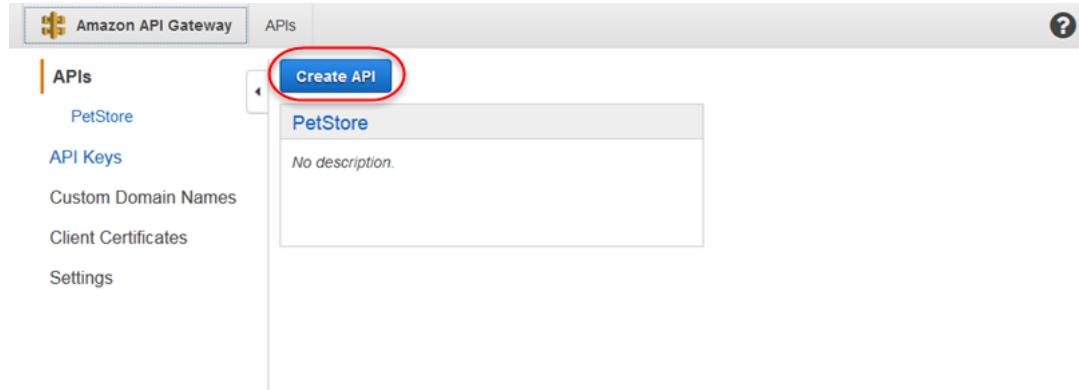
- GET /：用於讀取存取未與任何後端端點整合的 API 根資源。API Gateway 會以 PetStore 網站的概觀來做回應。這是 MOCK 整合類型範例。
- GET /pets：適用於與相同具名後端 /pets 資源整合之 API /pets 資源的讀取存取。後端會傳回 PetStore 中可用寵物的頁面。這是 HTTP 整合類型範例。整合端點的 URL 是 `http://petstore-demo-endpoint.execute-api.com/petstore/pets`。
- POST /pets：適用於與後端 /pets 資源整合之 API /petstore/pets 資源的寫入存取。在收到正確的請求時，後端會將指定的寵物新增至 PetStore，並將結果傳回給發起人。此整合也是 HTTP。
- GET /pets/{petId}：適用於指定為傳入請求 URL 之路徑變數的 `petId` 值所識別寵物的讀取存取。這個方法也有 HTTP 整合類型。後端會傳回 PetStore 中找到的指定寵物。後端 HTTP 端點的 URL 是 `http://petstore-demo-endpoint.execute-api.com/petstore/pets/n`，其中 n 是作為所查詢寵物識別符的整數。

此 API 支援透過 MOCK 整合類型之 OPTIONS 方法的 CORS 存取。API Gateway 會傳回支援 CORS 存取的所需標頭。

下列程序會逐步解說如何使用 API Gateway 主控台從範例建立和測試 API。

### 匯入、建置和測試範例 API

1. 完成[the section called “先決條件：準備在 API Gateway 中建置 API” \(p. 9\)](#) 中的步驟 (如果您尚未這麼做)。
2. 登入位於 <https://console.aws.amazon.com/apigateway> 的 API Gateway 主控台。
3. 請執行下列其中一項：
  - a. 如果這是您帳戶中的第一個 API，請從 API Gateway 主控台歡迎使用頁面中選擇 Get Started (開始使用)。  
如果出現提示，請選擇 OK (確定) 予以關閉並繼續。
  - b. 如果這不是您的第一個 API，請從 API Gateway API 首頁中選擇 Create API (建立 API)：



4. 在 Create new API (建立新的 API) 下，選擇 Example API (範例 API)，然後選擇 Import (匯入) 來建立範例 API。針對您的第一個 API，API Gateway 主控台預設會從這個選項開始。

#### Create new API

In Amazon API Gateway, an API refers to a collection of resources and methods that can be invoked through HTTPS endpoints.

New API    Clone from existing API    Import from Swagger    Example API

#### Example API

Learn about the service by importing an example API and turning on hints throughout the console.

```
1 {  
2   "swagger": "2.0",  
3   "info": {  
4     "description": "Your first API with Amazon API Gateway. This is a sample API that integrates via HTTP with our demo Pet Store endpoints",  
5     "title": "PetStore"  
6   },  
7   "schemes": [  
8     "https"  
9   ],  
10  "paths": {  
11    "/": {  
12      "get": {  
13        "tags": [  
14          "pets"  
15        ],  
16        "description": "PetStore HTML web page containing API usage information",  
17        "consumes": [  
18          "application/json"  
19        ]  
20      }  
21    }  
22  }  
23}  
24
```

Fail on warnings    Ignore warnings

Import

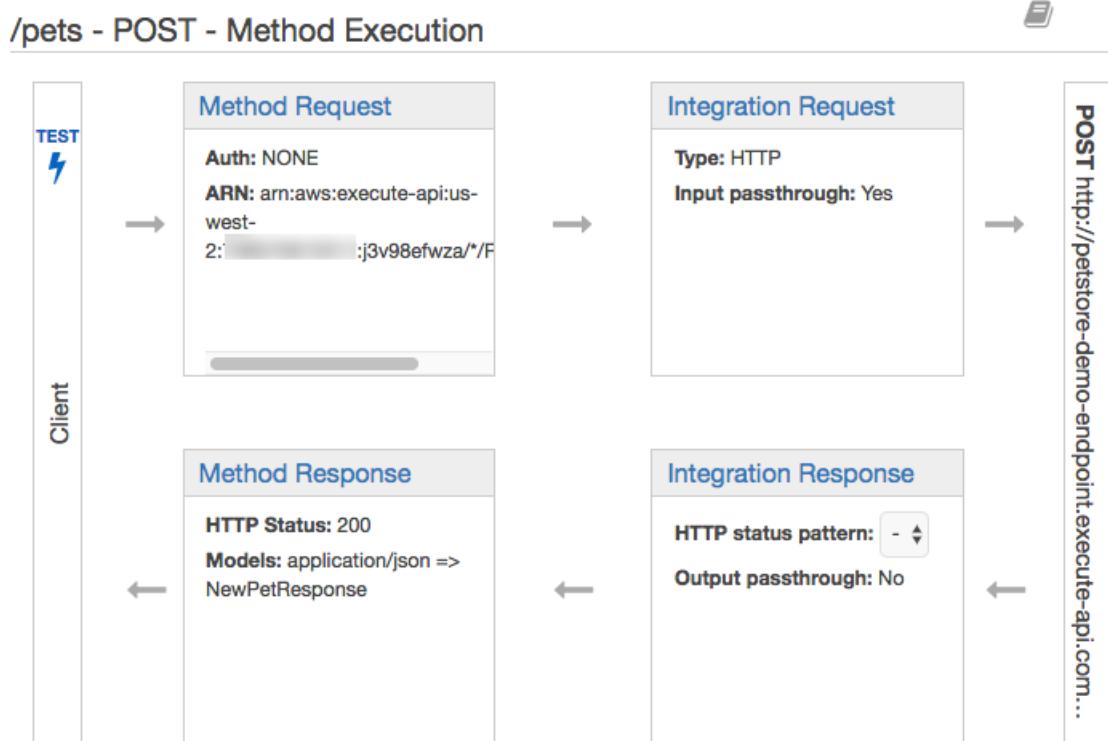
您可以先捲動 OpenAPI 定義以取得此範例 API 的詳細資訊，再選擇 Import (匯入)。

5. 新建立的 API 如下所示：

The screenshot shows the API Gateway console with the newly created 'PetStore' API. On the left, under 'Resources', there's a tree view showing the structure: '/' (GET), '/pets' (GET, OPTIONS, POST), and '/{petId}' (GET, OPTIONS). On the right, under '/ Methods', there's a detailed view for the '/' endpoint. It shows a 'GET' method with a blue header bar. Below the header, it says 'Mock Endpoint'. Under 'Authorization', it shows 'None'. Under 'API Key', it says 'Not required'.

Resources (資源) 窗格會將已建立 API 的結構顯示為節點樹狀結構。每個資源上所定義的 API 方法就是樹狀結構的邊緣。選取資源時，其所有方法都會列在右側的 Methods (方法) 窗格中。每個方法的下方都會顯示方法的簡短摘要，包含其端點 URL、授權類型和 API 金鑰需求。

6. 若要檢視方法的詳細資訊、修改其設定或測試方法呼叫，請從方法清單或資源樹狀結構中選擇方法名稱。在這裡，我們選擇 POST /pets 方法作為插圖：



產生的 Method Execution (方法執行) 窗格會呈現所選擇 (POST /pets) 方法的結構和行為之邏輯檢視：Method Request (方法請求) 和 Method Response (方法回應) 是 API 與 API 前端 (用戶端) 的界面，而 Integration Request (整合請求) 和 Integration Response (整合回應) 是 API 與後端 (<http://petstore-demo-endpoint.execute-api.com/petstore/pets>) 的界面。用戶端會使用 API 透過 Method Request (方法請求) 來存取後端功能。API Gateway 會在必要時先將用戶端請求翻譯為 Integration Request (整合請求) 中後端可接受的形式，再將傳入請求轉送至後端。轉換後的請求稱為整合請求。同樣地，後端會傳回 Integration Response (整合回應) 中的 API Gateway 回應。API Gateway 接著會先將它路由至 Method Response (方法回應)，再將它傳送至用戶端。同樣地，若有必要，API Gateway 可以將後端回應資料映射至用戶端所預期的形式。

針對 API 資源上的 POST 方法，如果方法請求承載的格式與整合請求承載的格式相同，則可以將方法請求承載傳遞至整合請求，而不需要修改。

GET / 方法請求會使用 MOCK 整合類型，而且未繫結至任何實際後端端點。對應的 Integration Response (整合回應) 設定為傳回靜態 HTML 頁面。呼叫方法時，API Gateway 只需要接受請求，並立即透過 Method Response (方法回應) 將已設定的整合回應傳回給用戶端。您可以使用模擬整合來測試 API，而不需要後端端點。您也可以使用它來服務從回應內文映射範本產生的本機回應。

身為 API 開發人員，您可以設定方法請求和方法回應來控制 API 前端互動的行為。您可以設定整合請求和整合回應來控制 API 後端互動的行為。這些涉及方法與其對應整合之間的資料映射。我們涵蓋「[教學：建置具有 HTTP 非代理整合的 API \(p. 52\)](#)」中的方法設定。現在，我們專注於測試 API，以提供端對端使用者體驗。

7. 選擇 Client (用戶端) 上顯示的 Test (測試) (如上一張圖片所示) 來開始測試。例如，若要測試 POST /pets 方法，請先將下列 {"type": "dog", "price": 249.99} 承載輸入至 Request Body (請求內容)，再選擇 Test (測試) 按鈕。

[← Method Execution](#) /pets - POST - Method Test

Make a test call to your method with the provided input

### Path

No path parameters exist for this resource. You can define path parameters by using the syntax {myPathParam} in a resource path.

### Query Strings

No query string parameters exist for this method. You can add them via Method Request.

### Headers

No header parameters exist for this method. You can add them via Method Request.

### Stage Variables

No stage variables exist for this method.

### Client Certificate

No client certificates have been generated.

### Request Body

```
1 {"type": "dog", "price": 249.99}
```

輸入指定我們想要新增至 PetStore 網站上寵物清單的寵物屬性。

8. 結果顯示如下：

Request: /pets

Status: 200

Latency: 566 ms

Response Body

```
{  
  "pet": {  
    "type": "dog",  
    "price": 249.99  
  },  
  "message": "success"  
}
```

Response Headers

```
{"Access-Control-Allow-Origin":"*","X-Amzn-Trace-Id":"Root=1-59287e14-bd5f1d07c673367be0739eae","Content-Type":"application/json"}
```

Logs

```
Execution log for request test-request  
Fri May 26 19:12:20 UTC 2017 : Starting execution for request: test-invoke-request  
Fri May 26 19:12:20 UTC 2017 : HTTP Method: POST, Resource Path: /pets  
Fri May 26 19:12:20 UTC 2017 : Method request path: {}  
Fri May 26 19:12:20 UTC 2017 : Method request query string: {}  
Fri May 26 19:12:20 UTC 2017 : Method request headers: {}  
Fri May 26 19:12:20 UTC 2017 : Method request body before transformations: {"type": "dog", "price": 249.99}  
Fri May 26 19:12:20 UTC 2017 : Endpoint request URI: http://petstore-demo-endpoint.execute-api.com/petstore/pets  
Fri May 26 19:12:20 UTC 2017 : Endpoint request headers: {x-amzn-apigateway-api-id=4wk1k4onj3, Accept=application/json, User-Agent=AmazonAPIGateway_4wk1k4onj3, X-Amzn-Trace-Id=Root=1-59287e14-bd5f1d07c673367be0739eae, Content-Type=application/json}  
Fri May 26 19:12:20 UTC 2017 : Endpoint request body after transformations: {"type": "dog", "price": 249.99}  
Fri May 26 19:12:20 UTC 2017 : Endpoint response body before transformations:  
  {  
    "pet": {  
      "type": "dog",  
      "price": 249.99  
    },  
    "message": "success"  
  }  
Fri May 26 19:12:20 UTC 2017 : Endpoint response headers: {Connection=keep-alive, Content-Length=81, Date=Fri, 26 May 2017 19:12:20 GMT, Content-Type=application/json; charset=utf-8, X-Powered-By=Express}  
Fri May 26 19:12:20 UTC 2017 : Method response body after transformations:  
  {  
    "pet": {  
      "type": "dog",  
      "price": 249.99  
    },  
    "message": "success"  
  }  
Fri May 26 19:12:20 UTC 2017 : Method response headers: {Access-Control-Allow-Origin=*, X-Amzn-Trace-Id=Root=1-59287e14-bd5f1d07c673367be0739eae, Content-Type=application/json}  
Fri May 26 19:12:20 UTC 2017 : Successfully completed execution  
Fri May 26 19:12:20 UTC 2017 : Method completed with status: 200
```

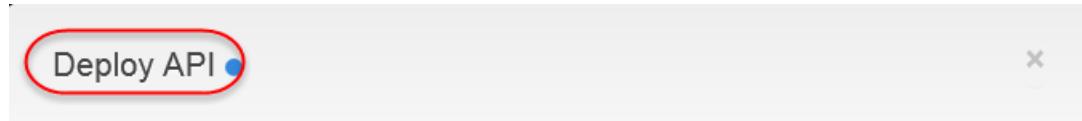
輸出的 Logs (日誌) 項目會顯示從方法請求到整合請求，以及從整合回應到方法回應的狀態變更。這適用於故障診斷任何會導致請求失敗的映射錯誤問題。在這個範例中，不會套用任何映射：方法請求承載會透過整合請求傳遞至後端；且同樣地，後端回應會透過整合回應傳遞至方法回應。

若要使用 API Gateway 測試呼叫請求功能以外的用戶端來測試 API，您必須先將 API 部署至階段。

9. 若要部署範例 API，請選取 PetStore API，然後從 Actions (動作) 選單中選擇 Deploy API (部署 API)。

The screenshot shows the AWS Lambda console interface. On the left, there's a tree view of resources: a root folder with a GET method, a '/pets' folder with a GET method, and a '/{petId}' folder with a GET, OPTIONS, POST, and PUT method. On the right, under 'Actions - / Methods', there are two sections: 'RESOURCE ACTIONS' and 'API ACTIONS'. The 'API ACTIONS' section contains 'Create Method', 'Create Resource', 'Enable CORS', 'Edit Resource Documentation', 'Deploy API', 'Import API', 'Edit API Documentation', and 'Delete API'. The 'Deploy API' button is circled in red.

在 Deploy API (部署 API) 中，針對 Deployment stage (部署階段)，選擇 [New Stage] ([新增階段])，因為這是第一次部署 API。在 Stage name (階段名稱) 中輸入名稱 (例如，test)，並選擇性地在 Stage description (階段說明) 和 Deployment description (部署說明) 中輸入描述。選擇 Deploy (部署)。



Choose a stage where your API will be deployed. For example, a test version of your API could be deployed to a stage named beta.

A modal dialog box titled 'Stage Editor'. It contains four input fields: 'Deployment stage' (set to '[New Stage]'), 'Stage name\*' (set to 'test'), 'Stage description' (set to 'test stage'), and 'Deployment description' (set to 'sample API first deployment'). At the bottom right are 'Cancel' and 'Deploy' buttons, with 'Deploy' circled in red.

在產生的 Stage Editor (階段編輯器) 窗格中，Invoke URL (呼叫 URL) 會顯示呼叫 API GET / 方法請求的 URL。

10. 在 Stage Editor (階段編輯器) 上，遵循 Invoke URL (呼叫 URL) 連結，以在瀏覽器中提交 GET / 方法請求。成功回應會傳回整合回應中從映射範本產生的結果。

11. 在 Stages (階段) 導覽窗格中，展開 test (測試) 階段，並選取 /pets/{petId} 上的 GET，然後複製 Invoke URL (呼叫 URL) 值 <https://api-id.execute-api.region.amazonaws.com/test/pets/{petId}>。{petId} 代表路徑變數。

將 Invoke URL (呼叫 URL) 值 (在前一個步驟中取得) 貼入瀏覽器的網址列 (例如，將 {petId} 取代為 1)，以及按 Enter 來提交請求。200 OK 回應傳回時應該具有下列 JSON 承載：

```
{  
    "id": 1,  
    "type": "dog",  
    "price": 249.99  
}
```

如所示呼叫 API 方法可能是因為其 Authorization (授權) 類型設定為 NONE。如果使用 AWS\_IAM 授權，請使用 Signature 第 4 版 (SigV4) 協定來簽署請求。如需這類請求的範例，請參閱[the section called “教學：建置具有 HTTP 非代理整合的 API” \(p. 52\)](#)。

## 建置具有 HTTP 整合的 API Gateway API

若要建置具有 HTTP 整合的 API，您可以使用 HTTP 代理整合或 HTTP 自訂整合。建議您盡可能使用 HTTP 代理整合，以簡化 API 設定，同時提供多樣化且強大的功能。如果需要為後端轉換用戶端請求資料，或為用戶端轉戶後端回應資料，則必須使用 HTTP 自訂整合。

### 主題

- [教學：建置具有 HTTP 代理整合的 API \(p. 47\)](#)
- [教學：建置具有 HTTP 非代理整合的 API \(p. 52\)](#)

## 教學：建置具有 HTTP 代理整合的 API

HTTP 代理整合是建置 API 的簡單、功能強大且多樣化的機制，可讓 Web 應用程式存取整合式 HTTP 端點 (例如整個網站) 的多個資源或功能，以簡化單一 API 方法的設定。在 HTTP 代理整合中，API Gateway 會將用戶端提交的方法請求傳遞至後端。傳遞的請求資料包含請求標頭、查詢字串參數、URL 路徑變數和承載。後端 HTTP 端點或 Web 伺服器會剖析傳入請求資料，以判斷其所傳回的回應。在設定 API 方法之後，HTTP 代理整合可讓用戶端和後端直接互動，而不需要 API Gateway 介入，除了[the section called “重要說明” \(p. 630\)](#)所列的一些已知問題，例如不支援的字元。

使用全能代理資源 {proxy+}，以及 HTTP 方法的 catch-all ANY 動詞，您可以使用 HTTP 代理整合來建立單一 API 方法的 API。此方法會公開網站的整個可公開存取的 HTTP 資源和操作集。後端 Web 伺服器開啟更多資源來進行公開存取時，用戶端可以搭配使用這些新的資源與相同的 API 設定。若要啟用此功能，網站開發人員必須與用戶端開發人員清楚地溝通新資源以及每個新資源適用的操作。

下列教學是快速簡介，可示範 HTTP 代理整合。在教學中，我們使用 API Gateway 主控台建立 API 以透過一般代理資源 {proxy+} 與 PetStore 網站整合，並建立 HTTP 方法預留位置 ANY。

### 主題

- [使用 API Gateway 主控台建立具有 HTTP 代理整合的 API \(p. 47\)](#)
- [測試具有 HTTP 代理整合的 API \(p. 49\)](#)

## 使用 API Gateway 主控台建立具有 HTTP 代理整合的 API

下列程序會逐步解說如何使用 API Gateway 主控台，建立和測試具有 HTTP 後端之代理資源的 API。HTTP 後端是[教學：建置具有 HTTP 非代理整合的 API \(p. 52\)](#)中的 PetStore 網站 (<http://petstore->

demo-endpoint.execute-api.com/petstore/pets)；其中，螢幕擷取畫面是用來顯示提醒，以說明 API Gateway UI 元素。如果您是第一次使用 API Gateway 主控台來建立 API，則建議您先遵循該節。

#### 透過代理資源建置具有 HTTP 代理與 PetStore 網站之整合的 API

1. 登入位於 <https://console.aws.amazon.com/apigateway> 的 API Gateway 主控台。
2. 若要建立 API，請選取 Create new API (建立新的 API) (用於建立第一個 API) 或 Create API (建立 API) (用於建立任何後續的 API)。接下來，請執行下列項目：
  - a. 選擇 New API (新增 API)。
  - b. 在 API Name (API 名稱) 中輸入名稱。
  - c. (選擇性) 在 Description (說明) 中輸入簡短說明。
  - d. 選擇 Create API (建立 API)。

在本教學中，使用 ProxyResourceForPetStore 做為 API 名稱。

3. 若要建立子資源，請在 Resources (資源) 樹狀結構中選取父資源項目，然後從 Actions (動作) 下拉式選單中選擇 Create Resource (建立資源)。然後，在 New Child Resource (新增子資源) 窗格中，執行下列項目。
  - a. 選擇 Configure as proxy resource (設定為代理資源) 選項以建立代理資源。否則，請維持未選取的狀態。
  - b. 在 Resource Name\* (資源名稱\*) 輸入欄位中輸入名稱。
  - c. 在 Resource Path\* (資源路徑\*) 輸入欄位中輸入新的或預設的名稱。
  - d. 選擇 Create Resource (建立資源)。
  - e. 如有需要，選擇 Enable API Gateway CORS (啟用 API Gateway CORS)。

在本教學中，選取 Configure as proxy resource (設定為代理資源)。針對 Resource Name (資源名稱)，使用預設值：proxy。針對 Resource Path (資源路徑)，使用 /{proxy+}。選取 Enable API Gateway CORS (啟用 &ABP; CORS)。

The screenshot shows the 'New Child Resource' creation dialog in the AWS API Gateway console. The URL in the address bar is > ProxyResourceForPetStore (miqyuu3lfg) > Resources > / (7ryftl47g0) > Create. The left sidebar shows 'Resources' and the current path '/'. The main area has a title 'New Child Resource' and a sub-instruction 'Use this page to create a new child resource for your resource.' It contains two sections: 'Configure as proxy resource' (checkbox checked) and 'Resource Name\*' (input field 'proxy'). Below it is another section for 'Resource Path\*' with the value '/ {proxy+}'. A note explains path parameters: 'You can add path parameters using brackets. For example, the resource path {username} represents a path parameter called 'username'. Configuring /{proxy+} as a proxy resource catches all requests to its sub-resources. For example, it works for a GET request to /foo. To handle requests to /, add a new ANY method on the / resource.' At the bottom, there's a 'Enable API Gateway CORS' checkbox (checked), a note '\* Required', and buttons for 'Cancel' and 'Create Resource'.

4. 若要設定 ANY 方法以整合 HTTP 後端，請執行以下項目：
  - a. 選擇剛建立的資源，然後從 Actions (動作) 下拉式選單中選取 Create Method (建立方法)。
  - b. 從 HTTP 方法下拉式清單中選擇 ANY，然後選擇核取記號圖示來儲存您的選擇。

- c. 針對 Integration type (整合類型) 選擇 HTTP Proxy。
- d. 在 Endpoint URL (端點 URL) 中輸入 HTTP 後端資源 URL。
- e. 其他欄位請使用預設設定。
- f. 選擇 Save (儲存) 以完成設定 ANY 方法。

在本教學中，使用 `http://petstore-demo-endpoint.execute-api.com/{proxy}` 做為 Endpoint URL (端點 URL)。

### /{{proxy+}} - ANY - Setup

API Gateway will configure your ANY method as a proxy integration. Proxy integrations can communicate with HTTP endpoints or Lambda functions. API Gateway sends the entire request to HTTP endpoints, including resource path, headers, query string parameters, and body. For Lambda integrations, API Gateway applies a default mapping to send all of the request information and responses follow a default interface. To learn more read our [documentation](#)

The screenshot shows the configuration for an ANY method. The 'Integration type' is set to 'HTTP Proxy'. The 'Endpoint URL' field contains 'emo-endpoint.execute-api.com/{{proxy}}'. The 'Content Handling' dropdown is set to 'Passthrough'. A blue 'Save' button is visible at the bottom right.

在剛建立的 API 中，`{proxy+}` 的 API 代理資源路徑會成為 `http://petstore-demo-endpoint.execute-api.com/` 下方之任何後端端點的預留位置。例如，它可以是 `petstore`、`petstore/pets` 和 `petstore/pets/{petId}`。在執行時間，ANY 方法用作任何支援 HTTP 動詞的預留位置。

## 測試具有 HTTP 代理整合的 API

特定用戶端請求成功與否取決於下列項目：

- 如果後端已讓對應的後端端點可供使用，因此已授予所需的存取許可。
- 如果用戶端提供正確的輸入。

例如，這裡使用的 PetStore API 未公開 `/petstore` 資源。因此，您會取得包含錯誤訊息 `404 Resource Not Found` 的 `Cannot GET /petstore` 回應。

此外，用戶端必須能夠處理後端的輸出格式，才能正確地剖析結果。API Gateway 不會居中來促進用戶端與後端之間的互動。

### 透過代理資源使用 HTTP 代理整合來測試與 PetStore 網站整合的 API

1. 若要使用 API Gateway 主控台來測試呼叫 API，請執行以下項目。
  - a. 在 Resources (資源) 樹狀結構中，選擇 代理資源 上的 ANY (任何)。
  - b. 在 Method Execution (方法執行) 窗格中，選擇 Test (測試)。
  - c. 從 Method (方法) 下拉式清單中，選擇後端支援的 HTTP 動詞。
  - d. 在 Path (路徑) 之下，輸入支援所選擇操作的 代理資源 的特定路徑。

- e. 如有需要，在 Query Strings (查詢字串) 標題之下，針對所選擇的操作，輸入支援的查詢表達式。
- f. 如有需要，在 Headers (標頭) 標題之下，針對所選擇的操作，輸入一或多個支援的標頭表達式。
- g. 如已設定，請在 Stage Variables (階段變數) 標題之下，針對所選擇的操作，設定必要的階段變數值。
- h. 如果系統提示或有需要，請在 Client Certificate (用戶端憑證) 標題之下選擇 API Gateway 產生的憑證，以便後端驗證該項操作。
- i. 如果系統提示，請輸入 Request Body (請求內文) 標題之下的文字編輯器中，輸入適當的請求內文。
- j. 選擇 Test (測試) 以測試呼叫方法。

在本教學中，使用 GET 做為 Method (方法) 來取代 ANY、使用 petstore/pets 做為 Path (路徑) 來取代代理資源路徑 ({proxy})，並使用 type=fish 做為 Query Strings (查詢字串)。

[Method Execution](#) /{proxy+} - ANY - Method Test

Make a test call to your method with the provided input

Method

GET

Path

{proxy}

petstore/pets

Query Strings

{proxy}

type=fish

Headers

{proxy}

Use a colon (:) to separate header name and value, and new lines to declare multiple headers. e.g.  
Accept:application/json.

Stage Variables

No [stage variables](#) exist for this method.

Client Certificate

No client certificates have been generated.

Request Body

Request Body is not supported for GET methods.

因為後端網站支援 GET /petstore/pets?type=fish 請求，所以它會傳回與下列類似的成功回應：

```
[  
  {  
    "id": 1,  
    "type": "fish",  
    "price": 249.99  
  },  
  {  
    "id": 2,  
    "type": "fish",  
    "price": 124.99  
  },  
  {  
    "id": 3,  
    "type": "fish",  
    "price": 0.99  
  }  
]
```

如果您嘗試呼叫 GET /petstore，則會取得具有錯誤訊息 404 的 Cannot GET /petstore 回應。原因是後端不支援指定的操作。如果您呼叫 GET /petstore/pets/1，則會取得具有下列承載的 200 OK 回應，因為 PetStore 網站支援請求。

```
{  
  "id": 1,  
  "type": "dog",  
  "price": 249.99  
}
```

2. 若要使用瀏覽器針對 API 的特定資源呼叫 GET 方法，請執行以下操作。
  - a. 如果您尚未這樣做，請從您建立的 API 的 Actions (動作) 下拉式選單中，選擇 Deploy API (部署 API)。依照說明，將 API 部署至特定階段。請記下最後顯示於 Stage Editor (階段編輯器) 上的 Invoke URL (呼叫 URL)。這是 API 的基底 URL。
  - b. 若要針對特定資源提交 GET 請求，請在上個步驟取得的 Invoke URL (呼叫 URL) 值之中附加資源路徑 (包括可能的查詢字串表達式)，並將完整的 URL 複製至瀏覽器的網址列，然後選擇 Enter。

在本教學中，將 API 部署至 test 階段，並將 petstore/pets?type=fish 附加至 API 的呼叫 URL。這會產生 URL <https://4z9giyi2c1.execute-api.us-west-2.amazonaws.com/test/petstore/pets?type=fish>。

結果應該與從 API Gateway 主控台使用 TestInvoke 時所傳回的結果相同。

## 教學：建置具有 HTTP 非代理整合的 API

在本教學中，您會使用 Amazon API Gateway 主控台從頭開始建立 API。您可以將主控台視為 API 設計工作室，並使用它來限定 API 功能範圍、測試其行為、建立 API，以及分階段部署您的 API。

### 主題

- [建立具有 HTTP 自訂整合的 API \(p. 53\)](#)
- [對應 API Gateway API 的請求參數 \(p. 61\)](#)
- [對應回應承載 \(p. 69\)](#)

## 建立具有 HTTP 自訂整合的 API

本節將逐步引導您建立資源、在資源上公開方法、設定方法來達到所需的 API 行為，以及測試與部署 API。

1. 登入位於 <https://console.aws.amazon.com/apigateway> 的 API Gateway 主控台。
2. 若要建立 API，請選取 Create new API (建立新的 API) (用於建立第一個 API) 或 Create API (建立 API) (用於建立任何後續的 API)。接下來，請執行下列項目：
  - a. 選擇 New API (新增 API)。
  - b. 在 API Name (API 名稱) 中輸入名稱。
  - c. (選擇性) 在 Description (說明) 中輸入簡短說明。
  - d. 選擇 Create API (建立 API)。

結果會建立一個空的 API。Resources (資源) 樹狀目錄顯示不含任何方法的根資源 (/)。在此練習中，我們將建立具有 PetStore 網站 (<http://petstore-demo-endpoint.execute-api.com/petstore/pets>) 之 HTTP 自訂整合的 API 為了方便說明，我們將建立 /pets 資源作為根目錄的子目錄，並在此資源上公開 GET 方法，讓用戶端可以從 PetStore 網站擷取可用的 Pets (寵物) 項目清單。

3. 若要建立 /pets 資源，請選取根目錄，然後依序選擇 Actions (動作) 與 Create Resource (建立資源)。

在 Resource Name (資源名稱) 中輸入 Pets，保留 Resource Path (資源路徑) 的指定值，選擇 Enable API Gateway CORS (啟用 API Gateway CORS)，再選擇 Create Resource (建立資源)。

### New Child Resource

Use this page to create a new child resource for your resource.

Configure as proxy resource 

**Resource Name\***

Pets 

**Resource Path\***

/ pets 

You can add path parameters using brackets. For example, the resource path `{username}` represents a path parameter called 'username'. Configuring `/{proxy+}` as a proxy resource catches all requests to its sub-resources. For example, it works for a GET request to `/foo`. To handle requests to `/`, add a new ANY method on the `/` resource.

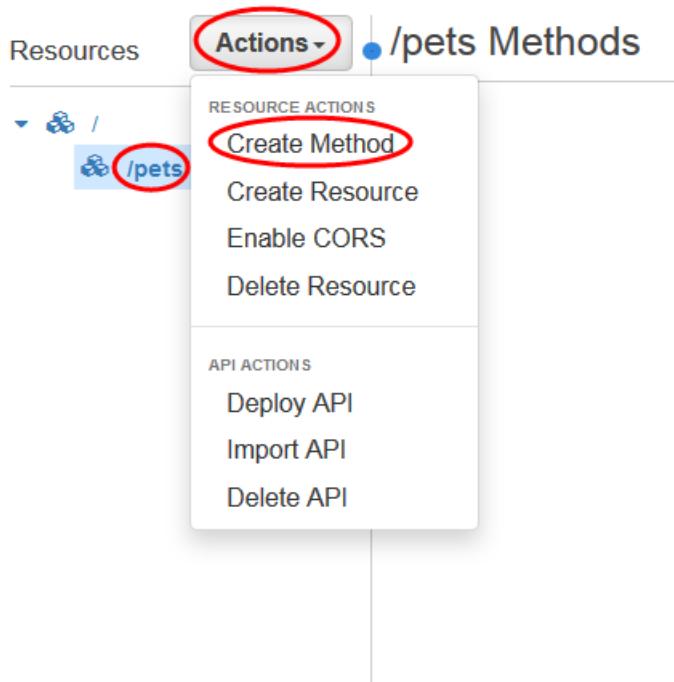
**Enable API Gateway CORS**  

\* Required

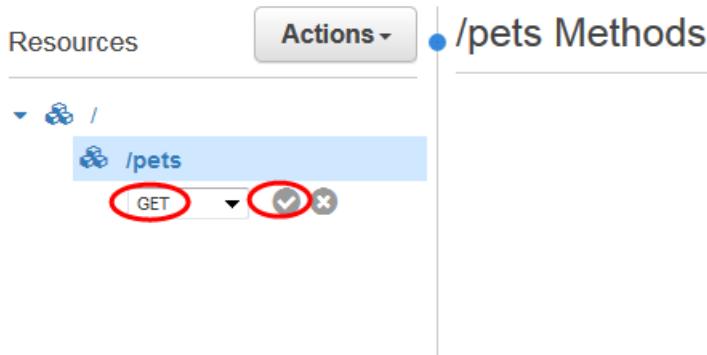
Cancel

Create Resource 

4. 若要在 /pets 資源上公開 GET 方法，請依序選擇 Actions (動作) 與 Create Method (建立方法)。



從 /pets 資源節點下的清單中選擇 GET，然後選擇核取記號圖示以完成建立方法。



#### Note

API 方法的其他選項包括：

- POST，主要用來建立子資源。
- PUT，主要用來更新現有的資源（也可用來建立子資源，但不建議）。
- DELETE，用來刪除資源。
- PATCH，用來更新資源。
- HEAD，主要用來測試案例。它與 GET 相同，但不會傳回資源顯示方式。
- OPTIONS，發起人可以使用它來取得目標服務之可用通訊選項的相關資訊。

建立的方法尚未與後端整合。下一個步驟會進行這項設定。

5. 在方法的 Setup (設定) 窗格中，針對 Integration type (整合類型) 選取 HTTP，從 HTTP method (HTTP 方法) 下拉式清單中選取 GET，輸入 `http://petstore-demo-endpoint.execute-api.com/petstore/pets` 做為 Endpoint URL (端點 URL) 值，保留所有其他設定的預設值，然後選擇 Save (儲存)。

### Note

對於整合請求的 HTTP method (HTTP 方法) , 您必須選擇後端支援的方法。對於 HTTP 或 Mock integration , 方法請求與整合請求最好使用相同的 HTTP 動詞。對於其他整合類型，方法請求可能會使用與整合請求不同的 HTTP 動詞。例如，若要呼叫 Lambda 函數，整合請求必須使用 POST 來呼叫函數，而方法請求則可根據 Lambda 函數的邏輯來使用任何 HTTP 動詞。

The screenshot shows the 'Integration type' section with 'HTTP' selected. Below it, the 'HTTP method' is set to 'GET'. The 'Endpoint URL' field contains 'Endpoint.execute-api.com/petstore/pets'. The 'Content Handling' dropdown is set to 'Passthrough'. A large red oval highlights the 'HTTP method' dropdown and the 'Endpoint URL' input field. A blue 'Save' button is at the bottom right.

當方法設定完成時，您會看到 Method Execution (方法執行) 窗格，其中您可以進一步設定方法請求，以新增查詢字串或自訂標頭參數。您也可以更新整合請求，將方法請求的輸入資料映射到後端所需的格式。

PetStore 網站可讓您依指定頁面上的寵物類型 (例如 "Dog" 或 "Cat") 來擷取 Pet 項目清單。它使用 type 與 page 查詢字串參數來接受這類輸入。因此，我們必須將查詢字串參數新增至方法請求，然後將它們映射到整合請求的對應查詢字串。

6. 在 GET 方法的 Method Execution (方法執行) 窗格中，選擇 Method Request (方法請求)，針對 Authorization (授權) 選取 AWS\_IAM，展開 URL Query String Parameters (URL 查詢字串參數) 區段，然後選擇 Add query string (新增查詢字串) 以建立名為 type 與 page 的兩個查詢字串參數。選擇核取記號圖示，以便在您新增每個查詢字串參數時將之儲存。

◀ Method Execution /pets - GET - Method Request



Provide information about this method's authorization settings and the parameters it can receive.

Settings

Authorization AWS\_IAM

Request Validator NONE

API Key Required false

▼ URL Query String Parameters

Name	Required	Caching	
page	<input type="checkbox"/>	<input type="checkbox"/>	
type	<input type="checkbox"/>	<input type="checkbox"/>	

[Add query string](#)

► HTTP Request Headers

► Request Body

► SDK Settings

用戶端現在可以在提交請求時，提供寵物類型與頁碼作為查詢字串參數。這些輸入參數必須映射到整合的查詢字串參數，以將輸入值轉送至後端的 PetStore 網站。由於方法使用 AWS\_IAM，因此您必須簽署請求才能呼叫方法。

7. 從方法的 Integration Request (整合請求) 頁面，展開 URL Query String Parameters (URL 查詢字串參數) 區段。根據預設，方法請求查詢字串參數會映射到相同名稱的整合請求查詢字串參數。此預設映射適用於我們的示範 API。因此，我們會保留指定值。若要將不同的方法請求參數映射到對應的整合請求參數，請針對參數選擇鉛筆圖示來編輯映射表達式，如 Mapped from (映射來源) 欄中所示。若要將方法請求參數映射到不同的整合請求參數，請先選擇刪除圖示以移除現有的整合請求參數，然後選擇 Add query string (新增查詢字串) 以指定新的名稱與所需的方法請求參數映射表達式。

[Method Execution](#) /pets - GET - Integration Request

Provide information about the target backend that this method will call and whether the incoming request data should be modified.

Integration type  Lambda Function [i](#)

HTTP [i](#)

Mock [i](#)

AWS Service [i](#)

Use HTTP Proxy integration  [i](#)

HTTP method GET [e](#)

Endpoint URL <http://petstore-demo-endpoint.execute-api.com/petstore/pets> [e](#)

▶ URL Path Parameters

▼ URL Query String Parameters

Name	Mapped from <a href="#">i</a>	Caching	
type	method.request.querystring.type	<input type="checkbox"/>	<a href="#">e</a> <a href="#">x</a>
page	method.request.querystring.page	<input type="checkbox"/>	<a href="#">e</a> <a href="#">x</a>

[+ Add query string](#)

▶ HTTP Headers

▶ Body Mapping Templates

如此即完成建立簡單的示範 API。接著可測試 API。

- 若要使用 API Gateway 主控台測試 API，請在 Method Execution (方法執行) 窗格上，針對 GET /pets 方法，選擇 Test (測試)。在 Method Test (方式測試) 窗格中，針對 type (類型) 與 page (頁面) 查詢字串分別輸入 Dog 與 2，然後選擇 Test (測試)。

[Method Execution](#) /pets - GET - Method Test

Make a test call to your method with the provided input

Path

No path parameters exist for this resource. You can define path parameters by using the syntax `{myPathParam}` in a resource path.

Query Strings

type

Dog

page

2

Headers

No header parameters exist for this method. You can add them via Method Request.

Stage Variables

No stage variables exist for this method.

Client Certificate

No client certificates have been generated.

Request Body

Request Body is not supported for GET methods.



結果如下所示(您可能需要向下捲動才會看到測試結果)。

Request: /pets?type=Dog&page=2

Status: 200

Latency: 1036 ms

Response Body

```
[  
  {  
    "id": 4,  
    "type": "Dog",  
    "price": 999.99  
  },  
  {  
    "id": 5,  
    "type": "Dog",  
    "price": 249.99  
  },  
  {  
    "id": 6,  
    "type": "Dog",  
    "price": 49.97  
  }  
]
```

Response Headers

```
{"Content-Type": "application/json"}
```

Logs

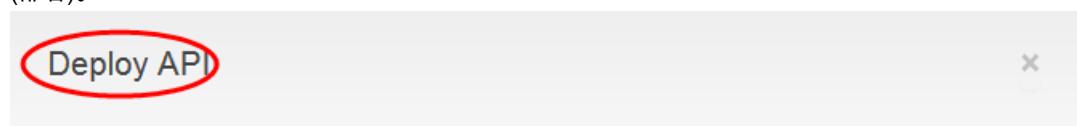
```
Execution log for request test-request  
Mon Apr 04 05:48:01 UTC 2016 : Starting execution for request: test-invoke-request  
Mon Apr 04 05:48:01 UTC 2016 : HTTP Method: GET, Resource Path: /pets  
Mon Apr 04 05:48:01 UTC 2016 : Method request path: {}  
Mon Apr 04 05:48:01 UTC 2016 : Method request query string: {page=2, type=Dog}  
Mon Apr 04 05:48:01 UTC 2016 : Method request headers: {}  
Mon Apr 04 05:48:01 UTC 2016 : Method request body before transformations: null
```

現在測試成功，我們可以部署 API 來公開提供使用。

- 若要部署 API，請選取 API，然後從 Actions (動作) 下拉式選單中選擇 Deploy API (部署 API)。

The screenshot shows the AWS Lambda & API Gateway interface. On the left, under 'APIs', 'myApi' is selected and highlighted with a red circle. The 'Resources' section shows a tree structure with a root node '/' and a child node '/pets'. Under the '/' node, there is a 'GET' method. On the right, a context menu is open from the 'Actions' button, also with a red circle around it. The menu has two sections: 'RESOURCE ACTIONS' and 'API ACTIONS'. In the 'API ACTIONS' section, 'Deploy API' is highlighted with a red circle.

在 Deploy API (部署 API) 對話方塊中，選擇一個階段 (若是 API 的第一個部署，則為 [New Stage])，然後在 Stage name (階段名稱) 輸入欄位中輸入名稱 (例如 "test"、"prod"、"dev" 等)；選擇性地在 Stage description (階段說明) 及/或 Deployment description (部署說明) 中提供說明，然後選擇 Deploy (部署)。



Choose a stage where your API will be deployed. For example, a test version of your API could be deployed to a stage named beta.

A screenshot of the 'Deploy API' configuration dialog. It contains four input fields: 'Deployment stage' with a dropdown menu showing '[New Stage]' (highlighted with a red circle), 'Stage name\*' with the value 'test' (highlighted with a red circle), 'Stage description' with the value 'GET on Pets only' (highlighted with a red circle), and 'Deployment description' with the value 'Initial deployment' (highlighted with a red circle). At the bottom right, there are 'Cancel' and 'Deploy' buttons, with 'Deploy' being highlighted with a red circle.

一旦完成部署，您就可以取得 API 端點的引動過程 URL (Invoke URL (呼叫 URL))。

如果 GET 方法支援開放式存取（亦即如果方法的授權類型已設定為 NONE），您可以按兩下 Invoke URL（呼叫 URL）連結在您的預設瀏覽器中呼叫方法。如果需要，您也可以將必要的查詢字串參數附加至引動過程 URL。對於此處所述的 AWS\_IAM 授權類型，您必須使用 AWS 帳戶之 IAM 使用者的存取金鑰 ID 與對應的秘密金鑰來簽署請求。若要執行這項操作，您必須使用支援 Signature 第 4 版 (SigV4) 協定的用戶端。這類用戶端的一個範例是使用其中一個 AWS 開發套件的應用程式，或是 Postman 應用程式或 cURL 命令。若要呼叫接受承載的 POST、PUT 或 PATCH 方法，您也需要使用這類用戶端來處理承載。

若要在 Postman 中呼叫此 API 方法，請將查詢字串參數附加至特定階段的方法引動過程 URL（如上圖所示），以建立完整的方法請求 URL：

```
https://api-id.execute-api.region.amazonaws.com/test/pets?type=Dog&page=2
```

在瀏覽器的網址列中指定此 URL。選擇 GET 作為 HTTP 動詞。針對 Authorization (授權) 標籤下的 Type (類型) 選項選取 AWS Signature (AWS 簽章)，然後指定下列必要的屬性，再傳送請求：

- 針對 AccessKey，輸入發起人的 AWS 存取金鑰，如 AWS IAM 所佈建。
- 針對 SecretKey，輸入發起人的 AWS 密密金鑰，如第一次建立存取金鑰時 AWS IAM 所佈建。
- 針對 AWS Region (AWS 區域)，輸入 API 託管的 AWS 區域，如引動過程 URL 中所指定。
- 針對 API Gateway 執行服務的 Service Name (服務名稱)，輸入 execute-api。

如果您使用開發套件建立用戶端，您可以呼叫開發套件所公開的方法來簽署請求。如需實作詳細資訊，請參閱您所選擇的 [AWS 開發套件](#)。

#### Note

當您的 API 變更時，您必須重新部署 API，以提供新的或更新的功能，再重新呼叫請求 URL。

## 對應 API Gateway API 的請求參數

在這個演練中，我們會說明如何將方法請求參數對應到對應的 API Gateway API 整合請求參數。我們會建立範例 API 與 HTTP 自訂整合，並用它示範如何使用 API Gateway 將方法請求參數對應到對應的整合請求參數。然後，我們會存取下列可公開存取的 HTTP 端點：

```
http://petstore-demo-endpoint.execute-api.com/petstore/pets
```

如果您複製上述 URL，將它貼入 Web 瀏覽器的網址列，再按 Enter 或 Return，您會收到下列 JSON 格式的回應內文：

```
[  
  {  
    "id": 1,  
    "type": "dog",  
    "price": 249.99  
  },  
  {  
    "id": 2,  
    "type": "cat",  
    "price": 124.99  
  },  
  {  
    "id": 3,  
    "type": "fish",  
    "price": 0.99  
  }]
```

]

上述端點可以接受兩個查詢參數：`type` 和 `page`。例如，請將 URL 變更為：

```
http://petstore-demo-endpoint.execute-api.com/petstore/pets?type=cat&page=2
```

您會收到下列 JSON 格式的回應承載，顯示只有貓的第 2 頁：

```
[  
  {  
    "id": 4,  
    "type": "cat",  
    "price": 999.99  
  },  
  {  
    "id": 5,  
    "type": "cat",  
    "price": 249.99  
  },  
  {  
    "id": 6,  
    "type": "cat",  
    "price": 49.97  
  }  
]
```

這個端點也支援使用項目 ID，如 URL 路徑參數所表示。例如，瀏覽至下列：

```
http://petstore-demo-endpoint.execute-api.com/petstore/pets/1
```

顯示下列 ID 為 1 的項目 JSON 格式資訊：

```
{  
  "id": 1,  
  "type": "dog",  
  "price": 249.99  
}
```

除了支援 GET 操作，這個端點也接受有承載的 POST 請求。例如，使用 Postman 傳送 POST 方法請求到下列項目：

```
http://petstore-demo-endpoint.execute-api.com/petstore/pets
```

包括標頭 Content-type: application/json 和下列請求內文：

```
{  
  "type": "dog",  
  "price": 249.99  
}
```

您在回應內文中會收到下列 JSON 物件：

```
{  
  "pet": {  
    "type": "dog",  
    "price": 249.99  
  }
```

```
    },
    "message": "success"
}
```

我們現在使用此 PetStore 網站的 HTTP 自訂整合建立 API Gateway API，公開這些和其他功能。任務包括下列：

- 以作用如同 `https://my-api-id.execute-api.region-id.amazonaws.com/test/petstorewalkthrough/pets` HTTP 端點代理的 `http://petstore-demo-endpoint.execute-api.com/petstore/pets` 資源建立 API。
- 讓 API 接受兩個方法請求查詢參數 `petType` 和 `petsPage`，並將它們分別對應到整合請求的 `type` 和 `page` 查詢參數，將請求傳送到 HTTP 端點。
- 支援 API 方法請求 URL 中的 `{petId}` 路徑參數，以指定項目 ID，將它對應到整合請求 URL 中的 `{id}` 路徑參數，將請求傳送到 HTTP 端點。
- 讓方法請求接受後端網站定義格式的 JSON 承載，將無修改的承載透過整合請求傳送到後端 HTTP 端點。

#### 主題

- [事前準備 \(p. 63\)](#)
- [步驟 1：建立資源 \(p. 63\)](#)
- [步驟 2：建立及測試方法 \(p. 64\)](#)
- [步驟 3：部署 API \(p. 66\)](#)
- [步驟 4：測試 API \(p. 67\)](#)

#### Note

請留意此演練步驟中使用的外殼。輸入小寫字母而不是大寫字母 (反之亦然)，稍後可能造成演練發生錯誤。

## 事前準備

開始此演練之前，您應該執行下列操作：

1. 完成 [先決條件：準備在 API Gateway 中建置 API \(p. 9\)](#) 中的步驟，包括將 API Gateway 存取許可指派給 IAM 使用者。
2. 至少要遵循 [教學：建置具有 HTTP 非代理整合的 API \(p. 52\)](#) 中的步驟，在 API Gateway 主控台建立名為 MyDemoAPI 的新 API。

## 步驟 1：建立資源

在此步驟中，您會建立三項資源，讓 API 與 HTTP 端點互動。

### 建立第一項資源

1. 在 Resources (資源) 窗格中，選取資源根，以一個正斜線表示 (/)，然後從 Actions (動作) 下拉式選單中選擇 Create Resource (建立資源)。
2. 針對 Resource Name (資源名稱) 輸入 `petstorewalkthrough`。
3. 針對 Resource Path (資源路徑)，接受預設值 `/petstorewalkthrough`，然後選擇 Create Resource (建立資源)。

### 建立第二項資源

1. 在 Resources (資源) 窗格中，選擇 `/petstorewalkthrough`，然後選擇 Create Resource (建立資源)。

2. 針對 Resource Name (資源名稱) 輸入 **pets**。
3. 針對 Resource Path (資源路徑)，接受預設值 /petstorewalkthrough/pets，然後選擇 Create Resource (建立資源)。

### 建立第三項資源

1. 在 Resources (資源) 窗格中，選擇 /petstorewalkthrough/pets，然後選擇 Create Resource (建立資源)。
  2. 針對 Resource Name (資源名稱) 輸入 **petId**。這會對應到 HTTP 端點的項目 ID。
  3. 針對 Resource Path (資源路徑)，以 **{petId}** 覆寫 petid。使用大括號 ({ }) 括住 petId，以顯示 /petstorewalkthrough/pets/{petId}，然後選擇 Create Resource (建立資源)。
- 這會對應到 HTTP 端點的 /petstore/pets/**my-item-id**。

## 步驟 2：建立及測試方法

在此步驟中，您要整合方法與後端 HTTP 端點，將 GET 方法請求參數對應到對應的整合請求參數，然後測試方法。

### 設定和測試第一個 GET 方法

此程序示範下列項目：

- 建立及整合 GET /petstorewalkthrough/pets 的方法請求與 GET `http://petstore-demo-endpoint.execute-api.com/petstore/pets` 的整合請求。
- 將 petType 和 petsPage 的方法請求查詢參數對應到 type 和 page 的整合請求查詢字串參數。

1. 在 Resources (資源) 窗格中選擇 /petstorewalkthrough/pets，從 Actions (動作) 選單中選擇 Create Method (建立方法)，然後從方法名稱的下拉式清單中，選擇 /pets (寵物) 下的 GET。
2. 在 /petstorewalkthrough/pets - GET - Setup (/petstorewalkthrough/pets - GET - 設定) 窗格中，針對 Integration type (整合類型) 選擇 HTTP，針對 HTTP method (HTTP 方法) 選擇 GET。
3. 針對 Endpoint URL (端點 URL)，輸入 `http://petstore-demo-endpoint.execute-api.com/petstore/pets`。
4. 選擇 Save (儲存)。
5. 在 Method Execution (方法執行) 窗格中，選擇 Method Request (方法請求)，然後選擇 URL Query String Parameters (URL 查詢字串參數) 旁的箭頭。
6. 選擇 Add query string (新增查詢字串)。
7. 在 Name (名稱) 輸入 **petType**。

這會在 API 方法請求中指定 petType 查詢參數。

8. 選擇核取記號圖示以完成建立方法請求 URL 查詢字串參數。
  9. 再次選擇 Add query string (新增查詢字串)。
  10. 在 Name (名稱) 輸入 **petsPage**。
- 這會在 API 方法請求中指定 petsPage 查詢參數。
11. 選擇核取記號圖示以完成建立方法請求 URL 查詢字串參數。
  12. 依序選擇 Method Execution (方法執行)、Integration Request (整合請求)，然後選擇 URL Query String Parameters (URL 查詢字串參數) 旁的箭頭。
  13. 刪除從 `method.request.querystring.petType` 對應的 **petType** 項目，以及從 `method.request.querystring.petsPage` 對應的 **petsPage** 項目。您之所以執行此步驟，是因為端點在要求 URL 需要名為 `type` 和 `page` 的查詢字串參數，而非預設值。
  14. 選擇 Add query string (新增查詢字串)。

15. 在 Name (名稱) 輸入 **type**。這會為整合請求 URL 建立所需的查詢字串參數。
16. 對於 Mapped from (對應來源)，輸入 **method.request.querystring.petType**。

這會將方法請求的 **petType** 查詢參數對應到整合請求的 **type** 查詢參數。

17. 選擇核取記號圖示以完成建立整合請求 URL 查詢字串參數。
18. 再次選擇 Add query string (新增查詢字串)。
19. 在 Name (名稱) 輸入 **page**。這會為整合請求 URL 建立所需的查詢字串參數。
20. 對於 Mapped from (對應來源)，輸入 **method.request.querystring.petsPage**。

這會將方法請求的 **petsPage** 查詢參數對應到整合請求的 **page** 查詢參數。

21. 選擇核取記號圖示以完成建立整合請求 URL 查詢字串參數。
22. 選擇 Method Execution (方法執行)。在 Client (用戶端) 方塊中，選擇 TEST。在 Query Strings (查詢字串) 區域的 **petType** (寵物類型) 中，輸入 **cat**。針對 **petsPage** (寵物頁數)，輸入 **2**。
23. 選擇 Test (測試)。若成功，Response Body (回應內文) 會顯示如下：

```
[  
  {  
    "id": 4,  
    "type": "cat",  
    "price": 999.99  
  },  
  {  
    "id": 5,  
    "type": "cat",  
    "price": 249.99  
  },  
  {  
    "id": 6,  
    "type": "cat",  
    "price": 49.97  
  }  
]
```

## 設定和測試第二個 GET 方法

此程序示範下列項目：

- 建立及整合 **GET /petstorewalkthrough/pets/{petId}** 的方法請求與 **GET http://petstore-demo-endpoint.execute-api.com/petstore/pets/{id}** 的整合請求。
- 這會將 **petId** 的方法請求路徑參數對應到 **id** 整合請求路徑參數。

1. 在 Resources (資源) 清單中選擇 **/petstorewalkthrough/pets/{petId}**，從 Actions (動作) 下拉式選單中選擇 Create Method (建立方法)，然後選擇 GET 做為方法的 HTTP 動詞。
2. 在 Setup (設定) 窗格中，針對 Integration type (整合類型) 選擇 HTTP，針對 HTTP method (HTTP 方法) 選擇 GET。
3. 針對 Endpoint URL (端點 URL)，輸入 **http://petstore-demo-endpoint.execute-api.com/petstore/pets/{id}**。
4. 選擇 Save (儲存)。
5. 在 Method Execution (方法執行) 窗格中，選擇 Integration Request (整合請求)，然後選擇 URL Path Parameters (URL 路徑參數) 旁的箭頭。
6. 選擇 Add path (新增路徑)。
7. 在 Name (名稱) 輸入 **id**。
8. 對於 Mapped from (對應來源)，輸入 **method.request.path.petId**。

這會將 petId 的方法請求路徑參數對應到 id 的整合請求路徑參數。

9. 選擇核取記號圖示以完成建立 URL 路徑參數。
10. 選擇 Method Execution (方法執行)，然後選擇 Client (用戶端) 方塊中的 TEST。在 Path (路徑) 區域的 petId 中，輸入 1。
11. 選擇 Test (測試)。若成功，Response Body (回應內文) 會顯示如下：

```
{  
    "id": 1,  
    "type": "dog",  
    "price": 249.99  
}
```

## 設定和測試 POST 方法

此程序示範下列項目：

- 建立及整合 POST /petstorewalkthrough/pets 的方法請求與 POST <http://petstore-demo-endpoint.execute-api.com/petstore/pets> 的整合請求。
  - 將方法請求 JSON 承載傳遞到整合請求承載，無需修改。
1. 在 Resources (資源) 清單中選擇 /petstorewalkthrough/pets/，從 Actions (動作) 下拉式選單中選擇 Create Method (建立方法)，然後選擇 POST 做為方法的 HTTP 動詞。
  2. 在 Setup (設定) 窗格中，針對 Integration type (整合類型) 選擇 HTTP，針對 HTTP method (HTTP 方法) 選擇 POST。
  3. 針對 Endpoint URL (端點 URL)，輸入 <http://petstore-demo-endpoint.execute-api.com/petstore/pets>。
  4. 選擇 Save (儲存)。
  5. 在 Method Execution (方法執行) 窗格的 Client (用戶端) 方塊中選擇 TEST。展開 Request Body (請求內文)，然後輸入下列資訊：

```
{  
    "type": "dog",  
    "price": 249.99  
}
```

6. 選擇 Test (測試)。若成功，Response Body (回應內文) 會顯示如下：

```
{  
    "pet": {  
        "type": "dog",  
        "price": 249.99  
    },  
    "message": "success"  
}
```

## 步驟 3：部署 API

在此步驟中，將會部署 API，如此您即開始從 API Gateway 主控台之外呼叫 API。

### 部署 API

1. 在 Resources (資源) 窗格中，選擇 Deploy API (部署 API)。
2. 針對 Deployment stage (部署階段)，選擇 test。

### Note

輸入必須為 UTF-8 編碼 (例如未本地化的) 文字。

3. 針對 Deployment description (部署描述)，輸入 **Calling HTTP endpoint walkthrough**。
4. 選擇 Deploy (部署)。

## 步驟 4：測試 API

在此步驟中，您會在 API Gateway 主控台外使用您的 API 存取 HTTP 端點。

1. 在 Stage Editor (階段編輯器) 窗格中，將 Invoke URL (呼叫 URL) 旁的 URL 複製到剪貼簿。此 URL 看起來如下：

```
https://my-api-id.execute-api.region-id.amazonaws.com/test
```

2. 將此 URL 貼至新瀏覽器標籤的網址方塊中。
3. 附加 /petstorewalkthrough/pets，然後其看起來如下：

```
https://my-api-id.execute-api.region-id.amazonaws.com/test/petstorewalkthrough/pets
```

瀏覽至該 URL。顯示的資訊應如下：

```
[  
 {  
   "id": 1,  
   "type": "dog",  
   "price": 249.99  
 },  
 {  
   "id": 2,  
   "type": "cat",  
   "price": 124.99  
 },  
 {  
   "id": 3,  
   "type": "fish",  
   "price": 0.99  
 }  
]
```

4. 在 petstorewalkthrough/pets 後，輸入 **?petType=cat&petsPage=2**，使其如下所示：

```
https://my-api-id.execute-api.region-id.amazonaws.com/test/petstorewalkthrough/pets?  
petType=cat&petsPage=2
```

5. 瀏覽至該 URL。顯示的資訊應如下：

```
[  
 {  
   "id": 4,  
   "type": "cat",  
   "price": 999.99  
 },  
 {  
   "id": 5,  
   "type": "cat",  
   "price": 249.99  
 },  
 ]
```

```
{  
    "id": 6,  
    "type": "cat",  
    "price": 49.97  
}  
]
```

6. 在 `petstorewalkthrough/pets` 後，將 `?petType=cat&petsPage=2` 取代為 `/1`，使其如下所示：

```
https://my-api-id.execute-api.region-id.amazonaws.com/test/petstorewalkthrough/pets/1
```

7. 瀏覽至該 URL。顯示的資訊應如下：

```
{  
    "id": 1,  
    "type": "dog",  
    "price": 249.99  
}
```

8. 使用 Web 偵錯代理工具或 cURL 命令列工具，將 POST 方法請求傳送到上個程序的 URL。附加 `/petstorewalkthrough/pets`，然後其看起來如下：

```
https://my-api-id.execute-api.region-id.amazonaws.com/test/petstorewalkthrough/pets
```

附加下列標頭：

```
Content-Type: application/json
```

將下列程式碼新增到請求內文中：

```
{  
    "type": "dog",  
    "price": 249.99  
}
```

例如，如果您使用 cURL 命令列工具，請執行類似下列的命令：

```
curl -H "Content-Type: application/json" -X POST -d "{\"type\": \"dog\",  
\"price\": 249.99}" https://my-api-id.execute-api.region-id.amazonaws.com/test/  
petstorewalkthrough/pets
```

回應內文應該會傳回下列資訊：

```
{  
    "pet": {  
        "type": "dog",  
        "price": 249.99  
    },  
    "message": "success"  
}
```

本演練到此結束。

## 對應回應承載

此演練示範如何在 API Gateway 中使用模型及對應範本，將 API 呼叫的輸出轉換成不同的資料結構描述。此演練採用「[Amazon API Gateway 入門 \(p. 9\)](#)」與「[對應 API Gateway API 的請求參數 \(p. 61\)](#)」中的說明與概念為基礎。若尚未完成這些演練，建議先行完成。

此演練使用 API Gateway 從一般大眾均可存取的 HTTP 端點及您建立的 AWS Lambda 函數取得範例資料。HTTP 終端節點及 Lambda 函數都會傳回相同的範例資料：

```
[  
  {  
    "id": 1,  
    "type": "dog",  
    "price": 249.99  
  },  
  {  
    "id": 2,  
    "type": "cat",  
    "price": 124.99  
  },  
  {  
    "id": 3,  
    "type": "fish",  
    "price": 0.99  
  }  
]
```

您將可使用模型及對應範本，將此資料轉換成一或多種輸出格式。在 API Gateway 中，模型可定義某些資料的格式，亦稱為結構描述或形狀。在 API Gateway 中，映射範本用於將一些資料從一種格式轉換為另一種格式。如需詳細資訊，請參閱[針對請求與回應對應建立模型與對應範本 \(p. 244\)](#)。

第一個模型與對應範本可用於將 id 重新命名為 number；將 type 重新命名為 class；以及將 price 重新命名為 salesPrice，如下所示：

```
[  
  {  
    "number": 1,  
    "class": "dog",  
    "salesPrice": 249.99  
  },  
  {  
    "number": 2,  
    "class": "cat",  
    "salesPrice": 124.99  
  },  
  {  
    "number": 3,  
    "class": "fish",  
    "salesPrice": 0.99  
  }  
]
```

第二個模型與對應範本可用於將 id 與 type 合併成 description，以及將 price 重新命名為 askingPrice，如下所示：

```
[  
  {  
    "description": "Item 1 is a dog.",  
    "askingPrice": 249.99  
  },  
  {  
    "description": "Item 2 is a cat.",  
    "askingPrice": 124.99  
  }  
]
```

```
        "askingPrice": 124.99
    },
{
    "description": "Item 3 is a fish.",
    "askingPrice": 0.99
}
]
```

第三個模型與對應範本可用於將 id、type 及 price 合併成一組 listings，如下所示：

```
{
    "listings": [
        "Item 1 is a dog. The asking price is 249.99.",
        "Item 2 is a cat. The asking price is 124.99.",
        "Item 3 is a fish. The asking price is 0.99."
    ]
}
```

### 主題

- [步驟 1：建立模型 \(p. 70\)](#)
- [步驟 2：建立資源 \(p. 72\)](#)
- [步驟 3：建立 GET 方法 \(p. 73\)](#)
- [步驟 4：建立 Lambda 函數 \(p. 73\)](#)
- [步驟 5：設定及測試方法 \(p. 74\)](#)
- [步驟 6：部署 API \(p. 78\)](#)
- [步驟 7：測試 API \(p. 78\)](#)
- [步驟 8：清除 \(p. 79\)](#)

## 步驟 1：建立模型

在此步驟中，將會建立四個模型。前三個模型代表在 HTTP 端點與 Lambda 函數中使用的資料輸出格式。最後一個模型代表要在 Lambda 函數中使用的資料輸入結構描述。

### 建立第一個輸出模型

1. 登入位於 <https://console.aws.amazon.com/apigateway> 的 API Gateway 主控台。
2. 顯示出 MyDemoAPI 時，選擇 Models (模型)。
3. 選擇 Create (建立)。
4. 針對 Model name (模型名稱)，輸入 **PetsModelNoFlatten**。
5. 針對 Content type (內容類型) 輸入 **application/json**。
6. 針對 Model description (模型描述)，輸入 **Changes id to number, type to class, and price to salesPrice**。
7. 在 Model schema (模型結構描述) 中，鍵入下列 JSON 結構描述相容定義：

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "title": "PetsModelNoFlatten",
    "type": "array",
    "items": {
        "type": "object",
        "properties": {
            "number": { "type": "integer" },
            "class": { "type": "string" },
            "salesPrice": { "type": "number" }
        }
    }
}
```

```
    }  
}
```

8. 選擇 Create model (建立模型)。

### 建立第二個輸出模型

1. 選擇 Create (建立)。
2. 針對 Model name (模型名稱)，輸入 **PetsModelFlattenSome**。
3. 針對 Content type (內容類型) 輸入 **application/json**。
4. 針對 Model description (模型描述)，輸入 **Combines id and type into description, and changes price to askingPrice**。
5. 在 Model schema (模型結構描述) 中鍵入如下內容：

```
{  
  "$schema": "http://json-schema.org/draft-04/schema#",  
  "title": "PetsModelFlattenSome",  
  "type": "array",  
  "items": {  
    "type": "object",  
    "properties": {  
      "description": { "type": "string" },  
      "askingPrice": { "type": "number" }  
    }  
  }  
}
```

6. 選擇 Create model (建立模型)。

### 建立第三個輸出模型

1. 選擇 Create (建立)。
2. 針對 Model name (模型名稱)，輸入 **PetsModelFlattenAll**。
3. 針對 Content type (內容類型) 輸入 **application/json**。
4. 針對 Model description (模型描述)，輸入 **Combines id, type, and price into a set of listings**。
5. 在 Model schema (模型結構描述) 中鍵入如下內容：

```
{  
  "$schema": "http://json-schema.org/draft-04/schema#",  
  "title": "PetsModelFlattenAll",  
  "type": "object",  
  "properties": {  
    "listings": {  
      "type": "array",  
      "items": {  
        "type": "string"  
      }  
    }  
  }  
}
```

6. 選擇 Create model (建立模型)。

### 建立輸入模型

1. 選擇 Create (建立)。

2. 針對 Model name (模型名稱) , 輸入 **PetsLambdaModel**。
3. 針對 Content type (內容類型) 輸入 **application/json**。
4. 針對 Model description (模型描述) , 輸入 **GetPetsInfo model**。
5. 在 Model schema (模型結構描述) 中鍵入如下內容：

```
{  
  "$schema": "http://json-schema.org/draft-04/schema#",  
  "title": "PetsLambdaModel",  
  "type": "array",  
  "items": {  
    "type": "object",  
    "properties": {  
      "id": { "type": "integer" },  
      "type": { "type": "string" },  
      "price": { "type": "number" }  
    }  
  }  
}
```

6. 選擇 Create model (建立模型)。

## 步驟 2：建立資源

在此步驟中，將會建立四項資源。前三項資源可讓您能從 HTTP 端點取得三種輸出格式的範例資料。最後一項資源可讓您能從 Lambda 函數取得範例資料，而其輸出的結構描述會將 id 與 type 合併為 description，並會將 price 重新命名為 askingPrice。

### 建立第一項資源

1. 在連結清單中，選擇 Resources (資源)。
2. 在 Resources (資源) 窗格中，選擇 /petstorewalkthrough，然後選擇 Create Resource (建立資源)。
3. 針對 Resource Name (資源名稱) 輸入 **NoFlatten**。
4. 針對 Resource Path (資源路徑)，接受預設值 /petstorewalkthrough/noflatten，然後選擇 Create Resource (建立資源)。

### 建立第二項資源

1. 在 Resources (資源) 窗格中，再次選擇 /petstorewalkthrough，然後選擇 Create Resource (建立資源)。
2. 針對 Resource Name (資源名稱) 輸入 **FlattenSome**。
3. 針對 Resource Path (資源路徑)，接受預設值 /petstorewalkthrough/flattensome，然後選擇 Create Resource (建立資源)。

### 建立第三項資源

1. 在 Resources (資源) 窗格中，再次選擇 /petstorewalkthrough，然後選擇 Create Resource (建立資源)。
2. 針對 Resource Name (資源名稱) 輸入 **FlattenAll**。
3. 針對 Resource Path (資源路徑)，接受預設值 /petstorewalkthrough/flattenall，然後選擇 Create Resource (建立資源)。

### 建立第四項資源

1. 在 Resources (資源) 窗格中，再次選擇 /petstorewalkthrough，然後選擇 Create Resource (建立資源)。
2. 針對 Resource Name (資源名稱) 輸入 **LambdaFlattenSome**。

3. 針對 Resource Path (資源路徑) , 接受預設值 /petstorewalkthrough/lambdafattensome , 然後選擇 Create Resource (建立資源)。

## 步驟 3：建立 GET 方法

在此步驟中，將會為上一個步驟中所建立的每項資源建立 GET 方法。

### 建立第一個 GET 方法

1. 在 Resources (資源) 清單中，選擇 /petstorewalkthrough/flattenall ，然後選擇 Create Method (建立方法)。
2. 從下拉式清單中選擇 GET ，然後選擇核取記號圖示來儲存您的選擇。
3. 在 Setup (設定) 窗格中，為 Integration type (整合類型) 選擇 HTTP ，同時為 HTTP method (HTTP 方法) 選擇 GET ，然後在 Endpoint URL (端點 URL) 中，鍵入 `http://petstore-demo-endpoint.execute-api.com/petstore/pets`，最後選擇 Save (儲存)。

### 建立第二個 GET 方法

1. 在 Resources (資源) 清單中，選擇 /petstorewalkthrough/lambdafattensome ，然後選擇 Create Method (建立方法)。
2. 從下拉式清單中選擇 GET ，然後選擇核取記號儲存您的選擇。
3. 在 Setup (設定) 窗格中，為 Integration type (整合類型) 選擇 Lambda Function (Lambda 函數) ，再從 Lambda Region (Lambda 區域) 下拉式清單中，選擇 [GetPetsInfo Lambda 函數 \(p. 73\)](#) 的建立區域，同時為 Lambda Function (Lambda 函數) 選擇 `GetPetsInfo` ，然後選擇 Save (儲存)。出現提示要為 Lambda 函數新增許可時，選擇 OK (確定)。

### 建立第三個 GET 方法

1. 在 Resources (資源) 清單中，選擇 /petstorewalkthrough/flattensome ，然後選擇 Create Method (建立方法)。
2. 從下拉式清單中選擇 GET ，然後選擇核取記號圖示來儲存您的選擇。
3. 在 Setup (設定) 窗格中，為 Integration type (整合類型) 選擇 HTTP ，同時為 HTTP method (HTTP 方法) 選擇 GET、在 Endpoint URL (端點 URL) 中，鍵入 `http://petstore-demo-endpoint.execute-api.com/petstore/pets`，然後選擇 Save (儲存)。

### 建立第四個 GET 方法

1. 在 Resources (資源) 清單中，選擇 /petstorewalkthrough/noflatten ，然後選擇 Actions (動作)、Create Method (建立方法)。
2. 從下拉式清單中選擇 GET ，然後選擇核取記號圖示來儲存您的選擇。
3. 在 Setup (設定) 窗格中，為 Integration type (整合類型) 選擇 HTTP ，同時為 HTTP method (HTTP 方法) 選擇 GET、在 Endpoint URL (端點 URL) 中，鍵入 `http://petstore-demo-endpoint.execute-api.com/petstore/pets`，然後選擇 Save (儲存)。

## 步驟 4：建立 Lambda 函數

在此步驟中，將會建立 Lambda 函數來傳回範例資料。

### 建立 Lambda 函數

1. Open the AWS Lambda console at <https://console.aws.amazon.com/lambda/>.

2. 請執行下列其中一項：

- 出現歡迎頁面時，請選擇 Get Started Now (立即開始使用)。
  - 出現 Lambda: Function list (&LAM; :函數清單) 頁面時，請選擇 Create a Lambda function (建立 &LAM; 函數)。
3. 在 Name (名稱) 輸入 **GetPetsInfo**。
4. 針對 Description (描述)，輸入 **Gets information about pets**。
5. 為 Code template (程式碼範本) 選擇 None (無)。
6. 鍵入下列程式碼：

```
console.log('Loading event');

exports.handler = function(event, context, callback) {
  callback(null,
    [{"id": 1, "type": "dog", "price": 249.99},
     {"id": 2, "type": "cat", "price": 124.99},
     {"id": 3, "type": "fish", "price": 0.99}]); // SUCCESS with message
};
```

Tip

在以 Node.js 編寫的上列程式碼中，`console.log` 會將資訊寫入 Amazon CloudWatch 日誌內。`event` 包含 Lambda 函數的輸入。`context` 包含呼叫內容。`callback` 會傳回結果。如需如何撰寫 Lambda 函數程式碼的詳細資訊，請參閱 [AWS Lambda：運作方式](#) 中的「程式設計模型」一節，以及 AWS Lambda 開發人員指南中的範例演練。

7. 為 Handler name (處理器名稱)，保留預設值 `index.handler`。
8. 為 Role (角色) 選擇在 [建置具有 Lambda 整合的 API Gateway API \(p. 24\)](#) 中建立的 Lambda 執行角色 `APIGatewayLambdaExecRole`。
9. 選擇 Create Lambda function (建立 Lambda 函數)。
10. 在函數清單中，選擇 `GetPetsInfo`，以顯示函數的詳細資訊。
11. 請記下您建立此函數所在的 AWS 區域，供稍後使用。
12. 在快顯清單中，選擇 `Edit or test function` (編輯或測試函數)。
13. 為 Sample event (範例事件)，以下列內容取代出現的所有程式碼：

```
{  
}
```

Tip

空白大括號表示此 Lambda 函數沒有任何輸入值。因為此函數只會傳回包含寵物資訊的 JSON 物件，所以此處不需要這些金鑰/值對。

14. 選擇 Invoke (叫用)。Execution result (執行結果) 會顯示 `[{"id":1,"type":"dog","price":249.99}, {"id":2,"type":"cat","price":124.99}, {"id":3,"type":"fish","price":0.99}]`，而此內容也會寫入 CloudWatch Logs 日誌檔中。
15. 選擇 Go to function list (前往函數清單)。

## 步驟 5：設定及測試方法

在此步驟中，將會設定方法的回應、整合要求以及整合回應，以指定與 HTTP 端點和 Lambda 函數相關聯之 GET 方法的輸入及輸出資料結構描述 (或模型)。此外還會學習如何使用 API Gateway 主控台測試如何呼叫這些方法。

### 設定第一個 GET 方法的整合並加以測試

1. 從 API 的 Resources (資源) 樹狀目錄中，選擇 /petstorewalkthrough/flattenall 節點下的 GET。
2. 在 Method Execution (方法執行) 窗格中，選擇 Method Response (方法回應)，然後選擇 200 旁的箭頭。
3. 在 Response Models for 200 (200 的回應模型) 區域中，為 application/json 選擇鉛筆圖示，開始設定方法輸出的模型。為 Models (模型) 選擇 PetsModelFlattenAll，然後選擇核取記號圖示儲存設定。
4. 在 Method Execution (方法執行) 中，選擇 Integration Response (整合回應)，然後選擇 200 旁的箭頭。
5. 展開 Body Mapping Templates (內文對應範本) 區段，然後選擇 Content-Type 下的 application/json。
6. 為 Generate template from model (從模型產生範本) 選擇 PetsModelFlattenAll，在 PetsModelFlattenAll 模型之後顯示對應範本做為起點。
7. 修改對應範本程式碼，如下所示：

```
#set($inputRoot = $input.path('$'))  
{  
    "listings" : [  
#foreach($elem in $inputRoot)  
        "Item number $elem.id is a $elem.type. The asking price is  
        $elem.price."#if($foreach.hasNext),#end  
  
#end  
    ]  
}
```

8. 選擇 Save (儲存)。
9. 選擇 Method Execution (方法執行)，然後在 Client (用戶端) 方塊中選擇 TEST (測試)，最後再選擇 Test (測試)。若成功，Response Body (回應內文) 會顯示如下：

```
{  
    "listings" : [  
        "Item number 1 is a dog. The asking price is 249.99.",  
        "Item number 2 is a cat. The asking price is 124.99.",  
        "Item number 3 is a fish. The asking price is 0.99."  
    ]  
}
```

### 設定第二個 GET 方法的整合並加以測試

1. 從 API 的 Resources (資源) 樹狀目錄中，選擇 /petstorewalkthrough/lambdaflattensome 節點下的 GET。
2. 在 Method Execution (方法執行) 中，選擇 Method Response (方法回應)。然後選擇 200 旁的箭頭展開該區段。
3. 在 Response Models for 200 (200 的回應模型) 區域中，選擇內容類型為 application/json 之資料列上的鉛筆圖示。為 Models (模型) 選擇 PetsModelFlattenSome，然後選擇核取記號圖示儲存選項。
4. 返回 Method Execution (方法執行)。選擇 Integration Response (整合回應)，再選擇 200 旁的箭頭。
5. 在 Body Mapping Templates (內文對應範本) 區段中，選擇 Content-Type 下的 application/json。
6. 為 Generate template (產生範本) 選擇 PetsModelFlattenSome，顯示此方法輸出的對應指令碼範本。
7. 修改程式碼 (如下所示)，然後選擇 Save (儲存)：

```
#set($inputRoot = $input.path('$'))  
[  
#foreach($elem in $inputRoot)  
{  
    "description" : "Item $elem.id is a $elem.type.",  
    "askingPrice" : $elem.price
```

```
 }#if($foreach.hasNext),#end  
  
#end  
]
```

8. 選擇 Method Execution (方法執行) , 然後在 Client (用戶端) 方塊中選擇 TEST (測試) , 最後再選擇 Test (測試)。若成功 , Response Body (回應內文) 會顯示如下 :

```
[  
 {  
     "description" : "Item 1 is a dog.",  
     "askingPrice" : 249.99  
 },  
 {  
     "description" : "Item 2 is a cat.",  
     "askingPrice" : 124.99  
 },  
 {  
     "description" : "Item 3 is a fish.",  
     "askingPrice" : 0.99  
 }  
 ]
```

### 設定第二個 GET 方法的整合並加以測試

1. 從 API 的 Resources (資源) 樹狀目錄中 , 選擇 /petstorewalkthrough/flattensome 節點下的 GET。
2. 在 Method Execution (方法執行) 窗格中 , 選擇 Method Response (方法回應)。
3. 選擇 200 旁的箭頭。
4. 在 Response Models for 200 (200 的回應模型) 區域中 , 為 application/json 選擇鉛筆圖示。為 Models (模型) 選擇 PetsModelFlattenSome , 然後選擇核取記號圖示儲存選項。
5. 返回 Method Execution (方法執行) , 然後選擇 Integration Response (整合回應)。
6. 選擇 200 旁的箭頭以展開該區段。
7. 展開 Body Mapping Templates (內文對應範本) 區域。為 Content-Type 選擇 application/json。為 Generate template (產生範本) 選擇 PetsModelFlattenSome , 顯示此方法輸出的對應指令碼範本。
8. 修改程式碼 , 如下所示 :

```
#set($inputRoot = $input.path('$'))  
[  
 #foreach($elem in $inputRoot)  
 {  
     "description": "Item $elem.id is a $elem.type.",  
     "askingPrice": $elem.price  
 }#if($foreach.hasNext),#end  
  
#end  
]
```

9. 選擇 Save (儲存)。
10. 返回 Method Execution (方法執行) , 然後選擇 Client (用戶端) 方塊中的 TEST (測試)。最後再選擇 Test (測試)。若成功 , Response Body (回應內文) 會顯示如下 :

```
[  
 {  
     "description": "Item 1 is a dog.",  
     "askingPrice": 249.99  
 },  
 {  
     "description": "Item 2 is a cat.",  
 }
```

```
        "askingPrice": 124.99
    },
{
    "description": "Item 3 is a fish.",
    "askingPrice": 0.99
}
]
```

### 設定第四個 GET 方法的整合並加以測試

1. 從 API 的 Resources (資源) 樹狀目錄中，選擇 /petstorewalkthrough/noflatten 節點下的 GET。
2. 在 Method Execution (方法執行) 窗格中，選擇 Method Response (方法回應)，然後展開 200 區段。
3. 在 Response Models for 200 (200 的回應模型) 區域中，為 application/json 選擇鉛筆圖示，更新此方法的回應模型。
4. 選擇 PetsModelNoFlatten 做為內容類型 application/json 的模型，然後選擇核取記號圖示來儲存選擇。
5. 選擇 Method Execution (方法執行)，再選擇 Integration Response (整合回應)，然後選擇 200 旁的箭頭展開該區段。
6. 展開 Mapping Templates (對應範本) 區段。為 Content-Type 選擇 application/json。為 Generate templates (產生範本) 選擇 PetsModelNoFlatten，顯示此方法輸出的對應指令碼範本。
7. 修改程式碼，如下所示：

```
#set($inputRoot = $input.path('$'))
[
#foreach($elem in $inputRoot)
{
    "number": $elem.id,
    "class": "$elem.type",
    "salesPrice": $elem.price
}#if($foreach.hasNext),#end

#end
]
```

8. 選擇 Save (儲存)。
9. 返回 Method Execution (方法執行)，然後在 Client (用戶端) 方塊中選擇 TEST (測試)，最後再選擇 Test (測試)。若成功，Response Body (回應內文) 會顯示如下：

```
[
{
    "number": 1,
    "class": "dog",
    "salesPrice": 249.99
},
{
    "number": 2,
    "class": "cat",
    "salesPrice": 124.99
},
{
    "number": 3,
    "class": "fish",
    "salesPrice": 0.99
}
]
```

## 步驟 6：部署 API

在此步驟中，將會部署 API，如此您即開始從 API Gateway 主控台之外呼叫 API。

### 部署 API

1. 在 Resources (資源) 窗格中，選擇 Deploy API (部署 API)。
2. 針對 Deployment stage (部署階段)，選擇 test。
3. 針對 Deployment description (部署描述)，輸入 **Using models and mapping templates walkthrough**。
4. 選擇 Deploy (部署)。

## 步驟 7：測試 API

在此步驟中，將會從 API Gateway 主控台之外與 HTTP 端點及 Lambda 函數進行互動。

1. 在 Stage Editor (階段編輯器) 窗格中，將 Invoke URL (呼叫 URL) 旁的 URL 複製到剪貼簿。此 URL 看起來如下：

```
https://my-api-id.execute-api.region-id.amazonaws.com/test
```

2. 將此 URL 貼至新瀏覽器標籤的網址方塊中。
3. 附加 /petstorewalkthrough/noflatten，然後其看起來如下：

```
https://my-api-id.execute-api.region-id.amazonaws.com/test/petstorewalkthrough/noflatten
```

瀏覽至該 URL。顯示的資訊應如下：

```
[  
  {  
    "number": 1,  
    "class": "dog",  
    "salesPrice": 249.99  
  },  
  {  
    "number": 2,  
    "class": "cat",  
    "salesPrice": 124.99  
  },  
  {  
    "number": 3,  
    "class": "fish",  
    "salesPrice": 0.99  
  }  
]
```

4. 在 petstorewalkthrough/ 之後，以 noflatten 取代 flattensome。
5. 瀏覽至該 URL。顯示的資訊應如下：

```
[  
  {  
    "description": "Item 1 is a dog.",  
    "askingPrice": 249.99  
  },  
  {  
    "description": "Item 2 is a cat.",  
    "askingPrice": 124.99  
  }
```

```
},
{
  "description": "Item 3 is a fish.",
  "askingPrice": 0.99
}
]
```

6. 在 petstorewalkthrough/ 之後，以 flattensome 取代 flattenall。
7. 瀏覽至該 URL。顯示的資訊應如下：

```
{
  "listings" : [
    "Item number 1 is a dog. The asking price is 249.99.",
    "Item number 2 is a cat. The asking price is 124.99.",
    "Item number 3 is a fish. The asking price is 0.99."
  ]
}
```

8. 在 petstorewalkthrough/ 之後，以 flattenall 取代 lambdaflattensome。
9. 瀏覽至該 URL。顯示的資訊應如下：

```
[
  {
    "description" : "Item 1 is a dog.",
    "askingPrice" : 249.99
  },
  {
    "description" : "Item 2 is a cat.",
    "askingPrice" : 124.99
  },
  {
    "description" : "Item 3 is a fish.",
    "askingPrice" : 0.99
  }
]
```

## 步驟 8：清除

若您不再需要因為此演練所建立的 Lambda 函數，可在此步驟中將其刪除。您也可以刪除隨附的 IAM 資源。

### Warning

若刪除掉了您 API 相依的 Lambda 函數，這些 API 將會無法繼續運作。而刪除 Lambda 函數無法復原。所以若要再次使用該 Lambda 函數，必須加以重新建立。

若刪除資源 Lambda 函數相依的 IAM 資源，Lambda 函數及所有相依於該資源的 API 將無法繼續運作。而刪除 IAM 資源無法復原。所以若要再次使用該 IAM 資源，必須加以重新建立。若預計繼續使用為此演練及其他演練所建立的資源進行實驗，請勿刪除 Lambda 叫用角色或 Lambda 執行角色。

### 刪除 Lambda 函數

1. Sign in to the AWS Management Console and open the AWS Lambda console at <https://console.aws.amazon.com/lambda/>.
2. 在 Lambda: Function list (Lambda：函數清單) 頁面的函數清單中，選擇 GetPetsInfo 旁的按鈕，然後依序選擇 Actions (動作) 及 Delete (刪除)。出現提示時，再次選擇 Delete (刪除)。

### 刪除相關聯的 IAM 資源

1. Open the IAM console at <https://console.aws.amazon.com/iam/>.

2. 在 Details (詳細資訊) 區域中，選擇 Roles (角色)。
3. 選取 APIGatewayLambdaExecRole，然後依序選擇 Role Actions (角色動作) 及 Delete Role (刪除角色)。出現提示時，選擇 Yes, Delete (是，刪除)。
4. 在 Details (詳細資訊) 區域中，選擇 Policies (政策)。
5. 選擇 APIGatewayLambdaExecPolicy、Policy Actions (政策動作)，再選擇 Delete (刪除)。出現提示時，選擇 Delete (刪除)。

這份演練到此結束。

## 教學：建置具有 API Gateway 私有整合的 API

您可以建立具有私有整合的 API Gateway API，讓您的客戶能夠在 Amazon Virtual Private Cloud (Amazon VPC) 內存取 HTTP/HTTPS 資源。這種 VPC 資源是 VPC 中，網路負載平衡器後方之 EC2 執行個體上的 HTTP/HTTPS 端點。網路負載平衡器會封裝 VPC 資源，並將內送請求路由到目標資源。

當用戶端呼叫 API 時，API Gateway 會透過預先設定的 VPC 連結連線到網路負載平衡器。VPC 連結由 [VpcLink](#) 的 API Gateway 資源封裝。它負責將 API 方法請求轉送到 VPC 資源，再將後端回應傳回給發起人。對於 API 開發人員而言，[VpcLink](#) 功能等同於整合端點。

若要建立具有私有整合的 API，您必須建立新的 [VpcLink](#)，或選擇已連線到網路負載平衡器的現有 [VpcLink](#)，且該網路負載平衡器是以所需的 VPC 資源為目標。您必須擁有[適當的許可 \(p. 229\)](#)，才能建立和管理 [VpcLink](#)。然後，您要設定 API 方法，將 HTTP 或 HTTP\_PROXY 設定為整合類型、將 VPC\_LINK 設定為整合連線類型，並設定整合 [connectionId](#) 上的 [VpcLink](#) 識別符，整合此方法與 [VpcLink](#)。

為快速開始建立 API 以存取 VPC 資源，我們會使用 API Gateway 主控台逐步介紹建立具有私有整合 API 的重要步驟。建立 API 之前，請執行下列操作：

1. 建立 VPC 資源，在同區域的您的帳戶下建立或選擇網路負載平衡器，然後新增 EC2 執行個體，將資源裝載為網路負載平衡器的目標。如需更多詳細資訊，請參閱 [設定 API Gateway 私有整合的網路負載平衡器 \(p. 229\)](#)。
2. 授予建立私有整合 VPC 連結的許可。如需更多詳細資訊，請參閱 [授予建立 VPC 連結的許可 \(p. 229\)](#)。

使用在目標群組中設定的 VPC 資源建立您的 VPC 資源和網路負載平衡器之後，請依下列說明來建立 API，並在私有整合中透過 [VpcLink](#) 整合它與 VPC 資源。

### 使用 API Gateway 主控台建立具有私有整合的 API

1. 登入 API Gateway 主控台，在導覽列選擇一個區域，例如 us-west-2。
2. 如果您尚未完成此操作，請先建立 VPC 連結。
  - a. 請在主導覽窗格中選擇 VPC Links (VPC 連結)，然後選擇 + Create (+ 建立)。
  - b. 在 VPC Link (VPC 連結) 下的 Name (名稱) 欄位中，輸入名稱 (例如，my-test-vpc-link)。
  - c. (選擇性) 在 Description (說明) 文字區域中提供 VPC 連結的描述。
  - d. 從 Target NLB (目標 NLB) 下拉式清單中選擇網路負載平衡器。

在選擇清單中所要顯示的網路負載平衡器區域中，您必須已建立網路負載平衡器。

- e. 選擇 Create (建立) 開始建立 VPC 連結。

初始回應傳回的 [VpcLink](#) 資源表示中有 VPC 連結 ID 和 PENDING 狀態。因為這是非同步操作，完成時間約需 2-4 分鐘。成功完成時的狀態為 AVAILABLE。與此同時，您可以繼續建立 API。

3. 從主導覽窗格中選擇 APIs，然後選擇 + Create API (+ 建立 API) 建立邊緣最佳化或區域性端點類型的新 API。

4. 針對根資源 (/)，請從 Actions (動作) 下拉式選單中選擇 Create Method (建立方式)，然後選擇 GET。
5. 在 / GET - Setup (/ GET - 設定) 窗格中，初始化 API 方法整合，如下所示：
  - a. 針對 Integration type (整合類型) 選擇 VPC Link。
  - b. 選擇 Use Proxy Integration (使用代理整合)。
  - c. 從 Method (方法) 下拉式清單中，選擇 GET 做為整合方法。
  - d. 從 VPC Link (VPC 連結) 下拉式清單中，選擇 [Use Stage Variables]，然後在下面的文字方塊中輸入 \${stageVariables.vpcLinkId}。

我們會在將 API 部署到階段後，定義 vpcLinkId 階段變數，並將其值設定為 Step 1 (步驟 1) 建立之 VpcLink 的 ID。

- e. 針對 Endpoint URL (端點 URL) 輸入 URL，例如 <http://myApi.example.com>。

在此，主機名稱 (例如，myApi.example.com) 是用來設定整合請求的 Host 標頭。

#### Note

若為網路負載平衡器 (NLB)，請務必依[網路負載平衡器入門](#)所述使用 NLB DNS 名稱。

- f. 請將 Use Default Timeout (使用預設逾時) 選項保持現狀，除非您想要自訂整合逾時。
  - g. 選擇 Save (儲存) 完成整合設定。
- 使用代理整合，API 已備妥可供部署。否則，您需要繼續設定適當的方法回應和整合回應。
- h. 從 Actions (動作) 下拉式選單中，選擇 Deploy API (部署 API)，然後選擇新的或現有的階段來部署 API。

請注意產生的 Invoke URL (呼叫 URL)。您需要它來呼叫 API。執行這項操作之前，您必須設定 vpcLinkId 階段變數。

- i. 在 Stage Editor (階段編輯器) 中，選擇 Stage Variables (階段變數) 標籤，然後選擇 Add Stage Variable (新增階段變數)。
  - i. 在 Name (名稱) 欄位下，輸入 vpcLinkId。
  - ii. 在 Value (值) 欄位下，輸入 VPC\_LINK 的 ID，例如 gix6s7。
  - iii. 選擇核取記號圖示儲存此階段變數。

使用階段變數，您可以透過變更階段變數值，輕鬆切換到 API 的不同 VPC 連結。

這樣就完成 API 的建立工作。您可以如同其他整合來測試呼叫 API。

## 教學：建置具有 AWS 整合的 API Gateway API

[教學：建置具有 Lambda 代理整合的 Hello World API \(p. 24\)](#)和[建置具有 Lambda 整合的 API Gateway API \(p. 24\)](#)主題皆說明如何建立 API Gateway API 來公布整合的 Lambda 函數。此外，您也可以建立 API Gateway API 來公布其他 AWS 服務，例如，Amazon SNS、Amazon S3、Amazon Kinesis，甚至是 AWS Lambda。這可透過 AWS 整合進行。Lambda 整合或 Lambda 代理整合都是一種特殊案例，會透過 API Gateway API 來公布 Lambda 函數呼叫。

所有 AWS 服務都支援專用 API 來公布其功能。不過，應用程式協定或程式設計界面可能會因服務而不同。具有 AWS 整合的 API Gateway API 可提供一致的應用程式協定，讓您的用戶端可存取不同的 AWS 服務。

在本演練中，我們會建立 API 來公布 Amazon SNS。如需整合 API 與其他 AWS 服務的更多範例，請參閱教學課程 ([p. 24](#))。

與 Lambda 代理整合不同，其他 AWS 服務沒有對應的代理整合。因此，API 方法會與單一 AWS 動作整合。與代理整合類似，如需更多的彈性，您可以設定 Lambda 代理整合。Lambda 函數會接著剖析和處理其他 AWS 動作的請求。

如果端點逾時，則 API Gateway 不會重試。API 發起人必須實作重試邏輯來處理端點逾時。

本演練以「[建置具有 Lambda 整合的 API Gateway API \(p. 24\)](#)」中的說明和概念為基礎。如果您尚未完成該演練，建議您先進行該演練。

#### 主題

- [事前準備 \(p. 82\)](#)
- [步驟 1：建立資源 \(p. 82\)](#)
- [步驟 2：建立 GET 方法 \(p. 82\)](#)
- [步驟 3：建立 AWS 服務代理執行角色 \(p. 83\)](#)
- [步驟 4：指定方法設定並測試方法 \(p. 84\)](#)
- [步驟 5：部署 API \(p. 85\)](#)
- [步驟 6：測試 API \(p. 85\)](#)
- [步驟 7：清除 \(p. 85\)](#)

## 事前準備

開始此演練前，請執行下列操作：

1. 完成「[先決條件：準備在 API Gateway 中建置 API \(p. 9\)](#)」中的步驟。
2. 確保 IAM 使用者具有在 IAM 中建立政策和角色的存取權。您需要在本演練中建立 IAM 政策和角色。
3. 建立名為 MyDemoAPI 的新 API。如需詳細資訊，請參閱 [教學：建置具有 HTTP 非代理整合的 API \(p. 52\)](#)。
4. 至少將 API 部署至名為 test 的階段一次。如需詳細資訊，請參閱 [建置具有 Lambda 整合的 API Gateway API \(p. 24\) 中的部署 API \(p. 38\)](#)。
5. 完成 [建置具有 Lambda 整合的 API Gateway API \(p. 24\)](#) 中的其餘步驟。
6. 在 Amazon Simple Notification Service (Amazon SNS) 中建立至少一個主題。您將使用已部署的 API 來取得 Amazon SNS 中與 AWS 帳戶相關聯的主題清單。若要了解如何在 Amazon SNS 中建立主題，請參閱 [建立主題](#)。(您不需要複製步驟 5 中提到的主題 ARN。)

## 步驟 1：建立資源

在此步驟中，您會建立讓 AWS 服務代理可與 AWS 服務互動的資源。

#### 建立資源

1. 登入位於 <https://console.aws.amazon.com/apigateway> 的 API Gateway 主控台。
2. 如果顯示 MyDemoAPI，請選擇 Resources (資源)。
3. 在 Resources (資源) 窗格中，選擇資源根，其以一條正斜線表示 (/)，然後選擇 Create Resource (建立資源)。
4. 針對 Resource Name (資源名稱)，輸入 **MyDemoAWSProxy**，然後選擇 Create Resource (建立資源)。

## 步驟 2：建立 GET 方法

在此步驟中，您會建立讓 AWS 服務代理可與 AWS 服務互動的 GET 方法。

#### 建立 GET 方法

1. 在 Resources (資源) 窗格中，選擇 /mydemoawsproxy，然後選擇 Create Method (建立方法)。
2. 對於 HTTP 方法，選擇 GET，然後儲存您的選擇。

## 步驟 3：建立 AWS 服務代理執行角色

在此步驟中，您會建立 AWS 服務代理用來與 AWS 服務互動的 IAM 角色。我們將這個 IAM 角色稱為 AWS 服務代理執行角色。如果沒有此角色，API Gateway 就不能與 AWS 服務互動。在後續步驟中，您會在剛建立之 GET 方法的設定中指定此角色。

### 建立 AWS 服務代理執行角色和其政策

1. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.
2. 選擇 Policies (政策)。
3. 請執行下列其中一項：
  - 如果顯示 Welcome to Managed Policies (歡迎使用受管政策) 頁面，請選擇 Get Started (開始使用)，然後選擇 Create Policy (建立政策)。
  - 如果顯示政策清單，請選擇 Create Policy (建立政策)。
4. 在 Create Your Own Policy (建立您自己的政策) 旁邊，選擇 Select (選取)。
5. 針對 Policy Name (政策名稱)，輸入政策的名稱 (例如，**APIGatewayAWSProxyExecPolicy**)。
6. 針對 Description (描述)，輸入 **Enables API Gateway to call AWS services**。
7. 針對 Policy Document (政策文件)，輸入下列項目，然後選擇 Create Policy (建立政策)。

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Resource": ["  
                "*"  
            ],  
            "Action": [  
                "sns>ListTopics"  
            ]  
        }  
    ]  
}
```

此政策文件可讓發起人取得 AWS 帳戶的 Amazon SNS 主題清單。

8. 選擇 Roles (角色)。
9. 選擇 Create Role (建立角色)。
10. 選擇 Select role type (選取角色類型) 下的 AWS Service (AWS 服務)，然後選擇 API Gateway (API 通道)。
11. 選擇 Next: Permissions (下一步：許可)。
12. 選擇 Next:Review (下一步：檢閱)。
13. 針對 Role Name (角色名稱)，輸入執行角色的名稱 (例如，**APIGatewayAWSProxyExecRole**)，並可選擇輸入此角色的描述，然後選擇 Create role (建立角色)。
14. 在 Roles (角色) 清單中，選擇您剛剛建立的角色。您可能需要向下捲動清單。
15. 對於選取的角色，選擇 Attach policy (連接政策)。
16. 選取您先前建立之政策 (例如，**APIGatewayAWSProxyExecPolicy**) 旁邊的核取方塊，然後選擇 Attach policy (連接政策)。
17. 您剛剛建立的角色具有下列信任關係，讓 API Gateway 可擔任已連接政策所允許之任何動作的角色：

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Principal": "  
                arn:aws:iam::  
                    account-id:  
                        role/  
                            APIGatewayAWSProxyExecRole"  
            ",  
            "Action": [  
                "stsAssumeRole"  
            ]  
        }  
    ]  
}
```

```
{  
    "Sid": "",  
    "Effect": "Allow",  
    "Principal": {  
        "Service": "apigateway.amazonaws.com"  
    },  
    "Action": "sts:AssumeRole"  
}  
]  
}
```

針對 Role ARN (角色 ARN)，請記下執行角色的 Amazon Resource Name (ARN)。供稍後使用。ARN 應該類似：`arn:aws:iam::123456789012:role/APIGatewayAWSProxyExecRole`，其中 `123456789012` 是您的 AWS 帳戶 ID。

## 步驟 4：指定方法設定並測試方法

在此步驟中，您會指定 GET 方法的設定，讓它可以透過 AWS 服務代理與 AWS 服務互動。然後您會測試方法。

### 指定 GET 方法的設定，然後加以測試

1. 在 API Gateway 主控台中，於名為 MyDemoAPI 之 API 的 Resources (資源) 窗格中，選擇 /mydemawsproxy 中的 GET。
2. 在 Setup (設定) 窗格的 Integration type (整合類型) 中，選擇 Show advanced (顯示進階)，然後選擇 AWS Service Proxy (&AWS; 服務代理)。
3. 針對 AWS Region (&AWS; 區域)，選擇您要取得 Amazon SNS 主題之 AWS 區域的名稱。
4. 針對 AWS Service (&AWS; 服務)，選擇 SNS。
5. 針對 HTTP method (HTTP 方法)，選擇 GET。
6. 針對 Action (動作) 輸入 **ListTopics**。
7. 針對 Execution Role (執行角色)，輸入執行角色的 ARN。
8. 將 Path Override (路徑覆寫) 保留空白。
9. 選擇 Save (儲存)。
10. 在 Method Execution (方法執行) 窗格的 Client (用戶端) 方塊中，選擇 TEST (測試)，然後選擇 Test (測試)。若成功，Response Body (回應內文) 會顯示與下列類似的回應：

```
{  
    "ListTopicsResponse": {  
        "ListTopicsResult": {  
            "NextToken": null,  
            "Topics": [  
                {  
                    "TopicArn": "arn:aws:sns:us-east-1:80398EXAMPLE:MySNSTopic-1"  
                },  
                {  
                    "TopicArn": "arn:aws:sns:us-east-1:80398EXAMPLE:MySNSTopic-2"  
                },  
                ...  
                {  
                    "TopicArn": "arn:aws:sns:us-east-1:80398EXAMPLE:MySNSTopic-N"  
                }  
            ]  
        },  
        "ResponseMetadata": {  
            "RequestId": "abc1de23-45fa-6789-b0c1-d2e345fa6b78"  
        }  
    }  
}
```

}

## 步驟 5：部署 API

在此步驟中，您會部署 API，以從 API Gateway 主控台外部呼叫 API。

### 部署 API

1. 在 Resources (資源) 窗格中，選擇 Deploy API (部署 API)。
2. 針對 Deployment stage (部署階段)，選擇 test。
3. 針對 Deployment description (部署描述)，輸入 Calling AWS service proxy walkthrough。
4. 選擇 Deploy (部署)。

## 步驟 6：測試 API

在此步驟中，您會前往 API Gateway 主控台外部，並使用 AWS 服務代理與 Amazon SNS 服務互動。

1. 在 Stage Editor (階段編輯器) 窗格中，將 Invoke URL (呼叫 URL) 旁的 URL 複製到剪貼簿。它應該如下所示：

`https://my-api-id.execute-api.region-id.amazonaws.com/test`

2. 將 URL 貼入新瀏覽器標籤的網址方塊中。
3. 附加 /mydemoawsproxy，然後其看起來如下：

`https://my-api-id.execute-api.region-id.amazonaws.com/test/mydemoawsproxy`

瀏覽至該 URL。應該會顯示類似如下的資訊：

```
{"ListTopicsResponse": {"ListTopicsResult": {"NextToken": null, "Topics": [{"TopicArn": "arn:aws:sns:us-east-1:80398EXAMPLE:MySNSTopic-1"}, {"TopicArn": "arn:aws:sns:us-east-1:80398EXAMPLE:MySNSTopic-2"}, {"TopicArn": "arn:aws:sns:us-east-1:80398EXAMPLE:MySNSTopic-N"}]}, "ResponseMetadata": {"RequestId": "abc1de23-45fa-6789-b0c1-d2e345fa6b78"}}}
```

## 步驟 7：清除

您可以刪除 AWS 服務代理需要處理的 IAM 資源。

### Warning

如果您刪除 AWS 服務代理相依的 IAM 資源，則該 AWS 服務代理和所有與其相依的 API 將無法繼續運作。而刪除 IAM 資源無法復原。如果您要再次使用 IAM 資源，則必須予以重建。

### 刪除相關聯的 IAM 資源

1. Open the IAM console at <https://console.aws.amazon.com/iam/>.
2. 在 Details (詳細資訊) 區域中，選擇 Roles (角色)。
3. 選取 APIGatewayAWSProxyExecRole，然後依序選擇 Role Actions (角色動作) 及 Delete Role (刪除角色)。出現提示時，選擇 Yes, Delete (是，刪除)。
4. 在 Details (詳細資訊) 區域中，選擇 Policies (政策)。

5. 選擇 APIGatewayAWSProxyExecPolicy、Policy Actions (政策動作)，再選擇 Delete (刪除)。出現提示時，選擇 Delete (刪除)。

本演練到此結束。如需建立 API 做為 AWS 服務代理的更詳細討論，請參閱「[教學：在 API Gateway 中建立 REST API 做為 Amazon S3 代理 \(p. 104\)](#)」、「[教學課程：使用兩個 AWS 服務整合和一個 Lambda 非代理整合建立 Calc REST API \(p. 86\)](#)」或「[教學：在 API Gateway 中建立 REST API 做為 Amazon Kinesis 代理 \(p. 129\)](#)」。

## 教學課程：使用兩個 AWS 服務整合和一個 Lambda 非代理整合建立 Calc REST API

[非代理整合入門教學 \(p. 31\)](#)只會使用 Lambda Function 整合。Lambda Function 整合是 AWS Service 整合類型的特殊案例，此整合類型會為您執行許多整合設定，例如自動新增 Lambda 函數呼叫所需的資源型許可。在這裡，三種整合的兩種會使用 AWS Service 整合。在此整合類型中，您具有更多的控制，但需要手動執行任務，像是建立和指定 IAM 角色，其中包含適當的許可。

在本教學中，您將需要建立 Calc Lambda 函數來實作基本算術運算，並接受和傳回 JSON 格式的輸入和輸出。然後，您將建立一個 REST API，並使用下列方式將其與 Lambda 函數整合：

1. 在 /calc 資源上公開 GET 方法來叫用 Lambda 函數，並提供輸入做為查詢字串參數 (AWS Service 整合)
2. 在 /calc 資源上公開 POST 方法來叫用 Lambda 函數，並在方法請求承載中提供輸入 (AWS Service 整合)
3. 在巢狀 /calc/{operand1}/{operand2}/{operator} 資源上公開 GET 來叫用 Lambda 函數，並提供輸入做為路徑參數 (Lambda Function 整合)

除了試做本教學之外，您可能還想要研究 Calc API 的 [OpenAPI 定義檔 \(p. 100\)](#)，您可以依照[the section called “匯入 REST API” \(p. 319\)](#)中的指示，將此定義檔匯入至 API Gateway。

### 主題

- [建立 AWS 帳戶 \(p. 86\)](#)
- [建立可擔任的 IAM 角色 \(p. 87\)](#)
- [建立 Calc Lambda 函數 \(p. 88\)](#)
- [測試 Calc Lambda 函數 \(p. 88\)](#)
- [建立 Calc API \(p. 90\)](#)
- [整合 1：建立 GET 方法與查詢參數搭配來呼叫 Lambda 函數 \(p. 91\)](#)
- [整合 2：建立 POST 方法與 JSON 承載搭配來呼叫 Lambda 函數 \(p. 93\)](#)
- [整合 3：建立 GET 方法與路徑參數搭配來呼叫 Lambda 函數 \(p. 95\)](#)
- [與 Lambda 函數整合之範例 API 的 OpenAPI 定義 \(p. 100\)](#)

## 建立 AWS 帳戶

您將需要一個 AWS 帳戶，才能開始本教學。

If you do not have an AWS account, complete the following steps to create one.

To sign up for an AWS account

1. Open <https://portal.aws.amazon.com/billing/signup>.

2. Follow the online instructions.

Part of the sign-up procedure involves receiving a phone call and entering a verification code on the phone keypad.

## 建立可擔任的 IAM 角色

為了讓 API 叫用 Calc Lambda 函數，您將需要具有 API Gateway 可擔任 IAM 角色，這是具有以下信任關係的 IAM 角色：

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

您建立的角色將需要具有 Lambda [InvokeFunction](#) 許可。否則，API 發起人將會收到 500 Internal Server Error 回應。若要將此許可給與角色，請將下列 IAM 政策連接至其中：

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": "lambda:InvokeFunction",  
            "Resource": "*"  
        }  
    ]  
}
```

以下是如何完成此全部操作的方式：

### 建立 API Gateway 可擔任 IAM 角色

1. 登入 IAM 主控台。
2. 選擇 Roles (角色)。
3. 選擇 Create Role (建立角色)。
4. 在 Select type of trusted entity (選擇信任的實體類型) 下，選擇 AWS Service (AWS 服務)。
5. 在 Choose the service that will use this role (選擇將使用此角色的服務) 下，選擇 Lambda。
6. 選擇 Next: Permissions (下一步：許可)。
7. 選擇 Create Policy (建立政策)。

Create Policy (建立政策) 主控台將開啟新的時段。在該視窗中，執行下列作業：

- a. 在 JSON 標籤中，用以下政策取代現有政策。

```
{  
    "Version": "2012-10-17",
```

```
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "lambda:InvokeFunction",
            "Resource": "*"
        }
    ]
}
```

- b. 選擇 Review policy (檢閱政策)。
- c. 在 Review Policy (檢閱政策) 下，執行下列操作：
  - i. 在 Name (名稱) 中輸入名稱，例如 `lambda_execute`。
  - ii. 選擇 Create Policy (建立政策)。
8. 在原本的 Create Role (建立角色) 主控台視窗中，執行下列動作：
  - a. 在 Attach permissions policies (連接許可政策) 下方，從下拉式清單中選擇您的 `lambda_execute` 政策。

如果您在清單中沒有看到您的政策，請選擇清單頂端的重新整理按鈕。(請不要重新整理瀏覽器頁面！)
  - b. 選擇 Next: Add Tags (下一步：新增標籤)。
  - c. 選擇 Next:Review (下一步：檢閱)。
  - d. 在 Role name (角色名稱) 中輸入名稱，例如 `lambda_invoke_function_assume_apigw_role`。
  - e. 選擇 Create Role (建立角色)。
9. 從角色清單中選擇您的 `lambda_invoke_function_assume_apigw_role`。
10. 選擇 Trust Relationships (信任關係) 標籤。
11. 選擇 Edit trust relationship (編輯信任關係)。
12. 用以下內容取代現有政策：

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

13. 選擇 Update Trust Policy (更新信任政策)。
14. 請記住您剛建立角色的角色 ARN。以供稍後使用。

## 建立 Calc Lambda 函數

接下來，您將使用 Lambda 主控台建立 Lambda 函數。

1. 在 Lambda 主控台中，請選擇 Create function (建立函數)。
2. 選擇 Author from Scratch (從頭開始撰寫)。
3. 在 Name (名稱) 輸入 **Calc**。
4. 將 Runtime (執行時間) 設定為 Node.js 8.10。
5. 選擇 Create function (建立函數)。
6. 在 Lambda 主控台中，複製下列 Lambda 函數，並將其貼入程式碼編輯器。

```
console.log('Loading the Calc function');

exports.handler = function(event, context, callback) {
    console.log('Received event:', JSON.stringify(event, null, 2));
    if (event.a === undefined || event.b === undefined || event.op === undefined) {
        callback("400 Invalid Input");
    }

    var res = {};
    res.a = Number(event.a);
    res.b = Number(event.b);
    res.op = event.op;

    if (isNaN(event.a) || isNaN(event.b)) {
        callback("400 Invalid Operand");
    }

    switch(event.op)
    {
        case "+":
        case "add":
            res.c = res.a + res.b;
            break;
        case "-":
        case "sub":
            res.c = res.a - res.b;
            break;
        case "*":
        case "mul":
            res.c = res.a * res.b;
            break;
        case "/":
        case "div":
            res.c = res.b==0 ? NaN : Number(event.a) / Number(event.b);
            break;
        default:
            callback("400 Invalid Operator");
            break;
    }
    callback(null, res);
};
```

7. 在執行角色下，選擇 Choose an existing role (選擇現有的角色)。
8. 為您先前建立的 **lambda\_invoke\_function\_assume\_apigw\_role** 角色輸入角色 ARN。
9. 選擇 Save (儲存)。

此函數需要 a 輸入參數有兩個運算元 (b 和 op) 和一個運算子 (event)。輸入是下列格式的 JSON 物件：

```
{
    "a": "Number" | "String",
    "b": "Number" | "String",
    "op": "String"
```

```
}
```

此函數會傳回計算的結果 (c) 和輸入。對於無效的輸入，該函數會傳回 null 值或「無效 op」字串做為結果。輸出具有下列 JSON 格式：

```
{
  "a": "Number",
  "b": "Number",
  "op": "String",
  "c": "Number" | "String"
}
```

您應該先在 Lambda 主控台中測試函數，再於後續步驟中將它與 API 整合。

## 測試 Calc Lambda 函數

以下是在 Lambda 主控台測試 Calc 函數的方式：

1. 在 Saved test events (儲存的測試事件) 下拉式功能表中，選擇 Configure test events (設定測試事件)。
2. 針對測試事件名稱，輸入 **calc2plus5**。
3. 將測試事件定義取代為下列內容：

```
{
  "a": "2",
  "b": "5",
  "op": "+"
}
```

4. 選擇 Save (儲存)。
5. 選擇 Test (測試)。
6. 展開 Execution result: succeeded (執行結果：成功)。請查看下列事項：

```
{
  "a": 2,
  "b": 5,
  "op": "+",
  "c": 7
}
```

## 建立 Calc API

下列程序說明如何為您剛建立的 Calc Lambda 函數建立 API。在後續幾節中，您會將資源和方法新增至其中。

### 建立 Calc API

1. 在 API Gateway 主控台中，選擇 Create API (建立 API)。

2. 針對 API Name (API 名稱) , 輸入 **LambdaCalc**。
3. 讓 Description (描述) 保留空白，並讓 Endpoint Type (端點類型) 保留設為 Regional (區域性)。
4. 選擇 Create API (建立 API)。

## 整合 1：建立 GET 方法與查詢參數搭配來呼叫 Lambda 函數

透過建立 GET 方法，將查詢字串參數傳遞至 Lambda 函數，您可以啟用 API，讓其可透過瀏覽器叫用。這種方法很有用，尤其適用於允許開放存取的 API。

### 設定具有查詢字串參數的 GET 方法

1. 在 API Gateway 主控台中，於 LambdaCalc API 的 Resources (資源) 下，選擇 **/**。
2. 在 Actions (動作) 下拉式功能表中，選擇 Create Resource (建立資源)。
3. 針對 Resource Name (資源名稱)，輸入 **calc**。
4. 選擇 Create Resource (建立資源)。
5. 選擇您剛建立的 **/calc** 資源。
6. 從 Actions (動作) 下拉式選單，選擇 Create Method (建立方法)。
7. 從出現的方法下拉式選單中，選擇 GET。
8. 選擇核取記號圖示來儲存選擇。
9. 在 Set up (設定) 窗格中：
  - a. 針對 Integration type (整合類型)，選擇 AWS Service (AWS 服務)。
  - b. 針對 AWS Region (AWS 區域)，選擇您針對 Lambda 函數所建立的區域 (例如 **us-west-2**)。
  - c. 針對 AWS Service (AWS 服務)，選擇 Lambda。
  - d. 將 AWS Subdomain (AWS 子網域) 空白，因為我們未在任何 AWS 子網域上託管 Lambda 函數。
  - e. 對於 HTTP method (HTTP 方法) 選擇 POST，然後選擇核取記號圖示來儲存您的選擇。Lambda 需要使用 POST 請求來呼叫任何 Lambda 函數。此範例示範前端方法請求中的 HTTP 方法可以與後端中的整合請求不同。
  - f. 針對 Action Type (動作類型) 選擇 Use path override。這個選項可讓我們指定 **Invoke** 動作的 ARN，來執行 Calc 函數。
  - g. 在 Path override (路徑覆寫) 中輸入 **/2015-03-31/functions/  
arn:aws:lambda:*region:account-id:function:Calc/invocations***，其中 **region** 是您建立 Lambda 函數的區域，而 **account-id** 是 AWS 帳戶的帳號。
  - h. 針對 Execution role (執行角色)，輸入您稍早 (p. 87) 建立之 **lambda\_invoke\_function\_assume\_apigw\_role** IAM 角色的角色 ARN。
  - i. 將 Content Handling (內容處理) 保留設定為 Passthrough (傳遞)，因為此方法將不會處理任何二進位資料。
  - j. 將 Use default timeout (使用預設逾時) 保留勾選。
  - k. 選擇 Save (儲存)。
10. 選擇 Method Request (方法請求)。

現在您可以在 **/calc** 上設定 GET 方法的查詢參數，用以代表後端 Lambda 函數接收輸入。

- a. 選擇 Request Validator (請求驗證程式) 旁的鉛筆圖示，然後從下拉式選單中選擇 Validate query string parameters and headers (驗證查詢字串參數和標頭)。如果用戶端未指定所需參數，則此設定會造成錯誤訊息還原為遺失所需參數的狀態。您將不會支付後端呼叫的費用。
- b. 選擇核取記號圖示來儲存變更。

- c. 展開 URL Query String Parameters (URL 查詢字串參數) 區段。
  - d. 選擇 Add query string (新增查詢字串)。
  - e. 在 Name (名稱) 輸入 **operand1**。
  - f. 選擇核取記號圖示以儲存參數。
  - g. 重複上述步驟，建立名為 **operand2** 和 **operator** 的參數。
  - h. 勾選每個參數的 Required (必要) 選項，確保它們已進行驗證。
11. 選擇 Method Execution (方法執行) 然後選擇 Integration Request (整合請求) 來設定對應範本，以將用戶端提供的查詢字串翻譯為 Calc 函數所需的整合請求承載。
- a. 展開 Mapping Templates (對應範本) 區段。
  - b. 為 Request body passthrough (請求內文傳遞) 選擇 When no template matches the request Content-Type header (沒有範本符合請求內容類型標頭時)。
  - c. 在 Content-Type (內容類型) 下方，選擇 Add mapping template (新增對應範本)。
  - d. 輸入 **application/json**，然後選擇核取記號圖示來開啟範本編輯器。
  - e. 選擇 Yes, secure this integration (是，保護此整合) 繼續進行。
  - f. 將下列對應指令碼複製到對應範本編輯器：

```
{  
    "a": "$input.params('operand1')",  
    "b": "$input.params('operand2')",  
    "op": "$input.params('operator')"  
}
```

此範本會將 Method Request (方法請求) 中所宣告的三個查詢字串參數對應至 JSON 物件的指定屬性值，做為後端 Lambda 函數的輸入。轉換過的 JSON 物件將會包含為整合請求承載。

- g. 選擇 Save (儲存)。
12. 選擇 Method Execution (方法執行)。
13. 您現在可以測試您的 GET 方法，確認它已正確設定，可以叫用 Lambda 函數：
- a. 對於 Query Strings (查詢字串)，輸入 **operand1=2&operand2=3&operator=+**。
  - b. 選擇 Test (測試)。

結果看起來會與下列類似：

**Request: /calc?operand1=2&operand2=3&operator=+**

**Status: 200**

**Latency: 414 ms**

**Response Body**

```
{  
    "a": 2,  
    "b": 3,  
    "op": "+",  
    "c": 5  
}
```

## 整合 2：建立 POST 方法與 JSON 承載搭配來呼叫 Lambda 函數

透過建立 POST 方法與 JSON 承載搭配來呼叫 Lambda 函數，讓用戶端必須在請求內文中提供後端函數的必要輸入。為了確保用戶端上傳正確的輸入資料，您將在承載上啟用請求驗證。

### 設定 POST 方法與 JSON 承載搭配來叫用 Lambda 函數

1. 前往 API Gateway 主控台。
2. 選擇 API。
3. 選擇先前建立的 LambdaCalc API。
4. 從 Resources (資源) 窗格中，選擇 /calc 資源。
5. 從 Actions (動作) 選單，選擇 Create Method (建立方法)。
6. 從方法下拉式清單中，選擇 POST。
7. 選擇核取記號圖示來儲存選擇。
8. 在 Set up (設定) 窗格中：
  - a. 針對 Integration type (整合類型)，選擇 AWS Service (AWS 服務)。
  - b. 針對 AWS Region (AWS 區域)，選擇您針對 Lambda 函數所建立的區域 (例如us-west-2)。
  - c. 針對 AWS Service (AWS 服務)，選擇 Lambda。
  - d. 將 AWS Subdomain (AWS 子網域) 空白，因為我們未在任何 AWS 子網域上託管 Lambda 函數。
  - e. 針對 HTTP method (HTTP 方法)，選擇 POST。此範例示範前端方法請求中的 HTTP 方法可以與後端中的整合請求不同。
  - f. 針對 Action (動作)，選擇 Use path override 做為 Action Type (動作類型)。這個選項可讓您指定 **Invoke** 動作的 ARN，來執行您的 Calc 函數。
  - g. 針對 Path override (路徑覆寫)，輸入 /2015-03-31/functions/  
`arn:aws:lambda:region:account-id:function:Calc/invocations`，其中 **region** 是您建立 Lambda 函數的區域，而 **account-id** 是 AWS 帳戶的帳號。
  - h. 針對 Execution role (執行角色)，輸入您稍早 (p. 87) 建立之 `lambda_invoke_function_assume_apigw_role` IAM 角色的角色 ARN。
  - i. 將 Content Handling (內容處理) 保留設定為 Passthrough (傳遞)，因為此方法將不會處理任何二進位資料。
  - j. 將 Use Default Timeout (使用預設逾時) 保留勾選。
  - k. 選擇 Save (儲存)。
9. 在 API Gateway 主控台的主導覽窗格中，選擇 LambdaCalc API 下的 Models (模型)，來建立方法輸入和輸出的資料模型：
  - a. 在 Models (模型) 窗格中，選擇 Create (建立)。在 Model name (模型名稱) 中輸入 Input、在 Content type (內容類型) 中輸入 application/json，然後將下列結構描述定義複製到 Model schema (模型結構描述)：

```
{  
    "type": "object",  
    "properties": {  
        "a": {"type": "number"},  
        "b": {"type": "number"},  
        "op": {"type": "string"}  
    },  
    "title": "Input"  
}
```

此模型說明輸入資料結構，將用來驗證傳入請求內文。

- b. 選擇 Create model (建立模型)。
- c. 在 Models (模型) 窗格中，選擇 Create (建立)。在 Model name (模型名稱) 中輸入 Output、在 Content type (內容類型) 中輸入 application/json，然後將下列結構描述定義複製到 Model schema (模型結構描述)：

```
{  
    "type": "object",  
    "properties": {  
        "c": {"type": "number"}  
    },  
    "title": "Output"  
}
```

此模型說明來自後端之計算輸出的資料結構。它可以用來將整合回應資料對應至不同模型。本教學依賴傳遞行為，而且不會使用此模型。

- d. 在主控台畫面上方尋找您 API 的 API ID，並記下它。它會出現在 API 名稱後面的括號中。
- e. 在 Models (模型) 窗格中，選擇 Create (建立)。
- f. 在 Model name (模型名稱) 中輸入 Result。
- g. 在 Content type (內容類型) 中輸入 application/json。
- h. 將下列結構描述定義 (其中 *restapi-id* 是您先前記下的 REST API ID) 複製到 Model schema (模型結構描述) 方塊：

```
{  
    "type": "object",  
    "properties": {  
        "input": {  
            "$ref": "https://apigateway.amazonaws.com/restapis/restapi-id/models/  
Input"  
        },  
        "output": {  
            "$ref": "https://apigateway.amazonaws.com/restapis/restapi-id/models/  
Output"  
        }  
    },  
    "title": "Output"  
}
```

此模型說明所傳回回應資料的資料結構。它參考所指定 API (Input) 中定義的 Output 和 *restapi-id* 結構描述。同樣地，本教學不會使用此模型，因為它利用傳遞行為。

- i. 選擇 Create model (建立模型)。
10. 在主導覽窗格的 LambdaCalc API 下，選擇 Resources (資源)。
11. 在 Resources (資源) 窗格中，選擇 API 的 POST 方法。
12. 選擇 Method Request (方法請求)。
13. 在 Method Request (方法請求) 組態設定中，執行下列操作，以啟用傳入請求內文上的請求驗證：
  - a. 選擇 Request Validator (請求驗證程式) 旁邊的鉛筆圖示，以選擇 Validate body。選擇核取記號圖示來儲存選擇。
  - b. 展開 Request Body (請求內文) 區段，並選擇 Add model (新增模型)
  - c. 在 Content-Type 輸入欄位中輸入 application/json，並從 Model name (模型名稱) 資料行的下拉式清單中選擇 Input。選擇核取記號圖示來儲存選擇。
14. 若要測試 POST 方法，請執行以下動作：
  - a. 選擇 Method Execution (方法執行)。
  - b. 選擇 Test (測試)。

15. 將下列 JSON 承載複製到 Request Body (請求內文)：

```
{  
    "a": 1,  
    "b": 2,  
    "op": "+"  
}
```

16. 選擇 Test (測試)。

您應該會在 Response Body (回應內文) 中看到以下輸出：

```
{  
    "a": 1,  
    "b": 2,  
    "op": "+",  
    "c": 3  
}
```

## 整合 3：建立 GET 方法與路徑參數搭配來呼叫 Lambda 函數

現在，您將在透過一系列路徑參數所指定的資源上建立 GET 方法，來呼叫後端 Lambda 函數。路徑參數值指定 Lambda 函數的輸入資料。您將會定義對應範本，以將傳入路徑參數值對應至所需的整合請求承載。

此時您將在 API Gateway 主控台中使用內建的 Lambda 整合支援來設定方法整合。

產生的 API 資源結構看起來像這樣：

The screenshot shows the AWS API Gateway Resources list. On the left, there's a tree view of resources under the root '/'. The first node is '/calc', which has two methods: POST and GET. Below '/calc' is another node '/{operator1}', which has two sub-nodes: '/{operator2}' and '/{operator}'. Each of these sub-nodes has a single GET method. The '/calc' node and its sub-nodes are circled in red.

### 設定 GET 方法與 URL 路徑參數搭配

1. 前往 API Gateway 主控台。
2. 在 APIs 下，選擇您先前建立的 LambdaCalc API。
3. 在 API 的 Resources (資源) 導覽窗格中，選擇 /calc。
4. 從 Actions (動作) 下拉式選單中，選擇 Create Resource (建立資源)。

5. 針對 Resource Name (資源名稱) 輸入 **{operand1}**。
6. 在 Resource Path (資源路徑) 中輸入 **{operand1}**。
7. 選擇 Create Resource (建立資源)。
8. 選擇您剛建立的 `/calc/{operand1}` 資源。
9. 從 Actions (動作) 下拉式選單中，選擇 Create Resource (建立資源)。
10. 針對 Resource Name (資源名稱) 輸入 **{operand2}**。
11. 在 Resource Path (資源路徑) 中輸入 **{operand2}**。
12. 選擇 Create Resource (建立資源)。
13. 選擇您剛建立的 `/calc/{operand1}/{operand2}` 資源。
14. 從 Actions (動作) 下拉式選單中，選擇 Create Resource (建立資源)。
15. 在 Resource Path (資源路徑) 中輸入 **{operator}**。
16. 針對 Resource Name (資源名稱) 輸入 **{operator}**。
17. 選擇 Create Resource (建立資源)。
18. 選擇您剛建立的 `/calc/{operand1}/{operand2}/{operator}` 資源。
19. 從 Actions (動作) 下拉式選單，選擇 Create Method (建立方法)。
20. 從方法下拉式選單中，選擇 GET。
21. 在 Setup (設定) 窗格中，針對 Integration type (整合類型) 選擇 Lambda Function，來使用主控台所啟用的簡化設定程序。
22. 針對 Lambda Region (&LAM; 區域) 選擇區域 (例如，us-west-2)。這個區域是託管 Lambda 函數的區域。
23. 針對 Lambda Function (&LAM; 函數) 選擇現有 Lambda 函數 (例如，calc)。
24. 選擇 Save (儲存)，然後選擇 OK (確定) 同意 Add Permissions to Lambda Function (新增 Lambda 函數的許可)。
25. 選擇 Integration Request (整合請求)。
26. 設定對應範本，如下所示：
  - a. 展開 Mapping Templates (對應範本) 區段。
  - b. 將設定保留為 When no template matches the requested Content-Type header (當沒有範本符合請求 Content-Type 標頭時)。
  - c. 選擇 Add mapping template (新增對應範本)。
  - d. 針對 Content-Type 輸入 application/json，然後選擇核取記號圖示來開啟範本編輯器。
  - e. 選擇 Yes, secure this integration (是，保護此整合) 繼續進行。
  - f. 將下列對應指令碼複製到範本編輯器：

```
{
  "a": "$input.params('operand1')",
  "b": "$input.params('operand2')",
  "op":
    #if($input.params('operator')=='%2F')"/#{else}"$input.params('operator')"#end
}
```

此範本會將建立 `/calc/{operand1}/{operand2}/{operator}` 資源時所宣告的三個 URL 路徑參數對應至 JSON 物件中的指定屬性值。因為 URL 路徑必須是 URL 編碼，所以必須將 division 運算子指定為 %2F，而非 /。此範本會先將 %2F 翻譯為 '/'，再將它傳遞給 Lambda 函數。

- g. 選擇 Save (儲存)。

方法的設定是正確時，設定應該或多或少如下所示：

◀ Method Execution /calc/{operand1}/{operand2}/{operator} - GET - Integration...

Provide information about the target backend that this method will call and whether the incoming request data should be modified.

Integration type  Lambda Function 

HTTP   
 Mock   
 AWS Service 

Use Lambda Proxy integration  

Lambda Region us-west-2 

Lambda Function Calc 

Invoke with caller credentials  

Credentials cache Do not add caller credentials to cache key 

▼ Body Mapping Templates

Request body passthrough  When no template matches the request Content-Type header   
 When there are no templates defined (recommended)   
 Never 

Content-Type
application/json

 Add mapping template

application/json

Generate template: 

```
1 ~ {  
2   "a": "$input.params('operand1')",  
3   "b": "$input.params('operand2')",  
4   "op": "#if($input.params('operator')=='%2F')/#{else}{$input.params('operator')}#end"  
5 }  
6 }
```

27. 若要測試 GET 函數，請執行以下作業：

- a. 選擇 Method Execution (方法執行)。
- b. 選擇 Test (測試)。
- c. 在 {operand1}、{operand2} 和 {operator} 欄位中，分別輸入 1、2 和 +。
- d. 選擇 Test (測試)。
- e. 結果應如下所示：

Request: /calc/1/1/+

Status: 200

Latency: 816 ms

Response Body

```
{  
  "a": 1,  
  "b": 1,  
  "op": "+",  
  "c": 2  
}
```

Response Headers

```
{"X-Amzn-Trace-Id":"sampled=0;root=1-58f7f1f6-57c26581ed7073a711c13216","Content-Type":"application/json"}
```

Logs

```
Execution log for request test-request  
Wed Apr 19 23:25:42 UTC 2017 : Starting execution for request: test-invoke-request  
Wed Apr 19 23:25:42 UTC 2017 : HTTP Method: GET, Resource Path: /calc/1/1/+  
Wed Apr 19 23:25:42 UTC 2017 : Method request path: {operand1=1, operand2=1, operator =+}  
Wed Apr 19 23:25:42 UTC 2017 : Method request query string: {}  
Wed Apr 19 23:25:42 UTC 2017 : Method request headers: {}  
Wed Apr 19 23:25:42 UTC 2017 : Method request body before transformations:  
Wed Apr 19 23:25:42 UTC 2017 : Endpoint request URI: https://lambda.us-west-2.amazonaws.com/2015-03-31/functions/arn:aws:lambda:us-west-2:738575810317:function:Calc/invocations  
Wed Apr 19 23:25:42 UTC 2017 : Endpoint request headers: {x-amzn-lambda-integration-tag=test-request, Authorization=*****}
```

此測試結果顯示後端 Lambda 函數的原始輸出，而此函數是透過整合回應所傳遞而不需要對應，因為沒有任何對應範本。接著，您將模型化 Result 結構描述後面之方法回應承載的資料結構。

28. 在預設情況下，方法回應內文會獲指派空白模型。這會造成傳遞整合回應內文，而不進行對應。不過，當您為一個強型別語言（例如 Java 或 Objective-C）產生開發套件時，開發套件使用者將會收到一個空白物件做為結果。為了確保 REST 用戶端和開發套件用戶端都收到所需的結果，您必須使用預先定義的結構描述來模型化回應資料。在這裡，您將定義方法回應內文的模型，以及如何建構對應範本，將整合回應內文翻譯為方法回應內文。
- 選擇 /calc/{operand1}/{operand2}/{operator}。
  - 選擇 GET。
  - 選擇 Method Execution (方法執行)。
  - 選擇 Method Response (方法回應)。
  - 展開 200 回應。
  - 在 Response Body for 200 (200 的回應內文) 下方選擇 application/json 內容類型模型旁邊的鉛筆圖示。
  - 從 Models (模型) 下拉式清單中，選擇 Result。
  - 選擇核取記號圖示來儲存選擇。

設定方法回應內文的模型，可確保將回應資料轉換為特定開發套件的 Result 物件。為了確保據此映射整合回應資料，您將需要映射範本。

29. 若要建立映射範本，請執行下列動作：

- a. 選擇 Method Execution (方法執行)。
- b. 選擇 Integration Response (整合回應)，並展開 200 方法回應項目。
- c. 展開 Mapping Templates (對應範本) 區段。
- d. 從 Content-Type (內容類型) 清單中選擇 application/json。
- e. 從 Generate template (產生範本) 下拉式清單中選擇 Result，以開啟 Result 範本藍圖。
- f. 將範本藍圖變更為以下內容：

```
#set($inputRoot = $input.path('$'))  
{  
    "input" : {  
        "a" : $inputRoot.a,  
        "b" : $inputRoot.b,  
        "op" : "$inputRoot.op"  
    },  
    "output" : {  
        "c" : $inputRoot.c  
    }  
}
```

- g. 選擇 Save (儲存)。

30. 若要測試映射範本，請執行下列動作：

- a. 選擇 Method Execution (方法執行)。
- b. 選擇 Test (測試)。
- c. 在 operand1、operand2 和 operator 輸入欄位中，分別輸入 1、2 和 +。

來自 Lambda 函數的整合回應現在映射至 Result 物件。

- d. 選擇 Test (測試)，您將會在主控台的 Response Body (回應內文) 下看到以下內容：

```
{  
    "input": {  
        "a": 1,  
        "b": 2,  
        "op": "+"  
    },  
    "output": {  
        "c": 3  
    }  
}
```

31. 此時只能透過 API Gateway 主控台中的 Test Invoke (測試叫用) 來呼叫 API。若要讓 API 可供用戶端使用，您將需要進行部署，如下所示：

- a. 從 Actions (動作) 下拉式功能表中，選擇 Deploy API (部署 API)。
- b. 從 Deployment Stage (部署階段) 下拉式功能表中選擇 [New Stage] ([新階段])。
- c. 針對 Stage Name (階段名稱)，輸入 **test**。
- d. 選擇 Deploy (部署)。
- e. 請注意，主控台視窗頂端的 Invoke URL (叫用 URL)。您可以使用此項與 Postman 和 cURL 這類工具搭配，來測試您的 API。

Note

每當新增、修改或刪除資源或方法、更新資料映射，或更新階段設定時，請一律務必重新部署您的 API。否則，您 API 的用戶端將無法使用新功能或更新。

## 與 Lambda 函數整合之範例 API 的 OpenAPI 定義

OpenAPI 2.0

```
{
  "swagger": "2.0",
  "info": {
    "version": "2017-04-20T04:08:08Z",
    "title": "LambdaCalc"
  },
  "host": "uojnr9hd57.execute-api.us-east-1.amazonaws.com",
  "basePath": "/test",
  "schemes": [
    "https"
  ],
  "paths": {
    "/calc": {
      "get": {
        "consumes": [
          "application/json"
        ],
        "produces": [
          "application/json"
        ],
        "parameters": [
          {
            "name": "operand2",
            "in": "query",
            "required": true,
            "type": "string"
          },
          {
            "name": "operator",
            "in": "query",
            "required": true,
            "type": "string"
          },
          {
            "name": "operand1",
            "in": "query",
            "required": true,
            "type": "string"
          }
        ],
        "responses": {
          "200": {
            "description": "200 response",
            "schema": {
              "$ref": "#/definitions/Result"
            },
            "headers": {
              "operand_1": {
                "type": "string"
              },
              "operand_2": {
                "type": "string"
              }
            }
          }
        }
      }
    }
  }
}
```

```
        "operator": {
            "type": "string"
        }
    }
},
"x-amazon-apigateway-request-validator": "Validate query string parameters and
headers",
"x-amazon-apigateway-integration": {
    "credentials": "arn:aws:iam::123456789012:role/apigAwsProxyRole",
    "responses": {
        "default": {
            "statusCode": "200",
            "responseParameters": {
                "method.response.header.operator": "integration.response.body.op",
                "method.response.header.operand_2": "integration.response.body.b",
                "method.response.header.operand_1": "integration.response.body.a"
            },
            "responseTemplates": {
                "application/json": "#set($res = $input.path('$'))\n{\n    \"result\": \"$res.a\", $res.b, $res.op => $res.c\",\\n    \"a\" : \"$res.a\",\\n    \"b\" : \"$res.b\",\\n    \"op\" : \"$res.op\",\\n    \"c\" : \"$res.c\"\n}"
            }
        }
    },
    "uri": "arn:aws:apigateway:us-west-2:lambda:path/2015-03-31/functions/
arn:aws:lambda:us-west-2:123456789012:function:Calc/invocations",
    "passthroughBehavior": "when_no_match",
    "httpMethod": "POST",
    "requestTemplates": {
        "application/json": "{\n    \"a\": \"$input.params('operand1')\",\\n    \"b\":
\": \"$input.params('operand2')\",\\n    \"op\": \"$input.params('operator')\"\n}"
    },
    "type": "aws"
},
"post": {
    "consumes": [
        "application/json"
    ],
    "produces": [
        "application/json"
    ],
    "parameters": [
        {
            "in": "body",
            "name": "Input",
            "required": true,
            "schema": {
                "$ref": "#/definitions/Input"
            }
        }
    ],
    "responses": {
        "200": {
            "description": "200 response",
            "schema": {
                "$ref": "#/definitions/Result"
            }
        }
    },
    "x-amazon-apigateway-request-validator": "Validate body",
    "x-amazon-apigateway-integration": {
        "credentials": "arn:aws:iam::123456789012:role/apigAwsProxyRole",
        "responses": {
            "default": {
```

```
        "statusCode": "200",
        "responseTemplates": {
            "application/json": "#set($inputRoot = $input.path('$'))\n{\n    \"a\" : $inputRoot.a,\n    \"b\" : $inputRoot.b,\n    \"op\" : $inputRoot.op,\n    \"c\" : $inputRoot.c\n}"
        }
    },
    "uri": "arn:aws:apigateway:us-west-2:lambda:path/2015-03-31/functions/
arn:aws:lambda:us-west-2:123456789012:function:Calc/invocations",
    "passthroughBehavior": "when_no_templates",
    "httpMethod": "POST",
    "type": "aws"
}
},
"/calc/{operand1}/{operand2}/{operator}": {
    "get": {
        "consumes": [
            "application/json"
        ],
        "produces": [
            "application/json"
        ],
        "parameters": [
            {
                "name": "operand2",
                "in": "path",
                "required": true,
                "type": "string"
            },
            {
                "name": "operator",
                "in": "path",
                "required": true,
                "type": "string"
            },
            {
                "name": "operand1",
                "in": "path",
                "required": true,
                "type": "string"
            }
        ],
        "responses": {
            "200": {
                "description": "200 response",
                "schema": {
                    "$ref": "#/definitions/Result"
                }
            }
        },
        "x-amazon-apigateway-integration": {
            "credentials": "arn:aws:iam::123456789012:role/apigAwsProxyRole",
            "responses": {
                "default": {
                    "statusCode": "200",
                    "responseTemplates": {
                        "application/json": "#set($inputRoot = $input.path('$'))\n{\n    \"$input\" : {\n        \"a\" : $inputRoot.a,\n        \"b\" : $inputRoot.b,\n        \"op\" : \"$inputRoot.op\"\n    },\n    \"output\" : {\n        \"c\" : $inputRoot.c\n    }\n}"
                    }
                }
            },
            "uri": "arn:aws:apigateway:us-west-2:lambda:path/2015-03-31/functions/
arn:aws:lambda:us-west-2:123456789012:function:Calc/invocations",
        }
    }
}
```

```
    "passthroughBehavior": "when_no_templates",
    "httpMethod": "POST",
    "requestTemplates": {
        "application/json": "{$input.params('operand1')}\n    \n    \"$input.params('operator')\"\n    \n    \"$input.params('operator')\"#end\n    \n}"
    },
    "contentHandling": "CONVERT_TO_TEXT",
    "type": "aws"
}
},
"definitions": {
    "Input": {
        "type": "object",
        "required": [
            "a",
            "b",
            "op"
        ],
        "properties": {
            "a": {
                "type": "number"
            },
            "b": {
                "type": "number"
            },
            "op": {
                "type": "string",
                "description": "binary op of ['+', 'add', '-', 'sub', '*', 'mul', '%2F', 'div']"
            }
        },
        "title": "Input"
    },
    "Output": {
        "type": "object",
        "properties": {
            "c": {
                "type": "number"
            }
        },
        "title": "Output"
    },
    "Result": {
        "type": "object",
        "properties": {
            "input": {
                "$ref": "#/definitions/Input"
            },
            "output": {
                "$ref": "#/definitions/Output"
            }
        },
        "title": "Result"
    }
},
"x-amazon-apigateway-requestValidators": {
    "Validate body": {
        "validateRequestParameters": false,
        "validateRequestBody": true
    },
    "Validate query string parameters and headers": {
        "validateRequestParameters": true,
        "validateRequestBody": false
    }
}
```

```
}
```

## 教學：在 API Gateway 中建立 REST API 做為 Amazon S3 代理

例如，若要示範如何在 API Gateway 中使用 REST API 來代理處理 Amazon S3，本節說明如何建立和設定 REST API 來公開下列 Amazon S3 操作：

- 在 API 根資源上公開 GET，以列出發起人的所有 Amazon S3 儲存貯體。
- 在 Folder 資源上公開 GET，以檢視 Amazon S3 儲存貯體中的所有物件清單。
- 在 Folder 資源上公開 PUT，以將儲存貯體新增至 Amazon S3。
- 在 Folder 資源上公開 DELETE，以從 Amazon S3 移除儲存貯體。
- 在 Folder/Item 資源上公開 GET，以檢視或下載 Amazon S3 儲存貯體中的物件。
- 在 Folder/Item 資源上公開 PUT，以將物件上傳至 Amazon S3 儲存貯體。
- 在 Folder/Item 資源上公開 HEAD，以取得 Amazon S3 儲存貯體中的物件中繼資料。
- 在 Folder/Item 資源上公開 DELETE，以移除 Amazon S3 儲存貯體中的物件。

### Note

若要整合 API Gateway API 與 Amazon S3，您必須選擇可使用 API Gateway 和 Amazon S3 服務的區域。如需區域可用性，請參閱[區域和端點](#)。

建議您匯入範例 API 做為 Amazon S3 代理，如[做為 Amazon S3 代理之範例 API 的 OpenAPI 定義 \(p. 119\)](#)中所示。如需如何使用 OpenAPI 定義匯入 API 的說明，請參閱[將 REST API 匯入 API Gateway \(p. 319\)](#)。

若要使用 API Gateway 主控台來建立 API，您必須先註冊 AWS 帳戶。

If you do not have an AWS account, complete the following steps to create one.

To sign up for an AWS account

1. Open <https://portal.aws.amazon.com/billing/signup>.
2. Follow the online instructions.

Part of the sign-up procedure involves receiving a phone call and entering a verification code on the phone keypad.

### 主題

- [設定 API 呼叫 Amazon S3 動作的 IAM 許可 \(p. 105\)](#)
- [建立 API 資源以代表 Amazon S3 資源 \(p. 106\)](#)
- [公開 API 方法來列出發起人的 Amazon S3 儲存貯體 \(p. 107\)](#)
- [公開 API 方法來存取 Amazon S3 儲存貯體 \(p. 112\)](#)
- [公開 API 方法來存取儲存貯體中的 Amazon S3 物件 \(p. 115\)](#)
- [使用 REST API 用戶端呼叫 API \(p. 118\)](#)

- 做為 Amazon S3 代理之範例 API 的 OpenAPI 定義 (p. 119)

## 設定 API 呼叫 Amazon S3 動作的 IAM 許可

若要允許 API 呼叫所需 Amazon S3 動作，您必須將適當的 IAM 政策連接至 IAM 角色。下節說明如何驗證與建立所需的 IAM 角色和政策 (必要時)。

若要讓您的 API 檢視或列出 Amazon S3 儲存貯體和物件，您可以在 IAM 角色中使用 IAM 所提供的 AmazonS3ReadOnlyAccess 政策。此政策的 ARN 是 `arn:aws:iam::aws:policy/AmazonS3ReadOnlyAccess`，如下所示：

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "s3:Get*",  
                "s3>List*"  
            ],  
            "Resource": "*"  
        }  
    ]  
}
```

此政策文件指出您可以在任何 Amazon S3 資源上呼叫任何 Amazon S3 Get\* 和 List\* 動作。

若要讓您的 API 更新 Amazon S3 儲存貯體和物件，您可以使用任何 Amazon S3 Put\* 動作的自訂政策，如下所示：

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": "s3:Put*",  
            "Resource": "*"  
        }  
    ]  
}
```

若要讓您的 API 使用 Amazon S3 Get\*、List\* 和 Put\* 動作，您可以將上述唯讀和僅放置政策新增至 IAM 角色。

若要讓您的 API 呼叫 Amazon S3 Post\* 動作，您必須在 IAM 角色中使用 s3:Post\* 動作的允許政策。如需完整 Amazon S3 動作清單，請參閱[在政策中指定 Amazon S3 許可](#)。

若要讓您的 API 建立、檢視、更新和刪除 Amazon S3 中的儲存貯體和物件，您可以在 IAM 角色中使用 IAM 所提供的 AmazonS3FullAccess 政策。ARN 是 `arn:aws:iam::aws:policy/AmazonS3FullAccess`。

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": "s3:*",  
            "Resource": "*"  
        }  
    ]  
}
```

```
    ]  
}
```

選擇所需的 IAM 政策以使用、建立 IAM 角色，並將它連接至政策。產生的 IAM 角色必須包含下列信任政策，讓 API Gateway 在執行時間擔任此角色。

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

使用 IAM 主控台建立角色時，請選擇 Amazon API Gateway 角色類型，以確保自動包含此信任政策。

## 建立 API 資源以代表 Amazon S3 資源

我們將使用 API 的根 (/) 資源做為已認證發起人之 Amazon S3 儲存貯體的容器。我們也會建立 Folder 和 Item 資源，分別代表特定 Amazon S3 儲存貯體和特定 Amazon S3 物件。發起人會將資料夾名稱和物件金鑰指定為請求 URL 的部分路徑參數。

### 建立可公開 Amazon S3 服務功能的 API 資源

1. 在 API Gateway 主控台中，建立名為 MyS3 的 API。這個 API 的根資源 (/) 代表 Amazon S3 服務。
2. 在 API 的根資源下，建立名為 Folder 的子資源，並將所需的 Resource Path (資源路徑) 設定為 /{folder}。
3. 針對 API 的 Folder 資源，建立 Item 子資源。將所需的 Resource Path (資源路徑) 設定為 /{item}。

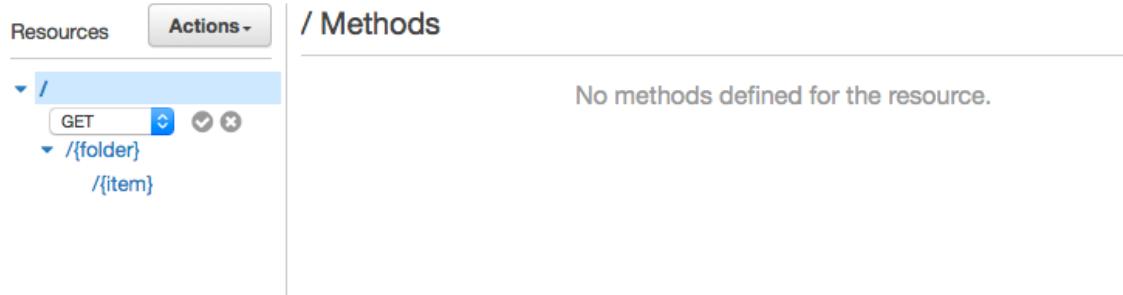
The screenshot shows the 'New Child Resource' dialog in the AWS API Gateway console. On the left, a sidebar shows the path structure: a root folder with a single child resource named '{folder}'. The '{folder}' resource is selected and highlighted with a red oval. On the right, the 'New Child Resource' form is displayed. It has fields for 'Resource Name\*' (set to 'Item') and 'Resource Path\*' (set to '/{folder}/{item}'). The '{item}' placeholder in the path field is also highlighted with a red oval. Below the path field, there is a note explaining path parameters. At the bottom, there are checkboxes for 'Enable API Gateway CORS' and 'Required'. The 'Create Resource' button at the bottom right is also highlighted with a red oval.

## 公開 API 方法來列出發起人的 Amazon S3 儲存貯體

取得發起人的 Amazon S3 儲存貯體清單包含在 Amazon S3 上呼叫 [GET 服務](#)動作。在 API 的根資源 (/) 上，建立 GET 方法。設定 GET 方法以與 Amazon S3 整合，如下所示。

### 建立和初始化 API 的 GET / 方法

1. 從 Resources (資源) 面板右上角的 Actions (動作) 下拉式選單中，在根節點 (/) 上選擇 Create method (建立方法)。
2. 從 HTTP 動詞的下拉式清單中選擇 GET，然後選擇核取記號圖示來開始建立方法。



3. 在 / - GET - Setup (/ - GET - 設定) 窗格中，針對 Integration type (整合類型) 選擇 AWS Service (AWS 服務)。
4. 從清單中，針對 AWS Region (&AWS; 區域) 選擇區域 (例如，us-west-2)。
5. 從 AWS Service (AWS 服務) 中，選擇 S3。
6. 對於 AWS Subdomain (AWS 子網域)，將其空白。
7. 從 HTTP method (HTTP 方法) 中，選擇 GET。
8. 對於 Action Type (動作類型)，選擇 Use path override (使用路徑覆寫)。使用路徑覆寫，API Gateway 會將用戶端請求轉送給 Amazon S3 以做為對應的 [Amazon S3 REST API 路徑樣式請求](#)；其中，Amazon S3 資源是以 s3-host-name/bucket/key 模式的資源路徑表示。API Gateway 設定 s3-host-name，並將用戶端指定的 bucket 和 key 從用戶端傳遞給 Amazon S3。
9. (選用) 在 Path override (路徑覆寫) 中，輸入 /。
10. 從 IAM 主控台中複製先前建立之 IAM 角色的 ARN，並將它貼入 Execution role (執行角色)。
11. 保留任何其他設定做為預設值。
12. 選擇 Save (儲存) 完成此方法的設定。

此設定會整合前端 GET `https://your-api-host/stage/` 請求與後端 GET `https://your-s3-host/`。

### Note

初始設定之後，您可以在方法的 Integration Request (整合請求) 頁面中修改這些設定。

為了控制誰可以呼叫 API 的這種方法，我們開啟方法授權旗標，並將它設定為 AWS\_IAM。

### 讓 IAM 控制 GET / 方法的存取

1. 從 Method Execution (方法執行) 中，選擇 Method Request (方法請求)。
2. 選擇 Authorization (授權) 旁邊的鉛筆圖示
3. 從下拉式清單中，選擇 AWS\_IAM。
4. 選擇核取記號圖示來儲存設定。

[Method Execution](#) / - GET - Method Request

Provide information about this method's authorization settings and the parameters it can receive.

Authorization Settings



API Key Required false

▶ URL Query String Parameters

▶ HTTP Request Headers

▶ Request Models [Create a Model](#)

為了讓我們的 API 將成功回應和例外狀況正確地傳回給發起人，請在 Method Response (方法回應) 中宣告 200、400 和 500 回應。我們使用 200 回應做為預設對應；因此，會將未在這裡宣告的狀態碼後端回應當成 200 回應傳回給發起人。

宣告 GET / 方法的回應類型

1. 從 Method Execution (方法執行) 窗格中，選擇 Method Response (方法回應) 方塊。API Gateway 預設會宣告 200 回應。
2. 選擇 Add response (新增回應)，並在輸入文字方塊中輸入 **400**，然後選擇核取記號以完成宣告。
3. 重複上述步驟來宣告 500 回應類型。最終設定顯示如下：

[Method Execution](#) / - GET - Method Response

Provide information about this method's response types, their headers and content types.

	HTTP Status	
▶	200	
▶	400	
▶	500	
	<a href="#">Add Response</a>	

因為來自 Amazon S3 的成功整合回應傳回儲存貯體清單做為 XML 承載，而來自 API Gateway 的預設方法回應傳回 JSON 承載，所以我們必須將後端 Content-Type 標頭參數值對應至前端相應實物。否則，回應內容實際上是 XML 字串時，用戶端將會收到內容類型的 application/json。下列程序示範如何設定此項目。此外，我們也想要向用戶端顯示其他標頭參數 (例如 Date 和 Content-Length)。

### 設定 GET / 方法的回應標頭對應

- 在 API Gateway 主控台中，選擇 Method Response (方法回應)。新增 200 回應類型的 Content-Type 標頭。

[Method Execution](#) / - GET - Method Response

Provide information about this method's response types, their headers and content types.

HTTP Status
▼ 200 <span style="float: right;">✖</span>

Response Headers for 200

Name
Timestamp
Content-Length
Content-Type

[+ Add Header](#)

Response Models for 200 [Create a model](#)

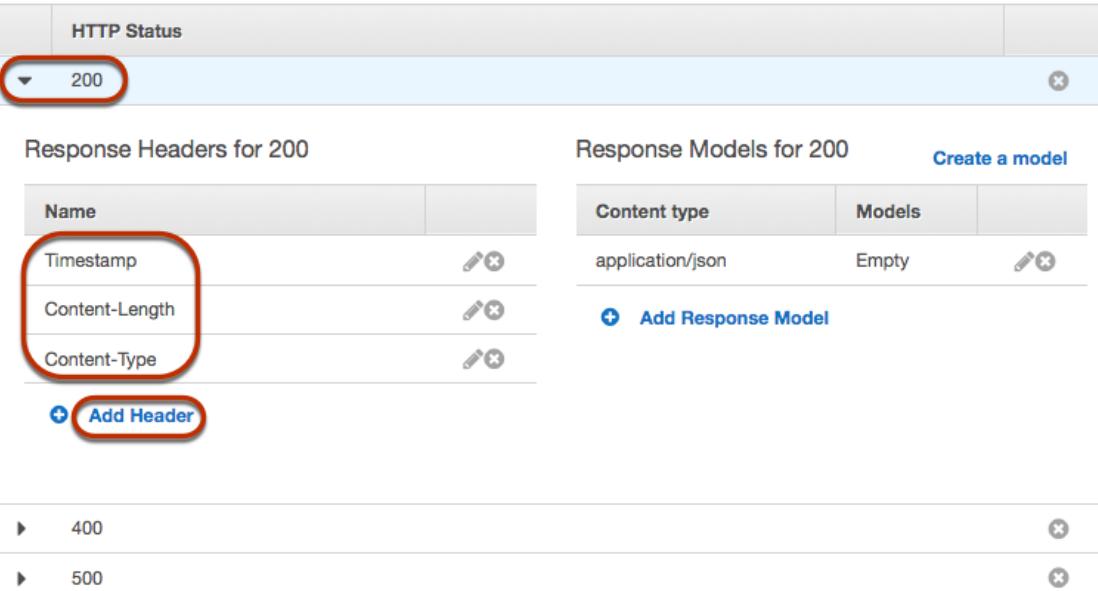
Content type	Models
application/json	Empty <span style="float: right;">✖</span>

[+ Add Response Model](#)

▶ 400 ✖

▶ 500 ✖

[+ Add Response](#)



- 在 Integration Response (整合回應) 中，針對 Content-Type，輸入方法回應的 `integration.response.header.Content-Type`。

[Method Execution](#) / - GET - Integration Response

First, declare response types using [Method Response](#). Then, map the possible responses from the backend to this method's response types.

	HTTP status regex	Method response status	Output model	Default mapping	
▼	-	200		Yes	X

Map the output from your HTTP endpoint to the headers and output model of the 200 method response.

HTTP status regex default Method response status200 ?

Cancel Save

▼ Header Mappings

Response header	Mapping value <span style="border: 1px solid #ccc; padding: 2px;">?</span>	
Timestamp	integration.response.header.Date	<span style="border: 1px solid #ccc; padding: 2px;">Edit</span>
Content-Length	integration.response.header.Content-Length	<span style="border: 1px solid #ccc; padding: 2px;">Edit</span>
Content-Type	integration.response.header.Content-Type	<span style="border: 1px solid #ccc; padding: 2px;">Edit</span>

► Body Mapping Templates

使用上述標頭對應，API Gateway 會將 Date 標頭從後端翻譯為用戶端的 Timestamp 標頭。

3. 仍然在 Integration Response (整合回應) 中，選擇 Add integration response (新增整合回應)，並在 HTTP status regex (HTTP 狀態 regex) 文字方塊中輸入其餘方法回應狀態的適當一般表達式。重複，直到涵蓋所有方法回應狀態。

[Method Execution](#) / - GET - Integration Response

First, declare response types using [Method Response](#). Then, map the possible responses from the backend to this method's response types.

	HTTP status regex	Method response status	Output model	Default mapping	
▶	-	200		Yes	×
▶	4\d{2}	400		No	×
▼	5\d{2}	500		No	×

Map the output from your HTTP endpoint to the headers and output model of the 500 method response.

**HTTP status regex** 5\d{2} i

Method response status 500

Cancel **Save**

▼ Header Mappings

Response header	Mapping value <span style="border: 1px solid #ccc; padding: 2px;">i</span>
No method response headers.	

▶ Body Mapping Templates

**Add integration response**

最好測試到目前為止已設定的 API。

測試 API 根資源上的 GET 方法

1. 返回 Method Execution (方法執行) , 然後從 Client (用戶端) 方塊中選擇 Test (測試)。
2. 在 GET / - Method Test (GET / - 方法測試) 窗格中 , 選擇 Test (測試)。範例結果顯示如下。

← Method Execution / - GET - Method Test

Make a test call to your method with the provided input

### Path

No path parameters exist for this resource. You can define path parameters by using the syntax `{myPathParam}` in a resource path.

### Query Strings

No query string parameters exist for this method. You can add them via Method Request.

### Headers

No header parameters exist for this method. You can add them via Method Request.

### Stage Variables

No stage variables exist for this method.

### Client Certificate

None

 Test

### Request Body

Request Body is not supported for GET methods.

Request: /

Status: 200

Latency: 746 ms

### Response Body

```
<?xml version="1.0" encoding="UTF-8"?>
<ListAllMyBucketsResult xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <Owner><ID>0e...</ID><DisplayName>aws-lambda-2015-02-12T22:05:55.000Z</DisplayName></Owner>
  <Buckets><Bucket><Name>pig-demo</Name><CreationDate>2016-02-12T22:05:55.000Z</CreationDate><Bucket><Name>apig-demo1</Name><CreationDate>2016-02-15T23:38:35.000Z</CreationDate><Bucket><Name>aws-devdoc-test-kd</Name><CreationDate>2015-10-27T22:59:29.000Z</CreationDate><Bucket><Name>aws-lambda-fileproc-test-kd-eventarchive-....17</Name><CreationDate>2015-10-21T22:39:57.000Z</CreationDate><Bucket><Name>aws-lambda-fileproc-test-kd-eventarchive-....0007-out</Name><CreationDate>2015-02-21T23:14:14.000Z</CreationDate><Bucket><Name>aws-lab-triggers-kd</Name><CreationDate>2015-10-22T23:48:01.000Z</CreationDate><Bucket><Name>cf-templates-1....19-us-east-1</Name><CreationDate>2015-10-21T20:59:45.000Z</CreationDate><Bucket><Name>elasticbeanstalk-us-east-1-700070000007</Name><CreationDate>2015-10-16T23:13:15.000Z</CreationDate><Bucket><Name>jaws.dev.u....com</Name><CreationDate>2015-12-01T19:34:56.000Z</CreationDate><Bucket><Name>jaws.dev.useast1.myapp-eyollwv.com</Name><CreationDate>2015-12-01T20:22:47.000Z</CreationDate><Bucket><Name>my-pictures-02-16-2016</Name><CreationDate>2016-02-18T08:14:40.000Z</CreationDate><Bucket><Name>myrv-contentdelivery-mobilehub-1161765204</Name><CreationDate>2015-10-26T05:07:51.000Z</CreationDate><Bucket><Name>myrv-userfiles-mobilehub-1161765204</Name><CreationDate>2015-10-26T05:09:30.000Z</CreationDate><Bucket><Name>node-sdk-sample-33358871-858c-40fd-8f90-9b0eeb096eee</Name><CreationDate>2015-10-21T23:49:15.000Z</CreationDate></Bucket></Buckets></ListAllMyBucketsResult>
```

### Response Headers

```
{"Timestamp": "Fri, 19 Feb 2016 03:09:52 GMT", "Content-Type": "application/xml"}
```

### Logs

Execution log for request test-request

Fri Feb 19 03:09:50 UTC 2016 : Starting execution for request: test-invok

## Note

若要使用 API Gateway 主控台來測試做為 Amazon S3 代理的 API，請確保目標 S3 儲存貯體來自 API 區域的不同區域。否則，您可能會收到「500 內部伺服器錯誤」回應。此限制不適用於任何已部署的 API。

## 公開 API 方法來存取 Amazon S3 儲存貯體

為了使用 Amazon S3 儲存貯體，我們在 `/{{folder}}` 資源上公開 GET、PUT 和 DELETE 方法，以列出儲存貯體中的物件、建立新的儲存貯體，以及刪除現有儲存貯體。這些說明與「[公開 API 方法來列出發起人的 Amazon S3 儲存貯體 \(p. 107\)](#)」中所述的說明類似。我們會在下列討論中概述一般任務，並強調相關的差異。

在資料夾資源上公開 GET、PUT 和 DELETE 方法

- 在 Resources (資源) 樹狀結構的 `{folder}` 節點上，逐一建立 DELETE、GET 和 PUT 方法。
  - 設定每個已建立方法與其對應 Amazon S3 端點的初始整合。下列螢幕擷取畫面說明 `PUT /{folder}` 方法的這個設定。針對 `DELETE /{folder}` 和 `GET /{folder}` 方法，將 HTTP method (HTTP 方法) 的 `PUT` 值分別取代為 `DELETE` 和 `GET`。

The screenshot shows the AWS Lambda function configuration page for the /{folder} - PUT - Setup method. On the left, there's a sidebar with resources like / (GET), /{folder} (DELETE, GET, PUT), and /{item}. The PUT option under /{folder} is selected. The main area is titled '/{folder} - PUT - Setup'. It asks to choose an integration point. The 'Integration type' section has 'AWS Service' selected. The 'AWS Region' dropdown is set to '<region>'. The 'AWS Service' dropdown is set to 'S3' (circled in red). The 'AWS Subdomain' field is empty. The 'HTTP method' dropdown is set to 'PUT' (circled in red). The 'Action Type' section has 'Use path override' selected. The 'Path override (optional)' field contains '{bucket}' (circled in red). The 'Execution role' dropdown is set to 'arn:aws:iam::...:role/apigAwsProxyRole'. At the bottom right is a blue 'Save' button.

請注意，我們在 Amazon S3 端點 URL 中使用 {bucket} 路徑參數來指定儲存貯體。我們需要將方法請求的 {folder} 路徑參數對應至整合請求的 {bucket} 路徑參數。

3. 將 {folder} 對應至 {bucket}：
  - a. 選擇 Method Execution (方法執行)，然後選擇 Integration Request (整合請求)。
  - b. 展開 URL Path Parameters (URL 路徑參數)，然後選擇 Add path (新增路徑)
  - c. 在 Name (名稱) 資料行中輸入 bucket，並在 Mapped from (對應來源) 資料行中輸入 method.request.path.folder。選擇核取記號圖示來儲存對應。

#### ▼ URL Path Parameters

Name	Mapped from	Caching
bucket	method.request.path.folder	<input checked="" type="checkbox"/>

4. 在 Method Request (方法請求) 中，將 Content-Type 新增至 HTTP Request Headers (HTTP 請求標頭) 區段。

▼ HTTP Request Headers

Name	Caching
Content-Type	<input type="checkbox"/>
<a href="#">+ Add header</a>	

如果您必須針對 XML 承載指定 **application/xml**，則這是使用 API Gateway 主控台時最需要進行測試的項目。

5. 遵循[公開 API 方法來列出發起人的 Amazon S3 儲存貯體 \(p. 107\)](#)中所述的說明，在 Integration Request (整合請求) 中設定下列標頭對應。

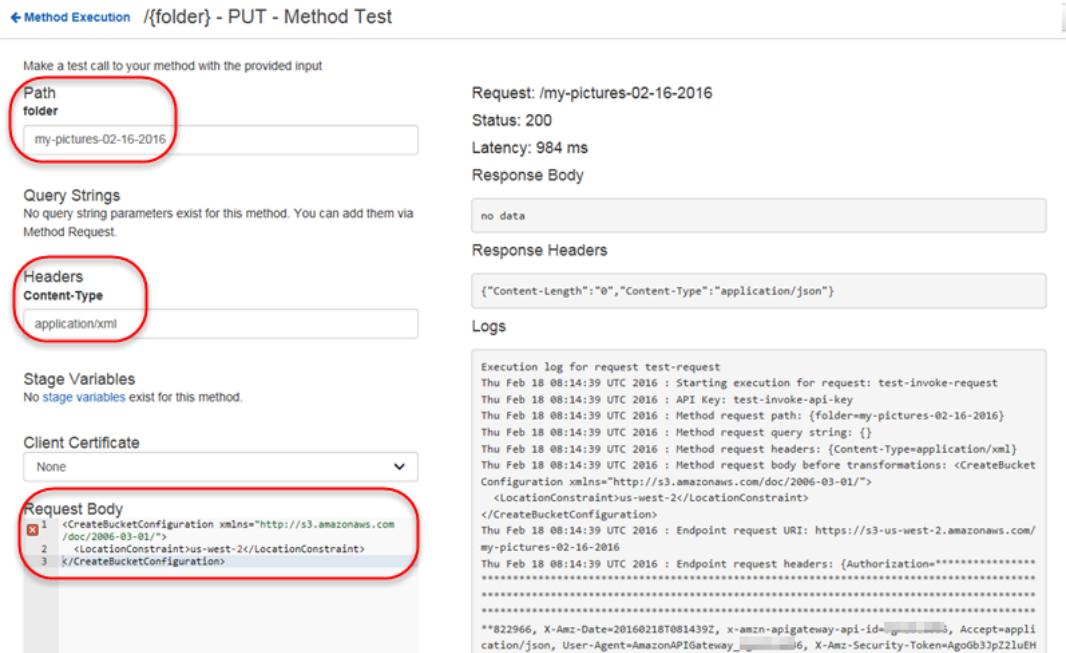
▼ HTTP Headers

Name	Mapped from ⓘ	Caching
x-amz-acl	'authenticated-read'	<input type="checkbox"/>  
Content-Type	method.request.header.Content-Type	<input type="checkbox"/>  
<a href="#">+ Add header</a>		

x-amz-acl 標頭是用於指定資料夾 (或相應 Amazon S3 儲存貯體) 的存取控制。如需詳細資訊，請參閱[Amazon S3 PUT Bucket 請求](#)。

6. 若要測試 PUT 方法，請從 Method Execution (方法執行) 的 Client (用戶端) 方塊中選擇 Test (測試)，然後輸入下列做為測試的輸入：
  - 在 folder (資料夾) 中，輸入儲存貯體名稱。
  - 對於 Content-Type 標頭，輸入 **application/xml**。
  - 在 Request Body (請求內文) 中，提供儲存貯體區域做為 XML 片段中宣告為請求承載的位置限制。例如：

```
<CreateBucketConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <LocationConstraint>{region}</LocationConstraint>
</CreateBucketConfiguration>
```



## 7. 重複先前的步驟，以在 API /{folder} 資源上建立和設定 GET 和 DELETE 方法。

上述範例說明如何在指定的區域中建立新的儲存貯體，以檢視儲存貯體中的物件清單，以及刪除儲存貯體。其他 Amazon S3 儲存貯體操作可讓您使用儲存貯體的中繼資料或屬性。例如，您可以設定 API 以呼叫 Amazon S3 的 [PUT /?notification](#) 動作來設定儲存貯體的通知、呼叫 [PUT /?acl](#) 來設定儲存貯體的存取控制清單等等。API 設定類似，差異處在您必須將適當的查詢參數附加至 Amazon S3 端點 URL 後面。在執行時間，您必須將適當的 XML 承載提供給方法請求。支援 Amazon S3 儲存貯體上的其他 GET 和 DELETE 操作時也相同。如需儲存貯體上可能 &S3; 動作的詳細資訊，請參閱 [儲存貯體上的 Amazon S3 操作](#)。

# 公開 API 方法來存取儲存貯體中的 Amazon S3 物件

Amazon S3 支援 GET、DELETE、HEAD、OPTIONS、POST 和 PUT 動作來存取和管理特定儲存貯體中的物件。如需完整支援動作的清單，請參閱 [物件上的 Amazon S3 操作](#)。

在本教學中，我們分別透過 API 方法 [PUT /{folder}/{item}](#)、[GET /{folder}/{item}](#)、[HEAD /{folder}/{item}](#) 和 [DELETE /{folder}/{item}](#) 來公開 [PUT Object](#) 操作、[GET Object](#) 操作、[HEAD Object](#) 操作和 [DELETE Object](#) 操作。

/ {folder} / {item} 上 PUT、GET 和 DELETE 方法的 API 設定與 / {folder} 上 PUT、GET 和 DELETE 方法的 API 設定類似，如「[公開 API 方法來存取 Amazon S3 儲存貯體 \(p. 112\)](#)」中所述。其中一個主要差異在於物件相關請求路徑具有額外的路徑參數 {item}，而且此路徑參數必須對應至整合請求路徑參數 {object}。

◀ Method Execution /{folder}/{item} - PUT - Integration Request

Provide information about the target backend that this method will call and whether the incoming request data should be modified.

Integration type  Lambda Function ?

HTTP ?

Mock ?

AWS Service ?

AWS Region u [ ] 2 edit

AWS Service S3 edit

AWS Subdomain edit

HTTP method PUT edit

Path override {bucket}/{object} edit

Execution role arn:aws:iam::[REDACTED]:role/apigAwsProxyRole edit

Credentials cache Do not add caller credentials to cache key edit

▼ URL Path Parameters

Name	Mapped from <small>?</small>	Caching	
object	method.request.path.item	<input type="checkbox"/>	<small>edit</small> <small>remove</small>
bucket	method.request.path.folder	<input type="checkbox"/>	<small>edit</small> <small>remove</small>

+ Add path

這適用於 GET 和 DELETE 方法。

做為說明，下列螢幕擷取畫面顯示使用 API Gateway 主控台測試 {folder}/{item} 資源上之 GET 方法時的輸出。請求會正確地傳回 ("Welcome to README.txt") 的純文字做為指定之 Amazon S3 儲存貯體 (apig-d демо) 中所指定檔案 (README.txt) 的內容。

← Method Execution /{folder}/{item} - GET - Method Test

Make a test call to your method with the provided input

Path

folder

apig-demo

item

README.txt

Request: /apig-demo/README.txt

Status: 200

Latency: 486 ms

Response Body

Welcome to README.txt

Response Headers

{"Content-Type": "text/plain"}

Logs

```
Execution log for request test-request
Fri Feb 19 03:35:20 UTC 2016 : Starting execution for request: test-invoke-request
Fri Feb 19 03:35:20 UTC 2016 : API Key: test-invoke-api-key
Fri Feb 19 03:35:20 UTC 2016 : Method request path: {item=README.txt, folder=apig-demo}
Fri Feb 19 03:35:20 UTC 2016 : Method request query string: {}
Fri Feb 19 03:35:20 UTC 2016 : Method request headers: {}
Fri Feb 19 03:35:20 UTC 2016 : Method request body before transformations: null
Fri Feb 19 03:35:20 UTC 2016 : Endpoint request URI: http://s3-us-west-2.amazonaws.com/apig-demo/README.txt
Fri Feb 19 03:35:20 UTC 2016 : Endpoint request headers: {Authorization=*****}
*****
*****
*****
*****
*****a99006, X-Amz
```

Query Strings

No query string parameters exist for this method.  
You can add them via Method Request.

Headers

No header parameters exist for this method. You can add them via Method Request.

Stage Variables

No stage variables exist for this method.

Client Certificate

None

Request Body

Request Body is not supported for GET methods.

Test

若要下載或上傳二進位檔案，而二進位檔案在 API Gateway 中視為 utf-8 編碼 JSON 內容以外的任何事項，則需要額外的 API 設定。這概述如下：

### 從 S3 下載或上傳二進位檔案

1. 將受影響檔案的媒體類型註冊到 API 的 binaryMediaTypes。您可以在主控台中執行這項操作：
  - a. 從 API Gateway 主導覽面板中，選擇 API 的 Binary Support (二進位支援)。
  - b. 選擇 Edit (編輯)。
  - c. 輸入所需的媒體類型 (例如，針對 Binary media types (二進位媒體類型) 輸入 image/png)。
  - d. 選擇 Add binary media type (新增二進位媒體類型) 來儲存設定。
2. 將 Content-Type (進行上傳) 和 (或) Accept (進行下載) 標頭新增至方法請求，以要求用戶端指定所需的二進位媒體類型，並將其對應至整合請求。
3. 在整合請求 (進行上傳) 和整合回應 (進行下載) 中，將 Content Handling (內容處理) 設定為 Passthrough。請確定未定義受影響內容類型的對應範本。如需詳細資訊，請參閱 [整合傳遞行為 \(p. 273\)](#) 和 [選取 VTL 對應範本 \(p. 272\)](#)。

承載大小上限為 10 MB。請參閱「[設定和執行 REST API 的 API Gateway 限制 \(p. 628\)](#)」。

確定 Amazon S3 上的檔案具有新增為檔案中繼資料的正確內容類型。針對可串流媒體內容，Content-Disposition:inline 也可能需要新增至中繼資料。

如需 API Gateway 中二進位支援的詳細資訊，請參閱[API Gateway 中的內容類型轉換 \(p. 283\)](#)。

## 使用 REST API 用戶端呼叫 API

我們現在示範如何使用 Postman 呼叫 API，該應用程式支援 AWS IAM 授權。

使用 Postman 呼叫 Amazon S3 代理 API

1. 部署或重新部署 API。請記下 API 的基本 URL，而此 API 顯示在 Stage Editor (階段編輯器) 頂端的 Invoke URL (呼叫 URL) 旁邊。
2. 啟動 Postman。
3. 選擇 Authorization (授權)，然後選擇 AWS Signature。在 AccessKey 和 SecretKey 輸入欄位中，分別輸入 IAM 使用者的存取金鑰 ID 和私密存取金鑰。在 AWS Region (AWS 區域) 文字方塊中，輸入在其中部署您 API 的 AWS 區域。在 Service Name (服務名稱) 輸入欄位中，輸入 execute-api。

您可以從 IAM 管理主控台之 IAM 使用者帳戶的 Security Credentials (安全登入資料) 標籤中建立一對金鑰。

4. 將名為 apig-demo-5 的儲存貯體新增至 *{region}* 區域中的 Amazon S3 帳戶：

Note

請確定儲存貯體名稱必須為全域唯一。

- a. 從下拉式方法清單中選擇 PUT，並輸入方法 URL (<https://api-id.execute-api.aws-region.amazonaws.com/stage/folder-name>)
- b. 將 Content-Type 標頭值設定為 application/xml。您可能需要先刪除任何現有標頭，再設定內容類型。
- c. 選擇 Body (內文) 選單項目，並輸入下列 XML 片段做為請求內文：

```
<CreateBucketConfiguration>
  <LocationConstraint>{region}</LocationConstraint>
</CreateBucketConfiguration>
```

- d. 選擇 Send (傳送)，以提交請求。如果成功，您應該會收到具有空承載的 200 OK 回應。
5. 若要將文字檔案新增至儲存貯體，請遵循上述說明。果您在 URL 中指定 *{folder}* 的儲存貯體名稱 apig-demo-5 以及 *{item}* 的檔案名稱 **Readme.txt**，並提供字串 **Hello, World!** 做為請求承載，則請求會成為

```
PUT /S3/apig-demo-5/Readme.txt HTTP/1.1
Host: 9gn28ca086.execute-api.{region}.amazonaws.com
Content-Type: application/xml
X-Amz-Date: 20161015T062647Z
Authorization: AWS4-HMAC-SHA256 Credential=access-key-id/20161015/{region}/execute-
api/aws4_request, SignedHeaders=content-length;content-type;host;x-amz-date,
Signature=ccadb877bdb0d395ca38cc47e18a0d76bb5eaf17007d11e40bf6fb63d28c705b
Cache-Control: no-cache
Postman-Token: 6135d315-9cc4-8af8-1757-90871d00847e

Hello, World!
```

如果一切順利，您應該會收到具有空承載的 200 OK 回應。

6. 若要取得剛剛新增至 **Readme.txt** 儲存貯體之 apig-demo-5 檔案的內容，請執行 GET 請求，如下所示：

```
GET /S3/apig-demo-5/Readme.txt HTTP/1.1
Host: 9gn28ca086.execute-api.{region}.amazonaws.com
Content-Type: application/xml
X-Amz-Date: 20161015T063759Z
Authorization: AWS4-HMAC-SHA256 Credential=access-key-id/20161015/{region}/
execute-api/aws4_request, SignedHeaders=content-type;host;x-amz-date,
Signature=ba09b72b585acf0e578e6ad02555c00e24b420b59025bc7bb8d3f7aed1471339
Cache-Control: no-cache
Postman-Token: d60fc59-d335-52f7-0025-5bd96928098a
```

如果成功，您應該會收到具有 200 OK 字串做為承載的 Hello, World! 回應。

- 若要列出 apig-demo-5 儲存貯體中的項目，請提交下列請求：

```
GET /S3/apig-demo-5 HTTP/1.1
Host: 9gn28ca086.execute-api.{region}.amazonaws.com
Content-Type: application/xml
X-Amz-Date: 20161015T064324Z
Authorization: AWS4-HMAC-SHA256 Credential=access-key-id/20161015/{region}/
execute-api/aws4_request, SignedHeaders=content-type;host;x-amz-date,
Signature=4ac9bd4574a14e01568134fd16814534d9951649d3a22b3b0db9f1f5cd4dd0ac
Cache-Control: no-cache
Postman-Token: 9c43020a-966f-61e1-81af-4c49ad8d1392
```

如果成功，除非先將更多檔案新增至儲存貯體再提交此請求，否則您應該會收到 200 OK 回應，而其 XML 承載顯示所指定儲存貯體中的單一項目。

```
<?xml version="1.0" encoding="UTF-8"?>
<ListBucketResult xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
    <Name>apig-demo-5</Name>
    <Prefix></Prefix>
    <Marker></Marker>
    <MaxKeys>1000</MaxKeys>
    <IsTruncated>false</IsTruncated>
    <Contents>
        <Key>Readme.txt</Key>
        <LastModified>2016-10-15T06:26:48.000Z</LastModified>
        <ETag>"65a8e27d8879283831b664bd8b7f0ad4"</ETag>
        <Size>13</Size>
        <Owner>
            <ID>06e4b09e9d...603add12ee</ID>
            <DisplayName>user-name</DisplayName>
        </Owner>
        <StorageClass>STANDARD</StorageClass>
    </Contents>
</ListBucketResult>
```

#### Note

若要上傳或下載影像，您需要將內容處理設定為 CONVERT\_TO\_BINARY。

## 做為 Amazon S3 代理之範例 API 的 OpenAPI 定義

下列 OpenAPI 定義說明本教學中所參考且做為 Amazon S3 代理的範例 API。

OpenAPI 2.0

```
{
```

```
"swagger": "2.0",
"info": {
    "version": "2016-10-13T23:04:43Z",
    "title": "MyS3"
},
"host": "9gn28ca086.execute-api.{region}.amazonaws.com",
"basePath": "/S3",
"schemes": [
    "https"
],
"paths": {
    "/": {
        "get": {
            "produces": [
                "application/json"
            ],
            "responses": {
                "200": {
                    "description": "200 response",
                    "schema": {
                        "$ref": "#/definitions/Empty"
                    },
                    "headers": {
                        "Content-Length": {
                            "type": "string"
                        },
                        "Timestamp": {
                            "type": "string"
                        },
                        "Content-Type": {
                            "type": "string"
                        }
                    }
                },
                "400": {
                    "description": "400 response"
                },
                "500": {
                    "description": "500 response"
                }
            },
            "security": [
                {
                    "sigv4": []
                }
            ],
            "x-amazon-apigateway-integration": {
                "credentials": "arn:aws:iam::<replaceable>123456789012</replaceable>:role/apigAwsProxyRole",
                "responses": {
                    "4\d{2}": {
                        "statusCode": "400"
                    },
                    "default": {
                        "statusCode": "200",
                        "responseParameters": {
                            "method.response.header.Content-Type": "integration.response.header.Content-Type",
                            "method.response.header.Content-Length": "integration.response.header.Content-Length",
                            "method.response.header.Timestamp": "integration.response.header.Date"
                        }
                    },
                    "5\d{2}": {
                        "statusCode": "500"
                    }
                }
            }
        }
    }
}
```

```
        },
        "uri": "arn:aws:apigateway:us-west-2:s3:path//",
        "passthroughBehavior": "when_no_match",
        "httpMethod": "GET",
        "type": "aws"
    }
}
},
"/{folder}": {
    "get": {
        "produces": [
            "application/json"
        ],
        "parameters": [
            {
                "name": "folder",
                "in": "path",
                "required": true,
                "type": "string"
            }
        ],
        "responses": {
            "200": {
                "description": "200 response",
                "schema": {
                    "$ref": "#/definitions/Empty"
                },
                "headers": {
                    "Content-Length": {
                        "type": "string"
                    },
                    "Date": {
                        "type": "string"
                    },
                    "Content-Type": {
                        "type": "string"
                    }
                }
            },
            "400": {
                "description": "400 response"
            },
            "500": {
                "description": "500 response"
            }
        },
        "security": [
            {
                "sigv4": []
            }
        ],
        "x-amazon-apigateway-integration": {
            "credentials": "arn:aws:iam::<replaceable>123456789012</replaceable>:role/apigAwsProxyRole",
            "responses": {
                "4\b\d{2}": {
                    "statusCode": "400"
                },
                "default": {
                    "statusCode": "200",
                    "responseParameters": {
                        "method.response.header.Content-Type": "integration.response.header.Content-Type",
                        "method.response.header.Date": "integration.response.header.Date",
                        "method.response.header.Content-Length": "integration.response.header.content-length"
                    }
                }
            }
        }
    }
}
```

```
        }
    },
    "5\\d{2}": {
        "statusCode": "500"
    }
},
"requestParameters": {
    "integration.request.path.bucket": "method.request.path.folder"
},
"uri": "arn:aws:apigateway:us-west-2:s3:path/{bucket}",
"passthroughBehavior": "when_no_match",
"httpMethod": "GET",
"type": "aws"
}
},
"put": {
    "produces": [
        "application/json"
    ],
    "parameters": [
        {
            "name": "Content-Type",
            "in": "header",
            "required": false,
            "type": "string"
        },
        {
            "name": "folder",
            "in": "path",
            "required": true,
            "type": "string"
        }
    ],
    "responses": {
        "200": {
            "description": "200 response",
            "schema": {
                "$ref": "#/definitions/Empty"
            },
            "headers": {
                "Content-Length": {
                    "type": "string"
                },
                "Content-Type": {
                    "type": "string"
                }
            }
        },
        "400": {
            "description": "400 response"
        },
        "500": {
            "description": "500 response"
        }
    },
    "security": [
        {
            "sigv4": []
        }
    ],
    "x-amazon-apigateway-integration": {
        "credentials": "arn:aws:iam::<replaceable>123456789012</replaceable>:role/apigAwsProxyRole",
        "responses": {
            "4\\d{2}": {
                "statusCode": "400"
            }
        }
    }
}
```

```
        },
        "default": {
            "statusCode": "200",
            "responseParameters": {
                "method.response.header.Content-Type":
                    "integration.response.header.Content-Type",
                "method.response.header.Content-Length":
                    "integration.response.header.Content-Length"
            }
        },
        "5\d{2}": {
            "statusCode": "500"
        }
    },
    "requestParameters": {
        "integration.request.header.x-amz-acl": "'authenticated-read'",
        "integration.request.path.bucket": "method.request.path.folder",
        "integration.request.header.Content-Type": "method.request.header.Content-
Type"
    },
    "uri": "arn:aws:apigateway:us-west-2:s3:path/{bucket}",
    "passthroughBehavior": "when_no_match",
    "httpMethod": "PUT",
    "type": "aws"
}
},
"delete": {
    "produces": [
        "application/json"
    ],
    "parameters": [
        {
            "name": "folder",
            "in": "path",
            "required": true,
            "type": "string"
        }
    ],
    "responses": {
        "200": {
            "description": "200 response",
            "schema": {
                "$ref": "#/definitions/Empty"
            },
            "headers": {
                "Date": {
                    "type": "string"
                },
                "Content-Type": {
                    "type": "string"
                }
            }
        },
        "400": {
            "description": "400 response"
        },
        "500": {
            "description": "500 response"
        }
    },
    "security": [
        {
            "sigv4": []
        }
    ],
    "x-amazon-apigateway-integration": {
```

```
    "credentials": "arn:aws:iam::<replaceable>123456789012</replaceable>:role/apigAwsProxyRole",
    "responses": {
        "4\\d{2}": {
            "statusCode": "400"
        },
        "default": {
            "statusCode": "200",
            "responseParameters": {
                "method.response.header.Content-Type": "integration.response.header.Content-Type",
                "method.response.header.Date": "integration.response.header.Date"
            }
        },
        "5\\d{2}": {
            "statusCode": "500"
        }
    },
    "requestParameters": {
        "integration.request.path.bucket": "method.request.path.folder"
    },
    "uri": "arn:aws:apigateway:us-west-2:s3:path/{bucket}",
    "passthroughBehavior": "when_no_match",
    "httpMethod": "DELETE",
    "type": "aws"
}
},
"/{folder}/{item)": {
    "get": {
        "produces": [
            "application/json"
        ],
        "parameters": [
            {
                "name": "item",
                "in": "path",
                "required": true,
                "type": "string"
            },
            {
                "name": "folder",
                "in": "path",
                "required": true,
                "type": "string"
            }
        ],
        "responses": {
            "200": {
                "description": "200 response",
                "schema": {
                    "$ref": "#/definitions/Empty"
                },
                "headers": {
                    "content-type": {
                        "type": "string"
                    },
                    "Content-Type": {
                        "type": "string"
                    }
                }
            },
            "400": {
                "description": "400 response"
            },
            "500": {

```

```
        "description": "500 response"
    },
},
"security": [
{
    "sigv4": []
}
],
"x-amazon-apigateway-integration": {
    "credentials": "arn:aws:iam::<replaceable>123456789012</replaceable>:role/
apigAwsProxyRole",
    "responses": {
        "4\d{2}": {
            "statusCode": "400"
        },
        "default": {
            "statusCode": "200",
            "responseParameters": {
                "method.response.header.content-type":
"integration.response.header.content-type",
                "method.response.header.Content-Type":
"integration.response.header.Content-Type"
            }
        },
        "5\d{2}": {
            "statusCode": "500"
        }
    },
    "requestParameters": {
        "integration.request.path.object": "method.request.path.item",
        "integration.request.path.bucket": "method.request.path.folder"
    },
    "uri": "arn:aws:apigateway:us-west-2:s3:path/{bucket}/{object}",
    "passthroughBehavior": "when_no_match",
    "httpMethod": "GET",
    "type": "aws"
}
},
"head": {
    "produces": [
        "application/json"
    ],
    "parameters": [
        {
            "name": "item",
            "in": "path",
            "required": true,
            "type": "string"
        },
        {
            "name": "folder",
            "in": "path",
            "required": true,
            "type": "string"
        }
    ],
    "responses": {
        "200": {
            "description": "200 response",
            "schema": {
                "$ref": "#/definitions/Empty"
            },
            "headers": {
                "Content-Length": {
                    "type": "string"
                }
            }
        }
    }
}
```

```
        "Content-Type": {
            "type": "string"
        }
    },
    "400": {
        "description": "400 response"
    },
    "500": {
        "description": "500 response"
    }
},
"security": [
{
    "sigv4": []
}
],
"x-amazon-apigateway-integration": {
    "credentials": "arn:aws:iam::<replaceable>123456789012</replaceable>:role/apigAwsProxyRole",
    "responses": {
        "4\b{2}": {
            "statusCode": "400"
        },
        "default": {
            "statusCode": "200",
            "responseParameters": {
                "method.response.header.Content-Type": "integration.response.header.Content-Type",
                "method.response.header.Content-Length": "integration.response.header.Content-Length"
            }
        },
        "5\b{2}": {
            "statusCode": "500"
        }
    },
    "requestParameters": {
        "integration.request.path.object": "method.request.path.item",
        "integration.request.path.bucket": "method.request.path.folder"
    },
    "uri": "arn:aws:apigateway:us-west-2:s3:path/{bucket}/{object}",
    "passthroughBehavior": "when_no_match",
    "httpMethod": "HEAD",
    "type": "aws"
}
},
"put": {
    "produces": [
        "application/json"
    ],
    "parameters": [
        {
            "name": "Content-Type",
            "in": "header",
            "required": false,
            "type": "string"
        },
        {
            "name": "item",
            "in": "path",
            "required": true,
            "type": "string"
        },
        {
            "name": "folder",
            "in": "path",
            "required": false,
            "type": "string"
        }
    ]
}
```

```
        "in": "path",
        "required": true,
        "type": "string"
    }
],
"responses": {
    "200": {
        "description": "200 response",
        "schema": {
            "$ref": "#/definitions/Empty"
        },
        "headers": {
            "Content-Length": {
                "type": "string"
            },
            "Content-Type": {
                "type": "string"
            }
        }
    },
    "400": {
        "description": "400 response"
    },
    "500": {
        "description": "500 response"
    }
},
"security": [
    {
        "sigv4": []
    }
],
"x-amazon-apigateway-integration": {
    "credentials": "arn:aws:iam::<replaceable>123456789012</replaceable>:role/apigAwsProxyRole",
    "responses": {
        "4\\\d{2}": {
            "statusCode": "400"
        },
        "default": {
            "statusCode": "200",
            "responseParameters": {
                "method.response.header.Content-Type": "integration.response.header.Content-Type",
                "method.response.header.Content-Length": "integration.response.header.Content-Length"
            }
        },
        "5\\\d{2}": {
            "statusCode": "500"
        }
    },
    "requestParameters": {
        "integration.request.path.object": "method.request.path.item",
        "integration.request.header.x-amz-acl": "'authenticated-read'",
        "integration.request.path.bucket": "method.request.path.folder",
        "integration.request.header.Content-Type": "method.request.header.Content-Type"
    },
    "uri": "arn:aws:apigateway:us-west-2:s3:path/{bucket}/{object}",
    "passthroughBehavior": "when_no_match",
    "httpMethod": "PUT",
    "type": "aws"
}
},
"delete": {
```

```
"produces": [
    "application/json"
],
"parameters": [
    {
        "name": "item",
        "in": "path",
        "required": true,
        "type": "string"
    },
    {
        "name": "folder",
        "in": "path",
        "required": true,
        "type": "string"
    }
],
"responses": {
    "200": {
        "description": "200 response",
        "schema": {
            "$ref": "#/definitions/Empty"
        },
        "headers": {
            "Content-Length": {
                "type": "string"
            },
            "Content-Type": {
                "type": "string"
            }
        }
    },
    "400": {
        "description": "400 response"
    },
    "500": {
        "description": "500 response"
    }
},
"security": [
    {
        "sigv4": []
    }
],
"x-amazon-apigateway-integration": {
    "credentials": "arn:aws:iam::<replaceable>123456789012</replaceable>:role/apigAwsProxyRole",
    "responses": {
        "4\\\d{2}": {
            "statusCode": "400"
        },
        "default": {
            "statusCode": "200"
        },
        "5\\\d{2}": {
            "statusCode": "500"
        }
    },
    "requestParameters": {
        "integration.request.path.object": "method.request.path.item",
        "integration.request.path.bucket": "method.request.path.folder"
    },
    "uri": "arn:aws:apigateway:us-west-2:s3:path/{bucket}/{object}",
    "passThroughBehavior": "when_no_match",
    "httpMethod": "DELETE",
    "type": "aws"
}
```

```
        }
    },
},
"securityDefinitions": {
    "sigv4": {
        "type": "apiKey",
        "name": "Authorization",
        "in": "header",
        "x-amazon-apigateway-authtype": "awsSigv4"
    }
},
"definitions": {
    "Empty": {
        "type": "object",
        "title": "Empty Schema"
    }
}
}
```

## 教學：在 API Gateway 中建立 REST API 做為 Amazon Kinesis 代理

本頁面說明如何建立和設定與 AWS 類型整合的 REST API，以存取 Kinesis。

### Note

若要整合 API Gateway API 與 Kinesis，您必須選擇可使用 API Gateway 和 Kinesis 服務的區域。如需區域可用性，請參閱[區域和端點](#)。

基於說明，我們建立範例 API，讓用戶端執行下列操作：

1. 列出 Kinesis 中的使用者可用串流
2. 建立、說明或刪除指定的串流
3. 從指定的串流讀取資料記錄或將資料記錄寫入其中

為了達成先前的任務，API 會分別公開各種資源的方法來呼叫下列項目：

1. Kinesis 中的 `ListStreams` 動作
2. `CreateStream`、`DescribeStream` 或 `DeleteStream` 動作
3. Kinesis 中的 `GetRecords` 或 `PutRecords` (包含 `PutRecord`) 動作

具體而言，我們會如下建置 API：

- 在 API /streams 資源上公開 HTTP GET 方法，並在 Kinesis 中整合此方法與 `ListStreams` 動作，以列出發起人帳戶中的串流。
- 在 API /streams/{stream-name} 資源上公開 HTTP POST 方法，並在 Kinesis 中整合此方法與 `CreateStream` 動作，以在發起人帳戶中建立具名串流。
- 在 API /streams/{stream-name} 資源上公開 HTTP GET 方法，並在 Kinesis 中整合此方法與 `DescribeStream` 動作，以說明發起人帳戶中的具名串流。
- 在 API /streams/{stream-name} 資源上公開 HTTP DELETE 方法，並在 Kinesis 中整合此方法與 `DeleteStream` 動作，以刪除發起人帳戶中的串流。
- 在 API /streams/{stream-name}/record 資源上公開 HTTP PUT 方法，並在 Kinesis 中整合此方法與 `PutRecord` 動作。這可讓用戶端將單一資料記錄新增至具名串流。

- 在 API /streams/{stream-name}/records 資源上公開 HTTP PUT 方法，並在 Kinesis 中整合此方法與 PutRecords 動作。這可讓用戶端將資料記錄清單新增至具名串流。
- 在 API /streams/{stream-name}/records 資源上公開 HTTP GET 方法，並在 Kinesis 中整合此方法與 GetRecords 動作。這可讓用戶端使用指定的碎片迭代運算來列出具名串流中的資料記錄。碎片迭代運算指定從中循序開始讀取資料記錄的碎片位置。
- 在 API /streams/{stream-name}/sharditerator 資源上公開 HTTP GET 方法，並在 Kinesis 中整合此方法與 GetShardIterator 動作。必須向 Kinesis 中的 ListStreams 動作提供此 helper 方法。

您可以將這裡看到的說明套用至其他 Kinesis 動作。如需完整 Kinesis 動作清單，請參閱 [Amazon Kinesis API 參考](#)。

除了使用 API Gateway 主控台來建立範例 API 之外，您還可以使用 API Gateway Import API，以將範例 API 匯入至 API Gateway。如需如何使用 Import API 的資訊，請參閱「[將 REST API 匯入 API Gateway \(p. 319\)](#)」。

If you do not have an AWS account, complete the following steps to create one.

To sign up for an AWS account

1. Open <https://portal.aws.amazon.com/billing/signup>.
2. Follow the online instructions.

Part of the sign-up procedure involves receiving a phone call and entering a verification code on the phone keypad.

## 建立 API 的 IAM 角色和政策來存取 Kinesis

若要允許 API 呼叫 Kinesis 動作，您必須將適當的 IAM 政策連接至 IAM 角色。本節說明如何驗證與建立所需的 IAM 角色和政策 (必要時)。

若要啟用 Kinesis 的唯讀存取權，您可以使用 AmazonKinesisReadOnlyAccess 政策，以允許在 Kinesis 中呼叫 Get\*、List\* 和 Describe\* 動作。

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "kinesis:Get*",  
                "kinesis>List*",  
                "kinesis:Describe*"  
            ],  
            "Resource": "*"  
        }  
    ]  
}
```

此政策可做為 IAM 佈建的受管政策，而其 ARN 是 arn:aws:iam::aws:policy/AmazonKinesisReadOnlyAccess。

若要啟用 Kinesis 中的讀寫動作，您可以使用 AmazonKinesisFullAccess 政策。

```
{
```

```
"Version": "2012-10-17",
"Statement": [
    {
        "Effect": "Allow",
        "Action": "kinesis:*",
        "Resource": "*"
    }
]
```

此政策也可以做為 IAM 佈建的受管政策。其 ARN 是 `arn:aws:iam::aws:policy/AmazonKinesisFullAccess`。

在您決定要使用的 IAM 政策之後，請將它連接至全新或現有 IAM 角色。確定 API Gateway 控制服務 (`apigateway.amazonaws.com`) 是角色的受信任實體，並允許擔任執行角色 (`sts:AssumeRole`)。

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

如果您在 IAM 主控台中建立執行角色，並選擇 Amazon API Gateway 角色類型，則會自動連接此信任政策。

請記下執行角色的 ARN。在建立 API 方法並設定其整合請求時，您會需要它。

## 開始建立 API 做為 Kinesis 代理

使用下列步驟可在 API Gateway 主控台中建立 API。

### 建立 API 做為 Kinesis 的 AWS 服務代理

1. 在 API Gateway 主控台中，選擇 Create API (建立 API)。
2. 選擇 New API (新增 API)。
3. 在 API name (API 名稱) 中，輸入 **KinesisProxy**。保留其他欄位中的預設值。
4. 如果您要的話，請在 Description (說明) 中輸入說明。
5. 選擇 Create API (建立 API)。

建立 API 之後，API Gateway 主控台會顯示 Resources (資源) 頁面，其中只包含 API 的根 (/) 資源。

## 列出 Kinesis 中的串流

Kinesis 透過下列 REST API 呼叫來支援 `ListStreams` 動作：

```
POST /?Action=ListStreams HTTP/1.1
```

```
Host: kinesis.<region>.<domain>
Content-Length: <PayloadSizeBytes>
User-Agent: <UserAgentString>
Content-Type: application/x-amz-json-1.1
Authorization: <AuthParams>
X-Amz-Date: <Date>

{
    ...
}
```

在上面的 REST API 請求中，動作指定於 Action 查詢參數中。或者，您可以改為在 X-Amz-Target 標頭中指定動作：

```
POST / HTTP/1.1
Host: kinesis.<region>.<domain>
Content-Length: <PayloadSizeBytes>
User-Agent: <UserAgentString>
Content-Type: application/x-amz-json-1.1
Authorization: <AuthParams>
X-Amz-Date: <Date>
X-Amz-Target: Kinesis_20131202.ListStreams
{
    ...
}
```

在本教學中，我們使用查詢參數來指定動作。

若要在 API 中公開 Kinesis 動作，請將 /streams 資源新增至 API 根。然後在資源上設定 GET 方法，並整合此方法與 Kinesis 的 ListStreams 動作。

下列程序說明如何使用 API Gateway 主控台來列出 Kinesis 串流。

### 使用 API Gateway 主控台列出 Kinesis 串流

- 選取 API 根資源。在 Actions (動作) 中，選擇 Create Resource (建立資源)。

在 Resource Name (資源名稱) 中，輸入 Streams，並將 Resource Path (資源路徑) 和其他欄位保留為預設值，然後選擇 Create Resource (建立資源)。

- 選擇 /Streams 資源。從 Actions (動作) 中，選擇 Create Method (建立方法)，並從清單中選擇 GET，然後選擇核取記號圖示來完成建立方法。

#### Note

用戶端所呼叫方法的 HTTP 動詞可能會與後端所需整合的 HTTP 動詞不同。我們在這裡選擇 GET，因為列出串流直覺上就是 READ 操作。

- 在方法的 Setup (設定) 窗格中，選擇 AWS Service (AWS 服務)。
  - 針對 AWS Region (AWS 區域)，選擇區域 (例如，us-east-1)。
  - 針對 AWS Service (AWS 服務)，選擇 Kinesis。
  - 將 AWS Subdomain (AWS 子網域) 保留空白。
  - 針對 HTTP method (HTTP 方法)，選擇 POST。

#### Note

我們在這裡選擇 POST，因為 Kinesis 需要使用它來呼叫 ListStreams 動作。

- 針對 Action Type (動作類型)，選擇 Use action name (使用動作名稱)。
- 針對 Action (動作) 輸入 **ListStreams**。
- 針對 Execution role (執行角色)，輸入執行角色的 ARN。

- h. 保留 Content Handling (內容處理) 之 Passthrough (傳遞) 的預設值。
- i. 選擇 Save (儲存) 來完成方法的初始設定。

## /streams - GET - Setup



Choose the integration point for your new method.

Integration type  Lambda Function i

HTTP i

Mock i

AWS Service i

AWS Region  i

AWS Service  i

AWS Subdomain

HTTP method  i

Action Type  Use action name

Use path override

Action

Execution role  i

Content Handling  i

**Save**

4. 仍在 Integration Request (整合請求) 窗格中，展開 HTTP Headers (HTTP 標頭) 區段：
  - a. 選擇 Add header (新增標頭)。
  - b. 在 Name (名稱) 欄位中，輸入 Content-Type。
  - c. 在 Mapped from (對應來源) 欄位中，輸入 'application/x-amz-json-1.1'。
  - d. 選擇核取記號圖示來儲存設定。

我們使用請求參數對應將 Content-Type 標頭設定為靜態值 'application/x-amz-json-1.1'，通知 Kinesis 有關輸入是特定版本的 JSON。

5. 展開 Body Mapping Templates (內文對應範本) 區段：

- a. 選擇 Add mapping template (新增對應範本)。
- b. 針對 Content-Type，輸入 application/json。

- c. 選擇核取記號圖示來儲存 Content-Type 設定。在 Change passthrough behavior (變更傳遞行為) 中，選擇 Yes, secure this integration (是，保護此整合)。
- d. 在範本編輯器中，輸入 {}。
- e. 選擇 Save (儲存) 按鈕來儲存對應範本。

[ListStreams](#) 請求採用下列 JSON 格式的承載：

```
{  
    "ExclusiveStartStreamName": "string",  
    "Limit": number  
}
```

不過，屬性是選用的。若要使用預設值，我們在這裡選擇空的 JSON 承載。

▼ HTTP Headers

Name	Mapped from ⓘ	Caching
Content-Type	'application/x-amz-json-1.1'	<input type="checkbox"/>  

**+ Add header**

▼ Body Mapping Templates

**Request body passthrough**  When there are no templates defined (recommended) ⓘ  When no template matches the request Content-Type header ⓘ  Never ⓘ

Content-Type
application/json 

**+ Add mapping template**

application/json

Generate template:  

1 [{}]

**Cancel** **Save**

6. 測試 Streams 資源上的 GET 方法，以在 Kinesis 中呼叫 ListStreams 動作：

從 API Gateway 主控台的 Resources (資源) 窗格中選取 /streams/GET 項目，並選擇 Test (測試) 呼叫選項，然後選擇 Test (測試)。

如果您已經在 Kinesis 中建立兩個名為 "myStream" 和 "yourStream" 的串流，則成功測試會傳回包含下列承載的 200 OK 回應：

```
{  
    "HasMoreStreams": false,  
    "StreamNames": [  
        "myStream",  
        "yourStream"  
    ]  
}
```

## 在 Kinesis 中建立、說明和刪除串流

在 Kinesis 中建立、說明和刪除串流會分別進行下列 Kinesis REST API 請求：

```
POST /?Action=CreateStream HTTP/1.1  
Host: kinesis.region.domain  
...  
Content-Type: application/x-amz-json-1.1  
Content-Length: PayloadSizeBytes  
  
{  
    "ShardCount": number,  
    "StreamName": "string"  
}
```

```
POST /?Action=DescribeStream HTTP/1.1  
Host: kinesis.region.domain  
...  
Content-Type: application/x-amz-json-1.1  
Content-Length: PayloadSizeBytes  
  
{  
    "ExclusiveStartShardId": "string",  
    "Limit": number,  
    "StreamName": "string"  
}
```

```
POST /?Action=DeleteStream HTTP/1.1  
Host: kinesis.region.domain  
...  
Content-Type: application/x-amz-json-1.1  
Content-Length: PayloadSizeBytes  
  
{  
    "StreamName": "string"  
}
```

我們可以建置 API 以接受所需的輸入做為方法請求的 JSON 承載，並將承載傳遞至整合請求。不過，若要提供方法與整合請求之間以及方法與整合回應之間的更多資料對應範例，我們會建立略為不同的 API。

我們會對指定的 GET 資源公開 POST、Delete 和 Stream HTTP 方法。我們使用 {stream-name} 路徑變數做為串流資源的預留位置，並分別整合這些 API 方法與 Kinesis DescribeStream、CreateStream 和 DeleteStream 動作。我們需要用戶端將其他輸入資料傳遞為方法請求的標頭、查詢參數或承載。我們提供對應範本將資料轉換為所需的整合請求承載。

### 設定和測試串流資源上的 GET 方法

1. 使用 {stream-name} 路徑變數，在先前建立的 /streams 資源下方建立子資源。

#### New Child Resource

Use this page to create a new child resource for your resource.

Configure as  proxy resource



Resource Name\*



Resource Path\*



You can add path parameters using brackets. For example, the resource path `{username}` represents a path parameter called 'username'. Configuring `/streams/{proxy+}` as a proxy resource catches all requests to its sub-resources. For example, it works for a GET request to `/streams/foo`. To handle requests to `/streams`, add a new ANY method on the `/streams` resource.

Enable API Gateway CORS



\* Required

[Cancel](#)

[Create Resource](#)

2. 將 POST、GET 和 DELETE HTTP 動詞新增至此資源。

在資源上建立方法之後，API 的結構如下所示：

Resources Actions ▾

- ▼ /
- ▼ /streams
  - GET
  - ▼ /{stream-name}
    - DELETE
    - POST
    - GET

3. 設定 GET /streams/{stream-name} 方法以在 Kinesis 中呼叫 POST /?Action=DescribeStream 動作，如下所示。

## /streams/{stream-name} - GET - Setup



Choose the integration point for your new method.

Integration type  Lambda Function ⓘ  
 HTTP ⓘ  
 Mock ⓘ  
 AWS Service ⓘ

AWS Region

AWS Service

AWS Subdomain

HTTP method

Action Type  Use action name  
 Use path override

Action

Execution role

Content Handling

**Save**

4. 將下列 Content-Type 標頭對應新增至整合請求：

```
Content-Type: 'x-amz-json-1.1'
```

任務所遵循的程序與設定 GET /streams 方法的請求參數對應相同。

5. 新增下列內文對應範本，以將資料從 GET /streams/{stream-name} 方法請求對應至 POST /?Action=DescribeStream 整合請求：

```
{  
    "StreamName": "$input.params('stream-name')"  
}
```

此對應範本會透過方法請求的 stream-name 路徑參數值，來產生 Kinesis 之 DescribeStream 動作所需的整合請求承載。

6. 測試 GET /stream/{stream-name} 方法，以在 Kinesis 中呼叫 DescribeStream 動作：

從 API Gateway 主控台的 Resources (資源) 窗格中，選取 /streams/{stream-name}/GET、選擇 Test (測試) 開始測試、在 Path (路徑) 欄位中輸入 stream-name 的現有 Kinesis 串流名稱，然後選擇 Test (測試)。如果測試成功，則會傳回承載與類似下列的 200 OK 回應：

```
{  
    "StreamDescription": {  
        "HasMoreShards": false,  
        "RetentionPeriodHours": 24,  
        "Shards": [  
            {  
                "HashKeyRange": {  
                    "EndingHashKey": "68056473384187692692674921486353642290",  
                    "StartingHashKey": "0"  
                },  
                "SequenceNumberRange": {  
                    "StartingSequenceNumber":  
                        "49559266461454070523309915164834022007924120923395850242"  
                },  
                "ShardId": "shardId-000000000000"  
            },  
            ...  
            {  
                "HashKeyRange": {  
                    "EndingHashKey": "340282366920938463463374607431768211455",  
                    "StartingHashKey": "272225893536750770770699685945414569164"  
                },  
                "SequenceNumberRange": {  
                    "StartingSequenceNumber":  
                        "49559266461543273504104037657400164881014714369419771970"  
                },  
                "ShardId": "shardId-000000000004"  
            }  
        ],  
        "StreamARN": "arn:aws:kinesis:us-east-1:12345678901:stream/myStream",  
        "StreamName": "myStream",  
        "StreamStatus": "ACTIVE"  
    }  
}
```

在您部署 API 之後，可以針對此 API 方法提出 REST 請求：

```
GET https://your-api-id.execute-api.region.amazonaws.com/stagestreams/myStream  
HTTP/1.1  
Host: your-api-id.execute-api.region.amazonaws.com  
Content-Type: application/json  
Authorization: ...  
X-Amz-Date: 20160323T194451Z
```

### 設定和測試串流資源上的 POST 方法

1. 設定 POST /streams/{stream-name} 方法，在 Kinesis 中呼叫 POST /?Action=CreateStream 動作。任務所遵循的程序與設定 GET /streams/{stream-name} 方法相同，但前提是將 DescribeStream 動作取代為 CreateStream。
2. 將下列 Content-Type 標頭對應新增至整合請求：

```
Content-Type: 'x-amz-json-1.1'
```

任務所遵循的程序與設定 GET /streams 方法的請求參數對應相同。

- 新增下列內文對應範本，以將資料從 POST /streams/{stream-name} 方法請求對應至 POST /?Action=CreateStream 整合請求：

```
{  
    "ShardCount": #if($input.path('$._ShardCount') == '') 5 #else  
    $input.path('$._ShardCount') #end,  
    "StreamName": "$input.params('stream-name')"  
}
```

如果用戶端未在方法請求承載中指定值，則在先前的對應範本中，我們將 ShardCount 設定為固定值 5。

- 測試 POST /streams/{stream-name} 方法，在 Kinesis 中建立具名串流：

從 API Gateway 主控台的 Resources (資源) 窗格中，選取 /streams/{stream-name}/POST、選擇 Test (測試) 開始測試、在 Path (路徑) 中輸入 stream-name 的現有 Kinesis 串流名稱，然後選擇 Test (測試)。如果測試成功，則會傳回 200 OK 回應，而沒有資料。

在您部署 API 之後，也可以對串流資源上的 POST 方法提出 REST API 請求，以在 Kinesis 中呼叫 CreateStream 動作：

```
POST https://your-api-id.execute-api.region.amazonaws.com/stagestreams/yourStream  
HTTP/1.1  
Host: your-api-id.execute-api.region.amazonaws.com  
Content-Type: application/json  
Authorization: ...  
X-Amz-Date: 20160323T194451Z  
  
{  
    "ShardCount": 5  
}
```

## 設定和測試串流資源上的 DELETE 方法

- 設定 DELETE /streams/{stream-name} 方法，以與 Kinesis 中的 POST /?Action=DeleteStream 動作整合。任務所遵循的程序與設定 GET /streams/{stream-name} 方法相同，但前提是將 DescribeStream 動作取代為 DeleteStream。
- 將下列 Content-Type 標頭對應新增至整合請求：

```
Content-Type: 'x-amz-json-1.1'
```

任務所遵循的程序與設定 GET /streams 方法的請求參數對應相同。

- 新增下列內文對應範本，以將資料從 DELETE /streams/{stream-name} 方法請求對應至對應的 POST /?Action=DeleteStream 整合請求：

```
{  
    "StreamName": "$input.params('stream-name')"  
}
```

此對應範本會從 DELETE /streams/{stream-name} 中由用戶端提供的 URL 路徑名稱，產生 stream-name 動作的所需輸入。

- 測試 DELETE 方法，以刪除 Kinesis 中的具名串流：

從 API Gateway 主控台的 Resources (資源) 窗格中，選取 /streams/{stream-name}/DELETE 方法節點、選擇 Test (測試) 開始測試、在 Path (路徑) 欄位中輸入 stream-name 的現有 Kinesis 串流名稱，然後選擇 Test (測試)。如果測試成功，則會傳回 200 OK 回應，而沒有資料。

在您部署 API 之後，也可以對串流資源上的 DELETE 方法提出下列 REST API 請求，以在 Kinesis 中呼叫 DeleteStream 動作：

```
DELETE https://your-api-id.execute-api.region.amazonaws.com/stage/streams/yourStream
HTTP/1.1
Host: your-api-id.execute-api.region.amazonaws.com
Content-Type: application/json
Authorization: ...
X-Amz-Date: 20160323T194451Z

{}
```

## 取得 Kinesis 中串流的記錄，並將記錄新增至其中

在您於 Kinesis 中建立串流之後，可以將資料記錄新增至串流，並讀取串流中的資料。新增資料記錄包含在 Kinesis 中呼叫 PutRecords 或 PutRecord 動作。前者會新增多筆記錄，而後者將單一記錄新增至串流。

```
POST /?Action=PutRecords HTTP/1.1
Host: kinesis.region.domain
Authorization: AWS4-HMAC-SHA256 Credential=..., ...
...
Content-Type: application/x-amz-json-1.1
Content-Length: PayloadSizeBytes

{
    "Records": [
        {
            "Data": blob,
            "ExplicitHashKey": "string",
            "PartitionKey": "string"
        }
    ],
    "StreamName": "string"
}
```

或

```
POST /?Action=PutRecord HTTP/1.1
Host: kinesis.region.domain
Authorization: AWS4-HMAC-SHA256 Credential=..., ...
...
Content-Type: application/x-amz-json-1.1
Content-Length: PayloadSizeBytes

{
    "Data": blob,
    "ExplicitHashKey": "string",
    "PartitionKey": "string",
    "SequenceNumberForOrdering": "string",
    "StreamName": "string"
}
```

```
}
```

在這裡，StreamName 識別要新增記錄的目標串流。StreamName、Data 和 PartitionKey 是必要輸入資料。在範例中，我們使用所有選用輸入資料的預設值，因此將不會在方法請求輸入中明確指定其值。

讀取 Kinesis 數量中的資料以呼叫 [GetRecords](#) 動作：

```
POST /?Action=GetRecords HTTP/1.1
Host: kinesis.region.domain
Authorization: AWS4-HMAC-SHA256 Credential=..., ...
...
Content-Type: application/x-amz-json-1.1
Content-Length: PayloadSizeBytes

{
    "ShardIterator": "string",
    "Limit": number
}
```

在這裡，從中取得記錄的來源串流指定於所需 ShardIterator 值，如可取得碎片迭代運算的下列 Kinesis 動作所示：

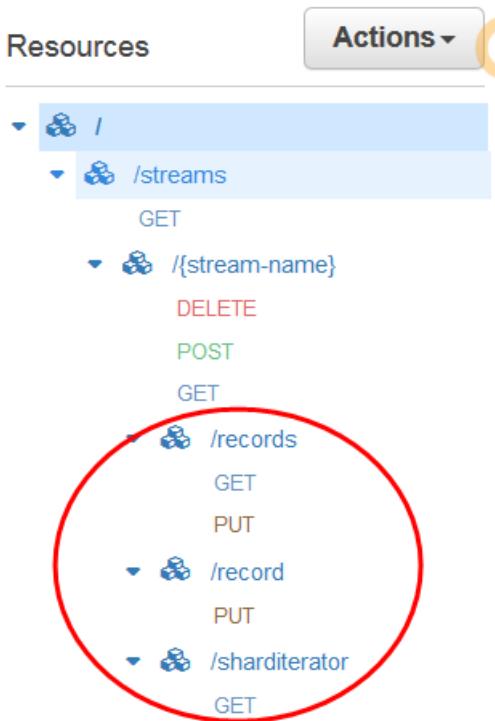
```
POST /?Action=GetShardIterator HTTP/1.1
Host: kinesis.region.domain
Authorization: AWS4-HMAC-SHA256 Credential=..., ...
...
Content-Type: application/x-amz-json-1.1
Content-Length: PayloadSizeBytes

{
    "ShardId": "string",
    "ShardIteratorType": "string",
    "StartingSequenceNumber": "string",
    "StreamName": "string"
}
```

針對 GetRecords 和 PutRecords 動作，我們分別在附加至具名串流資源 (GET) 的 PUT 資源上公開 /records 和 /{stream-name} 方法。同樣地，我們將 PutRecord 動作公開為 PUT 資源上的 /record 方法。

因為 GetRecords 動作將呼叫 ShardIterator helper 動作所取得的 GetShardIterator 值採用為輸入，所以我們在 GET 資源上公開 ShardIterator helper 方法 (/sharditerator)。

下圖顯示資源在建立方法之後的 API 結構：



下列四個程序說明如何設定每種方法、如何將資料從方法請求對應至整合請求，以及如何測試方法。

設定和測試 **PUT /streams/{stream-name}/record** 方法以在 Kinesis 中呼叫 **PutRecord**：

1. 設定 PUT 方法，如下所示：

[Method Execution](#)  
[/streams/{stream-name}/record - PUT - Integration Request](#) 

Provide information about the target backend that this method will call and whether the incoming request data should be modified.

Integration type  Lambda Function   
 HTTP   
 Mock   
 AWS Service 

AWS Region us-east-1 

AWS Service Kinesis 

AWS Subdomain 

HTTP method POST 

Action PutRecord 

Execution role arn:aws:iam::█████████████████████:role/apigAwsProxyRole 

Credentials cache Do not add caller credentials to cache key 

Content Handling Passthrough  

- 新增下列請求參數對應，以將 Content-Type 標頭設定為整合請求中 AWS 相容版本的 JSON：

```
Content-Type: 'x-amz-json-1.1'
```

任務所遵循的程序與設定 GET /streams 方法的請求參數對應相同。

- 新增下列內文對應範本，以將資料從 PUT /streams/{stream-name}/record 方法請求對應至對應的 POST /?Action=PutRecord 整合請求：

```
{  
    "StreamName": "$input.params('stream-name')",  
    "Data": "$util.base64Encode($input.json('$Data'))",  
    "PartitionKey": "$input.path('$PartitionKey')"  
}
```

此對應範本假設方法請求承載的格式如下：

```
{  
    "Data": "some data",  
    "PartitionKey": "some key"  
}
```

此資料可以透過下列 JSON 結構描述建立模型：

```
{  
  "$schema": "http://json-schema.org/draft-04/schema#",  
  "title": "PutRecord proxy single-record payload",  
  "type": "object",  
  "properties": {  
    "Data": { "type": "string" },  
    "PartitionKey": { "type": "string" }  
  }  
}
```

您可以建立模型來包含此結構描述，並使用此模型來促進產生對應範本。不過，您可以產生對應範本，而不使用任何模型。

4. 若要測試 `PUT /streams/{stream-name}/record` 方法，請將 `stream-name` 路徑變數設定為現有串流的名稱，並提供所需格式的承載，然後提交方法請求。成功結果是承載格式如下的 200 OK 回應：

```
{  
  "SequenceNumber": "49559409944537880850133345460169886593573102115167928386",  
  "ShardId": "shardId-00000000004"  
}
```

設定和測試 `PUT /streams/{stream-name}/records` 方法以在 Kinesis 中呼叫 **PutRecords**：

1. 設定 `PUT /streams/{stream-name}/records` 方法，如下所示：

← Method Execution

## /streams/{stream-name}/records - PUT - Integration Request

Provide information about the target backend that this method will call and whether the incoming request data should be modified.

Integration type  Lambda Function   
 HTTP   
 Mock   
 AWS Service 

AWS Region us-east-1 

AWS Service Kinesis 

AWS Subdomain 

HTTP method POST 

Action PutRecords 

Execution role arn:aws:iam::7...:role/apigAwsProxyRole 

Credentials cache Do not add caller credentials to cache key 

Content Handling Passthrough 

2. 新增下列請求參數對應，以將 Content-Type 標頭設定為整合請求中 AWS 相容版本的 JSON：

```
Content-Type: 'x-amz-json-1.1'
```

任務所遵循的程序與設定 GET /streams 方法的請求參數對應相同。

3. 新增下列內文對應範本，以將資料從 PUT /streams/{stream-name}/records 方法請求對應至對應的 POST /?Action=PutRecords 整合請求：

```
{
  "StreamName": "$input.params('stream-name')",
  "Records": [
    #foreach($elem in $input.path('$..records'))
    {
      "Data": "$util.base64Encode($elem.data)",
      "PartitionKey": "$elem.partition-key"
    }#if($foreach.hasNext),#end
    #end
  ]
}
```

此對應範本假設方法請求承載可以透過下列 JSON 結構描述建立模型：

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
```

```
"title": "PutRecords proxy payload data",
"type": "object",
"properties": {
    "records": {
        "type": "array",
        "items": {
            "type": "object",
            "properties": {
                "data": { "type": "string" },
                "partition-key": { "type": "string" }
            }
        }
    }
}
```

您可以建立模型來包含此結構描述，並使用此模型來促進產生對應範本。不過，您可以產生對應範本，而不使用任何模型。

在本教學中，我們使用兩個略為不同的承載格式來說明 API 開發人員可以選擇向用戶端公開後端資料格式，或向用戶端隱藏它。其中一種格式適用於 `PUT /streams/{stream-name}/records` 方法（上方）。另一種格式用於 `PUT /streams/{stream-name}/record` 方法（在先前的程序中）。在生產環境中，您應該保持這兩種格式一致。

4. 若要測試 `PUT /streams/{stream-name}/records` 方法，請將 `stream-name` 路徑變數設定為現有串流，並提供下列承載，然後提交方法請求。

```
{
    "records": [
        {
            "data": "some data",
            "partition-key": "some key"
        },
        {
            "data": "some other data",
            "partition-key": "some key"
        }
    ]
}
```

成功結果是承載與下列輸出類似的 200 OK 回應：

```
{
    "FailedRecordCount": 0,
    "Records": [
        {
            "SequenceNumber": "49559409944537880850133345460167468741933742152373764162",
            "ShardId": "shardId-000000000004"
        },
        {
            "SequenceNumber": "49559409944537880850133345460168677667753356781548470338",
            "ShardId": "shardId-000000000004"
        }
    ]
}
```

**設定和測試 GET /streams/{stream-name}/sharditerator 方法以在 Kinesis 中呼叫 GetShardIterator**

GET /streams/{stream-name}/sharditerator 方法是 helper 方法，可先獲得必要碎片迭代運算，再呼叫 GET /streams/{stream-name}/records 方法。

1. 設定 GET /streams/{stream-name}/sharditerator 方法的整合，如下所示：

[Method Execution](#)

**/streams/{stream-name}/sharditerator - GET - Integration ...**

Provide information about the target backend that this method will call and whether the incoming request data should be modified.

**Integration type**  Lambda Function [?](#)  
 HTTP [?](#)  
 Mock [?](#)  
 AWS Service [?](#)

**AWS Region** us-east-1 [edit](#)

**AWS Service** Kinesis [edit](#)

**AWS Subdomain** [edit](#)

**HTTP method** POST [edit](#)

**Action** GetShardIterator [edit](#)

**Execution role** arn:aws:iam::7...:role/apigAwsProxyRole [edit](#)

**Credentials cache** Do not add caller credentials to cache key [edit](#)

**Content Handling** Passthrough [edit](#) [?](#)

2. GetShardIterator 動作需要 ShardId 值的輸入。若要傳遞用戶端提供的 ShardId 值，我們將 shard-id 查詢參數新增至方法請求，如下所示：

[Method Execution](#)

/streams/{stream-name}/sharditerator - GET - Method Req...

Provide information about this method's authorization settings and the parameters it can receive.

Settings

Authorization: NONE [Edit](#) [Info](#)

Request Validator: NONE [Edit](#) [Info](#)

API Key Required: false [Edit](#)

▶ Request Paths

▼ URL Query String Parameters

Name	Required	Caching	
shard-id	<input type="checkbox"/>	<input type="checkbox"/>	<a href="#">Edit</a> <a href="#">Delete</a>

[+ Add query string](#)

▶ HTTP Request Headers

▶ Request Body [Edit](#)

▶ SDK Settings

在下列內文對應範本中，我們將 shard-id 查詢參數值設定為 JSON 承載的 ShardId 屬性值，做為 Kinesis 中 GetShardIterator 動作的輸入。

3. 設定內文對應範本，以從方法請求的 ShardId 和 StreamName 參數產生 GetShardIterator 動作所需的輸入 (shard-id 和 stream-name)。此外，對應範本也會將 ShardIteratorType 設定為 TRIM\_HORIZON 做為預設值。

```
{  
    "ShardId": "$input.params('shard-id')",  
    "ShardIteratorType": "TRIM_HORIZON",  
    "StreamName": "$input.params('stream-name')"  
}
```

4. 使用 API Gateway 主控台中的 Test (測試) 選項，輸入現有串流名稱做為 stream-name Path (路徑) 變數值，並將 shard-id Query string (查詢字串) 設定為現有 ShardId 值 (例如，shard-000000000004)，然後選擇 Test (測試)。

成功回應承載與下列輸出類似：

```
{
```

```
    "ShardIterator": "AAAAAAAAAAFYVN3VlFy..."  
}
```

記下 ShardIterator 值。您需要有它才能從串流取得記錄。

設定和測試 **GET /streams/{stream-name}/records** 方法，在 Kinesis 中呼叫 **GetRecords** 動作

1. 設定 **GET /streams/{stream-name}/records** 方法，如下所示：

[Method Execution](#)

**/streams/{stream-name}/records - GET - Integration Request**

Provide information about the target backend that this method will call and whether the incoming request data should be modified.

**Integration type**  Lambda Function [i](#)  
 HTTP [i](#)  
 Mock [i](#)  
 AWS Service [i](#)

**AWS Region** us-east-1 [e](#)

**AWS Service** Kinesis [e](#)

**AWS Subdomain** [e](#)

**HTTP method** POST [e](#)

**Action** GetRecords [e](#)

**Execution role** arn:aws:iam::7...:role/apigAwsProxyRole [e](#)

**Credentials cache** Do not add caller credentials to cache key [e](#)

**Content Handling** Passthrough [e](#) [i](#)

2. GetRecords 動作需要 ShardIterator 值的輸入。若要傳遞用戶端提供的 ShardIterator 值，我們將 Shard-Iterator 標頭參數新增至方法請求，如下所示：

◀ Method Execution /streams/{stream-name}/records - GET - Method Request 

Provide information about this method's authorization settings and the parameters it can receive.

Settings

Authorization NONE  

Request Validator NONE  

API Key Required false 

▶ Request Paths

▶ URL Query String Parameters

▼ HTTP Request Headers

Name	Required	Caching	
Shard-Iterator	<input type="checkbox"/>	<input type="checkbox"/>	 

 Add header

▶ Request Body 

▶ SDK Settings

3. 設定下列對應範本，以將 Shard-Iterator 標頭參數值對應至 Kinesis 中 GetRecords 動作之 JSON 承載的 ShardIterator 屬性值。

```
{  
    "ShardIterator": "$input.params('Shard-Iterator')"  
}
```

4. 使用 API Gateway 主控台中的 Test (測試) 選項，輸入現有串流名稱做為 stream-name Path (路徑) 變數值，並將 Shard-Iterator Header (標頭) 設定為取自 GET /streams/{stream-name}/sharditerator 方法 (上方) 測試回合的 ShardIterator 值，然後選擇 Test (測試)。

成功回應承載與下列輸出類似：

```
{  
    "MillisBehindLatest": 0,  
    "NextShardIterator": "AAAAAAAAAAF...",  
    "Records": [ ... ]  
}
```

## 做為 Kinesis 代理之範例 API 的 OpenAPI 定義

以下是用於本教學中做為 Kinesis 代理之範例 API 的 OpenAPI 定義。

### OpenAPI 3.0

```
{  
    "openapi": "3.0.0",  
    "info": {  
        "version": "2016-03-31T18:25:32Z",  
        "title": "KinesisProxy"  
    },  
    "paths": {  
        "/streams": {  
            "get": {  
                "responses": {  
                    "200": {  
                        "description": "200 response",  
                        "content": {  
                            "application/json": {  
                                "schema": {  
                                    "$ref": "#/components/schemas/Empty"  
                                }  
                            }  
                        }  
                    }  
                }  
            }  
        },  
        "x-amazon-apigateway-integration": {  
            "credentials": "arn:aws:iam::123456789012:role/apigAwsProxyRole",  
            "responses": {  
                "default": {  
                    "statusCode": "200"  
                }  
            },  
            "requestTemplates": {  
                "application/json": "{\n}"  
            },  
            "uri": "arn:aws:apigateway:us-east-1:kinesis:action/ListStreams",  
            "httpMethod": "POST",  
            "requestParameters": {  
                "integration.request.header.Content-Type": "'application/x-amz-  
json-1.1'"  
            },  
            "type": "aws"  
        }  
    },  
    "/streams/{stream-name)": {  
        "get": {  
            "parameters": [  
                {  
                    "name": "stream-name",  
                    "in": "path",  
                    "required": true,  
                    "schema": {  
                        "type": "string"  
                    }  
                }  
            ],  
            "responses": {  
                "200": {  
                    "description": "200 response",  
                    "content": {  
                        "application/json": {  
                            "schema": {  
                                "$ref": "#/components/schemas/Empty"  
                            }  
                        }  
                    }  
                }  
            }  
        }  
    }  
}
```

```

    },
    "x-amazon-apigateway-integration": {
        "credentials": "arn:aws:iam::123456789012:role/apigAwsProxyRole",
        "responses": {
            "default": {
                "statusCode": "200"
            }
        },
        "requestTemplates": {
            "application/json": "{\n              \"StreamName\": \"$input.params('stream-name')\"\n            }"
        },
        "uri": "arn:aws:apigateway:us-east-1:kinesis:action/DescribeStream",
        "httpMethod": "POST",
        "type": "aws"
    }
},
"post": {
    "parameters": [
        {
            "name": "stream-name",
            "in": "path",
            "required": true,
            "schema": {
                "type": "string"
            }
        }
    ],
    "responses": {
        "200": {
            "description": "200 response",
            "content": {
                "application/json": {
                    "schema": {
                        "$ref": "#/components/schemas/Empty"
                    }
                }
            }
        }
    }
},
"x-amazon-apigateway-integration": {
    "credentials": "arn:aws:iam::123456789012:role/apigAwsProxyRole",
    "responses": {
        "default": {
            "statusCode": "200"
        }
    },
    "requestTemplates": {
        "application/json": "{\n          \"ShardCount\": 5,\n          \"StreamName\": \"$input.params('stream-name')\"\n        }"
    },
    "uri": "arn:aws:apigateway:us-east-1:kinesis:action/CreateStream",
    "httpMethod": "POST",
    "requestParameters": {
        "integration.request.header.Content-Type": "'application/x-amz-json-1.1'"
    },
    "type": "aws"
}
},
"delete": {
    "parameters": [
        {
            "name": "stream-name",
            "in": "path",
            "required": true,
            "schema": {
                "type": "string"
            }
        }
    ]
}

```

```
        "required": true,
        "schema": {
            "type": "string"
        }
    }
],
"responses": {
    "200": {
        "description": "200 response",
        "headers": {
            "Content-Type": {
                "schema": {
                    "type": "string"
                }
            }
        },
        "content": {
            "application/json": {
                "schema": {
                    "$ref": "#/components/schemas/Empty"
                }
            }
        }
    },
    "400": {
        "description": "400 response",
        "headers": {
            "Content-Type": {
                "schema": {
                    "type": "string"
                }
            }
        }
    },
    "500": {
        "description": "500 response",
        "headers": {
            "Content-Type": {
                "schema": {
                    "type": "string"
                }
            }
        }
    }
},
"x-amazon-apigateway-integration": {
    "credentials": "arn:aws:iam::123456789012:role/apigAwsProxyRole",
    "responses": {
        "4\d{2}": {
            "statusCode": "400",
            "responseParameters": {
                "method.response.header.Content-Type": "integration.response.header.Content-Type"
            }
        },
        "default": {
            "statusCode": "200",
            "responseParameters": {
                "method.response.header.Content-Type": "integration.response.header.Content-Type"
            }
        },
        "5\d{2}": {
            "statusCode": "500",
            "responseParameters": {
```

```
        "method.response.header.Content-Type":  
        "integration.response.header.Content-Type"  
    }  
},  
"requestTemplates": {  
    "application/json": "{\n        \"StreamName\": \"$input.params('stream-  
name')\"\n    }  
},  
"uri": "arn:aws:apigateway:us-east-1:kinesis:action/DeleteStream",  
"httpMethod": "POST",  
"requestParameters": {  
    "integration.request.header.Content-Type": "'application/x-amz-  
json-1.1'"  
},  
    "type": "aws"  
}  
},  
"/streams/{stream-name}/record": {  
    "put": {  
        "parameters": [  
            {  
                "name": "stream-name",  
                "in": "path",  
                "required": true,  
                "schema": {  
                    "type": "string"  
                }  
            }  
        ],  
        "responses": {  
            "200": {  
                "description": "200 response",  
                "content": {  
                    "application/json": {  
                        "schema": {  
                            "$ref": "#/components/schemas/Empty"  
                        }  
                    }  
                }  
            }  
        }  
},  
"x-amazon-apigateway-integration": {  
    "credentials": "arn:aws:iam::123456789012:role/apigAwsProxyRole",  
    "responses": {  
        "default": {  
            "statusCode": "200"  
        },  
        "requestTemplates": {  
            "application/json": "{\n                \"StreamName\": \"$input.params('stream-  
name')\",\\n                \"Records\": [\n                    #foreach($elem in $input.path('$records'))\\n  
                    {\\n                        \"Data\": \"$util.base64Encode($elem.data)\"\\n                        \"PartitionKey\":  
                        \"$elem.partition-key\\n\" }#if($foreach.hasNext),#end\\n                    #end\\n                ]\\n            },"  
            "uri": "arn:aws:apigateway:us-east-1:kinesis:action/PutRecord",  
            "httpMethod": "POST",  
            "requestParameters": {  
                "integration.request.header.Content-Type": "'application/x-amz-  
json-1.1'"  
            },  
            "type": "aws"  
        }  
},  
}
```

```
"/streams/{stream-name}/records": {
    "get": {
        "parameters": [
            {
                "name": "stream-name",
                "in": "path",
                "required": true,
                "schema": {
                    "type": "string"
                }
            },
            {
                "name": "Shard-Iterator",
                "in": "header",
                "required": false,
                "schema": {
                    "type": "string"
                }
            }
        ],
        "responses": {
            "200": {
                "description": "200 response",
                "content": {
                    "application/json": {
                        "schema": {
                            "$ref": "#/components/schemas/Empty"
                        }
                    }
                }
            }
        }
    },
    "x-amazon-apigateway-integration": {
        "credentials": "arn:aws:iam::123456789012:role/apigAwsProxyRole",
        "responses": {
            "default": {
                "statusCode": "200"
            }
        },
        "requestTemplates": {
            "application/json": "{\n              \"ShardIterator\": \"$input.params('Shard-Iterator')\"\n            }"
        },
        "uri": "arn:aws:apigateway:us-east-1:kinesis:action/GetRecords",
        "httpMethod": "POST",
        "requestParameters": {
            "integration.request.header.Content-Type": "'application/x-amz-json-1.1'"
        },
        "type": "aws"
    }
},
"put": {
    "parameters": [
        {
            "name": "Content-Type",
            "in": "header",
            "required": false,
            "schema": {
                "type": "string"
            }
        },
        {
            "name": "stream-name",
            "in": "path",
            "required": true,
            "schema": {
                "type": "string"
            }
        }
    ]
}
```

```
        "schema": {
            "type": "string"
        }
    }
],
"responses": {
    "200": {
        "description": "200 response",
        "content": {
            "application/json": {
                "schema": {
                    "$ref": "#/components/schemas/Empty"
                }
            }
        }
    }
},
"x-amazon-apigateway-integration": {
    "credentials": "arn:aws:iam::123456789012:role/apigAwsProxyRole",
    "responses": {
        "default": {
            "statusCode": "200"
        }
    },
    "requestTemplates": {
        "application/json": "{\n            \"StreamName\": \"$input.params('stream-name')\", \n            \"Records\": [\n                {\n                    \"Data\": \"$util.base64Encode($elem.data)\", \n                    \"PartitionKey\": \"$elem.partition-key\"\n                }#if($foreach.hasNext),#end\n            ]\n        },\n        \"application/x-amz-json-1.1\": \"{\\n            \"StreamName\": \"$input.params('stream-name')\", \\n            \"records\": [\\n                {\\n                    \"Data\": \"$elem.data\", \\n                    \"PartitionKey\": \"$elem.partition-key\"\\n                }#if($foreach.hasNext),#end\\n            ]\\n        }\""
    },
    "uri": "arn:aws:apigateway:us-east-1:kinesis:action/PutRecords",
    "httpMethod": "POST",
    "requestParameters": {
        "integration.request.header.Content-Type": "'application/x-amz-json-1.1'"
    },
    "type": "aws"
},
"requestBody": {
    "content": {
        "application/json": {
            "schema": {
                "$ref": "#/components/schemas/PutRecordsMethodRequestPayload"
            }
        },
        "application/x-amz-json-1.1": {
            "schema": {
                "$ref": "#/components/schemas/PutRecordsMethodRequestPayload"
            }
        }
    },
    "required": true
}
}
},
"/streams/{stream-name}/sharditerator": {
    "get": {
        "parameters": [
            {
                "name": "stream-name",
                "in": "path",
                "required": true,
                "type": "string"
            }
        ]
    }
}
```

```
        "schema": {
            "type": "string"
        }
    },
    {
        "name": "shard-id",
        "in": "query",
        "required": false,
        "schema": {
            "type": "string"
        }
    }
],
"responses": {
    "200": {
        "description": "200 response",
        "content": {
            "application/json": {
                "schema": {
                    "$ref": "#/components/schemas/Empty"
                }
            }
        }
    }
},
"x-amazon-apigateway-integration": {
    "credentials": "arn:aws:iam::123456789012:role/apigAwsProxyRole",
    "responses": {
        "default": {
            "statusCode": "200"
        }
    },
    "requestTemplates": {
        "application/json": "{$input.params('shard-id')},\n      \"ShardIteratorType\": \"$TRIM_HORIZON\",\\n      \"StreamName\":\n      \"$input.params('stream-name')\"\n    },\n    \"uri\": \"arn:aws:apigateway:us-east-1:kinesis:action/GetShardIterator\",\n    \"httpMethod\": \"POST\",\n    \"requestParameters\": {\n        \"integration.request.header.Content-Type\": \"application/x-amz-json-1.1\"\n    },\n    \"type\": \"aws\"\n  }
}
},
"servers": [
{
    "url": "https://wd4zclrobb.execute-api.us-east-1.amazonaws.com/{basePath}",
    "variables": {
        "basePath": {
            "default": "/test"
        }
    }
}
],
"components": {
    "schemas": {
        "PutRecordsMethodRequestPayload": {
            "type": "object",
            "properties": {
                "records": {
                    "type": "array",
                    "items": {
                        "type": "string"
                    }
                }
            }
        }
    }
}
```

```
        "type": "object",
        "properties": {
            "data": {
                "type": "string"
            },
            "partition-key": {
                "type": "string"
            }
        }
    }
},
"Empty": {
    "type": "object"
}
}
}
```

### OpenAPI 2.0

```
{
  "swagger": "2.0",
  "info": {
    "version": "2016-03-31T18:25:32Z",
    "title": "KinesisProxy"
  },
  "host": "wd4zclrobb.execute-api.us-east-1.amazonaws.com",
  "basePath": "/test",
  "schemes": [
    "https"
  ],
  "paths": {
    "/streams": {
      "get": {
        "consumes": [
          "application/json"
        ],
        "produces": [
          "application/json"
        ],
        "responses": {
          "200": {
            "description": "200 response",
            "schema": {
              "$ref": "#/definitions/Empty"
            }
          }
        },
        "x-amazon-apigateway-integration": {
          "credentials": "arn:aws:iam::123456789012:role/apigAwsProxyRole",
          "responses": {
            "default": {
              "statusCode": "200"
            }
          },
          "requestTemplates": {
            "application/json": "{\n}"
          },
          "uri": "arn:aws:apigateway:us-east-1:kinesis:action/ListStreams",
          "httpMethod": "POST",
          "requestParameters": {
            "integration.request.header.Content-Type": "'application/x-amz-json-1.1'"
          }
        }
      }
    }
  }
}
```

```
        "type": "aws"
    }
}
},
"/streams/{stream-name}": {
    "get": {
        "consumes": [
            "application/json"
        ],
        "produces": [
            "application/json"
        ],
        "parameters": [
            {
                "name": "stream-name",
                "in": "path",
                "required": true,
                "type": "string"
            }
        ],
        "responses": {
            "200": {
                "description": "200 response",
                "schema": {
                    "$ref": "#/definitions/Empty"
                }
            }
        },
        "x-amazon-apigateway-integration": {
            "credentials": "arn:aws:iam::123456789012:role/apigAwsProxyRole",
            "responses": {
                "default": {
                    "statusCode": "200"
                }
            },
            "requestTemplates": {
                "application/json": "{\n        \"StreamName\": \"$input.params('stream-name')\"\n    }",
                "uri": "arn:aws:apigateway:us-east-1:kinesis:action/DescribeStream",
                "httpMethod": "POST",
                "type": "aws"
            }
        },
        "post": {
            "consumes": [
                "application/json"
            ],
            "produces": [
                "application/json"
            ],
            "parameters": [
                {
                    "name": "stream-name",
                    "in": "path",
                    "required": true,
                    "type": "string"
                }
            ],
            "responses": {
                "200": {
                    "description": "200 response",
                    "schema": {
                        "$ref": "#/definitions/Empty"
                    }
                }
            }
        }
    }
}
```

```
},
"x-amazon-apigateway-integration": {
  "credentials": "arn:aws:iam::123456789012:role/apigAwsProxyRole",
  "responses": {
    "default": {
      "statusCode": "200"
    }
  },
  "requestTemplates": {
    "application/json": "{$input.params('stream-name')}\n"
  },
  "uri": "arn:aws:apigateway:us-east-1:kinesis:action/CreateStream",
  "httpMethod": "POST",
  "requestParameters": {
    "integration.request.header.Content-Type": "'application/x-amz-json-1.1'"
  },
  "type": "aws"
},
"delete": {
  "consumes": [
    "application/json"
  ],
  "produces": [
    "application/json"
  ],
  "parameters": [
    {
      "name": "stream-name",
      "in": "path",
      "required": true,
      "type": "string"
    }
  ],
  "responses": {
    "200": {
      "description": "200 response",
      "schema": {
        "$ref": "#/definitions/Empty"
      },
      "headers": {
        "Content-Type": {
          "type": "string"
        }
      }
    },
    "400": {
      "description": "400 response",
      "headers": {
        "Content-Type": {
          "type": "string"
        }
      }
    },
    "500": {
      "description": "500 response",
      "headers": {
        "Content-Type": {
          "type": "string"
        }
      }
    }
  }
},
"x-amazon-apigateway-integration": {
  "credentials": "arn:aws:iam::123456789012:role/apigAwsProxyRole",
```

```
"responses": {
  "4\d{2}": {
    "statusCode": "400",
    "responseParameters": {
      "method.response.header.Content-Type":
        "integration.response.header.Content-Type"
    }
  },
  "default": {
    "statusCode": "200",
    "responseParameters": {
      "method.response.header.Content-Type":
        "integration.response.header.Content-Type"
    }
  },
  "5\d{2}": {
    "statusCode": "500",
    "responseParameters": {
      "method.response.header.Content-Type":
        "integration.response.header.Content-Type"
    }
  }
},
"requestTemplates": {
  "application/json": "{\n    \"StreamName\": \"$input.params('stream-name')\"\n}\n",
  "uri": "arn:aws:apigateway:us-east-1:kinesis:action/DeleteStream",
  "httpMethod": "POST",
  "requestParameters": {
    "integration.request.header.Content-Type": "'application/x-amz-json-1.1'"
  },
  "type": "aws"
}
},
"/streams/{stream-name}/record": {
  "put": {
    "consumes": [
      "application/json"
    ],
    "produces": [
      "application/json"
    ],
    "parameters": [
      {
        "name": "stream-name",
        "in": "path",
        "required": true,
        "type": "string"
      }
    ],
    "responses": {
      "200": {
        "description": "200 response",
        "schema": {
          "$ref": "#/definitions/Empty"
        }
      }
    }
  },
  "x-amazon-apigateway-integration": {
    "credentials": "arn:aws:iam::123456789012:role/apigAwsProxyRole",
    "responses": {
      "default": {
        "statusCode": "200"
      }
    }
  }
}
```



```
"consumes": [
    "application/json",
    "application/x-amz-json-1.1"
],
"produces": [
    "application/json"
],
"parameters": [
    {
        "name": "Content-Type",
        "in": "header",
        "required": false,
        "type": "string"
    },
    {
        "name": "stream-name",
        "in": "path",
        "required": true,
        "type": "string"
    },
    {
        "in": "body",
        "name": "PutRecordsMethodRequestPayload",
        "required": true,
        "schema": {
            "$ref": "#/definitions/PutRecordsMethodRequestPayload"
        }
    }
],
"responses": {
    "200": {
        "description": "200 response",
        "schema": {
            "$ref": "#/definitions/Empty"
        }
    }
},
"x-amazon-apigateway-integration": {
    "credentials": "arn:aws:iam::123456789012:role/apigAwsProxyRole",
    "responses": {
        "default": {
            "statusCode": "200"
        }
    },
    "requestTemplates": {
        "application/json": "{\n            \"StreamName\": \"$input.params('stream-name')\", \n            \"Records\": [\n                #foreach($elem in $input.path('$..records'))\n                {\n                    \"Data\": \"$util.base64Encode($elem.data)\" ,\n                    \"PartitionKey\": \"$elem.partition-key\"\n                }#if($foreach.hasNext),#end\n            ]\n        },\n        \"application/x-amz-json-1.1\": \"$set($inputRoot = $input.path('$'))\n        \"$inputRoot.StreamName\": \"$input.params('stream-name')\", \n        \"$inputRoot.records\" : [\n            #foreach($elem in $inputRoot.records)\n                {\n                    \"Data\" : \"$elem.data\", \n                    \"PartitionKey\" : \"$elem.partition-key\"\n                }#if($foreach.hasNext),#end\n            ]\n        ]\n    },\n    \"uri\": \"arn:aws:apigateway:us-east-1:kinesis:action/PutRecords\",\n    \"httpMethod\": \"POST\",\n    \"requestParameters\": {\n        \"integration.request.header.Content-Type\": \"application/x-amz-json-1.1\"\n    },\n    \"type\": \"aws\"\n}
},
"/streams/{stream-name}/sharditerator": {
```

```
"get": {
    "consumes": [
        "application/json"
    ],
    "produces": [
        "application/json"
    ],
    "parameters": [
        {
            "name": "stream-name",
            "in": "path",
            "required": true,
            "type": "string"
        },
        {
            "name": "shard-id",
            "in": "query",
            "required": false,
            "type": "string"
        }
    ],
    "responses": {
        "200": {
            "description": "200 response",
            "schema": {
                "$ref": "#/definitions/Empty"
            }
        }
    },
    "x-amazon-apigateway-integration": {
        "credentials": "arn:aws:iam::123456789012:role/apigAwsProxyRole",
        "responses": {
            "default": {
                "statusCode": "200"
            }
        },
        "requestTemplates": {
            "application/json": "{\n    \"ShardId\": \"$input.params('shard-\n        id')\", \n    \"ShardIteratorType\": \"TRIM_HORIZON\", \n    \"StreamName\":\n        \"$input.params('stream-name')\"\n}\n"
        },
        "uri": "arn:aws:apigateway:us-east-1:kinesis:action/GetShardIterator",
        "httpMethod": "POST",
        "requestParameters": {
            "integration.request.header.Content-Type": "application/x-amz-json-1.1"
        },
        "type": "aws"
    }
},
"definitions": {
    "PutRecordsMethodRequestPayload": {
        "type": "object",
        "properties": {
            "records": {
                "type": "array",
                "items": {
                    "type": "object",
                    "properties": {
                        "data": {
                            "type": "string"
                        },
                        "partition-key": {
                            "type": "string"
                        }
                    }
                }
            }
        }
    }
}
```

```
        }
      }
    },
    "Empty": {
      "type": "object"
    }
}
```

# 在 Amazon API Gateway 中建立、部署和呼叫 REST API

## 主題

- 在 Amazon API Gateway 中建立 REST API (p. 167)
- 在 API Gateway 中控制和管理 REST API 的存取 (p. 324)
- 記錄 API Gateway 中的 REST API (p. 411)
- 在 Amazon API Gateway 中更新和維護 REST API (p. 453)
- 在 Amazon API Gateway 中部署 REST API (p. 457)
- 在 Amazon API Gateway 中呼叫 REST API (p. 497)
- 追蹤、記錄並監控 API Gateway API (p. 518)
- OpenAPI 的 API Gateway 延伸 (p. 533)

## 在 Amazon API Gateway 中建立 REST API

在 Amazon API Gateway 中，您可以建置一個 REST API，其中包含可程式化的實體集合，稱為 API Gateway 資源。例如，您可以使用 [RestApi](#) 資源來代表可包含 [Resource](#) 實體集合的 API。每個 [Resource](#) 實體則可以有一或多個 [Method](#) 資源。[Method](#) 是以請求參數與內文表示，其會定義用戶端用來存取所公開之 [Resource](#) 的應用程式開發界面，並代表用戶端所提交的傳入請求。然後，您可以透過將傳入請求轉送到指定的整合端點 URI，建立 [Integration](#) 資源來整合 [Method](#) 與後端端點（也稱為整合端點）。如果需要，您可以轉換請求參數或內文，以符合後端需求。針對回應，您可以建立 [MethodResponse](#) 資源來代表用戶端接收的請求回應，並建立 [IntegrationResponse](#) 資源來代表後端傳回的請求回應。您可以設定整合回應來轉換後端回應資料，再將資料傳回用戶端或將後端回應依現狀傳遞到用戶端。

為了協助您的客戶了解 API，您也可以在建立 API 期間或之後提供 API 的文件。若要啟用此功能，請新增受支援 API 實體的 [DocumentationPart](#) 資源。

若要控制用戶端呼叫 API 的方式，請使用 [IAM 許可](#) (p. 333)、[Lambda 授權方](#) (p. 348) 或 [Amazon Cognito 使用者集區](#) (p. 363)。若要測量 API 的用量，請設定 [usage plans \(用量方案\)](#) (p. 397) 來調節 API 請求。您可以在建立或更新 API 時啟用這些功能。

您可以使用 API Gateway 主控台、API Gateway REST API、AWS CLI 或其中一個 AWS 開發套件，來執行這些與其他任務。我們接下來將會討論如何執行這些任務。

## 主題

- [選擇端點類型以設定 API Gateway API](#) (p. 168)
- [在 API Gateway 中初始化 REST API 設定](#) (p. 168)
- [在 API Gateway 中設定 REST API 方法](#) (p. 188)
- [在 API Gateway 中設定 REST API 整合](#) (p. 199)
- [設定閘道回應以自訂錯誤回應](#) (p. 237)
- [設定 API Gateway 請求和回應資料對應](#) (p. 242)
- [支援 API Gateway 中的二進位承載](#) (p. 283)
- [啟用 API 的承載壓縮功能](#) (p. 304)

- 在 API Gateway 中啟用請求驗證 (p. 307)
- 將 REST API 匯入 API Gateway (p. 319)

## 選擇端點類型以設定 API Gateway API

API 端點 (p. 5) 類型是指 API 的主機名稱。根據您大部分 API 流量的來源位置，API 端點類型可以是邊緣最佳化、區域或私有。

### 邊緣最佳化的 API 端點

邊緣最佳化 API 端點 (p. 6) 是最適用於散佈在多個地理位置的用戶端。API 請求會路由到最近的 CloudFront 出現點 (POP)。這是適用於 API Gateway REST API 的預設端點類型。

邊緣最佳化 API 會提供 HTTP 標頭的名稱 (例如，Cookie)。

在自然順序中 CloudFront 藉由 HTTP Cookie 名稱排序 Cookie，再轉送請求到原始伺服器。如需 CloudFront 處理 Cookie 方式的詳細資訊，請參閱 [根據 Cookie 快取內容](#)。

針對邊緣最佳化 API，您使用的任何自訂網域名稱適用於所有區域。

### 區域 API 端點

區域 API 端點 (p. 7) 適用於相同區域中的用戶端。當 EC2 執行個體上執行的用戶端呼叫相同區域中的 API 時，或當 API 是為了提供服務給具有高需求的少量用戶端時，區域 API 可降低連線成本。如需詳細資訊，請參閱 [the section called “設定區域 API” \(p. 181\)](#)。

針對區域 API，您使用的任何自訂網域名稱是專屬於 API 的部署區域。如果您部署在多個區域中的區域性 API，它可以在所有區域中擁有相同的自訂網域名稱。您可以使用自訂網域搭配 Amazon Route 53 來執行任務，例如 [以延遲為基礎的路由](#)。如需更多詳細資訊，請參閱 [the section called “設定區域性自訂網域名稱” \(p. 599\)](#) 及 [the section called “如何建立邊緣最佳化自訂網域名稱” \(p. 593\)](#)。

所有區域和 API 端點會依原狀傳遞所有標頭名稱。

### 私有 API 端點

私有 API 端點 (p. 7) 是僅能從 Amazon Virtual Private Cloud (VPC) 透過界面 VPC 端點存取的 API 端點；此端點是您在 VPC 中建立的端點網路界面 (ENI)。如需詳細資訊，請參閱 [the section called “建立私有 API” \(p. 183\)](#)。

所有私有 API 端點會依原狀傳遞所有標頭名稱。

## 在 API Gateway 中初始化 REST API 設定

在此範例中，我們使用簡化的 PetStore API，並搭配公開 GET /pets 與 GET /pets/{petId} 方法的 HTTP 整合。這些方法與兩個 HTTP 端點整合，分別是 `http://petstore-demo-endpoint.execute-api.com/petstore/pets` 與 `http://petstore-demo-endpoint.execute-api.com/petstore/pets/{petId}`。此 API 會處理 200 OK 回應。這些範例專注於在 API Gateway 中建立 API 的基本程式設計任務，並盡可能利用預設設定。

由於預設設定，產生的 API 是邊緣最佳化 API。或者，您也可以 [設定區域 API \(p. 181\)](#)。若要設定區域 API，您必須將 API 的端點類型明確設定為 REGIONAL。若要明確設定邊緣最佳化 API，您可以將 EDGE 設定為 endpointConfiguration 類型。

設定 API 時，您必須選擇區域。部署時，API 是區域特定的。邊緣最佳化 API 的基底 URL 格式為 `http[s]://{{restapi-id}}.execute-api.amazonaws.com/stage`，其中 `{{restapi-id}}` 是 API

Gateway 所產生之 API 的 [id](#) 值。您可以將自訂網域名稱 (例如 `apis.example.com`) 指派為 API 的主機名稱，並使用 `https://apis.example.com/myApi` 格式的基底 URL 呼叫 API。

#### 主題

- [使用 API Gateway 主控台設定 API \(p. 169\)](#)
- [使用 AWS CLI 命令設定邊緣最佳化 API \(p. 169\)](#)
- [使用適用於 Node.js 的 AWS 開發套件設定邊緣最佳化 API \(p. 173\)](#)
- [匯入 OpenAPI 定義以設定邊緣最佳化 API \(p. 180\)](#)
- [在 API Gateway 中設定區域 API \(p. 181\)](#)
- [在 Amazon API Gateway 中建立私有 API \(p. 183\)](#)

## 使用 API Gateway 主控台設定 API

若要使用 API Gateway 主控台設定 API Gateway API，請參閱[教學：建置具有 HTTP 非代理整合的 API \(p. 52\)](#)。

您可以遵循範例來了解如何設定 API。如需詳細資訊，請參閱[教學課程：匯入範例來建立 REST API \(p. 40\)](#)。

或者，您也可以使用 API Gateway [匯入 API \(p. 319\)](#) 功能上傳外部 API 定義來設定 API，例如透過 [OpenAPI 的 API Gateway 延伸 \(p. 533\)](#) 以 OpenAPI 2.0 表示的定義。「[教學課程：匯入範例來建立 REST API \(p. 40\)](#)」中提供的範例使用此匯入 API 功能。

## 使用 AWS CLI 命令設定邊緣最佳化 API

使用 AWS CLI 設定 API 需要用到 `create-rest-api`、`create-resource` 或 `get-resources`、`put-method`、`put-method-response`、`put-integration` 與 `put-integration-response` 命令。下列程序示範如何使用這些 AWS CLI 命令來建立 HTTP 整合類型的簡單 PetStore API。

### 使用 AWS CLI 建立簡單的 PetStore API

1. 呼叫 `create-rest-api` 命令，在特定區域 (`RestApi`) 中設定 `us-west-2`。

```
aws apigateway create-rest-api --name 'Simple PetStore (AWS CLI)' --region us-west-2
```

以下是此命令的輸出：

```
{  
  "name": "Simple PetStore (AWS CLI)",  
  "id": "vaz7da96z6",  
  "createdDate": 1494572809  
}
```

記下新建立之 `id` 的傳回 `RestApi`。您需要它才能設定 API 的其他部分。

2. 呼叫 `get-resources` 命令，擷取 `RestApi` 的根資源識別符。

```
aws apigateway get-resources --rest-api-id vaz7da96z6 --region us-west-2
```

以下是此命令的輸出：

```
{  
  "items": [  
    {  
      "path": "/",  
      "method": "GET",  
      "httpMethod": "GET",  
      "resourceId": "vaz7da96z6",  
      "parentResourceId": null  
    }  
  ]  
}
```

```
        "id": "begaltmsm8"
    }
}
```

記下根資源 Id。您需要它才能開始設定 API 的資源樹狀目錄，並設定方法與整合。

3. 呼叫 `create-resource` 命令，在根資源 (`pets`) 下附加子資源 (`begaltmsm8`)：

```
aws apigateway create-resource --rest-api-id vaz7da96z6 \
--region us-west-2 \
--parent-id begaltmsm8 \
--path-part pets
```

以下是此命令的輸出：

```
{
  "path": "/pets",
  "pathPart": "pets",
  "id": "6sxz2j",
  "parentId": "begaltmsm8"
}
```

若要在根資源下附加子資源，請將根資源 Id 指定為 `parentId` 屬性值。同樣地，若要在 `pets` 資源下附加子資源，請重複上述步驟，並以 `parent-id` 的 `pets` 資源 id 取代 `6sxz2j` 值：

```
aws apigateway create-resource --rest-api-id vaz7da96z6 \
--region us-west-2 \
--parent-id 6sxz2j \
--path-part '{petId}'
```

若要將路徑部分設為路徑參數，請以一對大括號括住。如果成功，此命令會傳回下列回應：

```
{
  "path": "/pets/{petId}",
  "pathPart": "{petId}",
  "id": "rjkmth",
  "parentId": "6sxz2j"
}
```

現在您已建立兩個資源：`/pets (6sxz2j)` 與 `/pets/{petId} (rjkmth)`，您可以繼續設定其方法。

4. 呼叫 `put-method` 命令，新增 GET 資源的 `/pets` HTTP 方法。這會建立 Method 之 API GET `/pets` 的開放式存取，並以 `/pets` 資源的 ID 值 `6sxz2j` 參考資源。

```
aws apigateway put-method --rest-api-id vaz7da96z6 \
--resource-id 6sxz2j \
--http-method GET \
--authorization-type "NONE" \
--region us-west-2
```

以下是此命令的成功輸出：

```
{
  "apiKeyRequired": false,
  "httpMethod": "GET",
  "authorizationType": "NONE"
}
```

此方法適用於開放式存取，因為 `authorization-type` 設定為 `NONE`。若只要允許已驗證的使用者呼叫方法，您可以使用 IAM 角色與政策、Lambda 授權方（先前稱為自訂授權方）或 Amazon Cognito 使用者集區。如需詳細資訊，請參閱[the section called “控制和管理 REST API 的存取” \(p. 324\)](#)。

若要啟用 `/pets/{petId}` 資源 (`rjkmth`) 的讀取權限，請對其新增 GET HTTP 方法來建立 Method 的 API GET `/pets/{petId}`，如下所示。

```
aws apigateway put-method --rest-api-id vaz7da96z6 \
    --resource-id rjkmth --http-method GET \
    --authorization-type "NONE" \
    --region us-west-2 \
    --request-parameters method.request.path.petId=true
```

以下是此命令的成功輸出：

```
{
  "apiKeyRequired": false,
  "httpMethod": "GET",
  "authorizationType": "NONE",
  "requestParameters": {
    "method.request.path.petId": true
  }
}
```

請注意，`petId` 的方法請求路徑參數必須指定為必要的請求參數，才能將其動態設定的值對應到相應的整合請求參數並傳遞到後端。

5. 呼叫 `put-method-response` 命令，設定 GET `/pets` 方法的 200 OK 回應，並以 `/pets` 資源的 ID 值 `6sxz2j` 指定資源。

```
aws apigateway put-method-response --rest-api-id vaz7da96z6 \
    --resource-id 6sxz2j --http-method GET \
    --status-code 200 --region us-west-2
```

以下是此命令的輸出：

```
{
  "statusCode": "200"
}
```

同樣地，若要設定 GET `/pets/{petId}` 方法的 200 OK 回應，請執行下列操作，並以 `/pets/{petId}` 資源的 ID 值 `rjkmth` 指定資源：

```
aws apigateway put-method-response --rest-api-id vaz7da96z6 \
    --resource-id rjkmth --http-method GET \
    --status-code 200 --region us-west-2
```

設定 API 的簡單用戶端界面之後，您可以繼續設定 API 方法與後端的整合。

6. 呼叫 `put-integration` 命令，為 Integration 方法設定與指定 HTTP 端點的 GET `/pets`。`/pets` 資源是由其資源 ID `6sxz2j` 識別：

```
aws apigateway put-integration --rest-api-id vaz7da96z6 \
    --resource-id 6sxz2j --http-method GET --type HTTP \
    --integration-http-method GET \
    --uri 'http://petstore-demo-endpoint.execute-api.com/petstore/pets' \
    --region us-west-2
```

以下是此命令的輸出：

```
{  
    "httpMethod": "GET",  
    "passthroughBehavior": "WHEN_NO_MATCH",  
    "cacheKeyParameters": [],  
    "type": "HTTP",  
    "uri": "http://petstore-demo-endpoint.execute-api.com/petstore/pets",  
    "cacheNamespace": "6sxz2j"  
}
```

請注意，uri 的整合 `http://petstore-demo-endpoint.execute-api.com/petstore/pets` 指定 GET /pets 方法的整合端點。

同樣地，您可以建立 GET /pets/{petId} 方法的整合請求，如下所示：

```
aws apigateway put-integration \  
    --rest-api-id vaz7da96z6 \  
    --resource-id rjkmth \  
    --http-method GET \  
    --type HTTP \  
    --integration-http-method GET \  
    --uri 'http://petstore-demo-endpoint.execute-api.com/petstore/pets/{id}' \  
    --request-parameters  
    '{"integration.request.path.id":"method.request.path.petId"}' \  
    --region us-west-2
```

在本例中，整合端點 (uri 的 `http://petstore-demo-endpoint.execute-api.com/petstore/pets/{id}`) 也使用路徑參數 (id)。其值是對應自 {petId} 的相應方法請求路徑參數。此對應會當做 request-parameters 的一部分來定義。如果未在這裡定義此對應，用戶端會在嘗試呼叫方法時收到錯誤回應。

以下是此命令的輸出：

```
{  
    "passthroughBehavior": "WHEN_NO_MATCH",  
    "cacheKeyParameters": [],  
    "uri": "http://petstore-demo-endpoint.execute-api.com/petstore/pets/{id}",  
    "httpMethod": "GET",  
    "cacheNamespace": "rjkmth",  
    "type": "HTTP",  
    "requestParameters": {  
        "integration.request.path.id": "method.request.path.petId"  
    }  
}
```

7. 呼叫 put-integration-response 命令，建立與 HTTP 後端整合之 IntegrationResponse 方法的 GET /pets。

```
aws apigateway put-integration-response --rest-api-id vaz7da96z6 \  
    --resource-id 6sxz2j --http-method GET \  
    --status-code 200 --selection-pattern "" \  
    --region us-west-2
```

以下是此命令的輸出：

```
{  
    "selectionPattern": "",
```

```
    "statusCode": "200"
}
```

同樣地，呼叫下列 `put-integration-response` 命令建立 `GET /pets/{petId}` 方法的 `IntegrationResponse`：

```
aws apigateway put-integration-response --rest-api-id vaz7da96z6 \
--resource-id rjkmth --http-method GET
--status-code 200 --selection-pattern ""
--region us-west-2
```

執行上述步驟之後，您已完成設定簡單的 API，讓您的客戶在 PetStore 網站上查詢可用的寵物，並檢視指定識別符的個別寵物。您必須部署 API，您的客戶才能呼叫它。

8. 例如，呼叫 `stage` 將 API 部署到 `create-deployment` 階段：

```
aws apigateway create-deployment --rest-api-id vaz7da96z6 \
--region us-west-2 \
--stage-name test \
--stage-description 'Test stage' \
--description 'First deployment'
```

您可以在瀏覽器中輸入 `https://vaz7da96z6.execute-api.us-west-2.amazonaws.com/test/pets` URL 來測試此 API，並以您 API 的識別符替代 `vaz7da96z6`。預期的輸出應如下所示：

```
[  
 {  
   "id": 1,  
   "type": "dog",  
   "price": 249.99  
 },  
 {  
   "id": 2,  
   "type": "cat",  
   "price": 124.99  
 },  
 {  
   "id": 3,  
   "type": "fish",  
   "price": 0.99  
 }]
```

若要測試 `GET /pets/{petId}` 方法，請在瀏覽器中輸入 `https://vaz7da96z6.execute-api.us-west-2.amazonaws.com/test/pets/3`。您應該會收到下列回應：

```
{  
   "id": 3,  
   "type": "fish",  
   "price": 0.99  
 }
```

## 使用適用於 Node.js 的 AWS 開發套件設定邊緣最佳化 API

做為說明，我們使用適用於 Node.js 的 AWS 開發套件說明如何使用 AWS 開發套件來建立 API Gateway API。如需使用 AWS 開發套件的詳細資訊，包括如何設定開發環境，請參閱 [AWS 開發套件](#)。

使用適用於 Node.js 的 AWS 開發套件設定 API 需要呼叫 `createRestApi`、`createResource` 或 `getResources`、`putMethod`、`putMethodResponse`、`putIntegration` 與 `putIntegrationResponse` 函數。

下列程序將引導您完成基本步驟，來使用這些開發套件命令設定支援 `GET /pets` 與 `GET /pets/{petId}` 方法的簡單 PetStore API。

使用適用於 Node.js 的 AWS 開發套件設定簡單的 PetStore API

1. 執行個體化開發套件：

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

AWS.config.region = 'us-west-2';
var apig = new AWS.APIGateway({apiVersion: '2015/07/09'});
```

2. 呼叫 `createRestApi` 函數，設定 RestApi 實體。

```
apig.createRestApi({
  name: "Simple PetStore (node.js SDK)",
  binaryMediaTypes: [
    '*'
  ],
  description: "Demo API created using the AWS SDK for node.js",
  version: "0.00.001"
}, function(err, data){
  if (!err) {
    console.log(data);
  } else {
    console.log('Create API failed:\n', err);
  }
});
```

此函數會傳回與下列結果類似的輸出：

```
{
  id: 'iuo308uaq7',
  name: 'PetStore (node.js SDK)',
  description: 'Demo API created using the AWS SDK for node.js',
  createdDate: 2017-09-05T19:32:35.000Z,
  version: '0.00.001',
  binaryMediaTypes: [ '*' ]
}
```

產生的 API 識別符為 `iuo308uaq7`。您需要提供此值才能繼續設定 API。

3. 呼叫 `getResources` 函數，擷取 RestApi 的根資源識別符。

```
apig.getResources({
  restApiId: 'iuo308uaq7'
}, function(err, data){
  if (!err) {
    console.log(data);
  } else {
    console.log('Get the root resource failed:\n', err);
  }
});
```

此函數會傳回與下列結果類似的輸出：

```
{  
    "items": [  
        {  
            "path": "/",  
            "id": "s4fb0trnk0"  
        }  
    ]  
}
```

根資源識別符為 s4fb0trnk0。這是您接下來建置 API 資源樹狀目錄的起點。

4. 呼叫 `createResource` 函數，設定 API 的 `/pets` 資源，並在 `s4fb0trnk0` 屬性上指定根資源識別符 (`parentId`)。

```
apig.createResource({  
    restApiId: 'iuo308uaq7',  
    parentId: 's4fb0trnk0',  
    pathPart: 'pets'  
}, function(err, data){  
    if (!err) {  
        console.log(data);  
    } else {  
        console.log("The '/pets' resource setup failed:\n", err);  
    }  
})
```

成功結果如下所示：

```
{  
    "path": "/pets",  
    "pathPart": "pets",  
    "id": "8sxa2j",  
    "parentId": "s4fb0trnk0"  
}
```

若要設定 `/pets/{petId}` 資源，請呼叫下列 `createResource` 函數，並在 `/pets` 屬性上指定新建立的 `8sxa2j` 資源 (`parentId`)。

```
apig.createResource({  
    restApiId: 'iuo308uaq7',  
    parentId: '8sxa2j',  
    pathPart: '{petId}'  
}, function(err, data){  
    if (:err) {  
        console.log(data);  
    } else {  
        console.log("The '/pets/{petId}' resource setup failed:\n", err);  
    }  
})
```

成功結果會傳回新建立的資源 id 值：

```
{  
    "path": "/pets/{petId}",  
    "pathPart": "{petId}",  
    "id": "au5df2",  
    "parentId": "8sxa2j"  
}
```

在此程序中，您會指定 /pets 資源的資源 ID 8sxa2j 來參考資源，並指定 /pets/{petId} 資源的資源 ID au5df2 來參考資源。

5. 呼叫 putMethod 函數，新增 GET 資源 (/pets) 的 8sxa2j HTTP 方法。這會設定 GET /pets Method 的開放式存取。

```
apig.putMethod({  
    restApiId: 'iuo308uaq7',  
    resourceId: '8sxa2j',  
    httpMethod: 'GET',  
    authorizationType: 'NONE'  
}, function(err, data){  
    if (!err) {  
        console.log(data);  
    } else {  
        console.log("The 'GET /pets' method setup failed:\n", err);  
    }  
})
```

此函數會傳回與下列結果類似的輸出：

```
{  
    "apiKeyRequired": false,  
    "httpMethod": "GET",  
    "authorizationType": "NONE"  
}
```

若要新增 /pets/{petId} 資源 (au5df2) 的 GET HTTP 方法，其設定 GET /pets/{petId} 之 API 方法的開放式存取，請呼叫 putMethod 函數，如下所示。

```
apig.putMethod({  
    restApiId: 'iuo308uaq7',  
    resourceId: 'au5df2',  
    httpMethod: 'GET',  
    authorizationType: 'NONE',  
    requestParameters: {  
        "method.request.path.petId" : true  
    }  
}, function(err, data){  
    if (!err) {  
        console.log(data);  
    } else {  
        console.log("The 'GET /pets/{petId}' method setup failed:\n", err);  
    }  
})
```

此函數會傳回與下列結果類似的輸出：

```
{  
    "apiKeyRequired": false,  
    "httpMethod": "GET",  
    "authorizationType": "NONE",  
    "requestParameters": {  
        "method.request.path.petId": true  
    }  
}
```

您需要如上述範例所示設定 requestParameters 屬性，以將用戶端提供的 petId 值對應並傳遞到後端。

6. 呼叫 `putMethodResponse` 函數，設定 GET /pets 方法的方法回應。

```
apig.putMethodResponse({
  restApiId: 'iuo308uaq7',
  resourceId: '8sxa2j',
  httpMethod: 'GET',
  statusCode: '200'
}, function(err, data){
  if (!err) {
    console.log(data);
  } else {
    console.log("Set up the 200 OK response for the 'GET /pets' method failed:\n", err);
  }
})
```

此函數會傳回與下列結果類似的輸出：

```
{
  "statusCode": "200"
}
```

若要設定 GET /pets/{petId} 方法的 200 OK 回應，請呼叫 `putMethodResponse` 函數，並在 /pets/{petId} 屬性上指定 au5df2 資源識別符 (resourceId)。

```
apig.putMethodResponse({
  restApiId: 'iuo308uaq7',
  resourceId: 'au5df2',
  httpMethod: 'GET',
  statusCode: '200'
}, function(err, data){
  if (!err) {
    console.log(data);
  } else {
    console.log("Set up the 200 OK response for the 'GET /pets/{petId}' method failed:\n", err);
  }
})
```

7. 呼叫 `putIntegration` 函數，為 Integration 方法設定與指定 HTTP 端點的 GET /pets，並在 /pets 屬性上提供 8sxa2j 資源識別符 (parentId)。

```
apig.putIntegration({
  restApiId: 'iuo308uaq7',
  resourceId: '8sxa2j',
  httpMethod: 'GET',
  type: 'HTTP',
  integrationHttpMethod: 'GET',
  uri: 'http://perstore-demo-endpoint.execute-api.com/pets'
}, function(err, data){
  if (!err) {
    console.log(data);
  } else {
    console.log("Set up the integration of the 'GET /' method of the API failed:\n", err);
  }
})
```

此函數會傳回類似如下的輸出：

```
{  
    "httpMethod": "GET",  
    "passthroughBehavior": "WHEN_NO_MATCH",  
    "cacheKeyParameters": [],  
    "type": "HTTP",  
    "uri": "http://petstore-demo-endpoint.execute-api.com/petstore/pets",  
    "cacheNamespace": "8sxa2j"  
}
```

若要設定 GET /pets/{petId} 方法與後端 http://perstore-demo-endpoint.execute-api.com/pets/{id} 之 HTTP 端點的整合，請呼叫下列 putIntegration 函數，並在 /pets/{petId} 屬性上提供 API 的 au5df2 資源識別碼 (parentId)。

```
apig.putIntegration({  
    restApiId: 'iuo308uaq7',  
    resourceId: 'au5df2',  
    httpMethod: 'GET',  
    type: 'HTTP',  
    integrationHttpMethod: 'GET',  
    uri: 'http://perstore-demo-endpoint.execute-api.com/pets/{id}',  
    requestParameters: {  
        "integration.request.path.id": "method.request.path.petId"  
    }  
}, function(err, data){  
    if (!err) {  
        console.log(data);  
    } else {  
        console.log("The 'GET /pets/{petId}' method integration setup failed:\n", err);  
    }  
})
```

此函數會傳回類似如下的成功輸出：

```
{  
    "httpMethod": "GET",  
    "passthroughBehavior": "WHEN_NO_MATCH",  
    "cacheKeyParameters": [],  
    "type": "HTTP",  
    "uri": "http://petstore-demo-endpoint.execute-api.com/petstore/pets/{id}",  
    "cacheNamespace": "au5df2",  
    "requestParameters": {  
        "integration.request.path.id": "method.request.path.petId"  
    }  
}
```

8. 呼叫 putIntegrationResponse 函數，設定 GET /pets 方法的 200 OK 整合回應，並在 /pets 屬性上指定 API 的 8sxa2j 資源識別符 (resourceId)。

```
apig.putIntegrationResponse({  
    restApiId: 'iuo308uaq7',  
    resourceId: '8sxa2j',  
    httpMethod: 'GET',  
    statusCode: '200',  
    selectionPattern: ''  
}, function(err, data){  
    if (!err) {  
        console.log(data);  
    } else {  
        console.log("The 'GET /pets' method integration response setup failed:\n", err);  
    }  
})
```

```
})
```

此函數會傳回與下列結果類似的輸出：

```
{  
    "selectionPattern": "",  
    "statusCode": "200"  
}
```

若要設定 GET /pets/{petId} 方法的 200 OK 整合回應，請呼叫 `putIntegrationResponse` 函數，並在 `resourceId` 屬性上指定 API 的 /pets/{petId} 資源識別符 (au5df2)。

```
apig.putIntegrationResponse({  
    restApiId: 'iuo308uaq7',  
    resourceId: 'au5df2',  
    httpMethod: 'GET',  
    statusCode: '200',  
    selectionPattern: ''  
}, function(err, data){  
    if (!err) {  
        console.log(data);  
    } else  
        console.log("The 'GET /pets/{petId}' method integration response setup failed:\n",  
err);  
});
```

- 建議您先測試呼叫 API，再進行部署。若要測試呼叫 GET /pets 方法，請呼叫 `testInvokeMethod`，並在 /pets 屬性上指定 8sxa2j 資源識別符 (resourceId)：

```
apig.testInvokeMethod({  
    restApiId: 'iuo308uaq7',  
    resourceId: '8sxa2j',  
    httpMethod: "GET",  
    pathWithQueryString: '/'  
}, function(err, data){  
    if (!err) {  
        console.log(data)  
    } else {  
        console.log('Test-invoke-method on 'GET /pets' failed:\n', err);  
    }  
})
```

若要測試呼叫 GET /pets/{petId} 方法，請呼叫 `testInvokeMethod`，並在 /pets/{petId} 屬性上指定 au5df2 資源識別符 (resourceId)：

```
apig.testInvokeMethod({  
    restApiId: 'iuo308uaq7',  
    resourceId: 'au5df2',  
    httpMethod: "GET",  
    pathWithQueryString: '/'  
}, function(err, data){  
    if (!err) {  
        console.log(data)  
    } else {  
        console.log('Test-invoke-method on 'GET /pets/{petId}' failed:\n', err);  
    }  
})
```

10. 最後，您可以部署 API 以供客戶進行呼叫。

```
apig.createDeployment({
  restApiId: 'iuo308uaq7',
  stageName: 'test',
  stageDescription: 'test deployment',
  description: 'API deployment'
}, function(err, data){
  if (err) {
    console.log('Deploying API failed:\n', err);
  } else {
    console.log("Deploying API succeeded\n", data);
  }
})
```

## 匯入 OpenAPI 定義以設定邊緣最佳化 API

您可以在 API Gateway 中透過指定適當 API Gateway API 實體的 OpenAPI 定義，再將 OpenAPI 定義匯入 API Gateway 來設定 API。

下列 OpenAPI 定義描述簡單的 API，只公開與後端 PetStore 網站之 HTTP 端點整合的 GET / 方法，並傳回 200 OK 回應。

OpenAPI 2.0

```
{
  "swagger": "2.0",
  "info": {
    "title": "Simple PetStore (OpenAPI)"
  },
  "schemes": [
    "https"
  ],
  "paths": {
    "/pets": {
      "get": {
        "responses": {
          "200": {
            "description": "200 response"
          }
        },
        "x-amazon-apigateway-integration": {
          "responses": {
            "default": {
              "statusCode": "200"
            }
          },
          "uri": "http://petstore-demo-endpoint.execute-api.com/petstore/pets",
          "passthroughBehavior": "when_no_match",
          "httpMethod": "GET",
          "type": "http"
        }
      }
    },
    "/pets/{petId)": {
      "get": {
        "parameters": [
          {
            "name": "petId",
            "in": "path",
            "required": true,
            "type": "string"
          }
        ]
      }
    }
  }
}
```

```
        },
    ],
    "responses": {
        "200": {
            "description": "200 response"
        }
    },
    "x-amazon-apigateway-integration": {
        "responses": {
            "default": {
                "statusCode": "200"
            }
        },
        "requestParameters": {
            "integration.request.path.id": "method.request.path.petId"
        },
        "uri": "http://petstore-demo-endpoint.execute-api.com/petstore/pets/{id}",
        "passthroughBehavior": "when_no_match",
        "httpMethod": "GET",
        "type": "http"
    }
}
}
```

下列程序說明如何使用 API Gateway 主控台來將這些 OpenAPI 定義匯入 API Gateway。

#### 使用 API Gateway 主控台匯入簡單的 OpenAPI 定義

1. 登入 API Gateway 主控台。
2. 選擇 Create API (建立 API)。
3. 選擇 Import from OpenAPI (從 OpenAPI 匯入)。
4. 如果您先前將 OpenAPI 定義儲存在某個檔案中，請選擇 Select OpenAPI File (選取 OpenAPI 檔案)。您也可以複製 OpenAPI 定義並將其貼到匯入文字編輯器中。
5. 選擇 Import (匯入) 完成匯入 OpenAPI 定義。

若要使用 AWS CLI 匯入 OpenAPI 定義，請將 OpenAPI 定義儲存到檔案中，然後執行下列命令 (假設您使用 us-west-2 區域且 OpenAPI 檔案絕對路徑為 file:///path/to/API\_OpenAPI\_template.json)：

```
aws apigateway import-rest-api --body 'file:///path/to/API_OpenAPI_template.json' --region us-west-2
```

## 在 API Gateway 中設定區域 API

如果 API 請求在部署 API 時主要源自相同區域內的 EC2 執行個體或服務，則區域 API 端點通常會降低連線延遲，並建議用於這類情況。

#### Note

如果地理上 API 用戶端分散各處，則使用區域 API 端點與自己的 Amazon CloudFront 分發搭配，以確保 API Gateway 不會將 API 與服務控制的 CloudFront 分發建立關聯，這樣做可能仍有意義。如需此使用案例的詳細資訊，請參閱[如何設定 API Gateway 與自己的 CloudFront 分發搭配？](#)。

若要建立區域 API，請遵循[建立邊緣最佳化的 API \(p. 168\)](#)中的步驟進行，但必須將 REGIONAL 類型明確設定為 API endpointConfiguration 的唯一選項。

下面示範如何使用 API Gateway 主控台、AWS CLI 和適用於 Javascript for Node.js 的 AWS 開發套件來建立區域 API。

## 主題

- [使用 API Gateway 主控台建立區域 API \(p. 182\)](#)
- [使用 AWS CLI 建立區域 API \(p. 182\)](#)
- [使用適用於 JavaScript 的 AWS 開發套件建立區域 API \(p. 182\)](#)
- [測試區域 API \(p. 183\)](#)

## 使用 API Gateway 主控台建立區域 API

### 使用 API Gateway 主控台建立區域 API

1. 登入 API Gateway 主控台，然後選擇 + Create API (+ 建立 API)。
2. 在 Create new API (建立新 API) 下，選擇 New API (新增 API) 選項。
3. 針對 API name (API 名稱)，輸入名稱 (例如，Simple PetStore (Console, Regional))。
4. 針對 Endpoint Type (端點類型) 選擇 Regional。
5. 選擇 Create API (建立 API)。

從這裡，您可以繼續設定 API 方法和其相關聯整合，如[建立邊緣最佳化的 API \(p. 169\)](#) 中所述。

## 使用 AWS CLI 建立區域 API

若要使用 AWS CLI 建立區域 API，請呼叫 `create-rest-api` 命令：

```
aws apigateway create-rest-api \
    --name 'Simple PetStore (AWS CLI, Regional)' \
    --description 'Simple regional PetStore API' \
    --region us-west-2 \
    --endpoint-configuration '{ "types": [ "REGIONAL" ] }'
```

成功回應會傳回與下列類似的承載：

```
{  
    "createdDate": "2017-10-13T18:41:39Z",  
    "description": "Simple regional PetStore API",  
    "endpointConfiguration": {  
        "types": "REGIONAL"  
    },  
    "id": "0qzs2sy7bh",  
    "name": "Simple PetStore (AWS CLI, Regional)"  
}
```

從這裡，您可以遵循「[the section called “使用 AWS CLI 命令設定邊緣最佳化 API” \(p. 169\)](#)」中提供的相同說明來設定此 API 的方法和整合。

## 使用適用於 JavaScript 的 AWS 開發套件建立區域 API

若要使用適用於 JavaScript 的 AWS 開發套件建立區域 API：

```
apig.createRestApi({
    name: "Simple PetStore (node.js SDK, regional)",
    endpointConfiguration: {
        types: ['REGIONAL']
    },
    description: "Demo regional API created using the AWS SDK for node.js",
    version: "0.0.001"
}, function(err, data){
```

```
if (!err) {
  console.log('Create API succeeded:\n', data);
  restApiId = data.id;
} else {
  console.log('Create API failed:\n', err);
}
});
```

成功回應會傳回與下列類似的承載：

```
{
  "createdDate": "2017-10-13T18:41:39Z",
  "description": "Demo regional API created using the AWS SDK for node.js",
  "endpointConfiguration": {
    "types": "REGIONAL"
  },
  "id": "0qzs2sy7bh",
  "name": "Simple PetStore (node.js SDK, regional)"
}
```

完成先前的步驟之後，您可以遵循 [the section called “使用適用於 Node.js 的 AWS 開發套件設定邊緣最佳化 API” \(p. 173\)](#) 中的說明來設定此 API 的方法和整合。

## 測試區域 API

部署之後，區域 API 的預設 URL 主機名稱格式如下：

```
{restapi-id}.execute-api.{region}.amazonaws.com
```

呼叫 API 的基本 URL 如下：

```
https://{restapi-id}.execute-api.{region}.amazonaws.com/{stage}
```

假設您在此範例中設定 GET /pets 和 GET /pets/{petId} 方法，則可以在瀏覽器中輸入下列 URL 來測試 API：

```
https://0qzs2sy7bh.execute-api.us-west-2.amazonaws.com/test/pets
```

和

```
https://0qzs2sy7bh.execute-api.us-west-2.amazonaws.com/test/pets/1
```

或者，您可以使用 cURL 命令：

```
curl -X GET https://0qzs2sy7bh.execute-api.us-west-2.amazonaws.com/test/pets
```

和

```
curl -X GET https://0qzs2sy7bh.execute-api.us-west-2.amazonaws.com/test/pets/2
```

## 在 Amazon API Gateway 中建立私有 API

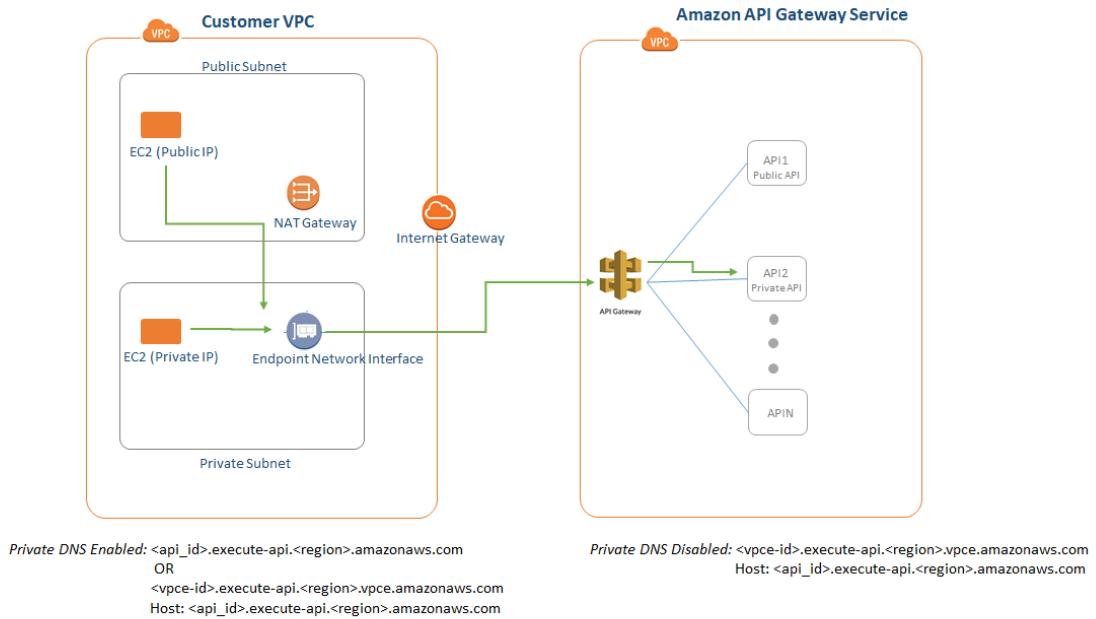
您可以使用 Amazon API Gateway，建立僅能從 Amazon Virtual Private Cloud (VPC) 透過 [界面 VPC 端點](#) 存取的私有 REST API；此端點是您在 VPC 中建立的端點網路界面 (ENI)。您可以使用 [資源政策 \(p. 187\)](#)，

允許或拒絕選定的 VPC 與 VPC 端點 (包括其他 AWS 帳戶) 存取您的 API。每個端點可用來存取多個私有 API。您也可以使用 AWS Direct Connect 在現場部署網路與 Amazon VPC 之間建立連線，並經由該連線來存取您的私有 API。在所有情況下，進出您私有 API 的流量都會使用安全連線，且留在 Amazon 網路中，並與公有網際網路隔離。

您可以透過 API Gateway 界面 VPC 端點來存取 (p. 517) 您的私有 API，如下圖所示。如果您已啟用私有 DNS，可以使用私有或公有 DNS 名稱來存取您的 API。如果您已停用私有 DNS，則只能使用公有 DNS 名稱。

Note

API Gateway 私有 API 僅支援 TLS 1.2。不支援舊版 TLS。



在高階流程中，建立私有 API 的步驟如下所示：

1. 首先，在 VPC 中為 API Gateway 元件服務建立界面 VPC 端點 (p. 185) (也就是 execute-api) 以執行 API。
2. 建立和測試您的私有 API。
  - a. 請依照下列程序擇一建立您的 API：
    - API Gateway 主控台 (p. 186)
    - API Gateway CLI (p. 186)
    - 適用於 JavaScript 的 AWS 開發套件 (p. 186)
  - b. 若要將存取權授與您的 VPC 端點，請建立一個資源政策，並將它連接到您的 API (p. 187)。
  - c. 測試您的 API (p. 517)。

Note

以下程序假設您已擁有完整設置的 VPC。如需詳細資訊，以及開始建立 VPC，請參閱 Amazon VPC User Guide 中的 [Amazon VPC 入門](#)。

主題

- 為 API Gateway execute-api 建立界面 VPC 端點 (p. 185)
- 使用 API Gateway 主控台建立私有 API (p. 186)
- 使用 AWS CLI 建立私有 API (p. 186)
- 使用適用於 JavaScript 的 AWS 開發套件建立私有 API (p. 186)
- 為私有 API 設定資源政策 (p. 187)
- 使用 API Gateway 主控台部署私有 API (p. 187)
- 私有 API 開發考量 (p. 187)

## 為 API Gateway execute-api 建立界面 VPC 端點

API 執行的 API Gateway 元件服務稱為 execute-api。若要存取您已部署的私有 API，您需要在 VPC 中建立界面 VPC 端點。

一旦建立 VPC 端點，您就可以使用它來存取多個私有 API。

### 為 API Gateway **execute-api** 建立界面 VPC 端點

1. 前往 <https://console.aws.amazon.com/vpc/> 以登入 Amazon VPC 主控台。
2. 在導覽窗格中，選擇 Endpoints (端點)，Create Endpoint (建立端點)。
3. 對於 Service category (服務類別)，請確定您已選擇 AWS services (AWS 服務)。
4. 對於 Service Name (服務名稱)，請選擇 API Gateway 服務端點，包括欲連線的區域。此格式將為 com.amazonaws.*region*.execute-api，例如 com.amazonaws.us-east-1.execute-api。

對於 Type (類型)，請確定其顯示 Interface (界面)。

5. 請填妥下列資訊：

- 對於 VPC，請選擇您欲建立端點的 VPC。
- 對於 Subnets (子網路)，請選擇欲建立端點的子網路 (可用區域)。

#### Note

並非所有可用區域皆可支援所有 AWS 服務。

- 對於 Enable Private DNS Name (啟用私有 DNS 名稱)，您可以選擇性選取核取方塊，以對該界面端點啟用私有 DNS。

如果您選擇啟用私有 DNS，您將可透過私有或公有 DNS 存取您的 API。(此設定不會影響可以存取您 API 的人，只會影響他們使用哪個 DNS 地址。) 不過，您無法使用已啟用私有 DNS 的 API GatewayVPC 端點，從 VPC 存取公有 API。請注意，如果您使用經最佳化的自訂網域名稱來存取公有 API，這些 DNS 設定不會影響呼叫 VPC 公有 API 的能力。使用經最佳化的自訂網域名稱來存取公有 API(同時使用私有 DNS 存取您的私有 API)，可以從建立端點和用已啟用私有 DNS 的 VPC，存取公有和私有 API。

#### Note

建議啟用私有 DNS。如果您選擇不要啟用私有 DNS，您將僅能透過公有 DNS 存取您的 API。

若要使用私有 DNS 選項，您 VPC 的 enableDnsSupport 和 enableDnsHostnames 屬性必須設為 true。如需詳細資訊，請參閱 Amazon VPC User Guide 中的 [您 VPC 中的 DNS 支援與為您的 VPC 更新 DNS 支援](#)。

- 對於 Security group (安全群組)，請選擇要與 VPC 端點網路界面建立關聯的安全群組。

您選擇的安全群組，必須設為允許從 VPC 的 IP 範圍或另一個安全群組的 TCP 連接埠 443 傳入 HTTPS 流量。

6. 選擇 Create endpoint (建立端點)。

## 使用 API Gateway 主控台建立私有 API

### 使用 API Gateway 主控台建立私有 API

1. 登入 API Gateway 主控台，然後選擇 + Create API (+ 建立 API)。
2. 在 Create new API (建立新 API) 下，選擇 New API (新增 API) 選項。
3. 針對 API name (API 名稱)，輸入名稱 (例如，Simple PetStore (Console, Private))。
4. 對於 Endpoint Type (端點類型)，選擇 Private。
5. 選擇 Create API (建立 API)。

從這裡，您可以依[??? \(p. 53\)](#)的步驟 1-6 所述，設定 API 方法及其相關的整合。

#### Note

在您的 API 擁有可對您的 VPC 或 VPC 端點 ([p. 185](#)) 授予存取權限的資源政策前，所有的 API 呼叫都將會失敗。測試和部署您的 API 之前，您將需要建立資源政策，並將它連接到 API，如[??? \(p. 187\)](#)中所述。

## 使用 AWS CLI 建立私有 API

若要使用 AWS CLI 建立私有 API，請呼叫 `create-rest-api` 命令：

```
aws apigateway create-rest-api \
    --name 'Simple PetStore (AWS CLI, Private)' \
    --description 'Simple private PetStore API' \
    --region us-west-2 \
    --endpoint-configuration '{ "types": [ "PRIVATE" ] }'
```

成功的呼叫會傳回類似如下的輸出：

```
{  
    "createdDate": "2017-10-13T18:41:39Z",  
    "description": "Simple private PetStore API",  
    "endpointConfiguration": {  
        "types": "PRIVATE"  
    },  
    "id": "0qzs2sy7bh",  
    "name": "Simple PetStore (AWS CLI, Private)"  
}
```

從這裡，您可以遵循「[the section called “使用 AWS CLI 命令設定邊緣最佳化 API” \(p. 169\)](#)」中提供的相同說明來設定此 API 的方法和整合。

準備好測試您的 API 前，務必建立資源政策並將它連接到 API，如[??? \(p. 187\)](#)中所述。

## 使用適用於 JavaScript 的 AWS 開發套件建立私有 API

若要使用適用於 JavaScript 的 AWS 開發套件建立私有 API：

```
apig.createRestApi({
    name: "Simple PetStore (node.js SDK, private)",
    endpointConfiguration: {
        types: ['PRIVATE']
    },
    description: "Demo private API created using the AWS SDK for node.js",
```

```
version: "0.0.0.001"
}, function(err, data){
  if (!err) {
    console.log('Create API succeeded:\n', data);
    restApiId = data.id;
  } else {
    console.log('Create API failed:\n', err);
  }
});
```

成功的呼叫會傳回類似如下的輸出：

```
{
  "createdDate": "2017-10-13T18:41:39Z",
  "description": "Demo private API created using the AWS SDK for node.js",
  "endpointConfiguration": {
    "types": "PRIVATE"
  },
  "id": "0qzs2sy7bh",
  "name": "Simple PetStore (node.js SDK, private)"
}
```

完成先前的步驟之後，您可以遵循「[the section called “使用適用於 Node.js 的 AWS 開發套件設定邊緣最佳化 API” \(p. 173\)](#)」中的說明來設定此 API 的方法和整合。

準備好測試您的 API 前，務必建立資源政策並將它連接到 API，如[??? \(p. 187\)](#)中所述。

## 為私有 API 設定資源政策

在您可以存取私有 API 之前，您需要建立資源政策並將它連接到 API；這將讓您或其他（您已明確授與存取權的）AWS 帳戶的 VPC 或 VPC 端點可以存取此 API。

若要執行此操作，請遵循 [the section called “建立一個 API Gateway 資源政策並將其連接到 API” \(p. 331\)](#) 中的說明。在步驟 4 中，選擇 Source VPC Whitelist（來源 VPC 允許名單）範例。用您的 VPC 端點 ID 替換 `{vpceID}`（包括大括號），然後選擇 Save（儲存），以儲存您的資源政策。

## 使用 API Gateway 主控台部署私有 API

若要部署您的私有 API，請在 API Gateway 主控台執行下列動作：

1. 在左側導覽窗格中選取 API，然後從 Actions（動作）下拉式選單中選擇 Deploy API（部署 API）。
2. 在 Deploy API（部署 API）對話方塊中，選擇一個階段（若是 API 的第一個部署，則為 [New Stage]），然後在 Stage name（階段名稱）輸入欄位中輸入名稱（例如 "test"、"prod"、"dev" 等）；選擇性地在 Stage description（階段說明）及/或 Deployment description（部署說明）中提供說明，然後選擇 Deploy（部署）。

## 私有 API 開發考量

- 您可以將現有的公有 API（區域或邊緣最佳化）轉換為私有 API，也可以將私有 API 轉換為區域 API。您不能將私有 API 轉換為邊緣最佳化 API。如需詳細資訊，請參閱[??? \(p. 454\)](#)。
- 要授與 VPC 和 VPC 端點對您私有 API 的存取權限，您將需要建立資源政策並將它連接到新建立（或轉換）的 API。到您執行此操作前，所有的 API 呼叫都將會失敗。如需詳細資訊，請參閱[??? \(p. 187\)](#)。
- 私有 API 並不支援[自訂網域名稱 \(p. 590\)](#)。
- 您可使用單一 VPC 端點來存取多個私有 API。
- 私有 API 的 VPC 端點會受到與其他界面 VPC 端點相同的限制。如需詳細資訊，請參閱 Amazon VPC User Guide 中的[界面端點屬性和限制](#)。

## 在 API Gateway 中設定 REST API 方法

在 API Gateway 中，API 方法包含[方法請求與方法回應](#)。您可以設定 API 方法，來定義用戶端應該或必須執行才能提交請求以存取後端服務的操作，以及定義用戶端接著會收到的回應。輸入時，您可以選擇方法請求參數或適用的承載，讓用戶端在執行階段提供必要或選用的資料。輸出時，您會決定方法回應狀態碼、標頭與適用的本文，以作為後端回應資料映射的目標，再將這些目標傳回用戶端。為了協助用戶端開發人員了解您的 API 行為以及輸入與輸出格式，您可以[記錄您的 API \(p. 411\)](#)並針對[無效的請求 \(p. 307\)](#)提供適當的錯誤訊息 (p. 237)。

API 方法請求是 HTTP 請求。若要設定方法請求，請設定 HTTP 方法 (或動詞)、API 資源的路徑、標頭、適用的查詢字串參數。當 HTTP 方法是 POST、PUT 或 PATCH 時，您也會設定承載。例如，若要使用 PetStore 範例 API (p. 40) 擷取寵物，請定義 GET /pets/{petId} 的 API 方法請求，其中 {petId} 是可在執行階段接受一個數值的路徑參數。

```
GET /pets/1
Host: apigateway.us-east-1.amazonaws.com
...
```

如果用戶端指定不正確的路徑，例如 /pet/1 或 /pets/one 而不是 /pets/1，則會擲回例外狀況。

API 方法回應是指定狀態碼的 HTTP 回應。對於非代理整合，您必須設定方法回應來指定映射的必要或選用目標。這會將整合回應標頭或本文轉換成相關聯的方法回應標頭或本文。映射可以像[身分轉換](#)一樣簡單，該轉換會依原狀透過整合來傳遞標頭或內文。例如，下列 200 方法回應顯示依現狀傳遞成功整合回應的範例。

```
200 OK
Content-Type: application/json
...
{
    "id": "1",
    "type": "dog",
    "price": "$249.99"
}
```

基本上，您可以定義對應到後端之特定回應的方法回應。一般而言，這涉及任何 2XX、4XX 與 5XX 回應。不過，這可能不可行，因為您通常不太可能會事先知道後端可能傳回的所有回應。實際操作時，您可以指定一個方法回應作為預設值，處理來自後端之不明或未映射的回應。您最好指定 500 回應做為預設值。在任何情況下，您都必須為非代理整合設定至少一個方法回應。否則，API Gateway 會將 500 錯誤回應傳回用戶端，即使請求在後端成功也一樣。

若要讓您的 API 支援強型別開發套件 (例如 Java 開發套件)，您應該定義方法請求輸入的資料模型，並定義方法回應輸出的資料模型。

### 主題

- [在 API Gateway 中設定方法請求 \(p. 188\)](#)
- [在 API Gateway 中設定方法回應 \(p. 194\)](#)
- [使用 API Gateway 主控台設定方法 \(p. 196\)](#)

## 在 API Gateway 中設定方法請求

設定方法請求需要在建立 RestApi 資源之後執行下列任務：

1. 建立新的 API 或選擇現有的 API 資源實體。
2. 在新的或選擇的 API Resource 上，建立 API 方法資源 (也就是特定 HTTP 動詞)。這項作業可進一步分為下列子任務：
  - 將 HTTP 方法新增至方法請求

- 設定請求參數
- 定義請求本文的模型
- 制定授權配置
- 啟用請求驗證

您可以使用下列方法來執行這些任務：

- [API Gateway 主控台 \(p. 196\)](#)
- AWS CLI 命令 ([create-resource](#) 與 [put-method](#))
- AWS 開發套件函數 (例如，在 Node.js 中為 [createResource](#) 與 [putMethod](#))
- API Gateway REST API ([resource:create](#) 與 [method:put](#))。

如需使用這些工具的範例，請參閱「[在 API Gateway 中初始化 REST API 設定 \(p. 168\)](#)」。

#### 主題

- [設定 API 資源 \(p. 189\)](#)
- [設定 HTTP 方法 \(p. 191\)](#)
- [設定方法請求參數 \(p. 192\)](#)
- [設定方法請求模型 \(p. 192\)](#)
- [設定方法請求授權 \(p. 193\)](#)
- [設定方法請求驗證 \(p. 194\)](#)

## 設定 API 資源

在 API Gateway API 中，您可以將可定址的資源公開為 API 資源實體的樹狀目錄，其根資源 (/) 在階層的最上層。此根資源與 API 的基底 URL 相關，其中包含 API 端點與階段名稱。在 API Gateway 主控台中，此基底 URI 稱為 Invoke URI (呼叫 URI)，並會在部署 API 之後顯示在 API 的階段編輯器中。

API 端點可以是預設主機名稱或自訂網域名稱。預設主機名稱的格式如下：

```
{api-id}.execute-api.{region}.amazonaws.com
```

在此格式中，`{api-id}` 表示 API Gateway 所產生的 API 識別符。`{region}` 變數表示您在建立 API 時所選擇的 AWS 區域 (例如 us-east-1)。自訂網域名稱是有效網際網路網域下的任何使用者易記名稱。例如，如果您已註冊網際網路網域 example.com，任何 \*.example.com 都是有效的自訂網域名稱。如需詳細資訊，請參閱[建立自訂網域名稱 \(p. 590\)](#)。

以 [PetStore 範例 API \(p. 40\)](#) 而言，根資源 (/) 會公開寵物店。/pets 資源表示寵物店中可用的寵物集合。/pets/{petId} 會公開指定識別符 (petId) 的個別寵物。`{petId}` 的路徑參數是請求參數的一部分。

若要設定 API 資源，您可以選擇現有資源作為其父系，然後在此父資源下建立子資源。您一開始會以根資源作為父系，然後將資源新增至此父系，再將另一項資源新增至此子資源作為新的父系，依此類推直到新增至其父識別符。然後，您可以將具名資源新增至父系。

透過 AWS CLI，您可以呼叫 `get-resources` 命令來了解哪些 API 資源可供使用：

```
aws apigateway get-resources --rest-api-id <apiId> \  
--region <region>
```

結果會列出目前可用的 API 資源。在我們的 PetStore 範例 API 中，此清單會顯示如下：

```
{
```

```
"items": [
  {
    "path": "/pets",
    "resourceMethods": {
      "GET": {}
    },
    "id": "6sxz2j",
    "pathPart": "pets",
    "parentId": "svzr2028x8"
  },
  {
    "path": "/pets/{petId}",
    "resourceMethods": {
      "GET": {}
    },
    "id": "rjkmth",
    "pathPart": "{petId}",
    "parentId": "6sxz2j"
  },
  {
    "path": "/",
    "id": "svzr2028x8"
  }
]
```

每個項目會列出資源的識別符 (id)、其直屬父系 (parentId) (根資源除外)，以及資源名稱 (pathPart)。根資源是特殊的，因為它沒有任何父系。選擇某個資源作為父系之後，請呼叫下列命令新增子資源。

```
aws apigateway create-resource --rest-api-id <apiId> \
--region <region> \
--parent-id <parentId> \
--path-part <resourceName>
```

例如，若要在 PetStore 網站上新增寵物食品進行促銷，請將 food 設定為 path-part 並將 food 設定為 parent-id，以將 svzr2028x8 資源新增至根目錄 (/)。結果看起來如下：

```
{
  "path": "/food",
  "pathPart": "food",
  "id": "xdsvhp",
  "parentId": "svzr2028x8"
}
```

## 使用代理資源來簡化 API 設定

隨著業務成長，PetStore 擁有者可能會決定新增食品、玩具與其他寵物相關項目來進行促銷。為了支援此目的，您可以在根資源下新增 /food、/toys 與其他資源。在每個銷售類別下，您可能還想要新增更多資源，例如 /food/{type}/{item}、/toys/{type}/{item} 等。這可能會變得很冗長。如果您決定將中介層 {subtype} 新增至資源路徑，以將路徑階層變更為 /food/{type}/{subtype}/{item}、/toys/{type}/{subtype}/{item} 等，變更會中斷現有的 API 設定。為了避免這種情況，您可以使用 API Gateway 代理資源 (p. 202) 一次完全公開一組 API 資源。

API Gateway 會將代理資源定義為要在提交請求時指定之資源的預留位置。代理資源是由 {proxy+} 的特殊路徑參數來表示，通常稱為 Greedy 路徑參數。+ 符號指出要附加的子資源。/parent/{proxy+} 預留位置代表符合 /parent/\* 之路徑模式的任何資源。Greedy 路徑參數名稱 proxy 可以取代為其他字串，就如同處理一般路徑參數名稱一樣。

您可以使用 AWS CLI，呼叫下列命令在根目錄下設定代理資源 (/{{proxy+}})：

```
aws apigateway create-resource --rest-api-id <apiId> \
```

```
--region <region> \
--parent-id <rootResourceId> \
--path-part {proxy+}
```

結果類似如下：

```
{
  "path": "/{proxy+}",
  "pathPart": "{proxy+}",
  "id": "234jdr",
  "parentId": "svzr2028x8"
}
```

在 PetStore API 範例中，您可以使用 `/{proxy+}` 來表示 `/pets` 與 `/pets/{petId}`。此代理資源也可以參考其他任何（現有或是要新增的）資源，例如 `/food/{type}/{item}`、`/toys/{type}/{item}` 等，或是 `/food/{type}/{subType}/{item}`、`/toys/{type}/{subType}/{item}` 等。後端開發人員會決定資源階層，而用戶端開發人員則負責了解此階層。API Gateway 只會將用戶端提交的任何項目傳遞到後端。

一個 API 可以有多個代理資源。例如，API 中允許下列代理資源。

```
/{proxy+}
/parent/{proxy+}
/parent/{child}/{proxy+}
```

當代理資源有非代理同層級資源時，則會從代理資源的顯示方式中排除這些同層級資源。在上述範例中，`/{proxy+}` 是指根資源下除了 `/parent[*]` 資源以外的任何資源。換言之，對特定資源提出的方法請求，會優先於對資源階層中同層級之一般資源提出的方法請求。

代理資源不能有任何子資源。`{proxy+}` 後面的任何 API 資源是多餘且模棱兩可的。API 中不允許下列代理資源。

```
/{proxy+}/child
/parent/{proxy+}/{child}
/parent/{child}/{proxy+}/{grandchild+}
```

## 設定 HTTP 方法

API 方法請求是由 API Gateway [Method](#) 資源所封裝。若要設定方法請求，您必須先具現化 Method 資源、設定至少一個 HTTP 方法，並在該方法上設定至少一種授權類型。

與代理資源密切相關的是，API Gateway 支援 HTTP 方法 ANY。此 ANY 方法代表要在執行階段提供的任何 HTTP 方法。它可讓您針對 DELETE、GET、HEAD、OPTIONS、PATCH、POST 與 PUT 之所有支援的 HTTP 方法，使用單一 API 方法設定。

您也可以在非代理資源上設定 ANY 方法。透過將 ANY 方法與代理資源合併，您就可以對 API 的任何資源，取得所有支援之 HTTP 方法的單一 API 方法設定。此外，後端可繼續發展而不需要中斷現有的 API 設定。

設定 API 方法之前，請考慮誰可以呼叫此方法。請根據您的方案設定授權類型。如需開放式存取，請將其設定為 NONE。若要使用 IAM 許可，請將授權類型設定為 AWS\_IAM。若要使用 Lambda 授權方函數，請將此屬性設定為 CUSTOM。若要使用 Amazon Cognito 使用者集區，請將授權類型設定為 COGNITO\_USER\_POOLS。

下列 AWS CLI 命令示範如何對指定的資源 (6sxz2j) 建立 ANY 動詞的方法請求，並使用 IAM 許可來控制其存取。

```
aws apigateway put-method --rest-api-id vaz7da96z6 \
  --resource-id 6sxz2j \
```

```
--http-method ANY \
--authorization-type AWS_IAM \
--region us-west-2
```

若要使用不同的授權類型來建立 API 方法請求，請參閱「[the section called “設定方法請求授權” \(p. 193\)](#)」。

## 設定方法請求參數

方法請求參數可讓用戶端提供完成方法請求所需的輸入資料或執行內容。方法參數可以是路徑參數、標頭或查詢字串參數。設定方法請求時，您必須宣告必要的請求參數以提供給用戶端。對於非代理整合，您可以將這些請求參數轉換成與後端需求相容的格式。

例如，對於 GET /pets/{petId} 方法請求，{petId} 路徑變數是必要的請求參數。您可以在呼叫 AWS CLI 的 put-method 命令時宣告此路徑參數，如下所示：

```
aws apigateway put-method --rest-api-id vaz7da96z6 \
    --resource-id rjkmth \
    --http-method GET \
    --authorization-type "NONE" \
    --region us-west-2 \
    --request-parameters method.request.path.petId=true
```

如果不需要參數，您可以在 false 中將它設定為 request-parameters。例如，如果 GET /pets 方法使用一個選用的查詢字串參數 type 與一個選用的標頭參數 breed，您可以使用下列 CLI 命令來宣告這些參數，並假設 /pets 資源 id 為 6sxz2j：

```
aws apigateway put-method --rest-api-id vaz7da96z6 \
    --resource-id 6sxz2j \
    --http-method GET \
    --authorization-type "NONE" \
    --region us-west-2 \
    --request-parameters
method.request.querystring.type=false,method.request.header.breed=false
```

除了此縮寫格式之外，您還可以使用 JSON 字串來設定 request-parameters 值：

```
'{"method.request.querystring.type":false,"method.request.header.breed":false}'
```

有了這項設定，用戶端就可依類型查詢寵物：

```
GET /pets?type=dog
```

此外，用戶端可以查詢貴賓狗品種的狗，如下所示：

```
GET /pets?type=dog
breed:poodle
```

如需如何將方法請求參數映射到整合請求參數的資訊，請參閱「[the section called “設定 REST API 整合” \(p. 199\)](#)」。

## 設定方法請求模型

若要讓 API 方法可接受承載中的輸入資料，您可以使用模型。模型是以 [JSON 結構描述草稿第 4 版](#)來表示，並描述請求本文的資料結構。透過模型，用戶端可以判斷如何建構方法請求承載作為輸入。更重要的是，API Gateway 可使用模型來驗證請求 (p. 307)、產生開發套件 (p. 481)，並初始化對應範本以在 API Gateway 主控台中設定整合。如需如何建立模型的資訊，請參閱模型與映射範本 (p. 244)。

視內容類型而定，一個方法承載可能會有不同的格式。模型會針對已套用承載的媒體類型來編製索引。若要設定方法請求模型，請在呼叫 AWS CLI `put-method` 命令時，將 "`<media-type>": "<model-name>"`" 格式的鍵值對新增至 `requestModels` 對應。

例如，若要在 PetStore 範例 API 之 `POST /pets` 方法請求的 JSON 承載上設定模型，您可以呼叫下列 AWS CLI 命令：

```
aws apigateway put-method \
  --rest-api-id vaz7da96z6 \
  --resource-id 6sxz2j \
  --http-method POST \
  --authorization-type "NONE" \
  --region us-west-2 \
  --request-models '{"application/json":"petModel"}'
```

在此範例中，`petModel` 是描述寵物之 `Model` 資源的 `name` 屬性值。實際結構描述定義會以 `Model` 資源之 `schema` 屬性的 JSON 字串值表示。

在 API 的 Java 開發套件或其他強型別開發套件中，輸入資料會轉換成衍生自結構描述定義的 `petModel` 類別。透過請求模型，所產生之開發套件中的輸入資料會轉換成衍生自預設 `Empty` 模型的 `Empty` 類別。在本例中，用戶端無法具現化正確的資料類別以提供必要的輸入。

## 設定方法請求授權

若要控制誰可以呼叫 API 方法，您可以在方法上設定 [授權類型](#)。您可以使用此類型制定其中一個支援的授權方，包括 IAM 角色與政策 (AWS\_IAM)、Amazon Cognito 使用者集區 (COGNITO\_USER\_POOLS) 或 Lambda 授權方 (CUSTOM)。

若要使用 IAM 許可來授權存取 API 方法，請將 `authorization-type` 輸入屬性設定為 AWS\_IAM。設定此選項時，API Gateway 會根據發起人之 IAM 使用者的存取金鑰識別符與秘密金鑰，來驗證請求上的發起人簽章。如果經驗證的使用者具備呼叫方法的許可，則會接受請求。否則，請求會遭到拒絕，且發起人會收到未經授權的錯誤回應。除非已授予發起人呼叫 API 方法的許可，或是如果發起人可擔任已獲得許可的角色，否則此方法的呼叫不會成功。如果發起人已將下列 IAM 政策連接到其 IAM 使用者，發起人便具備許可，能夠呼叫此方法與相同 AWS 帳戶之任何人所建立的任何其他 API 方法：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "execute-api:Invoke"
      ],
      "Resource": "arn:aws:execute-api:*.*.*"
    }
  ]
}
```

如需更多詳細資訊，請參閱 [the section called “使用 IAM 許可” \(p. 333\)](#)。

目前，這種政策只能授予 API 擁有者帳戶的 IAM 使用者。如果來自不同 AWS 的使用者可擔任 API 擁有者帳戶的角色，而且擔任的角色具備 `execute-api:Invoke` 動作的適當許可，則可以呼叫 API 方法。如需跨帳戶許可的資訊，請參閱 [使用 IAM 角色](#)。

您可以使用 AWS CLI、AWS 開發套件或 REST API 用戶端 (例如 Postman)，這會實作 [Signature 第 4 版簽署](#)。

若要使用 Lambda 授權方來授權存取 API 方法，請將 `authorization-type` 輸入屬性設定為 CUSTOM，並將 `authorizer-id` 輸入屬性設定為已存在之 Lambda 授權方的 `id` 屬性值。參考的 Lambda 授權方可以是 TOKEN 或 REQUEST 類型。如需建立 Lambda 授權方的資訊，請參閱[the section called “使用 Lambda 授權方” \(p. 348\)](#)。

若要使用 Amazon Cognito 使用者集區來授權存取 API 方法，請將 `authorization-type` 輸入屬性設定為 `COGNITO_USER_POOLS`，並將 `authorizer-id` 輸入屬性設定為已建立之 `COGNITO_USER_POOLS` 授權方的 `id` 屬性值。如需建立 Amazon Cognito 使用者集區授權方的資訊，請參閱[the section called “針對 REST API 使用 Cognito 使用者集區做為授權方” \(p. 363\)](#)。

## 設定方法請求驗證

您可以在設定 API 方法請求時啟用請求驗證。您必須先建立[請求驗證程式](#)：

```
aws apigateway create-request-validator \
--rest-api-id 7zw9uyk9kl \
--name bodyOnlyValidator \
--validate-request-body \
--no-validate-request-parameters
```

此 CLI 命令會建立僅限本文的請求驗證程式。範例輸出如下：

```
{
  "validateRequestParameters": true,
  "validateRequestBody": true,
  "id": "jgpwy6",
  "name": "bodyOnlyValidator"
}
```

透過此請求驗證程式，您可以在設定方法請求時啟用請求驗證：

```
aws apigateway put-method \
--rest-api-id 7zw9uyk9kl
--region us-west-2
--resource-id xdsvhp
--http-method PUT
--authorization-type "NONE"
--request-parameters '{"method.request.querystring.type": false,
"method.request.querystring.page":false}'
--request-models '{"application/json":"petModel"}'
--request-validator-id jgpwy6
```

您必須將請求參數宣告為必要，才能將它包含在請求驗證中。如果在請求驗證中使用頁面的查詢字串參數，上述範例的 `request-parameters` 映射必須指定為 `'{"method.request.querystring.type": false, "method.request.querystring.page":true}'`。

## 在 API Gateway 中設定方法回應

API 方法回應可封裝用戶端將會接收的 API 方法請求輸出。該輸出資料包含 HTTP 狀態碼、一些標頭，並可能包含本文。

透過非代理整合，指定的回應參數與本文可從相關聯的整合回應資料映射，或根據映射指派特定靜態值。這些映射是在整合回應中指定。此映射可以是依現狀傳遞整合回應的相同轉換。

透過代理整合，API Gateway 會自動將後端回應傳遞至方法回應。您不需要設定 API 方法回應。不過，若使用 Lambda 代理整合，Lambda 函數必須傳回[此輸出格式 \(p. 216\)](#)的結果，API Gateway 才能成功將整合回應對應至方法回應。

就程式設計而言，此方法回應設定相當於建立 API Gateway 的 `MethodResponse` 資源，然後設定 `statusCode`、`responseParameters` 與 `responseModels` 的屬性。

設定 API 方法的狀態碼時，您應該選擇一個預設值，來處理非預期狀態碼的任何整合回應。您可以設定 500 作為預設值，因為此值相當於將未映射的回應轉換為伺服器端錯誤。為了進行說明，API Gateway 主控台設定 200 回應做為預設值。但您可以將它重設為 500 回應。

若要設定方法回應，您必須已建立方法請求。

## 設定方法回應狀態碼

方法回應狀態碼可定義回應類型。例如，回應 200、400 與 500 分別表示成功、用戶端錯誤與伺服器端錯誤回應。

若要設定方法回應狀態碼，請將 `statusCode` 屬性設定為 HTTP 狀態碼。下列 AWS CLI 命令會建立方法回應 200。

```
aws apigateway put-method-response \
--region us-west-2 \
--rest-api-id vaz7da96z6 \
--resource-id 6sxz2j \
--http-method GET \
--status-code 200
```

## 設定方法回應參數

方法回應參數可定義用戶端所收到以回應相關聯方法請求的標頭。回應參數也可根據 API 方法的整合回應中所指定的映射，來指定 API Gateway 映射整合回應參數的目標。

若要設定方法回應參數，請新增至 "`{parameter-name}`": "`{boolean}`" 格式之 `MethodResponse` 鏈值對的 `responseParameters` 映射。下列 CLI 命令示範如何將 `my-header` 標頭、`petId` 路徑變數與 `query` 查詢參數設定為映射目標：

```
aws apigateway put-method-response \
--region us-west-2 \
--rest-api-id vaz7da96z6 \
--resource-id 6sxz2j \
--http-method GET \
--status-code 200 \
--response-parameters method.request.header.my-
header=false,method.request.path.petId=true,method.request.querystring.query=false
```

## 設定方法回應模型

方法回應模型可定義方法回應本文的格式。設定回應模型之前，您必須先在 API Gateway 中建立模型。若要這樣做，您可以呼叫 `create-model` 命令。下列範例示範如何建立 `PetStorePet` 模型，以描述 `GET /pets/{petId}` 方法請求的回應本文。

```
aws apigateway create-model \
--region us-west-2 \
--rest-api-id vaz7da96z6 \
--content-type application/json \
--name PetStorePet \
--schema '{ \
    "$$schema": "http://json-schema.org/draft-04/schema#", \
    "title": "PetStorePet", \
    "type": "object", \
    "properties": { \
        "id": { "type": "number" }, \
        "type": { "type": "string" }, \
        "price": { "type": "number" } \
    } \
}'
```

結果會建立為一個 API Gateway `Model` 資源。

若要設定方法回應模型來定義承載格式，請將 "application/json":"PetStorePet" 金鑰/值對新增至 MethodResponse 資源的 `requestModels` 映射。put-method-response 的下列 AWS CLI 命令示範如何完成此作業：

```
aws apigateway put-method-response \
    --region us-west-2 \
    --rest-api-id vaz7da96z6 \
    --resource-id 6szz2j \
    --http-method GET \
    --status-code 200 \
    --request-parameters method.request.header.my-
header=false,method.request.path.petId=true,method.request.querystring.query=false
    --request-models '{"application/json":"PetStorePet"}'
```

當您為 API 產生強型別開發套件時，需要設定方法回應模型。它可確保輸出在 Java 或 Objective-C 中轉換成適當的類別。在其他情況下，設定模型為選擇性。

## 使用 API Gateway 主控台設定方法

設定 API 方法之前，請驗證下列項目：

- 您必須在 API Gateway 中有可用的方法。請遵循[教學：建置具有 HTTP 非代理整合的 API \(p. 52\)](#)中的說明進行。
- 如果您想要讓方法與 Lambda 函數通訊，您必須已在 IAM 中建立 Lambda 叫用角色與 Lambda 執行角色。您也必須已建立您的方法在 AWS Lambda 中用來通訊的 Lambda 函數。若要建立角色與函數，請使用[建置具有 Lambda 整合的 API Gateway API \(p. 24\) 之為 Lambda 非代理整合建立 Lambda 函數 \(p. 31\)](#)中的說明。
- 如果您想要讓方法與 HTTP 或 HTTP 代理整合通訊，您必須已建立方法將用來通訊的 HTTP 端點 URL 並具備其存取權。
- 確認 API Gateway 是否支援 HTTP 與 HTTP 代理端點的憑證。如需詳細資訊，請參閱「[API Gateway 為 HTTP 與 HTTP 代理整合支援的憑證授權機構 \(p. 379\)](#)」。

### 主題

- 在 API Gateway 主控台中設定 API Gateway 方法請求 (p. 196)
- 使用 API Gateway 主控台設定 API Gateway 方法回應 (p. 198)

## 在 API Gateway 主控台中設定 API Gateway 方法請求

若要使用 API Gateway 主控台指定 API 的方法請求/回應，以及設定方法如何授權請求，請遵循下列說明進行。

### Note

這些說明假設您已完成[使用 API Gateway 主控台設定 API 整合請求 \(p. 203\)](#)中的步驟。這些說明最適合用來補充「[建置具有 Lambda 整合的 API Gateway API \(p. 24\)](#)」中提供的討論。

1. 選取 Resources (資源) 窗格中的方法時，從 Method Execution (方法執行) 窗格選擇 Method Request (方法請求)。
2. 在 Settings (設定) 下，選擇鉛筆圖示來開啟 Authorization (授權) 下拉式選單，然後選擇其中一個可用的授權方。
  - a. 若要對任何使用者授予方法的開放式存取，請選擇 NONE。如果尚未變更預設設定，則可以略過此步驟。
  - b. 若要使用 IAM 許可來控制方法的用戶端存取，請選擇 AWS\_IAM。使用此選項，只有連接正確 IAM 政策之 IAM 角色的使用者可以呼叫此方法。

若要建立 IAM 角色，請使用類似如下的格式來指定存取政策：

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "execute-api:Invoke"  
            ],  
            "Resource": [  
                "resource-statement"  
            ]  
        }  
    ]  
}
```

在此存取政策中，*resource-statement* 是 Authorization Settings (授權設定) 區段中的 ARN 欄位值。如需設定 IAM 許可的詳細資訊，請參閱[使用 IAM 許可控制 API 的存取 \(p. 333\)](#)。

若要建立 IAM 角色，您可以調整建置具有 Lambda 整合的 API Gateway API (p. 24)之建立 Lambda 函數 (p. 31)一節中「建立 Lambda 呼叫角色及其政策」和「建立 Lambda 執行角色及其政策」的指示。

若要儲存您的選擇，請選擇 Update (更新)。否則請選擇 Cancel (取消)。

- c. 若要使用 Lambda 授權方，請在 Token authorizer (字符串授權方) 下選擇一個授權方。您必須已建立 Lambda 授權方，才會在下拉式選單中顯示此選項。如需如何建立 Lambda 授權方的資訊，請參閱[使用 API Gateway Lambda 授權方 \(p. 348\)](#)。
- d. 若要使用 Amazon Cognito 使用者集區，請在 Cognito user pool authorizers (Cognito 使用者集區授權方) 下，選擇可用的使用者集區。您必須已在 Amazon Cognito 中建立使用者集區，並在 API Gateway 中建立 Amazon Cognito 使用者集區授權方，才會在下拉式選單中顯示此選項。如需如何建立 Amazon Cognito 使用者集區授權方的資訊，請參閱[使用 Amazon Cognito User Pools 做為授權方來控制 REST API 的存取 \(p. 363\)](#)。
3. 若要啟用或停用請求驗證，請從 Request Validator (請求驗證程式) 下拉式選單中選擇鉛筆圖示，然後選擇其中一個列出的選項。如需每個選項的詳細資訊，請參閱「[在 API Gateway 中啟用請求驗證 \(p. 307\)](#)」。
4. 如需 API 金鑰，請選擇鉛筆圖示來開啟 API Key Required (需要 API 金鑰) 下拉式選單，然後根據您的 API 需求來選擇 true 或 false。啟用時，可使用[用量方案 \(p. 397\)](#)中的 API 金鑰來調節用戶端流量。
5. 若要將查詢字串參數新增至方法，請執行下列動作：
  - a. 選擇 URL Query String Parameters (URL 查詢字串參數) 旁的箭頭，然後選擇 Add query string (新增查詢字串)。
  - b. 針對 Name (名稱)，輸入查詢字串參數的名稱。
  - c. 選擇核取記號圖示以儲存新的查詢字串參數名稱。
  - d. 若要使用新建立的查詢字串參數進行請求驗證，請選擇 Required (必要) 選項。如需請求驗證的詳細資訊，請參閱「[在 API Gateway 中啟用請求驗證 \(p. 307\)](#)」。
  - e. 若要將新建立的查詢字串參數當作快取金鑰的一部分來使用，請核取 Caching (快取) 選項。這僅適用於啟用快取時。如需快取的詳細資訊，請參閱「[使用方法/整合參數作為快取金鑰 \(p. 465\)](#)」。

#### Tip

若要移除查詢字串參數，請選擇與其關聯的 x 圖示，然後選擇 Remove this parameter and any dependent parameters (移除此參數與任何相依參數) 以確認移除。

若要變更查詢字串參數的名稱，請將它移除，然後建立新的名稱。

6. 若要將標頭參數新增至方法，請執行下列操作：

- a. 選擇 HTTP Request Headers (HTTP 請求標頭) 旁的箭頭，然後選擇 Add header (新增標頭)。
- b. 針對 Name (名稱)，輸入標頭參數的名稱，然後選擇核取記號圖示以儲存設定。
- c. 若要使用新建立的標頭參數進行請求驗證，請選擇 Required (必要) 選項。如需請求驗證的詳細資訊，請參閱「[在 API Gateway 中啟用請求驗證 \(p. 307\)](#)」。
- d. 若要將新建立的標頭參數當作快取金鑰的一部分來使用，請選擇 Caching (快取) 選項。這僅適用於啟用快取時。如需快取的詳細資訊，請參閱「[使用方法/整合參數作為快取金鑰 \(p. 465\)](#)」。

Tip

若要移除標頭參數，請選擇與其關聯的 x 圖示，然後選擇 Remove this parameter and any dependent parameters (移除此參數與任何相依參數) 以確認移除。

若要變更標頭參數的名稱，請將它移除，然後建立新的名稱。

7. 若要使用 POST、PUT 或 PATCH HTTP 動詞來宣告方法請求的承載格式，請展開 Request Body (請求本文)，然後執行下列操作：

- a. 選擇 Add model (新增模型)。
- b. 針對 Content type (內容類型) 輸入 MIME 類型 (例如 application/json)。
- c. 開啟 Model name (模型名稱) 下拉式選單來選擇承載的可用模型，然後選擇核取記號圖示以儲存設定。

API 的目前可用模型包括預設的 Empty 與 Error 模型，以及您已建立並新增至 API 之[模型](#)集合的任何模型。如需建立模型的詳細資訊，請參閱「[建立模型 \(p. 250\)](#)」。

Note

此模型可用來通知用戶端預期的承載資料格式。它對產生骨架映射範本很有幫助。請務必使用 Java、C#、Objective-C 與 Swift 等語言，來產生 API 的強型別開發套件。只有對承載啟用請求驗證時才需要這樣做。

8. 若要在此 API 的 Java 開發套件中指派 API Gateway 所產生的操作名稱，請展開 SDK Settings (開發套件設定)，然後在 Operation name (操作名稱) 中輸入名稱。例如，對於 GET /pets/{petId} 的方法請求，對應的 Java 開發套件操作名稱預設為 GetPetsPetId。此名稱是從方法的 HTTP 動詞 (GET) 以及資源路徑變數名稱 (Pets 與 PetId) 建構而來。如果您將操作名稱設定為 getPetById，開發套件操作名稱會變成 GetPetById。

## 使用 API Gateway 主控台設定 API Gateway 方法回應

一個 API 方法可以有一或多個回應。每個回應是由其 HTTP 狀態碼編製索引。API Gateway 主控台預設會將 200 回應新增至方法回應。您可以修改它；例如，讓方法改為傳回 201。您可以新增其他回應；例如，409 表示拒絕存取，而 500 表示使用了未初始化的階段變數。

若要使用 API Gateway 主控台來修改、刪除回應或將回應新增至 API 方法，請遵循下列說明進行。

1. 針對 API 資源的指定方法，從 Method Execution (方法執行) 選擇 Method Response (方法回應)。
2. 若要新增回應，請選擇 Add Response (新增回應)。
  - a. 輸入 HTTP 狀態碼；例如針對 HTTP Status (HTTP 狀態) 輸入 200、400 或 500，然後選擇核取記號以儲存選擇。

當後端傳回的回應未定義對應的方法回應時，API Gateway 無法將回應傳回至用戶端。相反地，它會傳回 500 Internal server error 錯誤回應。

- b. 展開指定狀態碼的回應。
- c. 選擇 Add Header (新增標頭)。

- d. 針對 Response Headers for **{status}** ({status} 的回應標頭) 下的 Name (名稱) 輸入名稱，然後選擇核取記號圖示以儲存選擇。

如果您需要將任何後端傳回的標頭轉譯為方法回應中定義的一個標頭，您必須先新增方法回應標頭，如這個步驟中所述。

- e. 在 Response Body for **{status}** ({status} 的回應本文) 下，選擇 Add Response Model (新增回應模型)。
- f. 針對 Content type (內容類型) 輸入回應承載的媒體類型，然後從 Models (模型) 下拉式選單中選擇一個模型。
- g. 選擇核取記號圖示以儲存設定。
3. 若要修改現有的回應，請展開回應，然後遵循上述的步驟 2 進行。
4. 若要移除回應，請選擇回應的 x 圖示，然後確認您要刪除回應。

對於從後端傳回的每個回應，您必須將相容的回應設定為方法回應。不過，除非您將結果從後端映射到方法回應，再傳回用戶端，否則設定方法回應標頭與承載模型為選擇性。此外，如果您想要為 API 產生強型別開發套件，方法回應承載模型就很重要。

## 在 API Gateway 中設定 REST API 整合

設定 API 方法後，您必須整合它與後端的端點。後端端點也稱為整合端點，可以是 Lambda 函數、HTTP 網頁或 AWS 服務動作。如同使用 API 方法一樣，API 整合有整合請求和整合回應。整合請求會封裝後端收到的 HTTP 請求。它可能會，也可能不會與用戶端提交的方法不同。整合回應是封裝後端傳回輸出的 HTTP 回應。

設定整合請求包括以下內容：設定如何將用戶端提交的方法請求傳送到後端、設定如何轉換請求資料；如有必要，轉換成整合請求資料、指定呼叫哪些 Lambda 函數、指定傳入請求要轉送到哪些 HTTP 伺服器、或指定要呼叫的 AWS 服務動作。

設定只適用於非代理整合的整合回應，包括下列內容：設定如何將後端傳回的結果傳送到指定狀態碼的方法回應、設定如何將指定的整合回應參數轉換成預先設定的方法回應參數、以及設定如何根據指定的內文映射範本，將整合回應內文映射到方法回應內文。

以程式設計方式，整合請求由 [Integration](#) 資源封裝，而整合回應由 API Gateway 的 [IntegrationResponse](#) 資源封裝。若要設定整合請求，您要建立 [Integration](#) 資源，並用它設定整合端點 URL。然後，設定 IAM 許可存取後端，並指定映射先轉換傳入的請求資料，再傳送到後端。若要設定非代理整合的整合回應，您要建立 [IntegrationResponse](#) 資源，並用它設定其目標方法回應。然後，設定如何將後端輸出映射到方法回應。

### 主題

- [在 API Gateway 中設定整合請求 \(p. 199\)](#)
- [在 API Gateway 中設定整合回應 \(p. 205\)](#)
- [在 API Gateway 中設定 Lambda 整合 \(p. 205\)](#)
- [在 API Gateway 中設定 HTTP 整合 \(p. 224\)](#)
- [設定 API Gateway 私有整合 \(p. 229\)](#)
- [在 API Gateway 中設定模擬整合 \(p. 234\)](#)

## 在 API Gateway 中設定整合請求

若要設定整合請求，您要執行下列必要和選用任務：

1. 選擇決定如何將方法請求資料傳送到後端的整合類型。
2. 針對非模擬整合，指定 HTTP 方法和目標整合端點的 URI，MOCK 整合除外。

3. 針對 Lambda 函數和其他 AWS 服務動作的整合，為 API Gateway 設定具有所需許可的 IAM 角色，代替您呼叫後端。
4. 針對非代理整合，設定必要的參數映射，將預先定義的方法請求參數映射到合適的整合請求參數。
5. 針對非代理整合，設定必要的內文映射，根據指定的映射範本映射傳入的特定內容類型方法請求內文。
6. 針對非代理整合，指定依現狀將傳入的方法請求資料傳送到後端的條件。
7. (選擇性) 也可以指定如何處理二進位承載的類型轉換。
8. (選擇性) 告知快取命名空間名稱和快取金鑰參數，啟用 API 快取。

執行這些任務包括建立 API Gateway 的 [Integration](#) 資源，以及設定適當的屬性值。您可以使用 API Gateway 主控台、AWS CLI、命令、AWS 開發套件或 API Gateway REST API 來執行這項操作。

#### 主題

- [API 整合請求的基本任務 \(p. 200\)](#)
- [選擇 API Gateway API 整合類型 \(p. 201\)](#)
- [設定代理整合與代理資源 \(p. 202\)](#)
- [使用 API Gateway 主控台設定 API 整合請求 \(p. 203\)](#)

## API 整合請求的基本任務

整合請求是一項 HTTP 請求，它是由 API Gateway 提交到後端，隨用戶端提交的請求資料傳送，並在有必要時轉換資料。HTTP 方法（或動詞）和整合請求的 URI 是由後端（即整合端點）提出。它們分別和方法請求的 HTTP 方法和 URI 可以相同或不同。例如，當 Lambda 函數傳回從 Amazon S3 擷取的檔案時，您可以直覺地向用戶端公開此操作為 GET 方法請求，即使對應的整合請求需要使用 POST 請求呼叫 Lambda 函數。若為 HTTP 端點，有可能方法請求和對應的整合請求都使用相同的 HTTP 動詞。不過，這不是必要的。您可以整合以下方法請求：

```
GET /{var}?query=value
Host: api.domain.net
```

使用以下整合請求：

```
POST /
Host: service.domain.com
Content-Type: application/json
Content-Length: ...

{
    path: "{var}'s value",
    type: "value"
}
```

身為 API 開發人員，您可以使用任何滿足您需求的方法請求 HTTP 動詞和 URI。但是，您必須遵循整合端點的需求。當方法請求資料和整合請求資料不同時，您可以提供方法請求資料到整合請求資料的映射，以調節差異。在上述範例中，映射會將 {var} 方法請求的路徑變數 (query) 和查詢參數 (GET) 值轉譯為整合請求承載屬性 path 和 type 的值。其他可映射請求資料包括請求標頭和內文。「[使用 API Gateway 主控台設定請求與回應資料映射 \(p. 242\)](#)」中會詳加說明。

設定 HTTP 或 HTTP 代理整合請求時，您要將後端 HTTP 端點 URL 指派為整合請求 URI 值。例如，在 PetStore API 中，取得寵物頁面的方法請求有以下整合請求 URI：

```
http://petstore-demo-endpoint.execute-api.com/petstore/pets
```

設定 Lambda 或 Lambda 代理整合時，您要將呼叫 Lambda 函數的 Amazon Resource Name (ARN) 指派為整合請求 URI 值。此 ARN 的格式如下：

```
arn:aws:apigateway:api-region:lambda:path//2015-03-31/functions/arn:aws:lambda:lambda-region:account-id:function:lambda-function-name/invocations
```

arn:aws:apigateway:*api-region*:lambda:path/ 後面的部分，亦即 /2015-03-31/functions/arn:aws:lambda:*lambda-region:account-id:function:lambda-function-name*/invocations，是 Lambda [Invoke](#) 動作的 REST API URI 路徑。如果您使用 API Gateway 主控台設定 Lambda 整合，API Gateway 就會建立 ARN，並在提示您從區域選擇 *lambda-function-name* 後，將它指派給整合 URI。

設定另一項 AWS 服務動作的整合請求時，整合請求 URI 也是 ARN，類似 Lambda [Invoke](#) 動作的整合。例如，Amazon S3 [GetBucket](#) 動作的整合，整合請求 URI 是以下格式的 ARN：

```
arn:aws:apigateway:api-region:s3:path/{bucket}
```

整合請求 URI 是指定動作的路徑慣例，其中 *{bucket}* 是儲存貯體名稱的預留位置。或者，您可以依其名稱參考 AWS 服務動作。使用動作名稱，Amazon S3 [GetBucket](#) 動作的整合請求 URI 會變成下面這樣：

```
arn:aws:apigateway:api-region:s3:action/GetBucket
```

使用動作型整合請求 URI，您必須在整合請求內文 ({ Bucket: "*{bucket}*" }) 中指定儲存貯體名稱 (*{bucket}*)，遵循 [GetBucket](#) 動作的輸入格式。

若為 AWS 整合，您還必須設定 [登入資料](#)，讓 API Gateway 呼叫整合的動作。您可以針對 API Gateway 建立新的或選擇現有的 IAM 角色來呼叫動作，然後使用其 ARN 指定角色。以下所示為此 ARN 範例：

```
arn:aws:iam::account-id:role/iam-role-name
```

此 IAM 角色必須包含允許執行動作的政策。它還必須讓 API Gateway 壓告 (在角色的信任關係中) 為信任的實體以擔任該角色。這種許可能夠授予動作本身。它們稱為資源型許可。針對 Lambda 整合，您可以呼叫 Lambda 的 [addPermission](#) 動作來設定資源型許可，然後在 API Gateway 整合請求中將 `credentials` 設定為 null。

我們已討論過基本的整合設定。進階設定涉及將方法請求資料映射到整合請求資料。討論過整合回應的基本設定之後，我們會在「[使用 API Gateway 主控台設定請求與回應資料映射 \(p. 242\)](#)」中涵蓋進階主題，同時論及傳送承載和處理內容編碼。

## 選擇 API Gateway API 整合類型

您可以根據您要使用的整合端點類型以及希望資料移入移出整合端點的方式，選擇 API 整合類型。Lambda 函數可以有 Lambda 代理整合或 Lambda 自訂整合。HTTP 端點可以有 HTTP 代理整合或 HTTP 自訂整合。AWS 服務動作則僅能有非代理類型的 AWS 整合。API Gateway 也支援 API Gateway 做為整合端點回應方法請求的模擬整合。

Lambda 自訂整合是特殊的 AWS 整合案例，在此案例中整合端點會映射到 Lambda 服務的 [函數呼叫動作](#)。

您可以程式設計的方式，在 [Integration](#) 資源上設定 `type` 屬性來選擇整合類型。Lambda 代理整合的值為 `AWS_PROXY`。至於 Lambda 自訂整合和所有其他 AWS 整合，它是 AWS。HTTP 代理整合和 HTTP 整合的值分別為 `HTTP_PROXY` 和 `HTTP`。若為模擬整合，`type` 值為 `MOCK`。

Lambda 代理整合支援以單一 Lambda 函數簡化的整合設定。設定很簡單，而且可隨後端發展，不必縮減現有的設定。基於這些原因，強烈建議您用於使用 Lambda 函數的整合。反之，Lambda 自訂整合允許輸入和輸出資料格式有類似需求的各種整合端點，重複使用設定的映射範本。更進階的應用程式案例中，建議使用更為複雜的設定。

同樣地，HTTP 代理整合有簡化的整合設定，可隨後端發展，不必縮減現有的設定。HTTP 自訂整合與設定更密切，但允許其他整合端點重複使用設定的映射範本。

以下清單摘要說明支援的整合類型：

- AWS：這類整合可讓 API 公開 AWS 服務動作。在 AWS 整合中，您必須同時設定整合請求和整合回應，並設定從方法請求到整合請求以及從整合回應到方法回應的必要資料映射。
- AWS\_PROXY：此整合類型可以整合 API 方法與 Lambda 函數呼叫動作，提供靈活、多樣化和簡化的整合設定。這個整合依賴用戶端和已整合 Lambda 函數之間的直接互動。使用這類整合，也稱為 Lambda 代理整合，您就不需要設定整合請求或整合回應。API Gateway 將來自用戶端的傳入請求當做輸入傳送到後端 Lambda 函數。已整合的 Lambda 函數採用 [此格式的輸入 \(p. 212\)](#)並剖析所有可用來源的輸入，包括請求標頭、URL 路徑變數、查詢字串參數和適用的內文。函數會按照這個 [輸出格式 \(p. 216\)](#)傳回結果。這種慣用的整合類型，會透過 API Gateway 呼叫 Lambda 函數，但並不適用任何其他 AWS 服務動作，包括函數呼叫動作以外的 Lambda 動作。
- HTTP：這類整合可讓 API 公開後端的 HTTP 端點。使用 HTTP 整合，也稱為 HTTP 自訂整合，您必須設定整合請求和整合回應，缺一不可。您必須設定從方法請求到整合請求以及從整合回應到方法回應的必要資料映射。
- HTTP\_PROXY：HTTP 代理整合可讓用戶端使用單一 API 方法的簡化整合設定，存取後端的 HTTP 端點。您不用設定整合請求或整合回應。API Gateway 會將傳入請求從用戶端傳送到 HTTP 端點，將傳出回應從 HTTP 端點傳送到用戶端。
- MOCK：這類整合可讓 API Gateway 傳回回應，卻無需進一步將請求傳送到後端。這對 API 測試很有用，因為它可用來測試整合設定，卻不會產生使用後端的費用，而且可以啟用 API 協作開發。在協作開發中，小組可以使用 MOCK 整合設定模擬其他小組擁有的 API 元件，區隔他們的開發成果。它也可用來傳回 CORS 相關的標頭，確保 API 方法允許 CORS 存取。事實上，API Gateway 主控台會整合 OPTIONS 方方法來支援使用模擬整合的 CORS。[閘道回應 \(p. 237\)](#)是其他模擬整合的範例。

## 設定代理整合與代理資源

若要在具有代理資源的 API Gateway API 中設定代理整合，您可以執行下列任務：

- 使用 Greedy 路徑變數 `{proxy+}` 建立代理資源。
- 在代理資源上設定 ANY 方法。
- 使用 HTTP 或 Lambda 整合類型來整合資源/方法與後端。

### Note

Greedy 路徑變數、ANY 方法與代理整合類型是獨立功能，但通常會一起使用。您可以在 Greedy 資源上設定特定 HTTP 方法，或將非代理整合類型套用至代理資源。

使用 Lambda 代理整合或 HTTP 代理整合處理方法時，API Gateway 會實施特定約束與限制。如需詳細資訊，請參閱 [the section called “重要說明” \(p. 630\)](#)。

### Note

搭配傳遞使用代理整合時，如果未指定承載的內容類型，API Gateway 會傳回預設 Content-Type:application/json 標頭。

使用 HTTP 代理整合或 Lambda 代理整合與後端整合時，代理資源最強大。

## HTTP 代理整合與代理資源

HTTP 代理整合是由 API Gateway REST API 中的 `HTTP_PROXY` 所指定，適用於整合方法請求與後端 HTTP 端點。有了此整合類型，API Gateway 可根據特定 [約束與限制 \(p. 630\)](#)，直接在前端與後端之間傳遞整個請求與回應。

#### Note

HTTP 代理整合支援多值標頭和查詢字串。

將 HTTP 代理整合套用至代理資源時，您可以設定 API 透過單一整合設定，公開 HTTP 後端的部分或整個端點階層。例如，假設網站的後端從根節點 (/site) 組織成樹狀目錄節點的多個分支：/site/a<sub>0</sub>/a<sub>1</sub>/.../a<sub>N</sub>、/site/b<sub>0</sub>/b<sub>1</sub>/.../b<sub>M</sub> 等等。如果您將 /api/{proxy+} 之代理資源上的 ANY 方法與 /site/{proxy} 之 URL 路徑的後端端點整合，單一整合請求可支援任何 [a<sub>0</sub>, a<sub>1</sub>, ..., a<sub>N</sub>, b<sub>0</sub>, b<sub>1</sub>, ..., b<sub>M</sub>, ...] 上的 HTTP 操作 (GET、POST 等)。如果您將代理整合套用至特定 HTTP 方法 (例如 GET)；相反地，產生的整合請求適用於任何後端節點上的指定 (也就是 GET) 操作。

#### Lambda 代理整合與代理資源

Lambda 代理整合是由 API Gateway REST API 中的 AWS\_PROXY 所指定，適用於整合方法請求與後端 Lambda 函數。有了此整合類型，API Gateway 可套用預設對應範本將整個請求傳送到 Lambda 函數，並將來自 Lambda 函數的輸出轉換成 HTTP 回應。

同樣地，您可以將 Lambda 代理整合套用至 /api/{proxy+} 的代理資源來設定單一整合，讓後端 Lambda 函數對 /api 下的任何 API 資源變更個別做出回應。

#### 使用 API Gateway 主控台設定 API 整合請求

API 方法設定會定義方法並描述其行為。若要設定方法，您必須指定資源，包括公開方法的根目錄 ("")、HTTP 方法 (GET、POST 等等)，以及與目標後端整合的方法。方法請求和回應會指定呼叫應用程式的合約，規定 API 收到哪些參數以及回應的外觀。

下列程序說明如何使用 API Gateway 主控台指定方法設定。

1. 在 Resources (資源) 窗格中，選擇方法。
2. 在 Method Execution (方法執行) 窗格中，選擇 Integration Request (整合請求)。針對 Integration type (整合類型)，選擇以下其中一項：
  - 如果您的 API 要與 Lambda 函數整合，請選擇 Lambda Function (&LAM; 函數)。在 API 層級，這是 AWS 整合類型。
  - 如果您的 API 要與 HTTP 端點整合，請選擇 HTTP。在 API 層級，這是 HTTP 整合類型。
  - 如果您的 API 要與 AWS 服務直接整合，請選擇 AWS Service (&AWS; 服務)。在 API 層級，這是 AWS 整合類型。上述的 Lambda Function (&LAM; 函數) 選項是呼叫 Lambda 函數的 AWS 整合特殊案例，只能在 API Gateway 主控台中使用。若要設定 API Gateway API 在 AWS Lambda 中建立新的 Lambda 函數；要對 Lambda 函數設定資源許可；或要執行任何其他 Lambda 服務動作，您必須在這裡選擇 AWS Service (&AWS; 服務) 選項。
  - 如果您希望 API Gateway 做為後端傳回靜態回應，請選擇 Mock (模擬)。在 API 層級，這是 MOCK 整合類型。一般而言，當您的 API 尚未到達最終形態，但您希望產生 API 回應解鎖相依小組進行測試時，您可以使用 MOCK 整合。針對 OPTION 方法，API Gateway 會將 MOCK 整合設為預設值，針對已套用的 API 資源傳回 CORS 啟用的標頭。如果您選擇此選項，請跳過本主題的其他指示並參閱 [在 API Gateway 中設定模擬整合 \(p. 234\)](#)。
3. 如果您選擇 Lambda Function (&LAM; 函數)，請執行下列操作：
  - a. 針對 Lambda Region (&LAM; 區域)，選擇對應到您 Lambda 函數建立所在區域的區域識別符。例如，如已在 US East (N. Virginia) 區域建立 Lambda 函數，請選擇 us-east-1。如需區域名稱和識別符清單，請參閱 Amazon Web Services 一般參考中的 [AWS Lambda](#)。
  - b. 針對 Lambda Function (&LAM; 函數)，輸入 Lambda 函數的名稱，然後選擇函數的對應 ARN。
  - c. 選擇 Save (儲存)。
4. 如果您選擇 HTTP，請執行下列操作：
  - a. 針對 HTTP method (HTTP 方法)，選擇最符合 HTTP 後端中方法的 HTTP 方法類型。
  - b. 針對 Endpoint URL (端點 URL)，輸入您希望此方法使用之 HTTP 後端的 URL。

- c. 選擇 Save (儲存)。
5. 如果您選擇 Mock (模擬) , 請執行下列操作 :
  - 選擇 Save (儲存)。
6. 如果您選擇 AWS Service (&AWS; 服務) , 請執行下列操作 :
  - a. 針對 AWS Region (&AWS; 區域) , 選擇您希望此方法用來呼叫動作的 AWS 區域。
  - b. 針對 AWS Service (&AWS; 服務) , 選擇您希望此方法呼叫的 AWS 服務。
  - c. 針對 AWS Subdomain (&AWS; 子網域) , 輸入 AWS 服務使用的子網域。這個項目一般會保持空白。有些 AWS 服務會支援子網域作為主機的一部分。請參閱服務文件以了解可用性及詳細資訊 (如有)。
  - d. 針對 HTTP method (HTTP 方法) , 選擇對應動作的 HTTP 方法類型。如需 HTTP 方法類型 , 請參閱您針對 AWS Service (&AWS; 服務) 選擇之 AWS 服務的 API 參考文件。
  - e. 針對 Action (動作) , 輸入您要使用的動作。如需可用動作的清單 , 請參閱您針對 AWS Service (&AWS; 服務) 選擇之 AWS 服務的 API 參考文件。
  - f. 針對 Execution Role (執行角色) , 輸入方法要用來呼叫動作之 IAM 角色的 ARN。

若要建立 IAM 角色 , 您可以調整[建立 Lambda 函數 \(p. 31\)](#)一節中「建立 Lambda 叫用角色及其政策」和「建立 Lambda 執行角色及其政策」的指示。指定以下格式的存取政策 , 包含所需動作數和資源陳述式 :

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "action-statement"  
            ],  
            "Resource": [  
                "resource-statement"  
            ]  
        },  
        ...  
    ]  
}
```

如需動作和資源陳述式語法 , 請參閱您針對 AWS Service (&AWS; 服務) 選擇的 AWS 服務文件。

如需 IAM 角色的信任關係 , 請指定以下動作 , 讓 API Gateway 代表您的 AWS 帳戶採取行動 :

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

- g. 如果您針對 Action (動作) 輸入的動作提供您希望此方法使用的自訂資源路徑 , 請在 Path Override (路徑覆寫) 中輸入此自訂資源路徑。如需自訂資源路徑 , 請參閱您針對 AWS Service (&AWS; 服務) 選擇之 AWS 服務的 API 參考文件。

- h. 選擇 Save (儲存)。

## 在 API Gateway 中設定整合回應

針對非代理整合，您必須至少設定一個整合回應，並讓它成為預設的回應，才能將後端傳回的結果傳送到用戶端。您可以選擇依現狀傳送結果，或將整合回應資料轉換成方法回應資料，如果它們兩個格式不同。

如需代理整合，API Gateway 會自動將後端輸出當作 HTTP 回應傳送至用戶端。您不用設定整合回應或方法回應。

若要設定整合回應，您要執行下列必要和選用任務：

1. 指定整合回應資料映射的方法回應 HTTP 狀態碼。這是必要的。
2. 定義規則表達式選取此整合回應要代表的後端輸出。如果此項目保留空白，此回應是用來擷取所有尚未設定回應的預設回應。
3. 如有需要，請宣告使用鍵值對組成的映射，將指定的整合回應參數映射到指定的方法回應參數。
4. 如有需要，請新增內文映射範本，將指定的整合回應承載傳送到指定的方法回應承載。
5. 如有需要，請指定如何處理二進位承載的類型轉換。

整合回應是封裝後端回應的 HTTP 回應。HTTP 端點的後端回應是 HTTP 回應。整合回應狀態碼可以採用後端傳回的狀態碼，整合回應內文是後端傳回的承載。Lambda 端點的後端回應是從 Lambda 函數傳回的輸出。使用 Lambda 整合，Lambda 函數輸出會傳回為 200 OK 回應。承載可以包含結果當作 JSON 資料，包括 JSON 字串或 JSON 物件，或當作 JSON 物件的錯誤訊息。您可以將常規表達式指派給 `selectionPattern` 屬性，將錯誤回應映射到適當的 HTTP 錯誤回應。如需 Lambda 函數錯誤回應的詳細資訊，請參閱 [在 API Gateway 中處理 Lambda 錯誤 \(p. 221\)](#)。使用 Lambda 代理整合，Lambda 函數必須傳回格式如下的輸出：

```
{  
  statusCode: "...",           // a valid HTTP status code  
  headers: {  
    custom-header: "..."      // any API-specific custom header  
  },  
  body: "...",                // a JSON string.  
  isBase64Encoded: true|false // for binary support  
}
```

您不必將 Lambda 函數回應映射到其正確的 HTTP 回應。

若要將結果傳回給用戶端，請設定整合回應依現狀將端點回應傳送到對應的方法回應。或者，您可以將端點回應資料映射到方法回應資料。可映射的回應資料包括回應狀態碼、回應標頭參數和回應內文。傳回的狀態碼如果未定義任何方法回應，API Gateway 會傳回 500 錯誤。如需更多詳細資訊，請參閱 [針對請求與回應對應建立模型與對應範本 \(p. 244\)](#)。

## 在 API Gateway 中設定 Lambda 整合

您可以使用 Lambda 代理整合或 Lambda 非代理（自訂）整合，將 API 方法與 Lambda 函數整合。

在 Lambda 代理整合中，設定非常簡單。如果您的 API 不需要內容編碼或快取，則只需要將整合的 HTTP 方法設定為 POST、將整合端點 URI 設定為特定 Lambda 函數之 Lambda 函數呼叫動作的 ARN，並且將登入資料設定為 IAM 角色，而此角色具有允許 API Gateway 代表您呼叫 Lambda 函數的許可。

在 Lambda 非代理整合中，除了代理整合設定步驟之外，您也可以指定如何將傳入請求資料映射至整合請求，以及如何將產生的整合回應資料映射至方法回應。

主題

- 在 API Gateway 中設定 Lambda 代理整合 (p. 206)
- 在 API Gateway 中設定 Lambda 自訂整合 (p. 217)
- 設定後端 Lambda 函數的非同步呼叫 (p. 220)
- 在 API Gateway 中處理 Lambda 錯誤 (p. 221)

## 在 API Gateway 中設定 Lambda 代理整合

### 主題

- [了解 API Gateway Lambda 代理整合 \(p. 206\)](#)
- [支援多值標頭和查詢字串參數 \(p. 207\)](#)
- [設定代理資源與 Lambda 代理整合 \(p. 207\)](#)
- [使用 AWS CLI 設定 Lambda 代理整合 \(p. 209\)](#)
- [代理整合之 Lambda 函數的輸入格式 \(p. 212\)](#)
- [代理整合之 Lambda 函數的輸出格式 \(p. 216\)](#)

### 了解 API Gateway Lambda 代理整合

Amazon API Gateway Lambda 代理整合是一個簡單、強大且靈活的機制，可透過設定單一 API 方法來建置 API。Lambda 代理整合可讓用戶端在後端呼叫單一 Lambda 函數。此函數可存取其他 AWS 服務的許多資源或功能，包括呼叫其他 Lambda 函數。

在 Lambda Proxy 整合中，當用戶端提交一個 API 請求時，API Gateway 會按原狀將原始請求傳遞至整合的 Lambda 函數，但不會保留請求參數的順序。此[請求資料 \(p. 212\)](#)包含請求標頭、查詢字串參數、URL 路徑變數、承載與 API 組態資料。組態資料可包含目前的部署階段名稱、階段變數、使用者身分或授權內容 (如果有)。後端 Lambda 函數會剖析傳入請求資料，以判斷其所傳回的回應。若要讓 API Gateway 將 Lambda 輸出當做 API 回應傳遞至用戶端，Lambda 函數必須以此格式 (p. 216) 傳回結果。

由於 API Gateway 不用在用戶端與後端 Lambda 函數之間介入太多就可進行 Lambda 代理整合，因此用戶端與整合的 Lambda 函數可以根據彼此的變更進行調整，而不需要中斷 API 的現有整合設定。若要啟用這項功能，用戶端必須遵循後端 Lambda 函數所制定的應用程式通訊協定。

您可以設定任何 API 方法的 Lambda 代理整合。但針對涉及一般代理資源的 API 方法設定時，Lambda 代理整合會更有效。一般代理資源可由 {proxy+} 的特殊樣板化路徑變數、catch-all ANY 方法預留位置或兩者來表示。用戶端可以在傳入請求中將輸入當作請求參數或適用的承載傳遞到後端 Lambda 函數。這些請求參數包含標頭、URL 路徑變數、查詢字串參數與適用的承載。整合的 Lambda 函數會驗證所有輸入來源，再處理請求，如果遺失所要求的任何輸入，則會以有意義的錯誤訊息來回應用戶端。

呼叫與 ANY 的泛型 HTTP 方法以及 {proxy+} 的一般資源整合的 API 方法時，用戶端會以特定 HTTP 方法取代 ANY 來提交請求。用戶端也會指定特定 URL 路徑 (而不是 {proxy+})，並包含任何必要的標頭、查詢字串參數或適用的承載。

下列清單摘要說明不同 API 方法與 Lambda 代理整合的執行階段行為：

- ANY /{proxy+}：用戶端必須選擇特定 HTTP 方法、必須設定特定資源路徑階層，並可設定任何標頭、查詢字串參數與適用的承載，以將資料做為輸入傳遞至整合的 Lambda 函數。
- ANY /res：用戶端必須選擇特定 HTTP 方法，並可設定任何標頭、查詢字串參數與適用的承載，以將資料做為輸入傳遞至整合的 Lambda 函數。
- GET|POST|PUT|... /{proxy+}：用戶端可設定特定資源路徑階層、任何標頭、查詢字串參數與適用的承載，以將資料做為輸入傳遞至整合的 Lambda 函數。
- GET|POST|PUT|... /res/{path}/...：用戶端必須選擇特定路徑區段 (針對 {path} 變數)，並可設定任何請求標頭、查詢字串參數與適用的承載，以將輸入資料傳遞至整合的 Lambda 函數。

- GET | POST | PUT | ... /res : 用戶端可選擇任何請求標頭、查詢字串參數與適用的承載，以將輸入資料傳遞至整合的 Lambda 函數。

{proxy+} 的代理資源與 {custom} 的自訂資源都可使用樣板化路徑變數來表示。不過，{proxy+} 可參考沿著路徑階層的任何資源，但 {custom} 只能參考特定路徑區段。例如，雜貨店可能會依部門名稱、產品類別與產品類型，來組織其線上產品庫存。雜貨店的網站可接著透過自訂資源的下列樣板化路徑變數來表示可用的產品：/{department}/{produce-category}/{product-type}。例如，蘋果是以 /produce/fruit/apple 表示，而紅蘿蔔是以 /produce/vegetables/carrot 表示。它也可以使用 /{proxy+} 來表示客戶在線上商店購物時可搜尋的任何部門、任何產品類別或任何產品類型。例如，/{proxy+} 可參考下列任何項目：

- /produce
- /produce/fruit
- /produce/vegetables/carrot

若要讓客戶搜尋任何可用的產品、其產品類別與相關聯的商店部門，您可以公開 GET /{proxy+} 的單一方法並授予唯讀許可。同樣地，若要讓主管更新 produce 部門的庫存，您可以設定 PUT /produce/{proxy+} 的另一個單一方法並授予讀取/寫入許可。若要讓出納員更新蔬菜的計算加總，您可以設定 POST /produce/vegetables/{proxy+} 方法並授予讀取/寫入許可。若要讓商店經理對任何可用的產品執行任何可能的動作，線上商店開發人員可以公開 ANY /{proxy+} 方法並授予讀取/寫入許可。在任何情況下，客戶或員工都必須在執行階段選取所選部門中指定類型的特定產品、所選部門中的特定產品類別或特定部門。

如需設定 API Gateway 代理整合的詳細資訊，請參閱[設定代理整合與代理資源 \(p. 202\)](#)。

代理整合需要用戶端更詳細了解後端需求。因此，若要確保最佳應用程式效能與使用者體驗，後端開發人員必須向用戶端開發人員清楚表達後端的需求，並提供未符合需求時的完善錯誤回饋機制。

## 支援多值標頭和查詢字串參數

API Gateway 支援具有相同名稱的多個標頭和查詢字串參數。多值標頭以及單值標頭和參數可在相同的請求和回應中結合使用。如需更多詳細資訊，請參閱[代理整合之 Lambda 函數的輸入格式 \(p. 212\)](#) 及[代理整合之 Lambda 函數的輸出格式 \(p. 216\)](#)。

## 設定代理資源與 Lambda 代理整合

若要設定代理資源與 Lambda 代理整合類型搭配，請建立 API 資源與 Greedy 路徑參數（例如 /parent/{proxy+}）搭配，並將此資源與 ANY 方法上的 Lambda 函數後端（例如 arn:aws:lambda:us-west-2:123456789012:function:SimpleLambda4ProxyResource）整合。Greedy 路徑參數必須位於 API 資源路徑結尾。如同非代理資源，您可以使用 API Gateway 主控台、匯入 OpenAPI 定義檔或直接呼叫 API Gateway REST API，來設定代理資源。

下列 OpenAPI API 定義檔顯示 API 範例，其中具有與名為 SimpleLambda4ProxyResource 的 Lambda 函數整合的代理資源。

### OpenAPI 3.0

```
{
  "openapi": "3.0.0",
  "info": {
    "version": "2016-09-12T17:50:37Z",
    "title": "ProxyIntegrationWithLambda"
  },
  "paths": {
    "/{proxy+}": {
      "x-amazon-apigateway-any-method": {
        "parameters": [

```

```
{  
    "name": "proxy",  
    "in": "path",  
    "required": true,  
    "schema": {  
        "type": "string"  
    }  
},  
],  
"responses": {},  
"x-amazon-apigateway-integration": {  
    "responses": {  
        "default": {  
            "statusCode": "200"  
        }  
    },  
    "uri": "arn:aws:apigateway:us-east-1:lambda:path/2015-03-31/functions/  
arn:aws:lambda:us-east-1:123456789012:function:SimpleLambda4ProxyResource/invocations",  
    "passthroughBehavior": "when_no_match",  
    "httpMethod": "POST",  
    "cacheNamespace": "roq9wj",  
    "cacheKeyParameters": [  
        "method.request.path.proxy"  
    ],  
    "type": "aws_proxy"  
}  
}  
}  
},  
"servers": [  
    {  
        "url": "https://gy415nuibc.execute-api.us-east-1.amazonaws.com/{basePath}",  
        "variables": {  
            "basePath": {  
                "default": "/testStage"  
            }  
        }  
    }  
]  
}
```

#### OpenAPI 2.0

```
{  
    "swagger": "2.0",  
    "info": {  
        "version": "2016-09-12T17:50:37Z",  
        "title": "ProxyIntegrationWithLambda"  
    },  
    "host": "gy415nuibc.execute-api.us-east-1.amazonaws.com",  
    "basePath": "/testStage",  
    "schemes": [  
        "https"  
    ],  
    "paths": {  
        "/{proxy+}": {  
            "x-amazon-apigateway-any-method": {  
                "produces": [  
                    "application/json"  
                ],  
                "parameters": [  
                    {  
                        "name": "proxy",  
                        "in": "path",  
                        "required": true,  
                        "type": "string"  
                    }  
                ]  
            }  
        }  
    }  
}
```

```
        "type": "string"
    }
],
"responses": {},
"x-amazon-apigateway-integration": {
    "responses": {
        "default": {
            "statusCode": "200"
        }
    },
    "uri": "arn:aws:apigateway:us-east-1:lambda:path/2015-03-31/functions/
arn:aws:lambda:us-east-1:123456789012:function:SimpleLambda4ProxyResource/invocations",
    "passthroughBehavior": "when_no_match",
    "httpMethod": "POST",
    "cacheNamespace": "roq9wj",
    "cacheKeyParameters": [
        "method.request.path.proxy"
    ],
    "type": "aws_proxy"
}
}
}
```

使用 Lambda 代理整合時，API Gateway 會在執行時間將傳入請求對應至 Lambda 函數的輸入 event 參數。輸入包含請求方法、路徑、標頭、任何查詢字串參數、任何承載、相關聯內容和任何已定義的階段變數。「[代理整合之 Lambda 函數的輸入格式 \(p. 212\)](#)」說明輸入格式。若要讓 API Gateway 成功將 Lambda 輸出對應至 HTTP 回應，Lambda 函數必須輸出[代理整合之 Lambda 函數的輸出格式 \(p. 216\)](#)中所述格式的結果。

使用透過 ANY 方法之代理資源的 Lambda 代理整合，單一後端 Lambda 函數透過代理資源用作所有請求的事件處理常式。例如，若要記錄流量模式，您可以在代理資源的 URL 路徑中使用 /state/city/street/house 提交請求，讓行動裝置傳送其州/省、城市、街道和建築物的位置資訊。後端 Lambda 函數接著可以剖析 URL 路徑，並將位置元組插入至 DynamoDB 資料表。

### 使用 AWS CLI 設定 Lambda 代理整合

在本節中，我們示範如何使用 AWS CLI 來設定具有 Lambda 代理整合的 API。

#### Note

如需使用 API Gateway 主控台來設定代理資源與 Lambda 代理整合搭配的詳細指示，請參閱教學：[建置具有 Lambda 代理整合的 Hello World API \(p. 24\)](#)。

例如，我們使用下列範例 Lambda 函數做為 API 的後端：

```
exports.handler = function(event, context, callback) {
    console.log('Received event:', JSON.stringify(event, null, 2));
    var res ={
        "statusCode": 200,
        "headers": {
            "Content-Type": "*/*"
        }
    };
    var greeter = 'World';
    if (event.greeter && event.greeter!=="") {
        greeter = event.greeter;
    } else if (event.body && event.body !== "") {
        var body = JSON.parse(event.body);
        if (body.greeter && body.greeter !== "") {
            greeter = body.greeter;
        }
    }
    callback(null, res);
}
```

```
        }
    } else if (event.queryStringParameters && event.queryStringParameters.greeter &&
event.queryStringParameters.greeter != "") {
        greeter = event.queryStringParameters.greeter;
    } else if (event.multiValueHeaders && event.multiValueHeaders.greeter &&
event.multiValueHeaders.greeter != "") {
        greeter = event.multiValueHeaders.greeter.join(" and ");
    } else if (event.headers && event.headers.greeter && event.headers.greeter != "") {
        greeter = event.headers.greeter;
    }

    res.body = "Hello, " + greeter + "!";
    callback(null, res);
};
```

將此項目與 [Lambda 自訂整合設定 \(p. 217\)](#)進行比較，可以在請求參數內文中表示此 Lambda 函數的輸入。您可以有更多的自由，讓用戶端傳遞相同的輸入資料。在這裡，用戶端可以傳入接待員名稱以做為查詢字串參數、標頭或內文屬性。此函數也可以支援 Lambda 自訂整合。API 設定較為簡單。您根本不需要設定方法回應或整合回應。

### 使用 AWS CLI 設定 Lambda 代理整合

1. 呼叫 `create-rest-api` 命令來建立 API：

```
aws apigateway create-rest-api --name 'HelloWorld (AWS CLI)' --region us-west-2
```

請記下回應中所產生 API 的 id 值 (`te6si5ach7`)：

```
{
  "name": "HelloWorldProxy (AWS CLI)",
  "id": "te6si5ach7",
  "createdDate": 1508461860
}
```

在本節中，您需要 API id。

2. 呼叫 `get-resources` 命令來取得根資源 id：

```
aws apigateway get-resources --rest-api-id te6si5ach7 --region us-west-2
```

成功回應顯示如下：

```
{
  "items": [
    {
      "path": "/",
      "id": "krznpq9xpg"
    }
  ]
}
```

記下根資源 id 值 (`krznpq9xpg`)。下一個步驟和之後的步驟都需要它。

3. 呼叫 `create-resource` 來建立 API Gateway /greeting 資源：

```
aws apigateway create-resource --rest-api-id te6si5ach7 \
  --region us-west-2 \
  --parent-id krznpq9xpg \
  --path-part {proxy+}
```

成功回應類似如下：

```
{  
    "path": "/{proxy+}",  
    "pathPart": "{proxy+}",  
    "id": "2jf6xt",  
    "parentId": "krznpq9xpg"  
}
```

記下所產生 `{proxy+}` 資源的 `id` 值 (`2jf6xt`)。在下一個步驟中，您需要它才能在 `/{proxy+}` 資源上建立方法。

4. 呼叫 `put-method` 來建立 ANY 方法請求 ANY `/{proxy+}` :

```
aws apigateway put-method --rest-api-id te6si5ach7 \  
    --region us-west-2 \  
    --resource-id 2jf6xt \  
    --http-method ANY \  
    --authorization-type "NONE"
```

成功回應類似如下：

```
{  
    "apiKeyRequired": false,  
    "httpMethod": "ANY",  
    "authorizationType": "NONE"  
}
```

這個 API 方法可讓用戶端從後端的 Lambda 函數接收或傳送問候語。

5. 呼叫 `put-integration` 來設定具有 Lambda 函數 (即 `HelloWorld`) 之 ANY `/{proxy+}` 方法的整合。如果提供 "`Hello, {name}!`" 參數，則此函數會以 `greeter` 訊息來回應請求，如果未設定查詢字串參數，則會以 "`Hello, World!`" 來回應請求。

```
aws apigateway put-integration \  
    --region us-west-2  
    --rest-api-id vaz7da96z6 \  
    --resource-id 2jf6xt \  
    --http-method ANY \  
    --type AWS_PROXY \  
    --integration-http-method POST \  
    --uri arn:aws:apigateway:us-east-1:lambda:path/2015-03-31/functions/  
arn:aws:lambda:us-east-1:123456789012:function:HelloWorld/invocations \  
    --credentials arn:aws:iam::123456789012:role/apigAwsProxyRole
```

### Important

針對 Lambda 整合，您必須根據 [指定進行函數呼叫的 Lambda 服務動作](#)，為整合請求使用 POST 的 HTTP 方法。IAM 角色 `apigAwsProxyRole` 的政策必須允許 `apigateway` 服務呼叫 Lambda 函數。如需 IAM 許可的詳細資訊，請參閱[the section called “呼叫 API 的 API Gateway 許可模型” \(p. 334\)](#)。

成功輸出類似如下：

```
{  
    "passthroughBehavior": "WHEN_NO_MATCH",  
    "cacheKeyParameters": [],  
    "uri": "arn:aws:apigateway:us-east-1:lambda:path/2015-03-31/functions/  
arn:aws:lambda:us-east-1:123456789012:function:HelloWorld/invocations",
```

```
    "httpMethod": "POST",
    "cacheNamespace": "vvom7n",
    "credentials": "arn:aws:iam::1234567890:role/apigAwsProxyRole",
    "type": "AWS_PROXY"
}
```

您可以呼叫 [add-permission](#) 命令來新增資源型許可，而不提供 `credentials` 的 IAM 角色。這是 API Gateway 主控台的作用。

6. 呼叫 `create-deployment` 來將 API 部署至 `test` 階段：

```
aws apigateway create-deployment --rest-api-id te6si5ach7 --stage-name test
```

7. 在終端機中使用下列 cURL 命令，以測試 API。

使用查詢字串參數 `?greeter=jane` 呼叫 API：

```
curl -X GET 'https://te6si5ach7.execute-api.us-west-2.amazonaws.com/test/greeting?
greeter=jane' \
-H 'authorization: AWS4-HMAC-SHA256 Credential={access_key}/20171020/us-west-2/
execute-api/aws4_request, \
SignedHeaders=content-type;host;x-amz-date, Signature=f327...5751'
```

使用標頭參數 `greeter:jane` 呼叫 API：

```
curl -X GET https://te6si5ach7.execute-api.us-west-2.amazonaws.com/test/hi \
-H 'authorization: AWS4-HMAC-SHA256 Credential={access_key}/20171020/us-west-2/
execute-api/aws4_request, \
SignedHeaders=content-type;host;x-amz-date, Signature=f327...5751' \
-H 'content-type: application/json' \
-H 'greeter: jane'
```

使用內文 `{"greeter": "jane"}` 呼叫 API：

```
curl -X POST https://te6si5ach7.execute-api.us-west-2.amazonaws.com/test \
-H 'authorization: AWS4-HMAC-SHA256 Credential={access_key}/20171020/us-west-2/
execute-api/aws4_request, \
SignedHeaders=content-type;host;x-amz-date, Signature=f327...5751' \
-H 'content-type: application/json' \
-d '{ "greeter": "jane" }'
```

在所有情況下，輸出都是具有下列回應內文的 200 回應：

```
Hello, jane!
```

## 代理整合之 Lambda 函數的輸入格式

使用 Lambda 代理整合，API Gateway 會將整個用戶端請求對應至後端 Lambda 函數的輸入 `event` 參數，如下所示：

```
{
  "resource": "Resource path",
  "path": "Path parameter",
  "httpMethod": "Incoming request's method name"
  "headers": {String containing incoming request headers}
  "multiValueHeaders": {List of strings containing incoming request headers}
  "queryStringParameters": {query string parameters }
```

```
"multiValueQueryStringParameters": {List of query string parameters}
"pathParameters": {path parameters}
"stageVariables": {Applicable stage variables}
"requestContext": {Request context, including authorizer-returned key-value pairs}
"body": "A JSON string of the request payload."
"isBase64Encoded": "A boolean flag to indicate if the applicable request payload is
Base64-encode"
}
```

#### Note

在輸入中：

- `headers` 鍵只能包含單一值標頭。
- `multiValueHeaders` 鍵可以包含多值標頭以及單一值標頭。
- 如果您同時指定 `headers` 和 `multiValueHeaders` 的值，API Gateway 會將它們合併成一個清單。如果在兩者指定了相同的鍵值對，則只有 `multiValueHeaders` 中的值會出現在合併清單。

下列 POST 請求顯示透過階段變數 `stageVariableName=stageVariableValue` 來部署至 `testStage` 的 API：

```
POST /testStage/hello/world?name=me HTTP/1.1
Host: gy415nuibc.execute-api.us-east-1.amazonaws.com
Content-Type: application/json
headerName: headerValue

{
    "a": 1
}
```

此請求會產生下列回應承載，內含從後端 Lambda 函數所傳回的輸出，而 `input` 已設定為 Lambda 函數的 `event` 參數。

```
{
    "message": "Hello me!",
    "input": {
        "resource": "/{proxy+}",
        "path": "/hello/world",
        "httpMethod": "POST",
        "headers": {
            "Accept": "*/*",
            "Accept-Encoding": "gzip, deflate",
            "cache-control": "no-cache",
            "CloudFront-Forwarded-Proto": "https",
            "CloudFront-Is-Desktop-Viewer": "true",
            "CloudFront-Is-Mobile-Viewer": "false",
            "CloudFront-Is-SmartTV-Viewer": "false",
            "CloudFront-Is-Tablet-Viewer": "false",
            "CloudFront-Viewer-Country": "US",
            "Content-Type": "application/json",
            "headerName": "headerValue",
            "Host": "gy415nuibc.execute-api.us-east-1.amazonaws.com",
            "Postman-Token": "9f583ef0-ed83-4a38-aef3-eb9ce3f7a57f",
            "User-Agent": "PostmanRuntime/2.4.5",
            "Via": "1.1 d98420743a69852491bbdea73f7680bd.cloudfront.net (CloudFront)",
            "X-Amz-Cf-Id": "pn-PWIJc6thYnZm5P0NMgOUgll1DYtl0gdeJky8tqsg8iS_sgsKD1A==",
            "X-Forwarded-For": "54.240.196.186, 54.182.214.83",
            "X-Forwarded-Port": "443",
            "X-Forwarded-Proto": "https"
        }
    }
}
```

```
},
"multiValueHeaders": {
  'Accept': [
    "*/*"
  ],
  'Accept-Encoding': [
    "gzip, deflate"
  ],
  'cache-control': [
    "no-cache"
  ],
  'CloudFront-Forwarded-Proto': [
    "https"
  ],
  'CloudFront-Is-Desktop-Viewer': [
    "true"
  ],
  'CloudFront-Is-Mobile-Viewer': [
    "false"
  ],
  'CloudFront-Is-SmartTV-Viewer': [
    "false"
  ],
  'CloudFront-Is-Tablet-Viewer': [
    "false"
  ],
  'CloudFront-Viewer-Country': [
    "US"
  ],
  '': [
    ""
  ],
  'Content-Type': [
    "application/json"
  ],
  'headerName': [
    "headerValue"
  ],
  'Host': [
    "gy415nuibc.execute-api.us-east-1.amazonaws.com"
  ],
  'Postman-Token': [
    "9f583ef0-ed83-4a38-aef3-eb9ce3f7a57f"
  ],
  'User-Agent': [
    "PostmanRuntime/2.4.5"
  ],
  'Via': [
    "1.1 d98420743a69852491bbdea73f7680bd.cloudfront.net (CloudFront)"
  ],
  'X-Amz-Cf-Id': [
    "pn-PWIJc6thYnZm5P0NMgOUglL1DYtl0gdeJky8tqsg8iS_sgsKD1A=="
  ],
  'X-Forwarded-For': [
    "54.240.196.186, 54.182.214.83"
  ],
  'X-Forwarded-Port': [
    "443"
  ],
  'X-Forwarded-Proto': [
    "https"
  ]
},
"queryStringParameters": {
  "name": "me",
  "multivalueName": "me"
}
```

```
},
"multiValueQueryStringParameters": {
    "name": [
        "me"
    ],
    "multivalueName": [
        "you",
        "me"
    ]
},
"pathParameters": {
    "proxy": "hello/world"
},
"stageVariables": {
    "stageVariableName": "stageVariableValue"
},
"requestContext": {
    "accountId": "12345678912",
    "resourceId": "roq9wj",
    "stage": "testStage",
    "requestId": "deef4878-7910-11e6-8f14-25afc3e9ae33",
    "identity": {
        "cognitoIdentityPoolId": null,
        "accountId": null,
        "cognitoIdentityId": null,
        "caller": null,
        "apiKey": null,
        "sourceIp": "192.168.196.186",
        "cognitoAuthenticationType": null,
        "cognitoAuthenticationProvider": null,
        "userArn": null,
        "userAgent": "PostmanRuntime/2.4.5",
        "user": null
    },
    "resourcePath": "/{proxy+}",
    "httpMethod": "POST",
    "apiId": "gy415nuibc"
},
"body": "{\r\n\t\"a\": 1\r\n}",
"isBase64Encoded": false
}
}
```

在後端 Lambda 函數的輸入中，`requestContext` 物件是鍵值對的映射。在每一對中，鍵是 [\\$context \(p. 274\)](#) 變數屬性的名稱，而值是該屬性的值。API Gateway 可能會將新鍵加入映射中。

根據啟用的功能，`requestContext` 映射可能會因 API 而不同。例如，在上述範例中，未指定任何授權類型，因此沒有 `$context.authorizer.*` 或 `$context.identity.*` 屬性存在。指定授權類型時，這會導致 API Gateway 將授權的使用者資訊傳遞至 `requestContext.identity` 物件中的整合端點，如下所示：

- 當授權類型為 AWS\_IAM 時，授權的使用者資訊包含 `$context.identity.*` 屬性。
- 當授權類型為 COGNITO\_USER\_POOLS (Amazon Cognito 授權方) 時，授權的使用者資訊包含 `$context.identity.cognito*` 和 `$context.authorizer.claims.*` 屬性。
- 當授權類型為 CUSTOM (Lambda 授權方) 時，授權的使用者資訊包含 `$context.authorizer.principalId` 和其他適用的 `$context.authorizer.*` 屬性。

以下範例顯示在授權類型設定為 AWS\_IAM 時，傳送到 Lambda 代理整合端點的 `requestContext`。

```
{
    ...
}
```

```
"requestContext": {  
    "requestTime": "20/Feb/2018:22:48:57 +0000",  
    "path": "/test/",  
    "accountId": "123456789012",  
    "protocol": "HTTP/1.1",  
    "resourceId": "yx5mhem7ye",  
    "stage": "test",  
    "requestTimeEpoch": 1519166937665,  
    "requestId": "3c3ecbaa-1690-11e8-ae31-8f39f1d24af",  
    "identity": {  
        "cognitoIdentityPoolId": null,  
        "accountId": "123456789012",  
        "cognitoIdentityId": null,  
        "caller": "AIDAJ.....4HCKVJZG",  
        "sourceIp": "51.240.196.104",  
        "accessKey": "IAM_user_access_key",  
        "cognitoAuthenticationType": null,  
        "cognitoAuthenticationProvider": null,  
        "userArn": "arn:aws:iam::123456789012:user/alice",  
        "userAgent": "PostmanRuntime/7.1.1",  
        "user": "AIDAJ.....4HCKVJZG"  
    },  
    "resourcePath": "/",  
    "httpMethod": "GET",  
    "apiId": "qr2gd9cfmf"  
,  
...  
}  
}
```

### 代理整合之 Lambda 函數的輸出格式

在 Lambda 代理整合中，API Gateway 需要後端 Lambda 函數根據下列 JSON 格式傳回輸出：

```
{  
    "isBase64Encoded": true/false,  
    "statusCode": httpStatusCode,  
    "headers": { "headerName": "HeaderValue", ... },  
    "multiValueHeaders": { "headerName": [ "HeaderValue", "HeaderValue2", ... ], ... },  
    "body": "..."  
}
```

在輸出中：

- 如果不再傳回額外的回應標頭，則可以不指定 `headers` 和 `multiValueHeaders` 鍵。
- `headers` 鍵只能包含單一值標頭。
- `multiValueHeaders` 鍵可以包含多值標頭以及單一值標頭。您可以使用 `multiValueHeaders` 鍵來指定所有額外標頭，包括任何單一值標頭。
- 如果您同時指定 `headers` 和 `multiValueHeaders` 的值，API Gateway 會將它們合併成一個清單。如果在兩者指定了相同的鍵值對，則只有 `multiValueHeaders` 中的值會出現在合併清單。

若要啟用 CORS 進行 Lambda 代理整合，您必須將 `Access-Control-Allow-Origin:domain-name` 新增至輸出 `headers`。`domain-name` 可以是 \*，其代表任何網域名稱。輸出 `body` 會封送處理至前端，以做為方法回應承載。如果 `body` 是二進位 Blob，您可以將其編碼為 Base64 編碼字串，方法是將 `isBase64Encoded` 設定為 `true`，並將 / 設定為 Binary Media Type (二進位媒體類型)。否則，您可以將它設定為 `false`，或保留未指定。

#### Note

如需啟用二進位支援的詳細資訊，請參閱[使用 API Gateway 主控台啟用二進位支援 \(p. 285\)](#)。

如果函數輸出的格式不同，則 API Gateway 會傳回 502 Bad Gateway 錯誤回應。

若要在 Node.js 的 Lambda 函數中傳回回應，您可以使用以下命令：

- 若要傳回成功結果，請呼叫 `callback(null, {"statusCode": 200, "body": "results"})`。
- 若要擲回例外狀況，請呼叫 `callback(new Error('internal server error'))`。
- 針對用戶端錯誤（例如，如果遺失必要參數），您可以呼叫 `callback(null, {"statusCode": 400, "body": "Missing parameters of ..."})` 傳回錯誤，而不擲回例外狀況。

在 Node.js 8.10 的 Lambda `async` 函數中，同等語法應為：

- 若要傳回成功結果，請呼叫 `return {"statusCode": 200, "body": "results"}`。
- 若要擲回例外狀況，請呼叫 `throw new Error("internal server error")`。
- 針對用戶端錯誤（例如，如果遺失必要參數），您可以呼叫 `return {"statusCode": 400, "body": "Missing parameters of ..."}` 傳回錯誤，而不擲回例外狀況。

## 在 API Gateway 中設定 Lambda 自訂整合

為了示範如何設定 Lambda 自訂整合，我們建立 API Gateway API 來公開 GET /greeting?greeter={name} 方法以呼叫 Lambda 函數。如果 "Hello, {name}!" 參數值是非空白字串，則該函數會以 greeter 訊息進行回應。如果 "Hello, World!" 值是空白字串，則它會傳回 greeter 訊息。如果未在傳入請求中設定 greeter 參數，則該函數會傳回錯誤訊息 "Missing the required greeter parameter."。我們將函數命名為 HelloWorld。

針對參考，Lambda 函數的 Node.js 版本顯示如下：

```
exports.handler = function(event, context, callback) {
    var res ={
        "statusCode": 200,
        "headers": {
            "Content-Type": "*/*"
        }
    };
    if (event.greeter==null) {
        callback(new Error('Missing the required greeter parameter.'));
    } else if (event.greeter === "") {
        res.body = "Hello, World";
        callback(null, res);
    } else {
        res.body = "Hello, " + event.greeter +"!";
        callback(null, res);
    }
};
```

您可以在 Lambda 主控台中或使用 AWS CLI 來建立它。在本節中，我們使用下列 ARN 來參考此函數：

```
arn:aws:lambda:us-east-1:123456789012:function>HelloWorld
```

在後端設定 Lambda 函數，即可繼續設定 API。

### 使用 AWS CLI 設定 Lambda 自訂整合

1. 呼叫 `create-rest-api` 命令來建立 API：

```
aws apigateway create-rest-api --name 'HelloWorld (AWS CLI)' --region us-west-2
```

請記下回應中所產生 API 的 id 值 (te6si5ach7) :

```
{  
    "name": "HelloWorld (AWS CLI)",  
    "id": "te6si5ach7",  
    "createdDate": 1508461860  
}
```

在本節中，您需要 API id。

2. 呼叫 `get-resources` 命令來取得根資源 id :

```
aws apigateway get-resources --rest-api-id te6si5ach7 --region us-west-2
```

成功回應如下：

```
{  
    "items": [  
        {  
            "path": "/",  
            "id": "krznpq9xpg"  
        }  
    ]  
}
```

記下根資源 id 值 (krznpq9xpg)。下一個步驟和之後的步驟都需要它。

3. 呼叫 `create-resource` 來建立 API Gateway /greeting 資源：

```
aws apigateway create-resource --rest-api-id te6si5ach7 \  
    --region us-west-2 \  
    --parent-id krznpq9xpg \  
    --path-part greeting
```

成功回應類似如下：

```
{  
    "path": "/greeting",  
    "pathPart": "greeting",  
    "id": "2jf6xt",  
    "parentId": "krznpq9xpg"  
}
```

記下所產生 greeting 資源的 id 值 (2jf6xt)。在下一個步驟中，您需要它才能在 /greeting 資源上建立方法。

4. 呼叫 `put-method` 來建立 API 方法請求 GET /greeting?greeter={name}：

```
aws apigateway put-method --rest-api-id te6si5ach7 \  
    --region us-west-2 \  
    --resource-id 2jf6xt \  
    --http-method GET \  
    --authorization-type "NONE" \  
    --request-parameters method.request.querystring.greeter=false
```

成功回應類似如下：

```
{
```

```
"apiKeyRequired": false,  
"httpMethod": "GET",  
"authorizationType": "NONE",  
"requestParameters": {  
    "method.request.querystring.greeter": false  
}
```

這個 API 方法可讓用戶端從後端的 Lambda 函數接收問候語。greeter 是選用參數，因為後端應該處理匿名發起人或自我識別發起人。

5. 呼叫 `put-method-response` 來設定方法請求 `GET /greeting?greeter={name}` 的 `200 OK` 回應：

```
aws apigateway put-method-response \  
    --region us-west-2  
    --rest-api-id te6si5ach7 \  
    --resource-id 2jf6xt  
    --http-method GET \  
    --status-code 200
```

6. 呼叫 `put-integration` 來設定具有 Lambda 函數(即 `HelloWorld`)之 `GET /greeting?greeter={name}` 方法的整合。如果提供 "`Hello, {name}!`" 參數，則此函數會以 `greeter` 訊息來回應請求，如果未設定查詢字串參數，則會以 "`Hello, World!`" 來回應請求。

```
aws apigateway put-integration \  
    --region us-west-2  
    --rest-api-id vaz7da96z6 \  
    --resource-id 2jf6xt \  
    --http-method GET \  
    --type AWS \  
    --integration-http-method POST \  
    --uri arn:aws:apigateway:us-east-1:lambda:path/2015-03-31/functions/  
arn:aws:lambda:us-east-1:123456789012:function:HelloWorld/invocations \  
    --request-templates file://path/to/integration-request-template.json \  
    --credentials arn:aws:iam::123456789012:role/apigAwsProxyRole
```

在這裡，`request-template` 參數值 `file://path/to/integration-request-template.json` 指向 JSON 檔案(在 `integration-request-template.json` 目錄中名為 `path/to`)，其中包含金鑰值對應做為 JSON 物件。金鑰是請求承載的媒體類型，而值是所指定內容類型內文的對應範本。在此範例中，JSON 檔案包含下列 JSON 物件：

```
{"application/json": "{\"greeter\": \"$input.params('greeter')\"}"}
```

這裡提供的對應範本會將 `greeter` 查詢字串參數翻譯為 JSON 承載的 `greeter` 屬性。這是必要的，因為 Lambda 函數中的 Lambda 函數輸入必須在內文中表示。您可以使用對應的 JSON 字串(例如，"`{\"greeter\": \"john\"}`")做為 `request-template` 命令的 `put-integration` 輸入值。不過，使用檔案輸入可避免將 JSON 物件字串化所需的困難(有時不可能)引號逸出。

#### Important

針對 Lambda 整合，您必須根據[指定進行函數呼叫的 Lambda 服務動作](#)，為整合請求使用 POST 的 HTTP 方法。`uri` 參數是函數呼叫動作的 ARN。  
成功輸出與下列輸出類似：

```
{  
    "passthroughBehavior": "WHEN_NO_MATCH",  
    "cacheKeyParameters": [],  
    "uri": "arn:aws:apigateway:us-east-1:lambda:path/2015-03-31/functions/  
arn:aws:lambda:us-east-1:123456789012:function:HelloWorld/invocations",
```

```
"httpMethod": "POST",
"requestTemplates": {
    "application/json": "{\"greeter\": \"$input.params('greeter')\"}"
},
"cacheNamespace": "krznpq9xpg",
"credentials": "arn:aws:iam::123456789012:role/apigAwsProxyRole",
"type": "AWS"
}
```

IAM 角色 apigAwsProxyRole 的政策必須允許 apigateway 服務呼叫 Lambda 函數。您可以呼叫 `add-permission` 命令來新增資源型許可，而不提供 `credentials` 的 IAM 角色。這是 API Gateway 主控台如何新增這些許可。

7. 呼叫 `put-integration-response` 來設定整合回應，以將 Lambda 函數輸出當做 200 OK 方法回應傳遞至用戶端。

```
aws apigateway put-integration-response \
--region us-west-2 \
--rest-api-id te6si5ach7 \
--resource-id 2jf6xt \
--http-method GET \
--status-code 200 \
--selection-pattern ""
```

透過將選取模式設定為空白字串，200 OK 回應是預設值。

成功回應應該類似如下：

```
{
    "selectionPattern": "",
    "statusCode": "200"
}
```

8. 呼叫 `create-deployment` 來將 API 部署至 test 階段：

```
aws apigateway create-deployment --rest-api-id te6si5ach7 --stage-name test
```

9. 在終端機中使用下列 cURL 命令，以測試 API：

```
curl -X GET 'https://te6si5ach7.execute-api.us-west-2.amazonaws.com/test/greeting?
greeter=me' \
-H 'authorization: AWS4-HMAC-SHA256 Credential={access_key}/20171020/us-
west-2/execute-api/aws4_request, SignedHeaders=content-type;host;x-amz-date,
Signature=f327...5751'
```

## 設定後端 Lambda 函數的非同步呼叫

在 Lambda 代理整合和 Lambda 非代理 (自訂) 整合中，預設為同步叫用後端 Lambda 函數。這是大部分 REST API 操作所需的行為。不過，有些應用程式需要工作以非同步方式 (以批次操作方式或長時間延遲操作方式) 執行，通常是由個別後端元件執行。在此情況下，會非同步叫用後端 Lambda 函數，且前端 REST API 方法不會傳回結果。

您可以將 Lambda 函數設定為以非同步方式叫用，方法是將 'Event' 指定為 [Lambda 呼叫類型](#)。此作法如下所示：

在 API Gateway 主控台中設定 Lambda 同步呼叫

1. 在 Integration Request (整合請求) 中，新增 X-Amz-Invocation-Type 標頭。

- 在 Method Request (方法請求) 中，新增 X-Amz-Invocation-Type 標頭，並將其映射至 Integration Request (整合請求) 中的 InvocationType 標頭，靜態值為 'Event' 或標頭映射表達式為 method.request.header.InvocationType。針對後者，用戶端必須在對 API 方法提出請求時包含 InvocationType:Event 標頭。

## 在 API Gateway 中處理 Lambda 錯誤

對於 Lambda 自訂整合，您必須將整合回應中 Lambda 傳回的錯誤，對應到您用戶端的標準 HTTP 錯誤回應。否則，Lambda 錯誤預設傳回為 200 OK 回應，此結果對您的 API 使用者不是直覺式。

Lambda 會傳回兩種錯誤：標準錯誤和自訂錯誤。在您的 API 中，您必須以不同的方式處理它們。

使用 Lambda 代理整合，Lambda 必須傳回格式如下的輸出：

```
{  
    "isBase64Encoded": "boolean",  
    "statusCode": "number",  
    "headers": { ... },  
    "body": "JSON string"  
}
```

在這個輸出中，statusCode 的用戶端錯誤通常是 4XX，伺服器錯誤通常是 5XX。API Gateway 處理這些錯誤的方法是根據指定的 statusCode，將 Lambda 錯誤對應到 HTTP 錯誤回應。如需 API Gateway 將錯誤類型（例如 InvalidParameterException）當做回應的一部分傳送到用戶端，Lambda 函數在 headers 屬性中必須包含標頭（例如 "X-Amzn-ErrorType": "InvalidParameterException"）。

### 主題

- [在 API Gateway 中處理標準 Lambda 錯誤 \(p. 221\)](#)
- [在 API Gateway 中處理自訂 Lambda 錯誤 \(p. 223\)](#)

## 在 API Gateway 中處理標準 Lambda 錯誤

標準 AWS Lambda 錯誤的格式如下：

```
{  
    "errorMessage": "<replaceable>string</replaceable>",  
    "errorType": "<replaceable>string</replaceable>",  
    "stackTrace": [  
        "<replaceable>string</replaceable>",  
        ...  
    ]  
}
```

這裡的 errorMessage 是錯誤的字串表達式。errorType 是語言相關錯誤或例外狀況類型。stackTrace 是字串表達式清單，顯示造成錯誤出現的堆疊追蹤。

例如，請考慮下列 JavaScript (Node.js) Lambda 函數。

```
exports.handler = function(event, context, callback) {  
    callback(new Error("Malformed input ..."));  
};
```

此函數會傳回下列標準 Lambda 錯誤，包含 Malformed input ... 做為錯誤訊息：

```
{  
    "errorMessage": "Malformed input ...",  
    "errorType": "Error",  
}
```

```
    "stackTrace": [
        "exports.handler (/var/task/index.js:3:14)"
    ]
}
```

同樣地，請考慮下列 Python Lambda 函數，它會引發具有相同 `Malformed input ...` 錯誤訊息的 `Exception`。

```
def lambda_handler(event, context):
    raise Exception('Malformed input ...')
```

此函數會傳回下列標準 Lambda 錯誤：

```
{
    "stackTrace": [
        [
            "/var/task/lambda_function.py",
            3,
            "lambda_handler",
            "raise Exception('Malformed input ...')"
        ]
    ],
    "errorType": "Exception",
    "errorMessage": "Malformed input ..."
}
```

請注意，`errorType` 和 `stackTrace` 屬性值為語言相關。標準錯誤也適用於為 `Error` 物件延伸或 `Exception` 類別子類別的任何錯誤物件。

若要將標準 Lambda 錯誤對應到方法回應，您必須先決定指定 Lambda 錯誤的 HTTP 狀態碼。然後，您要在與指定的 HTTP 狀態碼相關聯之 `IntegrationResponse` 的 `selectionPattern` 屬性上，設定常規表達式模式。在 API Gateway 主控台中，這個 `selectionPattern` 在 Integration Response (整合回應) 組態編輯器中表示為 Lambda Error Regex (&LAM; 錯誤 Regex)。

**Note**

API Gateway 使用 Java 模式 regex 進行回應對應。如需詳細資訊，請參閱 Oracle 文件中的 [模式一節](#)。

例如，若要使用 AWS CLI 設定新的 `selectionPattern` 表達式，請呼叫下列 `put-integration-response` 命令：

```
aws apigateway put-integration-response --rest-api-id z0vprf0mdh --resource-id x3o5ih --
http-method GET --status-code 400 --selection-pattern "Invalid*" --region us-west-2
```

請務必也要在[方法回應 \(p. 195\)](#)上設定對應的錯誤碼 (400)。否則，API Gateway 在執行時間會擲出無效的組態錯誤回應。

**Note**

在執行時間，API Gateway 會根據 `selectionPattern` 屬性規則運算式的模式比對 Lambda 錯誤的 `errorMessage`。若出現相符項目，API Gateway 會傳回 Lambda 錯誤做為對應 HTTP 狀態碼的 HTTP 回應。如果沒有相符項目，API Gateway 會傳回錯誤做為預設回應；或，如未設定預設回應，則擲出無效的組態例外狀況。

將指定回應數量的 `selectionPattern` 值設為 `.*`，將這個回應重新設定為預設回應。這是因為這種選取模式會比對所有的錯誤訊息，包括 Null，即任何未指定的錯誤訊息。產生的對應會覆寫預設的對應。

若要使用 AWS CLI 更新現有的 `selectionPattern` 值，請呼叫 `update-integration-response` 操作，將 / `selectionPattern` 路徑值替換成 `Malformed*` 模式的指定 regex 表達式。

若要使用 API Gateway 主控台設定 `selectionPattern` 表達式，當設定或更新指定的 HTTP 狀態碼的整合回應時，請在 Lambda Error Regex (Lambda 錯誤 Regex) 文字方塊中輸入表達式。

### 在 API Gateway 中處理自訂 Lambda 錯誤

AWS Lambda 可讓您傳回自訂錯誤物件當做 JSON 字串，不是上節所述的標準錯誤。此錯誤可以是任何有效的 JSON 物件。例如，下列 JavaScript (Node.js) Lambda 函數會傳回自訂錯誤：

```
exports.handler = (event, context, callback) => {
  ...
  // Error caught here:
  var myErrorObj = {
    errorType : "InternalServerError",
    httpStatus : 500,
    requestId : context.awsRequestId,
    trace : {
      "function": "abc()",
      "line": 123,
      "file": "abc.js"
    }
  }
  callback(JSON.stringify(myErrorObj));
};
```

您必須先將 `myErrorObj` 物件轉換為 JSON 字串，再呼叫 `callback` 結束函數。否則，`myErrorObj` 會傳回為 "[object Object]" 的字串。當您的 API 方法與前述 Lambda 函數整合時，API Gateway 會收到包含下列承載的整合回應：

```
{  
  "errorMessage": "{\"errorType\":\"InternalServerError\",\"httpStatus\":500,\"requestId\":\"e5849002-39a0-11e7-a419-5bb5807c9fb2\",\"trace\":{\"function\":\"abc()\",\"line\":123,\"file\":\"abc.js\"}}"  
}
```

就和任何整合回應一樣，您可以將這個錯誤回應依現狀傳遞到方法回應。或者，您可以讓內文對應範本將承載轉換成不同的格式。例如，500 狀態碼的方法回應請考慮下列內文對應範本：

```
{  
  errorMessage: $input.path('$.errorMessage');  
}
```

此範本會將包含自訂錯誤 JSON 字串的整合回應內文，轉譯成下列方法回應內文。這個方法回應內文包含自訂錯誤 JSON 物件：

```
{  
  "errorMessage" : {  
    errorType : "InternalServerError",  
    httpStatus : 500,  
    requestId : context.awsRequestId,  
    trace : {  
      "function": "abc()",  
      "line": 123,  
      "file": "abc.js"  
    }  
  }  
};
```

視您的 API 請求而定，您可能需要將部分或全部自訂錯誤屬性傳送為方法回應標頭參數。您可以將自訂錯誤對應從整合回應內文套用到方法回應標頭，完成此作業。

例如，下列 OpenAPI 延伸分別定義從

`errorMessage.errorType`、`errorMessage.httpStatus`、`errorMessage.trace.function` 和 `errorMessage.trace` 屬性到 `error_type`、`error_status`、`error_trace_function` 和 `error_trace` 標頭的對應。

```
"x-amazon-apigateway-integration": {
    "responses": {
        "default": {
            "statusCode": "200",
            "responseParameters": {
                "method.response.header.error_trace_function": "integration.response.body.errorMessage.trace.function",
                "method.response.header.error_status": "integration.response.body.errorMessage.httpStatus",
                "method.response.header.error_type": "integration.response.body.errorMessage.errorType",
                "method.response.header.error_trace": "integration.response.body.errorMessage.trace"
            },
            ...
        }
    }
}
```

在執行時間，API Gateway 會在執行標頭對應時還原序列化 `integration.response.body` 參數。不過，此還原序列化僅適用於 Lambda 自訂錯誤回應的內文到標頭對應，不適用於使用 `$input.body` 的內文到內文對應。如果在方法回應中宣告 `error_status`、`error_trace`、`error_trace_function` 和 `error_type` 標頭，使用這些自訂錯誤內文到標頭的對應，用戶端會收到屬於方法回應的下列標頭。

```
"error_status": "500",
"error_trace": "{\"function\": \"abc()\", \"line\": 123, \"file\": \"abc.js\"}",
"error_trace_function": "abc()",
"error_type": "InternalServerError"
```

整合回應內文的 `errorMessage.trace` 屬性是複雜屬性。它會對應到 `error_trace` 標頭做為 JSON 字串。

## 在 API Gateway 中設定 HTTP 整合

您可以使用 HTTP 代理整合或 HTTP 自訂整合，將 API 方法與 HTTP 端點進行整合。

API Gateway 支援以下端點連接埠：80、443 及 1024-65535。

使用代理整合時，設定非常簡單。如果您不在乎內容編碼或快取，您只需要根據後端需求來設定 HTTP 方法與 HTTP 端點 URI。

使用自訂整合時，需要進行更多的設定。除了代理整合設定步驟，您還需要指定傳入請求資料如何對應到整合請求，以及產生的整合回應資料如何對應到方法回應。

### 主題

- [在 API Gateway 中設定 HTTP 代理整合 \(p. 224\)](#)
- [在 API Gateway 中設定 HTTP 自訂整合 \(p. 228\)](#)

## 在 API Gateway 中設定 HTTP 代理整合

若要設定代理資源與 HTTP 代理整合類型搭配，請建立 API 資源與 Greedy 路徑參數（例如 `/parent/{proxy+}`）搭配，並將此資源與 ANY 方法上的 HTTP 後端端點（例如 `https://petstore-demo-endpoint.execute-api.com/petstore/{proxy}`）整合。Greedy 路徑參數必須位於資源路徑結尾。

如同非代理資源，您可以使用 API Gateway 主控台、匯入 OpenAPI 定義檔或直接呼叫 API Gateway REST API，來設定具有 HTTP 代理整合的代理資源。如需使用 API Gateway 主控台來設定代理資源與 HTTP 整合搭配的詳細說明，請參閱[教學：建置具有 HTTP 代理整合的 API \(p. 47\)](#)。

下列 OpenAPI 定義檔顯示一個其代理資源與 PetStore 網站整合的 API 範例。

#### OpenAPI 3.0

```
{
  "openapi": "3.0.0",
  "info": {
    "version": "2016-09-12T23:19:28Z",
    "title": "PetStoreWithProxyResource"
  },
  "paths": {
    "/{proxy+}": {
      "x-amazon-apigateway-any-method": {
        "parameters": [
          {
            "name": "proxy",
            "in": "path",
            "required": true,
            "schema": {
              "type": "string"
            }
          }
        ],
        "responses": {},
        "x-amazon-apigateway-integration": {
          "responses": {
            "default": {
              "statusCode": "200"
            }
          },
          "requestParameters": {
            "integration.request.path.proxy": "method.request.path.proxy"
          },
          "uri": "http://petstore-demo-endpoint.execute-api.com/petstore/{proxy}",
          "passthroughBehavior": "when_no_match",
          "httpMethod": "ANY",
          "cacheNamespace": "rbftud",
          "cacheKeyParameters": [
            "method.request.path.proxy"
          ],
          "type": "http_proxy"
        }
      }
    }
  },
  "servers": [
    {
      "url": "https://4z9giyi2c1.execute-api.us-east-1.amazonaws.com/{basePath}",
      "variables": {
        "basePath": {
          "default": "/test"
        }
      }
    }
  ]
}
```

#### OpenAPI 2.0

```
{
```

```
"swagger": "2.0",
"info": {
    "version": "2016-09-12T23:19:28Z",
    "title": "PetStoreWithProxyResource"
},
"host": "4z9giyi2c1.execute-api.us-east-1.amazonaws.com",
"basePath": "/test",
"schemes": [
    "https"
],
"paths": {
    "/{proxy+}": {
        "x-amazon-apigateway-any-method": {
            "produces": [
                "application/json"
            ],
            "parameters": [
                {
                    "name": "proxy",
                    "in": "path",
                    "required": true,
                    "type": "string"
                }
            ],
            "responses": {}
        },
        "x-amazon-apigateway-integration": {
            "responses": {
                "default": {
                    "statusCode": "200"
                }
            },
            "requestParameters": {
                "integration.request.path.proxy": "method.request.path.proxy"
            },
            "uri": "http://petstore-demo-endpoint.execute-api.com/petstore/{proxy}",
            "passthroughBehavior": "when_no_match",
            "httpMethod": "ANY",
            "cacheNamespace": "rbftud",
            "cacheKeyParameters": [
                "method.request.path.proxy"
            ],
            "type": "http_proxy"
        }
    }
}
}
```

在此範例中，快取金鑰是在代理資源的 `method.request.path.proxy` 路徑參數上宣告。當您使用 API Gateway 主控台建立 API 時，這是預設設定。API 的基底路徑 (對應到階段的 `/test`) 會對應到網站的 PetStore 頁面 (`/petstore`)。單一整合請求使用 API 的 Greedy 路徑變數與 catch-all ANY 方法來鏡射整個 PetStore 網站。下列範例說明此鏡射。

- 將 ANY 設定為 GET 並將 `{proxy+}` 設定為 pets

從前端啟動方法請求：

```
GET https://4z9giyi2c1.execute-api.us-west-2.amazonaws.com/test/pets HTTP/1.1
```

將整合請求傳送到後端：

```
GET http://petstore-demo-endpoint.execute-api.com/petstore/pets HTTP/1.1
```

ANY 方法與代理資源的執行時間執行個體皆有效。呼叫會傳回 200 OK 回應與含有從後端傳回之第一批寵物的承載。

- 將 ANY 設定為 GET 並將 {proxy+} 設定為 **pets?type=dog**

```
GET https://4z9giyi2c1.execute-api.us-west-2.amazonaws.com/test/pets?type=dog HTTP/1.1
```

將整合請求傳送到後端：

```
GET http://petstore-demo-endpoint.execute-api.com/petstore/pets?type=dog HTTP/1.1
```

ANY 方法與代理資源的執行時間執行個體皆有效。呼叫會傳回 200 OK 回應與含有從後端傳回之第一批指定狗類的承載。

- 將 ANY 設定為 GET 並將 {proxy+} 設定為 **pets/{petId}**

從前端啟動方法請求：

```
GET https://4z9giyi2c1.execute-api.us-west-2.amazonaws.com/test/pets/1 HTTP/1.1
```

將整合請求傳送到後端：

```
GET http://petstore-demo-endpoint.execute-api.com/petstore/pets/1 HTTP/1.1
```

ANY 方法與代理資源的執行時間執行個體皆有效。呼叫會傳回 200 OK 回應與含有從後端傳回之指定寵物的承載。

- 將 ANY 設定為 POST 並將 {proxy+} 設定為 **pets**

從前端啟動方法請求：

```
POST https://4z9giyi2c1.execute-api.us-west-2.amazonaws.com/test/pets HTTP/1.1
Content-Type: application/json
Content-Length: ...

{
    "type" : "dog",
    "price" : 1001.00
}
```

將整合請求傳送到後端：

```
POST http://petstore-demo-endpoint.execute-api.com/petstore/pets HTTP/1.1
Content-Type: application/json
Content-Length: ...

{
    "type" : "dog",
    "price" : 1001.00
}
```

ANY 方法與代理資源的執行時間執行個體皆有效。呼叫會傳回 200 OK 回應與含有從後端傳回之新建立寵物的承載。

- 將 ANY 設定為 GET 並將 {proxy+} 設定為 **pets/cat**

從前端啟動方法請求：

```
GET https://4z9giyi2c1.execute-api.us-west-2.amazonaws.com/test/pets/cat
```

將整合請求傳送到後端：

```
GET http://petstore-demo-endpoint.execute-api.com/petstore/pets/cat
```

代理資源路徑的執行時間執行個體未對應到後端端點，且產生的請求無效。因此會傳回 400 Bad Request 回應與下列錯誤訊息。

```
{
  "errors": [
    {
      "key": "Pet2.type",
      "message": "Missing required field"
    },
    {
      "key": "Pet2.price",
      "message": "Missing required field"
    }
  ]
}
```

- 將 ANY 設定為 GET 並將 {proxy+} 設定為 null

從前端啟動方法請求：

```
GET https://4z9giyi2c1.execute-api.us-west-2.amazonaws.com/test
```

將整合請求傳送到後端：

```
GET http://petstore-demo-endpoint.execute-api.com/petstore/pets
```

目標資源是代理資源的父系，但未在該資源的 API 中定義 ANY 方法的執行時間執行個體。因此，這個 GET 請求會傳回 403 Forbidden 回應與 API Gateway 所傳回的 Missing Authentication Token 錯誤訊息。如果 API 在父資源 (ANY) 上公開 GET 或 / 方法，呼叫會傳回 404 Not Found 回應與後端所傳回的 Cannot GET /petstore 訊息。

針對任何用戶端請求，如果目標端點 URL 無效，或 HTTP 動詞有效但不受支援，則後端會傳回 404 Not Found 回應。針對不支援的 HTTP 方法，則會傳回 403 Forbidden 回應。

## 在 API Gateway 中設定 HTTP 自訂整合

有了 HTTP 自訂整合，您可以更精細地控制要在 API 方法與 API 整合之間傳遞哪些資料，以及如何傳遞這些資料。您會透過資料對應來執行此作業。

在方法請求設定過程中，您可以在 Method 資源上設定 requestParameters 屬性。這會宣告哪些從用戶端佈建的方法請求參數，是要對應到整合請求參數或適用的內文屬性，再發送到後端。然後，在整合請求設定過程中，您可以在對應的 Integration 資源上設定 requestParameters 屬性，來指定參數對參數的對應。您也可以設定 requestTemplates 屬性，針對每個支援的內容類型各指定一個映射範本。對應範本會將方法請求參數或內文對應到整合請求內文。

同樣地，在方法回應設定過程中，您可以在 MethodResponse 資源上設定 responseParameters 屬性。這會宣告哪些要發送到用戶端的方法回應參數，是要從後端傳回之整合回應參數或特定適用的內文屬性對應而

來。然後，在整合回應設定過程中，您可以在對應的 [IntegrationResponse](#) 資源上設定 `responseParameters` 屬性，來指定參數對參數的對應。您也可以設定 `responseTemplates` 映射，針對每個支援的內容類型各指定一個映射範本。對應範本會將整合回應參數或整合回應內文屬性對應到方法回應內文。

如需設定映射範本的詳細資訊，請參閱[設定資料映射 \(p. 242\)](#)。

## 設定 API Gateway 私有整合

API Gateway 私有整合可以輕鬆地公開受 Amazon VPC 保護的 HTTP/HTTPS 資源，以供 VPC 外部的用戶端存取。若要將私有 VPC 資源的存取擴展到 VPC 邊界外部，您可以建立具有私有整合的 API 來進行開放存取或控制存取。做法是使用 IAM 許可、Lambda 授權方或 Amazon Cognito 使用者集區。

私有整合使用 API Gateway 資源 `VpcLink` 來封裝 API Gateway 與目標 VPC 資源之間的連線。您是 VPC 資源擁有者，則負責在 VPC 中建立網路負載平衡器，並新增 VPC 資源做為網路負載平衡器接聽程式的目標。如果您是 API 開發人員，若要設定具有私有整合的 API，則負責建立目標設為所指定網路負載平衡器的 `VpcLink`，然後將 `VpcLink` 視為有效的整合端點。

使用 API Gateway 私有整合，您可以啟用存取 VPC 內的 HTTP/HTTPS 資源，而不需要深入了解私有網路組態或技術特定設備。

### 主題

- [設定 API Gateway 私有整合的網路負載平衡器 \(p. 229\)](#)
- [授予建立 VPC 連結的許可 \(p. 229\)](#)
- [使用 API Gateway 主控台設定具有私有整合的 API Gateway API \(p. 230\)](#)
- [使用 AWS CLI 設定具有私有整合的 API Gateway API \(p. 230\)](#)
- [使用 OpenAPI 設定具有私有整合的 API \(p. 233\)](#)

## 設定 API Gateway 私有整合的網路負載平衡器

下列程序概述使用 Amazon EC2 主控台設定 API Gateway 私有整合之網路負載平衡器的步驟，並提供每個步驟之詳細說明的參考。

### 使用 API Gateway 主控台建立私有整合的網路負載平衡器

1. 在 <https://console.aws.amazon.com/ec2/> 登入 Amazon EC2 主控台，並在導覽列上選擇區域；例如，us-east-1。
2. 在 Amazon EC2 執行個體上設定 Web 伺服器。如需範例設定，請參閱[在 Amazon Linux 上安裝 LAMP Web Server](#)。
3. 建立網路負載平衡器、註冊具有目標群組的 EC2 執行個體，並將目標群組新增至網路負載平衡器接聽程式。如需詳細資訊，請參閱[開始使用網路負載平衡器](#)中的指示。

建立網路負載平衡器之後，請注意其 ARN。您需要它才能在 API Gateway 中建立 VPC 連結，以整合 API 與受網路負載平衡器保護的 VPC 資源。

## 授予建立 VPC 連結的許可

若要讓您或您帳戶中的使用者建立和維護 VPC 連結，您或使用者必須有權建立、刪除和檢視 VPC 端點服務組態、變更 VPC 端點服務許可，以及檢查負載平衡器。若要授予這類許可，請使用下列步驟。

### 授予建立和更新 `VpcLink` 的許可

1. 建立與下列類似的 IAM 政策：

```
{  
    "Version": "2012-10-17",  
    "Statement": [
```

```
{  
    "Effect": "Allow",  
    "Action": [  
        "ec2:CreateVpcEndpointServiceConfiguration",  
        "ec2>DeleteVpcEndpointServiceConfigurations",  
        "ec2:DescribeVpcEndpointServiceConfigurations",  
        "ec2:ModifyVpcEndpointServicePermissions"  
    ],  
    "Resource": "*"  
},  
{  
    "Effect": "Allow",  
    "Action": [  
        "elasticloadbalancing:DescribeLoadBalancers"  
    ],  
    "Resource": "*"  
}  
]  
}
```

2. 建立或選擇 IAM 角色，並將先前的政策連接至角色。
3. 將 IAM 角色指派給您或您帳戶中建立 VPC 連結的使用者。

## 使用 API Gateway 主控台設定具有私有整合的 API Gateway API

如需使用 API Gateway 主控台設定具有私有整合之 API 的說明，請參閱[教學：建置具有 API Gateway 私有整合的 API \(p. 80\)](#)。

## 使用 AWS CLI 設定具有私有整合的 API Gateway API

建立具有私有整合的 API 前，您必須設定 VPC 資源、使用 VPC 來源建立網路負載平衡器並將其設定為目標。如果需求皆不符合，請按照[設定 API Gateway 私有整合的網路負載平衡器 \(p. 229\)](#)來安裝 VPC 資源、建立 NLB、將 VPC 資源設定為網路負載平衡器的目標。

為了能夠建立和管理 VpcLink，您還必須設定適當的許可。如需詳細資訊，請參閱[授予建立 VPC 連結的許可 \(p. 229\)](#)。

### Note

您只需要在 API 中建立 VpcLink 的許可。您不需要使用 VpcLink 的許可。

建立網路負載平衡器之後，請注意其 ARN。您需要它才能建立私有整合的 VPC 連結。

## 使用 AWS CLI 設定具有私有整合的 API

1. 建立目標設為所指定網路負載平衡器的 VpcLink。

在本討論中，假設網路負載平衡器的 ARN 是 `arn:aws:elasticloadbalancing:us-east-1:123456789012:loadbalancer/net/my-vpc-link-test-nlb/1f8df693cd094a72`。

```
aws apigateway create-vpc-link \  
    --name my-test-vpc-link \  
    --target-arns arn:aws:elasticloadbalancing:us-east-1:123456789012:loadbalancer/net/  
    my-vpc-link-test-nlb/1f8df693cd094a72 \  
    --endpoint-url https://apigateway.us-east-1.amazonaws.com \  
    --region us-east-1
```

如果 AWS 組態使用 `us-east-1` 做為預設區域，則您可以略過先前輸入中的 `endpoint-url` 和 `region` 參數。

上述命令會立即傳回下列回應，並認可接收請求，以及顯示所建立 PENDING 的 VpcLink 狀態。

```
{  
    "status": "PENDING",  
    "targetArns": [  
        "arn:aws:elasticloadbalancing:us-east-1:123456789012:loadbalancer/net/my-vpc-link-test-nlb/1f8df693cd094a72"  
    ],  
    "id": "gim7c3",  
    "name": "my-test-vpc-link"  
}
```

需要 2-4 分鐘的時間，API Gateway 才能完成建立 VpcLink。操作順利完成時，status 是 AVAILABLE。驗證方式是呼叫下列 CLI 命令：

```
aws apigateway get-vpc-link --vpc-link-id gim7c3
```

如果操作失敗，您會取得 FAILED 狀態，以及包含錯誤訊息的 statusMessage。例如，如果您嘗試建立具有已與 VPC 端點建立關聯之網路負載平衡器的 VpcLink，則會在 statusMessage 屬性上收到下列訊息：

```
"NLB is already associated with another VPC Endpoint Service"
```

只有在順利建立 VpcLink 且準備好建立 API 並透過 VpcLink 將其與 VPC 資源整合之後。

請記下新建立 id 的 VpcLink 值 (先前輸出中的 gim7c3)。您需要它才能設定私有整合。

2. 建立 API Gateway RestApi 資源來設定 API：

```
aws apigateway create-rest-api --name 'My VPC Link Test'
```

我們已捨棄輸入參數 endpoint-url 和 region 來使用 AWS 組態中所指定的預設區域。

請記下所傳回結果中 RestApi 的 id 值。在此範例中，假設它是 6j4m3244we。您需要此值才能對 API 執行進一步操作，包含設定方法和整合。

基於說明，我們將建立根資源 (GET) 上只有 / 方法的 API，並整合此方法與 VpcLink。

3. 設定 GET / 方法。先取得根資源的識別符 (/)：

```
aws apigateway get-resources --rest-api-id 6j4m3244we
```

在輸出中，記下 id 路徑的 / 值。在此範例中，假設它是 skpp60rab7。

設定 API 方法 GET / 的方法請求：

```
aws apigateway put-method \  
    --rest-api-id 6j4m3244we \  
    --resource-id skpp60rab7 \  
    --http-method GET \  
    --authorization-type "NONE"
```

若要使用 IAM 許可、Lambda 授權方或 Amazon Cognito 使用者集區來驗證發起人，請將 authorization-type 分別設定為 AWS\_IAM、CUSTOM 或 COGNITO\_USER\_POOLS。

如果您未搭配使用代理整合與 VpcLink，則也必須至少設定 200 狀態碼的方法回應。我們將在這裡使用代理整合。

4. 設定 HTTP\_PROXY 類型的私有整合，並呼叫 put-integration 命令，如下所示：

```
aws apigateway put-integration \
--rest-api-id 6j4m3244we \
--resource-id skpp60rab7 \
--uri 'http://myApi.example.com' \
--http-method GET \
--type HTTP_PROXY \
--integration-http-method GET \
--connection-type VPC_LINK \
--connection-id gim7c3
```

針對私有整合，您必須將 `connection-type` 設定為 `VPC_LINK`，並將 `connection-id` 設定為您 `VpcLink` 的識別符或參考您 `VpcLink` ID 的階段變數。`uri` 參數不是用於將請求路由至端點，但用於設定 `Host` 標頭以及進行憑證驗證。

如果成功，此命令會傳回下列輸出：

```
{
  "passthroughBehavior": "WHEN_NO_MATCH",
  "timeoutInMillis": 29000,
  "connectionId": "gim7c3",
  "uri": "http://myApi.example.com",
  "connectionType": "VPC_LINK",
  "httpMethod": "GET",
  "cacheNamespace": "skpp60rab7",
  "type": "HTTP_PROXY",
  "cacheKeyParameters": []
}
```

使用階段變數，您可以在建立整合時設定 `connectionId` 屬性：

```
aws apigateway put-integration \
--rest-api-id 6j4m3244we \
--resource-id skpp60rab7 \
--uri 'http://myApi.example.com' \
--http-method GET \
--type HTTP_PROXY \
--integration-http-method GET \
--connection-type VPC_LINK \
--connection-id "\${stageVariables.vpcLinkId}"
```

請務必使用雙引號括住階段變數表達式 (`\${stageVariables.vpcLinkId}`) 並逸出 `$` 字元。

或者，您可以更新整合，以使用階段變數來重設 `connectionId` 值：

```
aws apigateway update-integration \
--rest-api-id 6j4m3244we \
--resource-id skpp60rab7 \
--http-method GET \
--patch-operations '[{"op":"replace","path":"/connectionId","value":"\${stageVariables.vpcLinkId}"}]'
```

請務必使用字串化 JSON 清單做為 `patch-operations` 參數值。

使用階段變數來設定 `connectionId` 值的優點是重設階段變數重，以具有與不同 `VpcLink` 整合的相同 API。這適用於將 API 切換至不同 VPC 連結，以遷移至不同的網路負載平衡器或不同的 VPC。

因為我們使用私有代理整合，所以 API 現在已準備好進行部署和測試執行。使用非代理整合，您也必須設定方法回應和整合回應，就像設定 [具有 HTTP 自訂整合的 API \(p. 53\)](#) 一樣。

5. 若要測試 API，請部署 API。如果您已使用階段變數做為 VpcLink ID 的預留位置，則這是必要的。若要使用階段變數來部署 API，請呼叫 `create-deployment` 命令，如下所示：

```
aws apigateway create-deployment \
--rest-api-id 6j4m3244we \
--stage-name test \
--variables vpcLinkId=gim7c3
```

若要更新具有不同 VpcLink ID 的階段變數（例如，`ASF9d7`），請呼叫 `update-stage` 命令：

```
aws apigateway update-stage \
--rest-api-id 6j4m3244we \
--stage-name test \
--patch-operations op=replace,path='/variables/vpcLinkId',value='ASF9d7'
```

若要測試 API，請使用下列 cURL 命令進行呼叫：

```
curl -X GET https://6j4m3244we.execute-api.us-east-1.amazonaws.com/test
```

或者，您可以在 Web 瀏覽器中輸入 API 的 invoke-URL 來檢視結果。

當您硬式編碼具有 `connection-id` ID 常值的 `VpcLink` 屬性時，也可以呼叫 `test-invoke-method` 來測試在部署 API 之前呼叫 API。

## 使用 OpenAPI 設定具有私有整合的 API

您可以匯入 API OpenAPI 檔案來設定具有私有整合的 API。這些設定類似具有 HTTP 整合之 API 的 OpenAPI 定義，但例外如下：

- 您必須將 `connectionType` 明確地設定為 `VPC_LINK`。
- 您必須將 `connectionId` 明確地設定為 `VpcLink` 的 ID 或參考 `VpcLink` ID 的階段變數。
- 私有整合中的 `uri` 參數指向 VPC 中的 HTTP/HTTPS 端點，但改為用來設定整合請求的 `Host` 標頭。
- 具有 VPC 中 HTTPS 端點之私有整合中的 `uri` 參數用來針對 VPC 端點上所安裝憑證中的網域名稱來驗證所指出的網域名稱。

您可以使用階段變數來參考 `VpcLink` ID。或者，您可以將 ID 值直接指派給 `connectionId`。

下列 JSON 格式化 OpenAPI 檔案會顯示 API 的範例，具有階段變數 (`#{stageVariables.vpcLinkId}`) 所參考的 VPC 連結：

OpenAPI 2.0

```
{
  "swagger": "2.0",
  "info": {
    "version": "2017-11-17T04:40:23Z",
    "title": "MyApiWithVpcLink"
  },
  "host": "p3wocvip9a.execute-api.us-west-2.amazonaws.com",
  "basePath": "/test",
  "schemes": [
    "https"
  ],
  "paths": {
    "/": {
      "get": {
        "responses": {
          "200": {
            "description": "Success"
          }
        }
      }
    }
  }
}
```

```
"produces": [
    "application/json"
],
"responses": {
    "200": {
        "description": "200 response",
        "schema": {
            "$ref": "#/definitions/Empty"
        }
    }
},
"x-amazon-apigateway-integration": {
    "responses": {
        "default": {
            "statusCode": "200"
        }
    },
    "uri": "http://myApi.example.com",
    "passthroughBehavior": "when_no_match",
    "connectionType": "VPC_LINK",
    "connectionId": "${stageVariables.vpcLinkId}",
    "httpMethod": "GET",
    "type": "http_proxy"
}
},
"definitions": {
    "Empty": {
        "type": "object",
        "title": "Empty Schema"
    }
}
}
```

## 在 API Gateway 中設定模擬整合

Amazon API Gateway 支援 API 方法的模擬整合。此功能可讓 API 開發人員直接從 API Gateway 產生 API 回應，而不需要整合後端。您是 API 開發人員，在專案開發完成之前，可以使用此功能來解鎖需要使用 API 的相依團隊。您也可以使用此功能來佈建您 API 的登錄頁面，以提供 API 的概觀，並導覽至 API。如需這類登錄頁面範例，請參閱範例 API 之根資源上 GET 方法的整合請求和回應，如[教學課程：匯入範例來建立 REST API \(p. 40\)](#)中所討論。

您是 API 開發人員，可決定 API Gateway 回應如何模擬整合請求。因此，您設定方法的整合請求和整合回應，以將回應與特定狀態碼建立關聯。若要讓具有模擬整合的方法傳回 200 回應，請設定整合請求內文映射範本來傳回下列內容。

```
{"statusCode": 200}
```

設定 200 整合回應，以具有下列內文映射範本，例如：

```
{
    "statusCode": 200,
    "message": "Go ahead without me."
}
```

例如，同樣地，若要讓方法傳回 500 錯誤回應，請設定整合請求內文映射範本來傳回下列內容。

```
{"statusCode": 500}
```

例如，設定具有下列映射範本的 500 整合回應：

```
{  
    "statusCode": 500,  
    "message": "The invoked method is not supported on the API resource."  
}
```

或者，您可以讓模擬整合方法傳回預設整合回應，而不需要定義整合請求映射範本。預設整合回應具有未定義的 HTTP status regex (HTTP 狀態 regex)。請確定已設定適當的傳遞行為。

使用整合請求映射範本，您可以注入應用程式邏輯，以根據特定條件來決定要傳回的模擬整合回應。例如，您可以在傳入請求上使用 scope 查詢參數，決定是要傳回成功回應還是錯誤回應：

```
{  
    #if( $input.params('scope') == "internal" )  
        "statusCode": 200  
    #else  
        "statusCode": 500  
    #end  
}
```

因此，模擬整合方法可讓內部呼叫通過，同時拒絕具有錯誤回應之其他類型的呼叫。

在這個部分中，我們會說明如何使用 API Gateway 主控台 來啟用 API 方法的模擬整合。

#### 主題

- [使用 API Gateway 主控台啟用模擬整合 \(p. 235\)](#)

## 使用 API Gateway 主控台啟用模擬整合

您必須在 API Gateway 中有可用的方法。請遵循「[教學：建置具有 HTTP 非代理整合的 API \(p. 52\)](#)」中的說明進行。

1. 選擇 API 資源，並建立方法。在方法 Setup (設定) 窗格的 Integration type (整合類型) 中選擇 Mock (模擬)，然後選擇 Save (儲存)。
2. 從 Method Execution (方法執行) 中，選擇 Method Request (方法請求)。展開 URL Query String Parameters (URL 檢索字串參數)。選擇 Add query string (新增檢索字串)，以新增 scope 檢索參數。這會判斷發起人是否為內部。
3. 從 Method Execution (方法執行) 中，選擇 Integration Request (整合請求)。展開 Body Mapping Templates (內文映射範本)。選擇或新增 application/json 映射範本。在範本編輯器中，輸入下列內容：

```
{  
    #if( $input.params('scope') == "internal" )  
        "statusCode": 200  
    #else  
        "statusCode": 500  
    #end  
}
```

選擇 Save (儲存)。

4. 從 Method Execution (方法執行) 中，選擇 Integration Response (整合回應)。展開 200 回應，然後展開 Body Mapping Templates (內文映射範本) 區段。選擇或新增應用程式/JSON 映射範本，並在範本編輯器中輸入下列回應內文映射範本。

```
{
```

```
        "statusCode": 200,
        "message": "Go ahead without me"
    }
```

選擇 Save (儲存)。

5. 滾動至 Integration Response (整合回應)。選擇 Add integration response (新增整合回應) 來新增 500 回應。在 HTTP status regex (HTTP 狀態 regex) 中，輸入 `5\d{2}`。展開 Body Mapping Templates (內文映射範本)，然後選擇 Add mapping template (新增映射範本)。針對 Content-Type 輸入 `application/json`，然後選擇核取記號圖示來儲存設定。在範本編輯器中，輸入下列整合回應內文映射範本：

```
{
    "statusCode": 500,
    "message": "The invoked method is not supported on the API resource."
}
```

選擇 Save (儲存)。

6. 在 Method Execution (方法執行) 中，選擇 Test (測試)。執行以下操作：

- a. 在 scope (範圍) 下，輸入 `internal`。選擇 Test. 測試結果顯示：

```
Request: /?scope=internal
Status: 200
Latency: 26 ms
Response Body

{
    "statusCode": 200,
    "message": "Go ahead without me"
}

Response Headers

{"Content-Type": "application/json"}
```

- b. 在 `public` 下輸入 `scope`，或將其空白。選擇 Test (測試)。測試結果顯示：

```
Request: /
Status: 500
Latency: 16 ms
Response Body

{
    "statusCode": 500,
    "message": "The invoked method is not supported on the API resource."
}

Response Headers

{"Content-Type": "application/json"}
```

您也可以在模擬整合回應中傳回標頭，方法是先將標頭新增至方法回應，然後在整合回應中設定標頭映射。事實上，這是 API Gateway 主控台如何透過傳回 CORS 必要標頭來啟用 CORS 支援。

## 設定閘道回應以自訂錯誤回應

如果 API Gateway 無法處理傳入請求，則它會傳送錯誤回應給用戶端，但不會將請求轉送到整合後端。錯誤回應預設包含一則簡短的描述性錯誤訊息。例如，如果您嘗試在未定義的 API 資源上呼叫操作，您會收到 { "message": "Missing Authentication Token" } 訊息的錯誤回應。如果您是初次使用 API Gateway，您可能會發現很難了解實際發生錯誤的原因。

針對一些錯誤回應，API Gateway 允許 API 開發人員進行自訂以傳回不同格式的回應。在 Missing Authentication Token 範例中，您可以將含有可能原因的提示新增至原始回應承載，如下列範例所示：{"message": "Missing Authentication Token", "hint": "The HTTP method or resources may not be supported."}。

當您的 API 連接外部 Exchange 與 AWS 雲端時，您可以對整合請求或整合回應使用 VTL 對應範本，將承載從某個格式對應到另一個格式。不過，VTL 對應範本僅適用於具有成功回應的有效請求。針對無效的請求，API Gateway 會完全略過整合並傳回錯誤回應。您必須使用自訂，將錯誤回應轉譯成 Exchange 相容的格式。在本例中，會在只支援簡單變數替換的非 VTL 對應範本中轉譯自訂。

將 API Gateway 所產生的錯誤回應一般化為 API Gateway 所產生的任何回應，此操作稱為「閘道回應」。這可區分 API Gateway 所產生的回應與整合回應。閘道回應對應範本可存取 \$context 變數值與 \$stageVariables 屬性值，以及 method.request.*param-position*.*param-name* 格式的方法請求參數。如需 \$context 變數的詳細資訊，請參閱「[適用於資料模型、授權方、映射範本及 CloudWatch 存取記錄的 \\$context 變數 \(p. 274\)](#)」。如需 \$stageVariables 的詳細資訊，請參閱 [\\$stageVariables \(p. 281\)](#)。如需方法請求參數的詳細資訊，請參閱 [映射範本可存取的請求參數 \(p. 270\)](#)。

### 主題

- [API Gateway 中的閘道回應 \(p. 237\)](#)
- [閘道回應類型 \(p. 237\)](#)
- [使用 API Gateway 主控台設定閘道回應 \(p. 239\)](#)
- [使用 API Gateway REST API 設定閘道回應 \(p. 241\)](#)
- [設定 OpenAPI 中的閘道回應自訂 \(p. 241\)](#)

## API Gateway 中的閘道回應

閘道回應是以 API Gateway 定義的回應類型識別。回應包含 HTTP 狀態碼、一組由參數對應指定的額外標頭，以及非 VTL 對應範本所產生的承載。

在 API Gateway REST API 中，閘道回應是以 [GatewayResponse](#) 表示。在 OpenAPI 中，GatewayResponse 執行個體是以 [x-amazon-apigateway-gateway-responses.gatewayResponse \(p. 540\)](#) 延伸來加以說明。

若要啟用閘道回應，您可以在 API 層級設定[支援回應類型 \(p. 237\)](#)的閘道回應。每當 API Gateway 傳回該類型的回應時，就會套用閘道回應中定義的標頭對應與承載對應範本，將對應的結果傳回給 API 發起人。

在下一節中，我們將示範如何使用 API Gateway 主控台與 API Gateway REST API 來設定閘道回應。

## 閘道回應類型

API Gateway 公開下列 API 開發人員可自訂的閘道回應。

閘道回應類型	預設狀態碼	描述
DEFAULT_4XX	Null	狀態碼為 4XX 之未指定回應類型的預設閘道回應。變更此後援閘道回應的狀態碼會將所有其他

閘道回應類型	預設狀態碼	描述
		4XX 回應的狀態碼變更為新的值。將此狀態碼重設為 Null 會將所有其他 4XX 回應的狀態碼還原成其原始值。
DEFAULT_5XX	Null	狀態碼為 5XX 之未指定回應類型的預設閘道回應。變更此後援閘道回應的狀態碼會將所有其他 5XX 回應的狀態碼變更為新的值。將此狀態碼重設為 Null 會將所有其他 5XX 回應的狀態碼還原成其原始值。
ACCESS_DENIED	403	授權失敗的閘道回應；例如，自訂或 Amazon Cognito 授權方拒絕存取時。如果未指定回應類型，此回應預設為 DEFAULT_4XX 類型。
API_CONFIGURATION_ERROR	500	無效 API 組態的閘道回應，包括提交的端點地址無效、制定二進位支援時二進位資料的 Base64 編碼失敗，或整合回應對應不符合任何範本且未設定預設範本。如果未指定回應類型，此回應預設為 DEFAULT_5XX 類型。
AUTHORIZER_CONFIGURATION_ERROR		無法連線到自訂或 Amazon Cognito 授權方的閘道回應。如果未指定回應類型，此回應預設為 DEFAULT_5XX 類型。
AUTHORIZER_FAILURE	500	自訂或 Amazon Cognito 授權方無法驗證發起人時的閘道回應。如果未指定回應類型，此回應預設為 DEFAULT_5XX 類型。
BAD_REQUEST_PARAMETERS	400	根據啟用的請求驗證程式無法驗證請求參數時的閘道回應。如果未指定回應類型，此回應預設為 DEFAULT_4XX 類型。
BAD_REQUEST_BODY	400	根據啟用的請求驗證程式無法驗證請求內文時的閘道回應。如果未指定回應類型，此回應預設為 DEFAULT_4XX 類型。
EXPIRED_TOKEN	403	AWS 身分驗證字符過期錯誤的閘道回應。如果未指定回應類型，此回應預設為 DEFAULT_4XX 類型。
INTEGRATION_FAILURE	504	整合失敗錯誤的閘道回應。如果未指定回應類型，此回應預設為 DEFAULT_5XX 類型。

閘道回應類型	預設狀態碼	描述
INTEGRATION_TIMEOUT	504	整合逾時錯誤的閘道回應。如果未指定回應類型，此回應預設為 DEFAULT_5XX 類型。
INVALID_API_KEY	403	針對需要 API 金鑰之方法提交的 API 金鑰無效的閘道回應。如果未指定回應類型，此回應預設為 DEFAULT_4XX 類型。
INVALID_SIGNATURE	403	AWS 簽章無效錯誤的閘道回應。如果未指定回應類型，此回應預設為 DEFAULT_4XX 類型。
MISSING_AUTHENTICATION_TOKEN	403	遺漏身分驗證字符錯誤的閘道回應，例如用戶端嘗試呼叫不支援的 API 方法或資源時。如果未指定回應類型，此回應預設為 DEFAULT_4XX 類型。
QUOTA_EXCEEDED	429	用量計劃超額錯誤的閘道回應。如果未指定回應類型，此回應預設為 DEFAULT_4XX 類型。
REQUEST_TOO_LARGE	413	請求太大錯誤的閘道回應。如果未指定回應類型，此回應預設為 DEFAULT_4XX 類型。
RESOURCE_NOT_FOUND	404	API 請求通過身分驗證與授權之後 (API 金鑰身分驗證與授權除外)，API Gateway 找不到指定資源時的閘道回應。如果未指定回應類型，此回應預設為 DEFAULT_4XX 類型。
THROTTLED	429	超過用量計劃、方法、階段或帳戶層級調節限制時的閘道回應。如果未指定回應類型，此回應預設為 DEFAULT_4XX 類型。
UNAUTHORIZED	401	自訂或 Amazon Cognito 授權方無法驗證發起人時的閘道回應。
UNSUPPORTED_MEDIA_TYPE	415	承載屬於不支援的媒體類型時的閘道回應 (如果啟用嚴格的傳遞行為)。如果未指定回應類型，此回應預設為 DEFAULT_4XX 類型。

## 使用 API Gateway 主控台設定閘道回應

### 使用 API Gateway 主控台自訂閘道回應

1. 登入 API Gateway 主控台。
2. 選擇您現有的 API 或建立新的 API。
3. 在主導覽窗格中展開 API，然後選擇 API 下的 Gateway Responses (閘道回應)。

4. 在 Gateway Responses (閘道回應) 窗格中，選擇回應類型。在此演練中，我們以 Missing Authentication Token (403) (遺漏身分驗證字符 (403)) 為例。
5. 您可以變更 API Gateway 所產生的 Status Code (狀態碼)，傳回符合您的 API 需求的不同狀態碼。在此範例中，自訂會將狀態碼從預設值 (403) 變更為 404，因為當用戶端呼叫可視為找不到的不支援或無效資源時會出現此錯誤訊息。
6. 若要傳回自訂標頭，請在 Response Headers (回應標頭) 下，選擇 Add Header (新增標頭)。為了方便說明，我們新增下列自訂標頭：

```
Access-Control-Allow-Origin:'a.b.c'  
x-request-id:method.request.header.x-amzn-RequestId  
x-request-path:method.request.path.petId  
x-request-query:method.request.querystring.q
```

在上述標頭對應中，靜態網域名稱 ('a.b.c') 會對應到 Allow-Control-Allow-Origin 標頭以允許 API 的 CORS 存取；x-amzn-RequestId 的輸入請求標頭會對應到回應中的 request-id；傳入請求的 petId 路徑變數會對應到回應中的 request-path 標頭；而原始請求的 q 查詢參數會對應到回應的 request-query 標頭。

7. 在 Body Mapping Templates (內文映射範本) 下，針對 Content Type (內容類型) 保留 application/json，然後在 Body Mapping Template (內文映射範本) 編輯器中輸入下列內文映射範本：

```
{  
    "message": "$context.error.messageString",  
    "type": "$context.error.responseType",  
    "statusCode": "'404'",  
    "stage": "$context.stage",  
    "resourcePath": "$context.resourcePath",  
    "stageVariables.a": "$stageVariables.a"  
}
```

此範例示範如何將 \$context 與 \$stageVariables 屬性對應到閘道回應內文的屬性。

8. 選擇 Save (儲存)。
9. 將 API 部署到新的或現有的階段。
10. 透過呼叫下列 CURL 命令進行測試，並假設對應 API 方法的 Invoke URL (呼叫 URL) 為 https://o81lxisefl.execute-api.us-east-1.amazonaws.com/custErr/pets/{petId}：

```
curl -v -H 'x-amzn-RequestId:123344566' https://o81lxisefl.execute-api.us-east-1.amazonaws.com/custErr/pets/5?type?q=1
```

由於額外查詢字串參數 q=1 與 API 不相容，因此會傳回錯誤來觸發指定的閘道回應。您應該取得類似如下的閘道回應：

```
> GET /custErr/pets/5?q=1 HTTP/1.1  
Host: o81lxisefl.execute-api.us-east-1.amazonaws.com  
User-Agent: curl/7.51.0  
Accept: */*  
  
HTTP/1.1 404 Not Found  
Content-Type: application/json  
Content-Length: 334  
Connection: keep-alive  
Date: Tue, 02 May 2017 03:15:47 GMT  
x-amzn-RequestId: a2be05a4-2ee5-11e7-bbf2-df131ec50ae6  
Access-Control-Allow-Origin: a.b.c  
x-amzn-ErrorType: MissingAuthenticationTokenException  
header-1: static  
x-request-query: 1  
x-request-path: 5
```

```
X-Cache: Error from cloudfront
Via: 1.1 441811a054e8d05b893175754efd0c3.cloudfront.net (CloudFront)
X-Amz-Cf-Id: nNDR-fX4csbRoAgtQJ16u0rTDz9FZWT-Mk93KgoxnfzDlTUh3flmzA==

{
    "message": "Missing Authentication Token",
    "type": MISSING_AUTHENTICATION_TOKEN,
    "statusCode": '404',
    "stage": custErr,
    "resourcePath": /pets/{petId},
    "stageVariables.a": a
}
```

上述範例假設 API 後端是[寵物店](#)，而且 API 已定義階段變數 a。

## 使用 API Gateway REST API 設定閘道回應

使用 API Gateway REST API 自訂閘道回應之前，您必須已建立 API 並已取得其識別符。若要擷取 API 識別符，您可以遵循[「restapi:gateway-responses」連結關係並查看結果](#)。

### 使用 API Gateway REST API 自訂閘道回應

1. 若要覆寫整個 [GatewayResponse](#) 執行個體，請呼叫 [gatewayresponse:put](#) 動作、在 URL 路徑參數中指定所需的 [responseType](#)，並在請求承載中提供 [statusCode](#)、[responseParameters](#) 與 [responseTemplates](#) 映射。
2. 若要更新 [GatewayResponse](#) 執行個體的一部分，請呼叫 [gatewayresponse:update](#) 動作、在 URL 路徑參數中指定所需的 [responseType](#)，並在請求承載中提供所需的個別 [GatewayResponse](#) 屬性，例如 [responseParameters](#) 或 [responseTemplates](#) 映射。

## 設定 OpenAPI 中的閘道回應自訂

您可以在 API 根層級使用 `x-amazon-apigateway-gateway-responses` 延伸，來自訂 OpenAPI 中的閘道回應。下列 OpenAPI 定義示範如何自訂 [MISSING\\_AUTHENTICATION\\_TOKEN](#) 類型的 [GatewayResponse](#)。

```
"x-amazon-apigateway-gateway-responses": {
    "MISSING_AUTHENTICATION_TOKEN": {
        "statusCode": 404,
        "responseParameters": {
            "gatewayresponse.header.x-request-path": "method.input.params.petId",
            "gatewayresponse.header.x-request-query": "method.input.params.q",
            "gatewayresponse.header.Access-Control-Allow-Origin": "'a.b.c'",
            "gatewayresponse.header.x-request-header": "method.input.params.Accept"
        },
        "responseTemplates": {
            "application/json": "{\n                \"message\": \"$context.error.messageString\",\n                \"type\": \"$context.error.responseType\",\n                \"stage\": \"$context.stage\",\n                \"resourcePath\": \"$context.resourcePath\",\n                \"stageVariables.a\": \"$stageVariables.a\"\n            }"
        }
    }
}
```

在此範例中，自訂會將狀態碼從預設值 (403) 變更為 404。它也會將 application/json 媒體類型的四個標頭參數與一個內文對應範本新增至閘道回應。

## 設定 API Gateway 請求和回應資料對應

### 主題

- [使用 API Gateway 主控台設定請求與回應資料映射 \(p. 242\)](#)
- [針對請求與回應對應建立模型與對應範本 \(p. 244\)](#)
- [Amazon API Gateway API 請求和回應資料對應參考 \(p. 270\)](#)
- [API Gateway 映射範本和存取記錄變數參考 \(p. 274\)](#)

## 使用 API Gateway 主控台設定請求與回應資料映射

若要使用 API Gateway 主控台來定義 API 的整合請求/回應，請遵循這些說明進行。

### Note

這些說明假設您已完成[使用 API Gateway 主控台設定 API 整合請求 \(p. 203\)](#)中的步驟。

1. 選取 Resources (資源) 窗格中的方法時，在 Method Execution (方法執行) 窗格中選擇 Integration Request (整合請求)。
2. 針對 HTTP 代理或 AWS 服務代理，若要將整合請求中定義的路徑參數、查詢字串參數或標頭參數，與 HTTP 代理或 AWS 服務代理之方法請求中的對應路徑參數、查詢字串參數或標頭參數建立關聯，請執行下列操作：
  - a. 分別選擇 URL Path Parameters (URL 路徑參數)、URL Query String Parameters (URL 查詢字串參數) 或 HTTP Headers (HTTP 標頭) 旁邊的箭頭，然後分別選擇 Add path (新增路徑)、Add query string (新增查詢字串) 或 Add header (新增標頭)。
  - b. 針對 Name (名稱)，輸入 HTTP 代理或 AWS 服務代理中路徑參數、查詢字串參數或標頭參數的名稱。
  - c. 針對 Mapped from (映射來源)，輸入路徑參數、查詢字串參數或標頭參數的映射值。請使用下列其中一個格式：
    - `method.request.path.parameter-name`，代表 Method Request (方法請求) 頁面中所定義之名為 `parameter-name` 的路徑參數。
    - `method.request.querystring.parameter-name`，代表 Method Request (方法請求) 頁面中所定義之名為 `parameter-name` 的查詢字串參數。
    - `method.request.multivaluequerystring.parameter-name`，代表 Method Request (方法請求) 頁面中所定義之名為 `parameter-name` 的多值查詢字串參數。
    - `method.request.header.parameter-name`，代表 Method Request (方法請求) 頁面中所定義之名為 `parameter-name` 的標頭參數。
- 或者，您可以將字串常值 (以一對單引號括住) 設定為整合標頭。
  - a. `method.request.multivalueheader.parameter-name`，代表 Method Request (方法請求) 頁面中所定義之名為 `parameter-name` 的多值標頭參數。
  - d. 選擇 Create (建立)。(若要刪除路徑參數、查詢字串參數或標頭參數，請選擇您要刪除之參數旁邊的 Cancel (取消) 或 Remove (移除))。
3. 在 Body Mapping Templates (內文對應範本) 區域中，選擇 Request body passthrough (請求內文傳遞) 選項，設定如何將未對應內容類型的方法請求內文透過整合請求傳遞，而不轉換成 Lambda 函數、HTTP 代理或 AWS 服務代理。共有三個選項：
  - 當方法請求內容類型不符合與下一個步驟中定義之映射範本相關聯的任何內容類型時，如果您想要讓方法請求內文透過整合請求傳遞到後端而不進行轉換，請選擇 When no template matches the request Content-Type header (沒有範本符合請求內容類型標頭時)。

#### Note

呼叫 API Gateway API 時，您可以透過在整合資源上將 WHEN\_NO\_MATCH 設定為 passthroughBehavior 屬性值，來選擇這個選項。

- 當整合請求中未定義任何映射範本時，如果您想要讓方法請求內文透過整合請求傳遞到後端而不進行轉換，請選擇 When there are no templates defined (recommended) (未定義範本時 (建議))。如果選取此選項時已定義範本，未映射內容類型的方法請求會遭到拒絕，並顯示 HTTP 415 Unsupported Media Type (不支援的媒體類型) 回應。

#### Note

呼叫 API Gateway API 時，您可以透過在整合資源上將 WHEN\_NO\_TEMPLATE 設定為 passthroughBehavior 屬性值，來選擇這個選項。

- 當方法請求內容類型不符合與整合請求中定義之映射範本相關聯的任何內容類型時，或是當整合請求中未定義任何映射範本時，如果您不想要讓方法請求傳遞，請選擇 Never (永不)。未映射內容類型的方法請求會遭到拒絕，並顯示 HTTP 415 Unsupported Media Type (不支援的媒體類型) 回應。

#### Note

呼叫 API Gateway API 時，您可以透過在整合資源上將 NEVER 設定為 passthroughBehavior 屬性值，來選擇這個選項。

- 如需整合傳遞行為的詳細資訊，請參閱「[整合傳遞行為 \(p. 273\)](#)」。
4. 若要定義傳入請求的映射範本，請在 Content-Type 下，選擇 Add mapping template (新增映射範本)。在輸入文字方塊中輸入內容類型 (例如 `application/json`)，再選擇核取記號圖示以儲存輸入。然後，手動輸入映射範本，或選擇 Generate template (產生範本) 從模型範本中建立一個範本。如需詳細資訊，請參閱[針對請求與回應對應建立模型與對應範本 \(p. 244\)](#)。
  5. 您可以將後端整合回應映射到傳回給呼叫應用程式的 API 方法回應。這包括將從後端可用回應標頭中選取的回應標頭傳回給用戶端，並將後端回應承載的資料格式轉換成 API 指定的格式。您可以從 Method Execution (方法執行) 頁面中，設定 Method Response (方法回應) 與 Integration Response (整合回應)，來指定這類對應。
    - a. 在 Method Execution (方法執行) 窗格中，選擇 Integration Response (整合回應)。選擇 200 旁邊的箭頭，指定從方法設定 200 HTTP 回應碼，或選擇 Add integration response (新增整合回應)，指定從方法設定任何其他 HTTP 回應狀態碼。
    - b. 針對 Lambda error regex (&LAM; 錯誤 regex) (適用於 Lambda 函數) 或 HTTP status regex (HTTP 狀態 regex) (適用於 HTTP 代理或 AWS 服務代理)，輸入規則表達式來指定要對應到此輸出對應的 Lambda 函數錯誤字串 (適用於 Lambda 函數) 或 HTTP 回應狀態碼 (適用於 HTTP 代理或 AWS 服務代理)。例如，若要將所有 2xx HTTP 回應狀態碼從 HTTP 代理對應到此輸出對應，請針對 HTTP status regex (HTTP 狀態 regex) 輸入「`2\\d{2}`」。若要從 Lambda 函數傳回包含「無效請求」的錯誤訊息給 400 Bad Request 回應，請輸入「`.*Invalid request.*`」做為 Lambda error regex (Lambda 錯誤 regex) 表達式。另一方面，若要對 Lambda 中未對應的錯誤訊息傳回 400 Bad Request，請在 Lambda error regex (Lambda 錯誤 regex) 中輸入「`(\\n|.)*`」。最後一個規則表達式可作為 API 的預設錯誤回應使用。

#### Note

API Gateway 使用 Java 模式 regex 進行回應映射。如需詳細資訊，請參閱 Oracle 文件中的[模式](#)一節。

錯誤模式會與 Lambda 回應中 errorMessage 屬性的整個字串進行比對，該屬性在 Node.js 中是由 `callback(errorMessage)` 填入，在 Java 中是由 `throw new MyException(errorMessage)` 填入。此外，逸出字元在套用規則表達式之前為未逸出。

如果您使用 '+' 做為選取模式來篩選回應，請注意它可能不會比對含有新行 ('\n') 字元的回應。

- c. 如果啟用，請針對 Method response status (方法回應狀態)，選擇您在 Method Response (方法回應) 頁面中定義的 HTTP 回應狀態碼。
- d. 針對您在 Method Response (方法回應) 頁面中為 HTTP 回應狀態碼定義之每個標頭的 Header Mappings (標頭映射)，選擇 Edit (編輯) 以指定映射值。對於 Mapping value (對應值)，請使用以下其中一個格式：
  - `integration.response.multivalueheaders.header-name`，其中 `header-name` 是後端多值回應標頭的名稱。

例如，若要傳回後端回應的 Date 標頭做為 API 方法回應的 Timestamp 標頭，Response header (回應標頭) 欄會包含 Timestamp (時間戳記) 項目，且相關聯的 Mapping value (對應值) 應設為 `integration.response.multivalueheaders.Date`。
  - `integration.response.header.header-name`，其中 `header-name` 是後端單一值回應標頭的名稱。

例如，若要傳回後端回應的 Date 標頭做為 API 方法回應的 Timestamp 標頭，Response header (回應標頭) 欄會包含 Timestamp (時間戳記) 項目，且相關聯的 Mapping value (對應值) 應設為 `integration.response.header.Date`。
- e. 在 Template Mappings (範本映射) 區域中，選擇 Content type (內容類型) 旁邊的 Add (新增)。在 Content type (內容類型) 方塊中，輸入要從 Lambda 函數、HTTP 代理或 AWS 服務代理傳遞給方法之資料的內容類型。選擇 Update (更新)。
- f. 如果您想要讓方法從 Lambda 函數、HTTP 代理或 AWS 服務代理接收資料但不進行修改，請選取 Output passthrough (輸出傳遞)。
- g. 如果已清除 Output passthrough (輸出傳遞)，請針對 Output mapping (輸出對應)，指定您想要讓 Lambda 函數、HTTP 代理或 AWS 服務代理用來將資料傳送給方法的輸出對應範本。您可以手動輸入映射範本，或從 Generate template from model (從模型產生範本) 中選擇一個模型。
- h. 選擇 Save (儲存)。

## 針對請求與回應對應建立模型與對應範本

在 API Gateway 中，API 的方法請求可視後端的需求，從對應的整合請求承載擷取不同格式的承載。同樣地，後端可能會依照前端的預期，傳回與方法回應承載不同的整合回應承載。API Gateway 可讓您使用對應範本，將承載從方法請求對應到對應的整合請求，以及從整合回應對應到對應的方法回應。

對應範本是以 [Velocity 範本語言 \(VTL\)](#) 表示，並使用 [JSONPath 表達式](#)套用至承載的指令碼。

根據 [JSON 結構描述草稿第 4 版](#)，承載可以有一個資料模型。您必須定義模型，才能讓 API Gateway 產生開發套件或為 API 啟用基本請求驗證。您不需要定義任何模型，即可建立對應範本。不過，模型可協助您建立範本，因為 API Gateway 會根據提供的模型產生範本藍圖。

### Note

除了 [JSON 結構描述草稿 4](#) 之外，API Gateway 還支援一些延伸關鍵字，例如，[了解 JSON 結構描述](#)中所述的 `additionalProperties` 關鍵字，這些關鍵字可與請求驗證搭配使用，讓包含額外屬性的請求失效。

本節說明如何使用模型與對應範本來對應 API 請求與回應承載。

### 主題

- [模型 \(p. 245\)](#)
- [對應範本 \(p. 248\)](#)
- [模型與對應範本的任務 \(p. 250\)](#)
- [在 API Gateway 中建立模型 \(p. 250\)](#)
- [在 API Gateway 中檢視模型清單 \(p. 250\)](#)
- [使用對應範本來覆寫 API 請求和回應參數和狀態碼 \(p. 251\)](#)

- 在 API Gateway 中刪除模型 (p. 255)
- 相片範例 (API Gateway 模型與對應範本) (p. 255)
- 新聞文章範例 (API Gateway 模型與對應範本) (p. 258)
- 銷售發票範例 (API Gateway 模型與對應範本) (p. 261)
- 員工記錄範例 (API Gateway 模型與對應範本) (p. 265)

## 模型

在 API Gateway 中，模型會定義承載的資料結構。在 API Gateway 中，模型是使用 JSON 結構描述草稿第 4 版來定義。

下列 JSON 物件說明一個範例資料，其說明超市之類的農產品部門中的水果或蔬菜庫存：

假設我們有一個 API，用於管理超市農產品部門中的水果與蔬菜庫存。當經理查詢後端的目前庫存時，伺服器傳回下列回應承載：

```
{  
  "department": "produce",  
  "categories": [  
    "fruit",  
    "vegetables"  
,  
  "bins": [  
    {  
      "category": "fruit",  
      "type": "apples",  
      "price": 1.99,  
      "unit": "pound",  
      "quantity": 232  
    },  
    {  
      "category": "fruit",  
      "type": "bananas",  
      "price": 0.19,  
      "unit": "each",  
      "quantity": 112  
    },  
    {  
      "category": "vegetables",  
      "type": "carrots",  
      "price": 1.29,  
      "unit": "bag",  
      "quantity": 57  
    }  
  ]  
}
```

此 JSON 物件包含三個屬性

- department 屬性有一個字串值 (produce)。
- categories 屬性是下列兩個字串的陣列：fruit 與 vegetables。
- bins 屬性是物件的陣列，每個物件有 category、type、price、unit 與 quantity 的字串或數值屬性。

我們可以使用下列 JSON 結構描述來定義上述資料的模型：

```
{
```

```
 "$schema": "http://json-schema.org/draft-04/schema#",
"title": "GroceryStoreInputModel",
"type": "object",
"properties": {
    "department": { "type": "string" },
    "categories": {
        "type": "array",
        "items": { "type": "string" }
    },
    "bins": {
        "type": "array",
        "items": {
            "type": "object",
            "properties": {
                "category": { "type": "string" },
                "type": { "type": "string" },
                "price": { "type": "number" },
                "unit": { "type": "string" },
                "quantity": { "type": "integer" }
            }
        }
    }
}
```

在上述範例模型中：

- `$schema` 物件代表有效的 JSON 結構描述版本識別符。在此範例中是指 JSON 結構描述草稿第 4 版。
- `title` 物件是人類看得懂的模型識別符。在此範例中為 `GroceryStoreInputModel`。
- JSON 資料中的最上層或根建構為物件。
- JSON 資料中的根物件包含 `department`、`categories` 與 `bins` 屬性。
- `department` 屬性是 JSON 資料中的字串物件。
- `categories` 屬性是 JSON 資料中的陣列。此陣列包含 JSON 資料中的字串值。
- `bins` 屬性是 JSON 資料中的陣列。此陣列包含 JSON 資料中的物件。JSON 資料中的每個物件都包含 `category` 字串、`type` 字串、`price` 數值、`unit` 字串與 `quantity` 整數(不含分數或指數部分的數值)。

或者，您可以在相同檔案的不同區段中包含此結構描述的一部分(例如 `bins` 陣列的項目定義)，並使用 `$ref` 基本類型在結構描述的其他部分參考此可重複使用的定義。使用 `$ref`，上述模型定義檔案會如下表示：

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "title": "GroceryStoreInputModel",
    "type": "object",
    "properties": {
        "department": { "type": "string" },
        "categories": {
            "type": "array",
            "items": { "type": "string" }
        },
        "bins": {
            "type": "array",
            "items": {
                "$ref": "#/definitions/Bin"
            }
        }
    },
    "definitions": {
        "Bin": {
            "type": "object",

```

```
    "properties": {
        "category": { "type": "string" },
        "type": { "type": "string" },
        "price": { "type": "number" },
        "unit": { "type": "string" },
        "quantity": { "type": "integer" }
    }
}
}
```

`definitions` 區段包含 Bin 項目的結構描述定義，在 bins 陣列中是以 `$ref": "#/definitions/Bin"` 參考。以此方式使用可重複使用的定義會讓您的模型定義更容易閱讀。

此外，您也可以參考外部模型檔案中定義的另一個模型結構描述，方法是將該模型的 URL 設定為 `$ref` 屬性的值：`$ref": "https://apigateway.amazonaws.com/restapis/{restapi_id}/models/{model_name}"`。例如，假設您在識別符為 Bin 的 API 下，建立了名為 fugvwdxtri 的下列成熟模型：

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "title": "GroceryStoreInputModel",
    "type": "object",
    "properties": {
        "Bin": {
            "type": "object",
            "properties": {
                "category": { "type": "string" },
                "type": { "type": "string" },
                "price": { "type": "number" },
                "unit": { "type": "string" },
                "quantity": { "type": "integer" }
            }
        }
    }
}
```

然後，您可以從相同 API 中的 `GroceryStoreInputModel` 來參考它，如下所示：

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "title": "GroceryStoreInputModel",
    "type": "object",
    "properties": {
        "department": { "type": "string" },
        "categories": {
            "type": "array",
            "items": { "type": "string" }
        },
        "bins": {
            "type": "array",
            "items": {
                "$ref": "https://apigateway.amazonaws.com/restapis/fugvwdxtri/models/Bin2"
            }
        }
    }
}
```

參考的模型與被參考的模型必須來自相同的 API。

這些範例不會使用進階 JSON 結構描述功能；例如，指定必要的項目，允許的字串長度、數值與陣列項目長度的上下限；規則表達式等等。如需詳細資訊，請參閱[介紹 JSON 與 JSON 結構描述草稿第 4 版](#)。

如需更複雜的 JSON 資料格式與其模型，請參閱下列範例：

- [相片範例 \(p. 255\)](#)中的輸入模型 (相片範例) (p. 256)與輸出模型 (相片範例) (p. 257)
- [新聞文章範例 \(p. 258\)](#)中的輸入模型 (新聞文章範例) (p. 259)與輸出模型 (新聞文章範例) (p. 260)
- [銷售發票範例 \(p. 261\)](#)中的輸入模型 (銷售發票範例) (p. 262)與輸出模型 (銷售發票範例) (p. 264)
- [員工記錄範例 \(p. 265\)](#)中的輸入模型 (員工記錄範例) (p. 266)與輸出模型 (員工記錄範例) (p. 268)

若要在 API Gateway 中測試模型，請遵循[對應回應承載 \(p. 69\)](#)中的說明進行，特別是步驟 1：建立模型 (p. 70)。

## 對應範本

當後端傳回查詢結果時 (如[模型 \(p. 245\)](#)一節中所示)，農產品部門經理可能對讀取以下內容有興趣：

```
{  
    "choices": [  
        {  
            "kind": "apples",  
            "suggestedPrice": "1.99 per pound",  
            "available": 232  
        },  
        {  
            "kind": "bananas",  
            "suggestedPrice": "0.19 per each",  
            "available": 112  
        },  
        {  
            "kind": "carrots",  
            "suggestedPrice": "1.29 per bag",  
            "available": 57  
        }  
    ]  
}
```

若要啟用此功能，我們需要提供對應範本給 API Gateway，以轉譯後端格式的資料。下列對應範本可執行此操作。

```
#set($inputRoot = $input.path('$'))  
{  
    "choices": [  
#foreach($elem in $inputRoot.bins)  
    {  
        "kind": "$elem.type",  
        "suggestedPrice": "$elem.price per $elem.unit",  
        "available": $elem.quantity  
    }#if($foreach.hasNext),#end  
  
#end  
    ]  
}
```

現在，讓我們來看上述輸出對應範本的一些詳細資訊：

- `$inputRoot` 變數代表上一節中原始 JSON 資料的根物件。輸出對應範本中的變數會對應到原始 JSON 資料，而不是所需之轉換後的 JSON 資料結構描述。
- 輸出對應範本中的 `choices` 陣列是從原始 JSON 資料 (`bins`) 中具有根物件的 `$inputRoot.bins` 陣列對應而來。
- 在輸出對應範本中，`choices` 陣列中的每個物件 (以 `$elem` 表示) 是從原始 JSON 資料根物件之 `bins` 陣列中的對應物件對應而來。

- 在輸出對應範本中，針對 `choices` 物件中的每個物件，`kind` 與 `available` 物件的值 (以 `$elem.type` 與 `$elem.quantity` 表示) 是從原始 JSON 資料 `type` 陣列之每個物件中的對應 `value` 與 `bins` 物件值分別對應而來。
- 在輸出對應範本中，針對 `choices` 物件中的每個物件，`suggestedPrice` 物件的值分別是原始 JSON 資料之每個物件中對應 `price` 與 `unit` 物件值的串連，每個值會以 `per` 一字分隔。

如需 Velocity 範本語言的詳細資訊，請參閱 [Apache Velocity - VTL Reference](#)。如需 JSONPath 的詳細資訊，請參閱 [JSONPath - XPath for JSON](#)。

對應範本假設基礎資料屬於 JSON 物件。它不需要定義資料模型。身為 API 開發人員，您知道前端與後端的資料格式。該知識可引導您明確定義必要的對應。

為了產生 API 的開發套件，上述資料會以特定語言物件傳回。針對強型別語言 (例如 Java、Objective-C 或 Swift)，物件會對應到使用者定義的資料類型 (UDT)。如果您透過資料模型提供，則 API Gateway 會建立這類 UDT。在上述方法回應範例中，您可以在整合回應中定義下列承載模型：

```
{  
  "$schema": "http://json-schema.org/draft-04/schema#",  
  "title": "GroceryStoreOutputModel",  
  "type": "object",  
  "properties": {  
    "choices": {  
      "type": "array",  
      "items": {  
        "type": "object",  
        "properties": {  
          "kind": { "type": "string" },  
          "suggestedPrice": { "type": "string" },  
          "available": { "type": "integer" }  
        }  
      }  
    }  
  }  
}
```

在此模型中，JSON 結構描述會如下表示：

- `$schema` 物件代表有效的 JSON 結構描述版本識別符。在此範例中是指 JSON 結構描述草稿第 4 版。
- `title` 物件是人類看得懂的模型識別符。在此範例中為 `GroceryStoreOutputModel`。
- JSON 資料中的最上層或根建構為物件。
- JSON 資料中的根物件包含物件陣列。
- 物件陣列中的每個物件都包含 `kind` 字串、`suggestedPrice` 字串與 `available` 整數 (不含分數或指數部分的數值)。

有了此模型，您可以呼叫開發套件分別讀取 `kind`、`suggestedPrice` 與 `available` 屬性，來擷取 `GroceryStoreOutputModel[i].kind`、`GroceryStoreOutputModel[i].suggestedPrice` 與 `GroceryStoreOutputModel[i].available` 屬性值。如果未提供模型，API Gateway 會使用空白模型來建立預設 UDT。在此情況下，您將無法使用強型別開發套件來讀取這些屬性。

若要探索更複雜的對應範本，請參閱下列範例：

- [相片範例 \(p. 255\)](#)中的 [輸入對應範本 \(相片範例\) \(p. 256\)](#)與 [輸出對應範本 \(相片範例\) \(p. 258\)](#)
- [新聞文章範例 \(p. 258\)](#)中的 [輸入對應範本 \(新聞文章範例\) \(p. 259\)](#)與 [輸出對應範本 \(新聞文章範例\) \(p. 261\)](#)
- [銷售發票範例 \(p. 261\)](#)中的 [輸入對應範本 \(銷售發票範例\) \(p. 263\)](#)與 [輸出對應範本 \(銷售發票範例\) \(p. 265\)](#)

- [員工記錄範例 \(p. 265\)](#)中的輸入對應範本 ([員工記錄範例 \(p. 267\)](#))與輸出對應範本 ([員工記錄範例 \(p. 269\)](#))

若要在 API Gateway 中測試對應範本，請遵循[對應回應承載 \(p. 69\)](#)中的說明進行，特別是[步驟 5：設定及測試方法 \(p. 74\)](#)。

## 模型與對應範本的任務

如需您可以使用模型與對應範本執行的其他作業，請參閱下列各項：

- [建立模型 \(p. 250\)](#)
- [檢視模型清單 \(p. 250\)](#)
- [刪除模型 \(p. 255\)](#)

## 在 API Gateway 中建立模型

使用 API Gateway 主控台建立 API 模型。

### 主題

- [先決條件 \(p. 250\)](#)
- [使用 API Gateway 主控台建立模型 \(p. 250\)](#)

### 先決條件

- 您必須擁有 API Gateway 中可用的 API。請遵循[在 Amazon API Gateway 中建立 REST API \(p. 167\)](#)中的說明進行。

## 使用 API Gateway 主控台建立模型

1. 登入位於 <https://console.aws.amazon.com/apigateway> 的 API Gateway 主控台。
2. 在包含您要建立模型之 API 名稱的方塊中，選擇 Models (模型)。
3. 選擇 Create (建立)。
4. 針對 Model Name (模型名稱)，輸入模型的名稱。
5. 針對 Content Type (內容類型)，輸入模型的內容類型 (例如 `application/json` 適用於 JSON)。
6. (選用) 針對 Model description (模型說明)，輸入模型的說明。
7. 針對 Model schema (模型結構描述)，輸入模型的結構描述。如需模型結構描述的詳細資訊，請參閱[針對請求與回應對應建立模型與對應範本 \(p. 244\)](#)。
8. 選擇 Create model (建立模型)。

## 在 API Gateway 中檢視模型清單

使用 API Gateway 主控台來檢視模型清單。

### 主題

- [先決條件 \(p. 250\)](#)
- [使用 API Gateway 主控台來檢視模型清單 \(p. 251\)](#)

### 先決條件

- 在 API Gateway 中，您必須至少有一個模型。請遵循[建立模型 \(p. 250\)](#)中的說明進行。

## 使用 API Gateway 主控台來檢視模型清單

1. 登入位於 <https://console.aws.amazon.com/apigateway> 的 API Gateway 主控台。
2. 在包含 API 名稱的方塊中，選擇 Models (模型)。

## 使用對應範本來覆寫 API 請求和回應參數和狀態碼

標準 API Gateway 參數和回應程式碼對應範本 (p. 244) 可讓您一對一對應參數和將一系列整合回應狀態碼 (透過一般表達式而相符) 對應至單一回應狀態碼。對應範本覆寫使您能夠靈活地執行許多一對一參數對應；在標準 API Gateway 對應後的覆寫參數已套用；根據內文內容或其他參數值有條件地對應參數；同時透過程式設計方式建立新的參數，以及覆寫整合端點傳回的狀態碼。可以覆寫任何類型的請求參數、回應標頭，或回應狀態碼。

以下是用於對應範本覆寫的範例：

- 若要建立新的標頭 (或覆寫現有標頭) 做為兩個參數的連接
- 若要根據內文內容將回應程式碼覆寫為成功或失敗程式碼
- 若要有條件地根據它的內容或一些其他參數的內容重新對應參數
- 若要反覆查看 json 內文的內容和將金鑰值對重新對應為標頭或查詢字串

若要建立對應範本覆寫，請使用一或多個對應範本 (p. 244) 中的下列 \$context 變數 (p. 274)：

請求內文對應範本	回應內文對應範本。
\$context.requestOverride.header.header_name	\$context.responseOverride.header.header_name
\$context.requestOverride.path.path_name	\$context.responseOverride.status
\$context.requestOverride.querystring.QueryString_name	

### Note

對應範本覆寫無法用於 Proxy 整合端點，其缺乏資料對應。如需整合類型的詳細資訊，請參閱選擇 API Gateway API 整合類型 (p. 201)。

### Important

覆寫是最終。覆寫只能套用到每個參數一次。多次嘗試覆寫相同的參數會導致來自 Amazon API Gateway 的 5xx 回應。如果您必須多次在整個範本中覆寫相同的參數，我們建議您建立變數與在範本結尾套用覆寫。請注意，只會在整個範本剖析後才會套用範本。請參閱「教學課程：使用 API Gateway 主控台覆寫 API 請求參數和標頭 (p. 252)」。

以下教學課程示範如何建立和測試在 API Gateway 主控台中的對應範本覆寫。這些教學課程是使用 PetStore 範例 API (p. 40) 做為起點。兩個教學都假設您已建立 PetStore 範例 API (p. 40)。

### 主題

- 教學課程：使用 API Gateway 主控台來覆寫 API 回應狀態程式碼 (p. 251)
- 教學課程：使用 API Gateway 主控台覆寫 API 請求參數和標頭 (p. 252)
- 範例：使用 API Gateway CLI 覆寫 API 請求參數和標頭 (p. 253)
- 範例：使用 SDK for JavaScript 覆寫 API 請求參數和標頭 (p. 254)

## 教學課程：使用 API Gateway 主控台來覆寫 API 回應狀態程式碼

若要使用 PetStore 範例 API 擷取寵物，請使用 GET /pets/{petId} 的 API 方法請求，其中 {petId} 是可在執行階段接受一個數值的路徑參數。

在此教學課程中，您將覆寫此 GET 方法的回應程式碼，方法是在偵測到錯誤條件時，建立將 \$context.responseOverride.status 對應到 400 的對應範本。

1. 登入位於 <https://console.aws.amazon.com/apigateway> 的 API Gateway 主控台。
2. 在 APIs 下，選擇一個 PetStore API。
3. 在 Resources (資源) 窗格中，選擇 /{petId} 下的 GET 方法。
4. 在 Client (用戶端) 方塊中，選擇 Test (測試)。
5. 對 {petId} ({petId}) 輸入 -1 然後選擇 Test (測試)。

您會在結果中注意到兩件事：

首先，Response Body (回應內文) 表示超出範圍錯誤：

```
{  
  "errors": [  
    {  
      "key": "GetPetRequest.petId",  
      "message": "The value is out of range."  
    }  
  ]  
}
```

其次，在 Logs (日誌) 方塊下的最後一列結尾為：Method completed with status: 200。

6. 返回 Method Execution (方法執行)。選擇 Integration Response (整合回應)，再選擇 200 旁的箭頭。
7. 展開 Mapping Templates (對應範本)，然後選擇 Add mapping template (新增對應範本)。
8. 針對 Content Type (內容類型) 輸入 **application/json**，然後選擇核取記號圖示來儲存選項。
9. 請將下列程式碼複製到範本區域中。

```
#set($inputRoot = $input.path('$'))  
$input.json("$")  
#if($inputRoot.toString().contains("error"))  
#set($context.responseOverride.status = 400)  
#end
```

10. 選擇 Save (儲存)。
11. 返回 Method Execution (方法執行)。
12. 在 Client (用戶端) 方塊中，選擇 Test (測試)。
13. 對 {petId} ({petId}) 輸入 -1 然後選擇 Test (測試)。

在結果中，Response Body (回應內文) 表示超出範圍錯誤：

```
{  
  "errors": [  
    {  
      "key": "GetPetRequest.petId",  
      "message": "The value is out of range."  
    }  
  ]  
}
```

然而，在 Logs (日誌) 方塊下的最後一列目前結尾為：Method completed with status: 400。

### 教學課程：使用 API Gateway 主控台覆寫 API 請求參數和標頭

在此教學課程中，您將覆寫 GET 方法的請求標頭程式碼，方法是建立會將 \$context.requestOverride.header.**header\_name** 對應到結合兩個其他標頭之新標頭的對應範本。

1. 登入位於 <https://console.aws.amazon.com/apigateway> 的 API Gateway 主控台。
2. 在 APIs 下，選擇一個 PetStore API。
3. 在 Resources (資源) 窗格中，選擇 /pets 下的 GET 方法。
4. 選擇 Method Request (方法請求)。
5. 建立參數，如下所示：
  - a. 展開 HTTP Request Headers (HTTP 請求標頭)。
  - b. 選擇 Add header (新增標頭)。
  - c. 在 Name (名稱) 中輸入 **header1**。
  - d. 選擇核取記號圖示來儲存選擇。

重複進行以上程序，以建立第二個名為 header2 的標頭。

6. 返回 Method Execution (方法執行)。
7. 選擇 Integration Request (整合請求)。
8. 展開 HTTP Headers (HTTP 標頭)。您將會看到您建立的兩個標頭 header1 和 header2 以及他們的預設對應 (在 Mapped from (對應來源) 下)。
9. 展開 Mapping Templates (對應範本)。
10. 選擇 Add mapping template (新增對應範本)。
11. 針對 Content Type (內容類型) 輸入 **application/json**，然後選擇核取記號圖示來儲存選項。
12. 系統會顯示一個快顯視窗，說明：Note: This template can map headers and body (請注意：此範本可以對應標頭和內文)。

選擇 Yes, secure this integration (是，保護此整合)。

13. 請將下列程式碼複製到範本區域中。

```
#set($header1Override = "foo")
#set($header3Value = "$input.params('header1')$input.params('header2')")
$input.json("$")
#set($context.requestOverride.header.header3 = $header3Value)
#set($context.requestOverride.header.header1 = $header1Override)
#set($context.requestOverride.header.multivalueheader=[ $header1Override,
$header3Value])
```

14. 選擇 Save (儲存)。
15. 返回 Method Execution (方法執行)。
16. 在 Client (用戶端) 方塊中，選擇 Test (測試)。
17. 在 {pets} ({pets}) 的 Headers (標頭) 下，複製下列程式碼：

```
header1:header1Val
header2:header2Val
```

18. 選擇 Test (測試)。

在日誌中，您應該會看到一個包括此文字的項目：

```
Endpoint request headers: {header3=header1Valheader2Val,
header2=header2Val, header1=foo, x-amzn-apigateway-api-id=<api-id>,
Accept=application/json, multivalueheader=foo,header1Valheader2Val}
```

### 範例：使用 API Gateway CLI 覆寫 API 請求參數和標頭

以下 CLI 範例示範如何使用 `put-integration` 命令來覆寫回應程式碼：

```
aws apigateway put-integration --rest-api-id <API_ID> --resource-id <PATH_TO_RESOURCE_ID>
--http-method <METHOD>
--type <INTEGRATION_TYPE> --request-templates <REQUEST_TEMPLATE_MAP>
```

其中 *<REQUEST\_TEMPLATE\_MAP>* 是從內容類型到要套用之範本字串的對應。對應的結構如下所示：

```
Content_type1=template_string,Content_type2=template_string
```

或者，在 JSON 語法中：

```
{"content_type1": "template_string"
...}
```

以下範例示範如何使用 `put-integration-response` 命令來覆寫 API 回應程式碼：

```
aws apigateway put-integration-response --rest-api-id <API_ID> --resource-
id <PATH_TO_RESOURCE_ID> --http-method <METHOD>
--status-code <STATUS_CODE> --response-templates <RESPONSE_TEMPLATE_MAP>
```

其中 *<RESPONSE\_TEMPLATE\_MAP>* 具有與上述 *<REQUEST\_TEMPLATE\_MAP>* 相同的格式。

#### 範例：使用 SDK for JavaScript 覆寫 API 請求參數和標頭

以下範例示範如何使用 `put-integration` 命令來覆寫回應程式碼：

要求：

```
var params = {
  httpMethod: 'STRING_VALUE', /* required */
  resourceId: 'STRING_VALUE', /* required */
  restApiId: 'STRING_VALUE', /* required */
  type: HTTP | AWS | MOCK | HTTP_PROXY | AWS_PROXY, /* required */
  requestTemplates: {
    '<Content_type>': 'TEMPLATE_STRING',
    /* '<String>': ... */
  },
};
apigateway.putIntegration(params, function(err, data) {
  if (err) console.log(err, err.stack); // an error occurred
  else    console.log(data);           // successful response
});
```

回應：

```
var params = {
  httpMethod: 'STRING_VALUE', /* required */
  resourceId: 'STRING_VALUE', /* required */
  restApiId: 'STRING_VALUE', /* required */
  statusCode: 'STRING_VALUE', /* required */
  responseTemplates: {
    '<Content_type>': 'TEMPLATE_STRING',
    /* '<String>': ... */
  },
};
apigateway.putIntegrationResponse(params, function(err, data) {
  if (err) console.log(err, err.stack); // an error occurred
  else    console.log(data);           // successful response
});
```

## 在 API Gateway 中刪除模型

您可以使用 API Gateway 主控台來刪除模型。

### Warning

刪除模型可能會導致 API 發起人變成無法使用部分或所有對應的 API。刪除模型無法復原。

### 使用 API Gateway 主控台刪除模型

1. 登入位於 <https://console.aws.amazon.com/apigateway> 的 API Gateway 主控台。
2. 在包含模型 API 之名稱的方塊中，選擇 Models (模型)。
3. 在 Models (模型) 窗格中，選擇您要刪除的模型，然後選擇 Delete Model (刪除模型)。
4. 出現提示時，選擇 Delete (刪除)。

## 相片範例 (API Gateway 模型與對應範本)

下列各節提供的模型與對應範本範例，可在 API Gateway 中用作範例相片 API。如需 API Gateway 中模型與對應範本的詳細資訊，請參閱[針對請求與回應對應建立模型與對應範本 \(p. 244\)](#)。

### 主題

- [原始資料 \(相片範例\) \(p. 255\)](#)
- [輸入模型 \(相片範例\) \(p. 256\)](#)
- [輸入對應範本 \(相片範例\) \(p. 256\)](#)
- [已轉換的資料 \(相片範例\) \(p. 257\)](#)
- [輸出模型 \(相片範例\) \(p. 257\)](#)
- [輸出對應範本 \(相片範例\) \(p. 258\)](#)

### 原始資料 (相片範例)

以下是相片範例的原始 JSON 資料：

```
{  
  "photos": {  
    "page": 1,  
    "pages": "1234",  
    "perpage": 100,  
    "total": "123398",  
    "photo": [  
      {  
        "id": "12345678901",  
        "owner": "23456789@A12",  
        "secret": "abc123d456",  
        "server": "1234",  
        "farm": 1,  
        "title": "Sample photo 1",  
        "ispublic": 1,  
        "isfriend": 0,  
        "isfamily": 0  
      },  
      {  
        "id": "23456789012",  
        "owner": "34567890@B23",  
        "secret": "bcd234e567",  
        "server": "2345",  
        "farm": 2,  
        "title": "Sample photo 2",  
        "ispublic": 1,  
        "isfriend": 0,  
        "isfamily": 0  
      }  
    ]  
  }  
}
```

```
        "isfriend": 0,
        "isfamily": 0
    }
]
}
```

### 輸入模型 (相片範例)

以下是對應到相片範例之原始 JSON 資料的輸入模型：

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "title": "PhotosInputModel",
  "type": "object",
  "properties": {
    "photos": {
      "type": "object",
      "properties": {
        "page": { "type": "integer" },
        "pages": { "type": "string" },
        "perpage": { "type": "integer" },
        "total": { "type": "string" },
        "photo": {
          "type": "array",
          "items": {
            "type": "object",
            "properties": {
              "id": { "type": "string" },
              "owner": { "type": "string" },
              "secret": { "type": "string" },
              "server": { "type": "string" },
              "farm": { "type": "integer" },
              "title": { "type": "string" },
              "ispublic": { "type": "integer" },
              "isfriend": { "type": "integer" },
              "isfamily": { "type": "integer" }
            }
          }
        }
      }
    }
  }
}
```

### 輸入對應範本 (相片範例)

以下是對應到相片範例之原始 JSON 資料的輸入對應範本：

```
#set($inputRoot = $input.path('$'))
{
  "photos": {
    "page": $inputRoot.photos.page,
    "pages": "$inputRoot.photos.pages",
    "perpage": $inputRoot.photos.perpage,
    "total": "$inputRoot.photos.total",
    "photo": [
      #foreach($elem in $inputRoot.photos.photo)
        {
          "id": "$elem.id",
          "owner": "$elem.owner",
          "secret": "$elem.secret",
          "server": "$elem.server",
        }
    ]
  }
}
```

```
        "farm": $elem.farm,
        "title": "$elem.title",
        "ispublic": $elem.ispublic,
        "isfriend": $elem.isfriend,
        "isfamily": $elem.isfamily
    }#if($foreach.hasNext),#end

}
]
}
```

### 已轉換的資料 (相片範例)

以下是如何轉換原始相片範例 JSON 資料以供輸出的範例之一：

```
{
    "photos": [
        {
            "id": "12345678901",
            "owner": "23456789@A12",
            "title": "Sample photo 1",
            "ispublic": 1,
            "isfriend": 0,
            "isfamily": 0
        },
        {
            "id": "23456789012",
            "owner": "34567890@B23",
            "title": "Sample photo 2",
            "ispublic": 1,
            "isfriend": 0,
            "isfamily": 0
        }
    ]
}
```

### 輸出模型 (相片範例)

以下是對應到轉換後 JSON 資料格式的輸出模型：

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "title": "PhotosOutputModel",
    "type": "object",
    "properties": {
        "photos": {
            "type": "array",
            "items": {
                "type": "object",
                "properties": {
                    "id": { "type": "string" },
                    "owner": { "type": "string" },
                    "title": { "type": "string" },
                    "ispublic": { "type": "integer" },
                    "isfriend": { "type": "integer" },
                    "isfamily": { "type": "integer" }
                }
            }
        }
    }
}
```

## 輸出對應範本 (相片範例)

以下是對應到轉換後 JSON 資料格式的輸出對應範本。此處的範本變數採用原始 JSON 資料格式，而不是轉換後的 JSON 資料格式：

```
#set($inputRoot = $input.path('$'))  
{  
    "photos": [  
        #foreach($elem in $inputRoot.photos.photo)  
        {  
            "id": "$elem.id",  
            "owner": "$elem.owner",  
            "title": "$elem.title",  
            "ispublic": $elem.ispublic,  
            "isfriend": $elem.isfriend,  
            "isfamily": $elem.isfamily  
        }#if($foreach.hasNext),#end  
  
    #end  
    ]  
}
```

## 新聞文章範例 (API Gateway 模型與對應範本)

下列各節提供的模型與對應範本範例，可在 API Gateway 中做為範例新聞文章 API 使用。如需 API Gateway 中模型與對應範本的詳細資訊，請參閱[針對請求與回應對應建立模型與對應範本 \(p. 244\)](#)。

### 主題

- [原始資料 \(新聞文章範例\) \(p. 258\)](#)
- [輸入模型 \(新聞文章範例\) \(p. 259\)](#)
- [輸入對應範本 \(新聞文章範例\) \(p. 259\)](#)
- [轉換後的資料 \(新聞文章範例\) \(p. 260\)](#)
- [輸出模型 \(新聞文章範例\) \(p. 260\)](#)
- [輸出對應範本 \(新聞文章範例\) \(p. 261\)](#)

### 原始資料 (新聞文章範例)

以下是新聞文章範例的原始 JSON 資料：

```
{  
    "count": 1,  
    "items": [  
        {  
            "last_updated_date": "2015-04-24",  
            "expire_date": "2016-04-25",  
            "author_first_name": "John",  
            "description": "Sample Description",  
            "creation_date": "2015-04-20",  
            "title": "Sample Title",  
            "allow_comment": "1",  
            "author": {  
                "last_name": "Doe",  
                "email": "johndoe@example.com",  
                "first_name": "John"  
            },  
            "body": "Sample Body",  
            "publish_date": "2015-04-25",  
            "version": "1",  
            "author_last_name": "Doe",  
            "author_email": "johndoe@example.com",  
            "author_id": "12345678901234567890123456789012",  
            "author_ispublic": true,  
            "author_isfriend": false,  
            "author_isfamily": false  
        }  
    ]  
}
```

```
        "parent_id": 2345678901,
        "article_url": "http://www.example.com/articles/3456789012"
    },
    "version": 1
}
```

### 輸入模型 (新聞文章範例)

以下是對應到新聞文章範例之原始 JSON 資料的輸入模型：

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "title": "NewsArticleInputModel",
  "type": "object",
  "properties": {
    "count": { "type": "integer" },
    "items": {
      "type": "array",
      "items": {
        "type": "object",
        "properties": {
          "last_updated_date": { "type": "string" },
          "expire_date": { "type": "string" },
          "author_first_name": { "type": "string" },
          "description": { "type": "string" },
          "creation_date": { "type": "string" },
          "title": { "type": "string" },
          "allow_comment": { "type": "string" },
          "author": {
            "type": "object",
            "properties": {
              "last_name": { "type": "string" },
              "email": { "type": "string" },
              "first_name": { "type": "string" }
            }
          },
          "body": { "type": "string" },
          "publish_date": { "type": "string" },
          "version": { "type": "string" },
          "author_last_name": { "type": "string" },
          "parent_id": { "type": "integer" },
          "article_url": { "type": "string" }
        }
      }
    },
    "version": { "type": "integer" }
  }
}
```

### 輸入對應範本 (新聞文章範例)

以下是對應到新聞文章範例之原始 JSON 資料的輸入對應範本：

```
#set($inputRoot = $input.path('$'))
{
  "count": $inputRoot.count,
  "items": [
#foreach($elem in $inputRoot.items)
  {
    "last_updated_date": "$elem.last_updated_date",
    "expire_date": "$elem.expire_date",
    "author_first_name": "$elem.author_first_name",
  }
]
```

```
"description": "$elem.description",
"creation_date": "$elem.creation_date",
"title": "$elem.title",
"allow_comment": "$elem.allow_comment",
"author": {
    "last_name": "$elem.author.last_name",
    "email": "$elem.author.email",
    "first_name": "$elem.author.first_name"
},
"body": "$elem.body",
"publish_date": "$elem.publish_date",
"version": "$elem.version",
"author_last_name": "$elem.author_last_name",
"parent_id": $elem.parent_id,
"article_url": "$elem.article_url"
}#if($foreach.hasNext),#end

#endif
],
"version": $inputRoot.version
}
```

### 轉換後的資料 (新聞文章範例)

以下是如何轉換原始新聞文章範例 JSON 資料以供輸出的範例之一：

```
{
  "count": 1,
  "items": [
    {
      "creation_date": "2015-04-20",
      "title": "Sample Title",
      "author": "John Doe",
      "body": "Sample Body",
      "publish_date": "2015-04-25",
      "article_url": "http://www.example.com/articles/3456789012"
    }
  ],
  "version": 1
}
```

### 輸出模型 (新聞文章範例)

以下是對應到轉換後 JSON 資料格式的輸出模型：

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "title": "NewsArticleOutputModel",
  "type": "object",
  "properties": {
    "count": { "type": "integer" },
    "items": {
      "type": "array",
      "items": {
        "type": "object",
        "properties": {
          "creation_date": { "type": "string" },
          "title": { "type": "string" },
          "author": { "type": "string" },
          "body": { "type": "string" },
          "publish_date": { "type": "string" },
          "article_url": { "type": "string" }
        }
      }
    }
  }
}
```

```
        },
        "version": { "type": "integer" }
    }
}
```

### 輸出對應範本 (新聞文章範例)

以下是對應到轉換後 JSON 資料格式的輸出對應範本。此處的範本變數採用原始 JSON 資料格式，而不是轉換後的 JSON 資料格式：

```
#set($inputRoot = $input.path('$'))
{
    "count": $inputRoot.count,
    "items": [
#foreach($elem in $inputRoot.items)
    {
        "creation_date": "$elem.creation_date",
        "title": "$elem.title",
        "author": "$elem.author.first_name $elem.author.last_name",
        "body": "$elem.body",
        "publish_date": "$elem.publish_date",
        "article_url": "$elem.article_url"
    }#if($foreach.hasNext),#end

#end
    ],
    "version": $inputRoot.version
}
```

### 銷售發票範例 (API Gateway 模型與對應範本)

下列章節提供的模型與對應範本範例，可在 API Gateway 中用為範例銷售發票 API。如需 API Gateway 中模型與對應範本的詳細資訊，請參閱針對請求與回應對應建立模型與對應範本 (p. 244)。

#### 主題

- [原始資料 \(銷售發票範例\) \(p. 261\)](#)
- [輸入模型 \(銷售發票範例\) \(p. 262\)](#)
- [輸入對應範本 \(銷售發票範例\) \(p. 263\)](#)
- [轉換後的資料 \(銷售發票範例\) \(p. 264\)](#)
- [輸出模型 \(銷售發票範例\) \(p. 264\)](#)
- [輸出對應範本 \(銷售發票範例\) \(p. 265\)](#)

### 原始資料 (銷售發票範例)

以下是銷售發票範例的原始 JSON 資料：

```
{
    "DueDate": "2013-02-15",
    "Balance": 1990.19,
    "DocNumber": "SAMP001",
    "Status": "Payable",
    "Line": [
        {
            "Description": "Sample Expense",
            "Amount": 500,
            "DetailType": "ExpenseDetail",
            "ExpenseDetail": {

```

```
"Customer": {  
    "value": "ABC123",  
    "name": "Sample Customer"  
},  
"Ref": {  
    "value": "DEF234",  
    "name": "Sample Construction"  
},  
"Account": {  
    "value": "EFG345",  
    "name": "Fuel"  
},  
"LineStatus": "Billable"  
}  
]  
],  
"Vendor": {  
    "value": "GHI456",  
    "name": "Sample Bank"  
},  
"APRef": {  
    "value": "HIJ567",  
    "name": "Accounts Payable"  
},  
"TotalAmt": 1990.19  
}
```

### 輸入模型 (銷售發票範例)

以下是對應到銷售發票範例之原始 JSON 資料的輸入模型：

```
{  
    "$schema": "http://json-schema.org/draft-04/schema#",  
    "title": "InvoiceInputModel",  
    "type": "object",  
    "properties": {  
        "DueDate": { "type": "string" },  
        "Balance": { "type": "number" },  
        "DocNumber": { "type": "string" },  
        "Status": { "type": "string" },  
        "Line": {  
            "type": "array",  
            "items": {  
                "type": "object",  
                "properties": {  
                    "Description": { "type": "string" },  
                    "Amount": { "type": "integer" },  
                    "DetailType": { "type": "string" },  
                    "ExpenseDetail": {  
                        "type": "object",  
                        "properties": {  
                            "Customer": {  
                                "type": "object",  
                                "properties": {  
                                    "value": { "type": "string" },  
                                    "name": { "type": "string" }  
                                }  
                            },  
                            "Ref": {  
                                "type": "object",  
                                "properties": {  
                                    "value": { "type": "string" },  
                                    "name": { "type": "string" }  
                                }  
                            }  
                        }  
                    }  
                }  
            }  
        }  
    }  
}
```

```
        },
        "Account": {
            "type": "object",
            "properties": {
                "value": { "type": "string" },
                "name": { "type": "string" }
            }
        },
        "LineStatus": { "type": "string" }
    }
}
},
"Vendor": {
    "type": "object",
    "properties": {
        "value": { "type": "string" },
        "name": { "type": "string" }
    }
}
},
"APRef": {
    "type": "object",
    "properties": {
        "value": { "type": "string" },
        "name": { "type": "string" }
    }
}
},
"TotalAmt": { "type": "number" }
}
}
```

### 輸入對應範本 (銷售發票範例)

以下是對應到銷售發票範例之原始 JSON 資料的輸入對應範本：

```
#set($inputRoot = $input.path('$'))
{
    "DueDate": "$inputRoot.DueDate",
    "Balance": $inputRoot.Balance,
    "DocNumber": "$inputRoot.DocNumber",
    "Status": "$inputRoot.Status",
    "Line": [
        #foreach($elem in $inputRoot.Line)
        {
            "Description": "$elem.Description",
            "Amount": $elem.Amount,
            "DetailType": "$elem.DetailType",
            "ExpenseDetail": {
                "Customer": {
                    "value": "$elem.ExpenseDetail.Customer.value",
                    "name": "$elem.ExpenseDetail.Customer.name"
                },
                "Ref": {
                    "value": "$elem.ExpenseDetail.Ref.value",
                    "name": "$elem.ExpenseDetail.Ref.name"
                },
                "Account": {
                    "value": "$elem.ExpenseDetail.Account.value",
                    "name": "$elem.ExpenseDetail.Account.name"
                },
                "LineStatus": "$elem.ExpenseDetail.LineStatus"
            }
        }#if($foreach.hasNext),#end
    ]
}
```

```
#end
],
"Vendor": {
    "value": "$inputRoot.Vendor.value",
    "name": "$inputRoot.Vendor.name"
},
"APRef": {
    "value": "$inputRoot.APRef.value",
    "name": "$inputRoot.APRef.name"
},
"TotalAmt": $inputRoot.TotalAmt
}
```

### 轉換後的資料 (銷售發票範例)

以下是如何轉換原始銷售發票範例 JSON 資料以供輸出的範例之一：

```
{
    "DueDate": "2013-02-15",
    "Balance": 1990.19,
    "DocNumber": "SAMP001",
    "Status": "Payable",
    "Line": [
        {
            "Description": "Sample Expense",
            "Amount": 500,
            "DetailType": "ExpenseDetail",
            "Customer": "ABC123 (Sample Customer)",
            "Ref": "DEF234 (Sample Construction)",
            "Account": "EFG345 (Fuel)",
            "LineStatus": "Billable"
        }
    ],
    "TotalAmt": 1990.19
}
```

### 輸出模型 (銷售發票範例)

以下是對應到轉換後 JSON 資料格式的輸出模型：

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "title": "InvoiceOutputModel",
    "type": "object",
    "properties": {
        "DueDate": { "type": "string" },
        "Balance": { "type": "number" },
        "DocNumber": { "type": "string" },
        "Status": { "type": "string" },
        "Line": {
            "type": "array",
            "items": {
                "type": "object",
                "properties": {
                    "Description": { "type": "string" },
                    "Amount": { "type": "integer" },
                    "DetailType": { "type": "string" },
                    "Customer": { "type": "string" },
                    "Ref": { "type": "string" },
                    "Account": { "type": "string" },
                    "LineStatus": { "type": "string" }
                }
            }
        }
    }
}
```

```
        },
        "TotalAmt": { "type": "number" }
    }
}
```

### 輸出對應範本 (銷售發票範例)

以下是對應到轉換後 JSON 資料格式的輸出對應範本。此處的範本變數採用原始 JSON 資料格式，而不是轉換後的 JSON 資料格式：

```
#set($inputRoot = $input.path('$'))
{
    "DueDate": "$inputRoot.DueDate",
    "Balance": $inputRoot.Balance,
    "DocNumber": "$inputRoot.DocNumber",
    "Status": "$inputRoot.Status",
    "Line": [
        #foreach($elem in $inputRoot.Line)
        {
            "Description": "$elem.Description",
            "Amount": $elem.Amount,
            "DetailType": "$elem.DetailType",
            "Customer": "$elem.ExpenseDetail.Customer.value ($elem.ExpenseDetail.Customer.name)",
            "Ref": "$elem.ExpenseDetail.Ref.value ($elem.ExpenseDetail.Ref.name)",
            "Account": "$elem.ExpenseDetail.Account.value ($elem.ExpenseDetail.Account.name)",
            "LineStatus": "$elem.ExpenseDetail.LineStatus"
        }#if($foreach.hasNext),#end
    ],
    "TotalAmt": $inputRoot.TotalAmt
}
```

### 員工記錄範例 (API Gateway 模型與對應範本)

下列各節提供的模型與對應範本範例，可在 API Gateway 中用作範例員工記錄 API。如需 API Gateway 中模型與對應範本的詳細資訊，請參閱[針對請求與回應對應建立模型與對應範本 \(p. 244\)](#)。

#### 主題

- [原始資料 \(員工記錄範例\) \(p. 265\)](#)
- [輸入模型 \(員工記錄範例\) \(p. 266\)](#)
- [輸入對應範本 \(員工記錄範例\) \(p. 267\)](#)
- [轉換後的資料 \(員工記錄範例\) \(p. 268\)](#)
- [輸出模型 \(員工記錄範例\) \(p. 268\)](#)
- [輸出對應範本 \(員工記錄範例\) \(p. 269\)](#)

#### 原始資料 (員工記錄範例)

以下是員工記錄範例的原始 JSON 資料：

```
{
    "QueryResponse": {
        "maxResults": "1",
        "startPosition": "1",
        "Employee": {
            "Organization": "false",
            "Title": "Mrs.",
            "GivenName": "Jane",
            "FamilyName": "Doe"
        }
    }
}
```

```
"MiddleName": "Lane",
"FamilyName": "Doe",
"DisplayName": "Jane Lane Doe",
"PrintOnCheckName": "Jane Lane Doe",
"Active": "true",
"PrimaryPhone": { "FreeFormNumber": "505.555.9999" },
"PrimaryEmailAddr": { "Address": "janedoe@example.com" },
"EmployeeType": "Regular",
"status": "Synchronized",
"Id": "ABC123",
"SyncToken": "1",
"MetaData": {
    "CreateTime": "2015-04-26T19:45:03Z",
    "LastUpdatedTime": "2015-04-27T21:48:23Z"
},
"PrimaryAddr": {
    "Line1": "123 Any Street",
    "City": "Any City",
    "CountrySubDivisionCode": "WA",
    "PostalCode": "01234"
}
},
"time": "2015-04-27T22:12:32.012Z"
}
```

### 輸入模型 (員工記錄範例)

以下是對應到員工記錄範例之原始 JSON 資料的輸入模型：

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "title": "EmployeeInputModel",
  "type": "object",
  "properties": {
    "QueryResponse": {
      "type": "object",
      "properties": {
        "maxResults": { "type": "string" },
        "startPosition": { "type": "string" },
        "Employee": {
          "type": "object",
          "properties": {
            "Organization": { "type": "string" },
            "Title": { "type": "string" },
            "GivenName": { "type": "string" },
            "MiddleName": { "type": "string" },
            "FamilyName": { "type": "string" },
            "DisplayName": { "type": "string" },
            "PrintOnCheckName": { "type": "string" },
            "Active": { "type": "string" },
            "PrimaryPhone": {
              "type": "object",
              "properties": {
                "FreeFormNumber": { "type": "string" }
              }
            },
            "PrimaryEmailAddr": {
              "type": "object",
              "properties": {
                "Address": { "type": "string" }
              }
            },
            "EmployeeType": { "type": "string" },
            "status": "Synchronized"
          }
        }
      }
    }
  }
}
```

```
        "status": { "type": "string" },
        "Id": { "type": "string" },
        "SyncToken": { "type": "string" },
        "MetaData": {
            "type": "object",
            "properties": {
                "CreateTime": { "type": "string" },
                "LastUpdatedTime": { "type": "string" }
            }
        },
        "PrimaryAddr": {
            "type": "object",
            "properties": {
                "Line1": { "type": "string" },
                "City": { "type": "string" },
                "CountrySubDivisionCode": { "type": "string" },
                "PostalCode": { "type": "string" }
            }
        }
    },
    "time": { "type": "string" }
}
```

### 輸入對應範本 (員工記錄範例)

以下是對應到員工記錄範例之原始 JSON 資料的輸入對應範本：

```
#set($inputRoot = $input.path('$'))
{
    "QueryResponse": {
        "maxResults": "$inputRoot.QueryResponse.maxResults",
        "startPosition": "$inputRoot.QueryResponse.startPosition",
        "Employee": {
            "Organization": "$inputRoot.QueryResponse.Employee.Organization",
            "Title": "$inputRoot.QueryResponse.Employee.Title",
            "GivenName": "$inputRoot.QueryResponse.Employee.GivenName",
            "MiddleName": "$inputRoot.QueryResponse.Employee.MiddleName",
            "FamilyName": "$inputRoot.QueryResponse.Employee.FamilyName",
            "DisplayName": "$inputRoot.QueryResponse.Employee.DisplayName",
            "PrintOnCheckName": "$inputRoot.QueryResponse.Employee.PrintOnCheckName",
            "Active": "$inputRoot.QueryResponse.Employee.Active",
            "PrimaryPhone": { "FreeFormNumber": "$inputRoot.QueryResponse.Employee.PrimaryPhone.FreeFormNumber" },
            "PrimaryEmailAddr": { "Address": "$inputRoot.QueryResponse.Employee.PrimaryEmailAddr.Address" },
            "EmployeeType": "$inputRoot.QueryResponse.Employee.EmployeeType",
            "status": "$inputRoot.QueryResponse.Employee.status",
            "Id": "$inputRoot.QueryResponse.Employee.Id",
            "SyncToken": "$inputRoot.QueryResponse.Employee.SyncToken",
            "MetaData": {
                "CreateTime": "$inputRoot.QueryResponse.Employee.MetaData.CreateTime",
                "LastUpdatedTime": "$inputRoot.QueryResponse.Employee.MetaData.LastUpdatedTime"
            },
            "PrimaryAddr": {
                "Line1": "$inputRoot.QueryResponse.Employee.PrimaryAddr.Line1",
                "City": "$inputRoot.QueryResponse.Employee.PrimaryAddr.City",
                "CountrySubDivisionCode": "$inputRoot.QueryResponse.Employee.PrimaryAddr.CountrySubDivisionCode",
                "PostalCode": "$inputRoot.QueryResponse.Employee.PrimaryAddr.PostalCode"
            }
        }
    }
}
```

```
        },
        "time": "$inputRoot.time"
    }
```

### 轉換後的資料 (員工記錄範例)

以下是如何轉換原始員工記錄範例 JSON 資料以供輸出的範例之一：

```
{
    "QueryResponse": {
        "maxResults": "1",
        "startPosition": "1",
        "Employees": [
            {
                "Title": "Mrs.",
                "GivenName": "Jane",
                "MiddleName": "Lane",
                "FamilyName": "Doe",
                "DisplayName": "Jane Lane Doe",
                "PrintOnCheckName": "Jane Lane Doe",
                "Active": "true",
                "PrimaryPhone": "505.555.9999",
                "Email": [
                    {
                        "type": "primary",
                        "Address": "janedoe@example.com"
                    }
                ],
                "EmployeeType": "Regular",
                "PrimaryAddr": {
                    "Line1": "123 Any Street",
                    "City": "Any City",
                    "CountrySubDivisionCode": "WA",
                    "PostalCode": "01234"
                }
            }
        ],
        "time": "2015-04-27T22:12:32.012Z"
    }
}
```

### 輸出模型 (員工記錄範例)

以下是對應到轉換後 JSON 資料格式的輸出模型：

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "title": "EmployeeOutputModel",
    "type": "object",
    "properties": {
        "QueryResponse": {
            "type": "object",
            "properties": {
                "maxResults": { "type": "string" },
                "startPosition": { "type": "string" },
                "Employees": {
                    "type": "array",
                    "items": {
                        "type": "object",
                        "properties": {
                            "Title": { "type": "string" },
                            "GivenName": { "type": "string" },
                            "MiddleName": { "type": "string" },
                            "FamilyName": { "type": "string" },
                            "DisplayName": { "type": "string" },
                            "PrintOnCheckName": { "type": "string" },
                            "Active": { "type": "boolean" },
                            "PrimaryPhone": { "type": "string" },
                            "Email": {
                                "type": "array",
                                "items": {
                                    "type": "object",
                                    "properties": {
                                        "type": { "type": "string" },
                                        "Address": { "type": "string" }
                                    }
                                }
                            },
                            "EmployeeType": { "type": "string" },
                            "PrimaryAddr": {
                                "type": "object",
                                "properties": {
                                    "Line1": { "type": "string" },
                                    "City": { "type": "string" },
                                    "CountrySubDivisionCode": { "type": "string" },
                                    "PostalCode": { "type": "string" }
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}
```

```
"MiddleName": { "type": "string" },
"FamilyName": { "type": "string" },
"DisplayName": { "type": "string" },
"PrintOnCheckName": { "type": "string" },
"Active": { "type": "string" },
"PrimaryPhone": { "type": "string" },
"Email": {
    "type": "array",
    "items": {
        "type": "object",
        "properties": {
            "type": { "type": "string" },
            "Address": { "type": "string" }
        }
    }
},
"EmployeeType": { "type": "string" },
"PrimaryAddr": {
    "type": "object",
    "properties": {
        "Line1": { "type": "string" },
        "City": { "type": "string" },
        "CountrySubDivisionCode": { "type": "string" },
        "PostalCode": { "type": "string" }
    }
}
},
"time": { "type": "string" }
}
```

### 輸出對應範本 (員工記錄範例)

以下是對應到轉換後 JSON 資料格式的輸出對應範本。此處的範本變數採用原始 JSON 資料格式，而不是轉換後的 JSON 資料格式：

```
#set($inputRoot = $input.path('$'))
{
    "QueryResponse": {
        "maxResults": "$inputRoot.QueryResponse.maxResults",
        "startPosition": "$inputRoot.QueryResponse.startPosition",
        "Employees": [
            {
                "Title": "$inputRoot.QueryResponse.Employee.Title",
                "GivenName": "$inputRoot.QueryResponse.Employee.GivenName",
                "MiddleName": "$inputRoot.QueryResponse.Employee.MiddleName",
                "FamilyName": "$inputRoot.QueryResponse.Employee.FamilyName",
                "DisplayName": "$inputRoot.QueryResponse.Employee.DisplayName",
                "PrintOnCheckName": "$inputRoot.QueryResponse.Employee.PrintOnCheckName",
                "Active": "$inputRoot.QueryResponse.Employee.Active",
                "PrimaryPhone": "$inputRoot.QueryResponse.Employee.PrimaryPhone.FreeFormNumber",
                "Email" : [
                    {
                        "type": "primary",
                        "Address": "$inputRoot.QueryResponse.Employee.PrimaryEmailAddr.Address"
                    }
                ],
                "EmployeeType": "$inputRoot.QueryResponse.Employee.EmployeeType",
                "PrimaryAddr": {
                    "Line1": "$inputRoot.QueryResponse.Employee.PrimaryAddr.Line1",
                    "City": "$inputRoot.QueryResponse.Employee.PrimaryAddr.City",
                    "CountrySubDivisionCode": "$inputRoot.QueryResponse.Employee.PrimaryAddr.CountrySubDivisionCode",
                    "PostalCode": "$inputRoot.QueryResponse.Employee.PrimaryAddr.PostalCode"
                }
            }
        ]
    }
}
```

```
"City": "$inputRoot.QueryResponse.Employee.PrimaryAddr.City",
"CountrySubDivisionCode":
"$inputRoot.QueryResponse.Employee.PrimaryAddr.CountrySubDivisionCode",
"PostalCode": "$inputRoot.QueryResponse.Employee.PrimaryAddr.PostalCode"
}
]
},
"time": "$inputRoot.time"
}
```

## Amazon API Gateway API 請求和回應資料對應參考

本節會說明如何從 API 方法請求資料設定將資料，包含使用 [context \(p. 274\)](#)、[stage \(p. 281\)](#) 或 [util \(p. 282\)](#) 變數存放的其他資料，對應到對應的整合請求參數；以及從整合回應資料設定將資料，包含其他資料，對應到方法回應參數。方法請求資料包含請求參數（路徑、查詢字串、標頭）和內文。整合回應資料包含回應參數（標頭）和內文。如需使用階段變數的詳細資訊，請參閱「[Amazon API Gateway 階段變數參考 \(p. 480\)](#)」。

### 主題

- 將方法請求資料對應到整合請求參數 ([p. 270](#))
- 將整合回應資料對應到方法回應標頭 ([p. 271](#))
- 對應方法和整合之間的請求和回應承載 ([p. 272](#))
- 整合傳遞行為 ([p. 273](#))

### 將方法請求資料對應到整合請求參數

格式為路徑變數、查詢字串或標頭的整合請求參數，可以從任何已定義的方法請求參數和承載對應。

在下表中，*PARAM\_NAME* 是指定參數類型的方法請求參數名稱。它必須符合規則表達式 '^[a-zA-Z0-9.\_\$-]+\$]'。它必須已先定義，您才能參考它。*JSONPath\_EXPRESSION* 是請求或回應內文之 JSON 欄位的 JSONPath 表達式。

#### Note

"\$" 字首在這個語法中予以省略。

#### 整合請求資料對應表達式

對應的資料來源	對應表達式
方法請求路徑	method.request.path. <i>PARAM_NAME</i>
方法請求查詢字串	method.request.querystring. <i>PARAM_NAME</i>
多值方法請求查詢字串	method.request.multivaluequerystring. <i>PARAM_NAME</i>
方法請求標頭	method.request.header. <i>PARAM_NAME</i>
多值方法請求標頭	method.request.multivalueheader. <i>PARAM_NAME</i>
方法請求內文	method.request.body
方法請求內文 (JsonPath)	method.request.body. <i>JSONPath_EXPRESSION</i> 。
階段變數	stageVariables. <i>VARIABLE_NAME</i>
環境變數	context. <i>VARIABLE_NAME</i> 必須是支援的環境變數 ( <a href="#">p. 274</a> )之一。

對應的資料來源	對應表達式
靜態值	' <b>STATIC_VALUE</b> '。 <b>STATIC_VALUE</b> 是字串常值，而且必須以一對單引號括住。

### Example 從 OpenAPI 的方法請求參數對應

下列範例顯示的 OpenAPI 程式碼片段會：

- 將名為 `methodRequestHeaderParam` 的方法請求標頭對應到名為 `integrationPathParam` 的整合請求路徑參數
- 將名為 `methodRequestQueryParam` 的多值方法請求查詢字串對應到名為 `integrationQueryParam` 的整合請求查詢字串

```
...
"requestParameters" : {
    "integration.request.path.integrationPathParam" :
    "method.request.header.methodRequestHeaderParam",
    "integration.request.querystring.integrationQueryParam" :
    "method.request.multivaluequerystring.methodRequestQueryParam"
}
...
```

您也可以使用 [JSONPath 表達式](#)從 JSON 請求內文中的欄位，對應整合請求參數。下表顯示方法請求內文的對應表達式及其 JSON 欄位。

### Example 從 OpenAPI 的方法請求內文對應

下列範例顯示的 OpenAPI 程式碼片段將 1) 方法請求內文對應到名為 `body-header` 的整合請求標頭；以及對應 2) 內文的 JSON 欄位，如 JSON 表達式所表示 (`petstore.pets[0].name`，無 `$.` 字首)。

```
...
"requestParameters" : {
    "integration.request.header.body-header" : "method.request.body",
    "integration.request.path.pet-name" : "method.request.body.petstore.pets[0].name",
}
...
```

## 將整合回應資料對應到方法回應標頭

您可以從任何整合回應標頭或整合回應內文、`$context` 變數或靜態值，對應方法回應標頭參數。

### 方法回應標頭對應表達式

對應的資料來源	對應表達式
整合回應標頭	<code>integration.response.header.PARAM_NAME</code>

對應的資料來源	對應表達式
整合回應標頭	integration.response.multivalueheader. <a href="#">PARAM_NAME</a>
整合回應內文	integration.response.body
整合回應內文 (JsonPath)	integration.response.body. <a href="#">JSONPath_EXPRESSION</a>
階段變數	stageVariables. <a href="#">VARIABLE_NAME</a>
環境變數	context. <a href="#">VARIABLE_NAME</a> 必須是 <a href="#">支援的環境變數</a> (p. 274)之一。
靜態值	' <a href="#">STATIC_VALUE</a> '。 <a href="#">STATIC_VALUE</a> 是字串常值，而且必須以一對單引號括住。

### Example 從 OpenAPI 整合回應對應的資料

下列範例顯示的 OpenAPI 程式碼片段將 1) 整合回應的 redirect.url，即 JSONPath 欄位對應到請求回應的 location 標頭；將 2) 整合回應的 x-app-id 標頭對應到方法回應的 id 標頭。

```
...
"responseParameters" : {
    "method.response.header.location" : "integration.response.body.redirect.url",
    "method.response.header.id" : "integration.response.header.x-app-id",
    "method.response.header.items" : "integration.response.multivalueheader.item",
}
...
```

## 對應方法和整合之間的請求和回應承載

API Gateway 使用 [Velocity 範本語言 \(VTL\)](#) 引擎來處理整合請求和整合回應的內文[對應範本](#) (p. 248)。對應範本會將方法請求承載轉譯為對應的整合請求承載，並將整合回應內文轉譯成方法回應內文。

VTL 範本使用 JSONPath 表達式、其他參數 (例如呼叫的環境與階段變數) 和公用程式函數來處理 JSON 資料。

如果模型定義成說明承載的資料結構，則 API Gateway 可以使用模型來產生整合請求或整合回應的骨架對應範本。您可以使用骨架範本協助自訂和擴展對應的 VTL 指令碼。不過，您可以從頭開始建立對應範本，且無須定義承載資料結構的模型。

### 選取 VTL 對應範本

API Gateway 使用下列邏輯來選取使用 [Velocity 範本語言 \(VTL\)](#) 的對應範本，將承載從方法請求對應到對應的整合請求，或將承載從整合回應對應到對應的方法回應。

針對請求承載，API Gateway 會將請求的 Content-Type 標頭值用作選取請求承載對應範本的金鑰。針對回應承載，API Gateway 會將傳入請求的 Accept 標頭值用作選取對應範本的金鑰。

當請求中沒有 Content-Type 標頭時，API Gateway 會假設其預設值為 application/json。對於這種請求，API Gateway 會將 application/json 用作預設金鑰來選取對應範本，如已有定義的對應範本。當沒有符合此金鑰的範本時，如果 `passthroughBehavior` 屬性設定為 WHEN\_NO\_MATCH 或 WHEN\_NO\_TEMPLATES，則 API Gateway 會傳送未映射的承載。

當請求中未指定 Accept 標頭時，API Gateway 會假設其預設值為 application/json。在這種情況下，API Gateway 會選取現有的 application/json 對應範本來對應回應承載。如果未針對 application/json 定義任何範本，則 API Gateway 會選取第一個現有的範本並使用它做為預設值來對應回應承載。同樣地，當指定的 Accept 標頭值不符合任何現有的範本金鑰時，API Gateway 會使用第一個現有的範本。如未定義任何範本，API Gateway 只會通過未對應的回應承載。

例如，假設 API 針對請求承載定義了 application/json 範本，且針對回應承載定義了 application/xml 範本。如果用戶端在請求中設定 "Content-Type : application/json" 和 "Accept : application/xml" 標頭，則請求和回應承載都會使用對應的對應範本來處理。如果沒有 Accept:application/xml 標頭，則會使用 application/xml 對應範本來對應回應承載。若要改傳回未對應的回應承載，您必須為 application/json 設定空的範本。

選取對應範本時，只有 MIME 類型是從 Accept 和 Content-Type 標頭使用。例如，"Content-Type: application/json; charset=UTF-8" 的標頭會有已選取 application/json 金鑰的請求範本。

## 整合傳遞行為

使用非代理整合，當方法請求傳送承載，但 Content-Type 標頭不符合任何指定的對應範本，或未定義任何對應範本時，您可以選擇將用戶端提供的請求承載透過整合請求傳送到後端，且不須轉換。這個過程稱為整合傳遞。

關於[代理整合 \(p. 202\)](#)，API Gateway 會將整個請求傳遞到您的後端，而您並無法修改傳遞行為。

傳入請求的實際傳遞行為，是由您在[整合請求設定 \(p. 242\)](#)期間針對指定對應範本所選擇的選項，以及用戶端在傳入請求中設定的內容類型標頭而決定。下列範例說明可能的傳遞行為。

**範例 1：**針對 application/json 內容類型在整合請求中定義的一個對應範本。

Content-Type 標頭\選取的傳遞選項	WHEN_NO_MATCH	WHEN_NO_TEMPLATE	NEVER
無 (application/json 預設值)	使用此範本轉換請求承載。	使用此範本轉換請求承載。	使用此範本轉換請求承載。
application/json	使用此範本轉換請求承載。	使用此範本轉換請求承載。	使用此範本轉換請求承載。
application/xml	請求承載未轉換，而且會依現狀傳送到後端。	請求遭到拒絕，回應為 HTTP 415 Unsupported Media Type。	請求遭到拒絕，回應為 HTTP 415 Unsupported Media Type。

**範例 2：**針對 application/xml 內容類型在整合請求中定義的一個對應範本。

Content-Type 標頭\選取的傳遞選項	WHEN_NO_MATCH	WHEN_NO_TEMPLATE	NEVER
無 (application/json 預設值)	請求承載未轉換，而且會依現狀傳送到後端。	請求遭到拒絕，回應為 HTTP 415 Unsupported Media Type。	請求遭到拒絕，回應為 HTTP 415 Unsupported Media Type。
application/json	請求承載未轉換，而且會依現狀傳送到後端。	請求遭到拒絕，回應為 HTTP 415 Unsupported Media Type。	請求遭到拒絕，回應為 HTTP 415 Unsupported Media Type。

Content-Type 標頭\選取的傳遞選項	<b>WHEN_NO_MATCH</b>	<b>WHEN_NO_TEMPLATE</b>	<b>NEVER</b>
application/xml	使用此範本轉換請求承載。	使用此範本轉換請求承載。	使用此範本轉換請求承載。

## API Gateway 映射範本和存取記錄變數參考

本節提供 Amazon API Gateway 定義之變數和函數的參考資訊，而這些變數和函數會與資料模型、授權方、映射範本及 CloudWatch 存取記錄搭配使用。如需如何使用這些變數和函數的詳細資訊，請參閱[the section called “建立模型與對應範本” \(p. 244\)](#)。

### 主題

- [適用於資料模型、授權方、映射範本及 CloudWatch 存取記錄的 \\$context 變數 \(p. 274\)](#)
- [\\$context 變數範本範例 \(p. 278\)](#)
- [僅適用於 CloudWatch 存取記錄的 \\$context 變數 \(p. 278\)](#)
- [\\$input 變數 \(p. 279\)](#)
- [\\$input 變數範本範例 \(p. 279\)](#)
- [\\$stageVariables \(p. 281\)](#)
- [\\$util 變數 \(p. 282\)](#)

### Note

若為 \$method 和 \$integration 變數，請參閱 [the section called “請求和回應資料對應參考” \(p. 270\)](#)。

### 適用於資料模型、授權方、映射範本及 CloudWatch 存取記錄的 \$context 變數

下列 \$context 變數可用於資料模型、授權方、映射範本及 CloudWatch 存取記錄中。

若為只能在 CloudWatch 存取記錄中使用的 \$context 變數，請參閱[the section called “僅適用於 CloudWatch 存取記錄的 \\$context 變數” \(p. 278\)](#)。

參數	描述
\$context.accountId	API 擁有者的 AWS 帳戶 ID。
\$context.apiId	API Gateway 指派給您 API 的識別符。
\$context.authorizer.claims. <i>property</i>	成功驗證方法發起人之後，從 Amazon Cognito 使用者集區中傳回之宣告的屬性。如需詳細資訊，請參閱 <a href="#">the section called “針對 REST API 使用 Cognito 使用者集區做為授權方” (p. 363)</a> 。
	<p><b>Note</b></p> <p>呼叫 \$context.authorizer.claims 會傳回 null。</p>
\$context.authorizer.principalId	與用戶端所傳送並從 API Gateway Lambda 授權方 (先前稱作自訂授權方) 所傳回之字符相關聯的主要使用者身分。如需詳細資訊，請參閱 <a href="#">the section called “使用 Lambda 授權方” (p. 348)</a> 。

參數	描述
<code>\$context.authorizer.property</code>	API Gateway Lambda 授權方函數所傳回 context 映射之指定索引鍵/值對的字串化值。例如，如果授權方傳回下列 context 映射：
	<pre>"context" : {     "key": "value",     "numKey": 1,     "boolKey": true }</pre>
	呼叫 <code>\$context.authorizer.key</code> 會傳回 "value" 字串、呼叫 <code>\$context.authorizer.numKey</code> 會傳回 "1" 字串，而呼叫 <code>\$context.authorizer.boolKey</code> 會傳回 "true" 字串。 如需詳細資訊，請參閱 <a href="#">the section called “使用 Lambda 授權方” (p. 348)</a> 。
<code>\$context.awsEndpointRequestId</code>	AWS 端點的請求 ID。
<code>\$context.domainName</code>	用來叫用 API 的完整網域名稱。這應該與傳入的 Host 標頭相同。
<code>\$context.domainPrefix</code>	<code>\$context.domainName</code> 的第一個標籤。這是通常用作發起人/客戶 ID。
<code>\$context.error.message</code>	包含 API Gateway 錯誤訊息的字串。此變數只能用於 Velocity 範本語言引擎無法處理的 <a href="#">GatewayResponse</a> 本文映射範本及存取記錄中的簡單變數替換。如需詳細資訊，請參閱 <a href="#">the section called “使用 CloudWatch 監控 WebSocket API 執行” (p. 568)</a> 及 <a href="#">the section called “設定閘道回應” (p. 237)</a> 。
<code>\$context.error.messageString</code>	<code>\$context.error.message</code> 的引用值，即 " <code>\$context.error.message</code> "。
<code>\$context.error.responseType</code>	<a href="#">GatewayResponse</a> 的類型。此變數只能用於 Velocity 範本語言引擎無法處理的 <a href="#">GatewayResponse</a> 本文映射範本及存取記錄中的簡單變數替換。如需詳細資訊，請參閱 <a href="#">the section called “使用 CloudWatch 監控 WebSocket API 執行” (p. 568)</a> 及 <a href="#">the section called “設定閘道回應” (p. 237)</a> 。
<code>\$context.error.validationErrorString</code>	字串，其中包含詳細的驗證錯誤訊息。
<code>\$context.extendedRequestId</code>	API Gateway 指派給 API 請求的延伸 ID，其中包含更有助於偵錯/疑難排解的資訊。
<code>\$context.httpMethod</code>	使用的 HTTP 方法。有效值包含：DELETE、GET、HEAD、OPTIONS、PATCH、POST 和 PUT。
<code>\$context.identity.accountId</code>	與請求相關聯的 AWS 帳戶 ID。

參數	描述
<code>\$context.identity.apiKey</code>	對於需要 API 金鑰的 API 方法，此變數是與方法請求相關聯的 API 金鑰。對於不需要 API 金鑰的方法，此變數為 null。如需詳細資訊，請參閱 <a href="#">the section called “使用 API 金鑰的用量計畫” (p. 397)</a> 。
<code>\$context.identity.apiKeyId</code>	與需要 API 金鑰之 API 請求相關聯的 API 金鑰 ID。
<code>\$context.identity.caller</code>	提出請求之發起人的委託人識別符。
<code>\$context.identity.cognitoAuthenticationProvider</code>	提出請求之發起人所使用的 Amazon Cognito 身分驗證提供者。只有在使用 Amazon Cognito 登入資料簽署請求時才能使用。如需詳細資訊，請參閱《Amazon Cognito 開發人員指南》中的 <a href="#">使用聯合身分</a> 。
<code>\$context.identity.cognitoAuthenticationType</code>	提出請求之發起人的 Amazon Cognito 身分驗證類型。只有在使用 Amazon Cognito 登入資料簽署請求時才能使用。
<code>\$context.identity.cognitoIdentityId</code>	提出請求之發起人的 Amazon Cognito 身分 ID。只有在使用 Amazon Cognito 登入資料簽署請求時才能使用。
<code>\$context.identity.cognitoIdentityPoolId</code>	提出請求之發起人的 Amazon Cognito 身分集區 ID。只有在使用 Amazon Cognito 登入資料簽署請求時才能使用。
<code>\$context.identity.principalOrgId</code>	AWS 組織 ID。
<code>\$context.identity.sourceIp</code>	對 API Gateway 提出請求之 TCP 連線的來源 IP 地址。  <b>Warning</b>  如果 X-Forwarded-For 標頭可能偽造，您不應該信任此數值。
<code>\$context.identity.user</code>	提出請求之使用者的委託人識別符。用於 Lambda 授權方。如需詳細資訊，請參閱 <a href="#">the section called “Amazon API Gateway Lambda 授權方的輸出” (p. 358)</a> 。
<code>\$context.identity.userAgent</code>	API 發起人的 User-Agent 標頭。
<code>\$context.identity.userArn</code>	身分驗證之後識別之有效使用者的 Amazon Resource Name (ARN)。如需詳細資訊，請參閱 <a href="https://docs.aws.amazon.com/IAM/latest/UserGuide/id_users.html">https://docs.aws.amazon.com/IAM/latest/UserGuide/id_users.html</a> 。
<code>\$context.path</code>	請求路徑。例如，對於 <code>https://rest-api-id.execute-api.{region}.amazonaws.com/{stage}/root/child</code> 的非代理請求 URL， <code>\$context.path</code> 值是 <code>/{stage}/root/child</code> 。
<code>\$context.protocol</code>	請求通訊協定，例如 HTTP/1.1。

參數	描述
<code>\$context.requestId</code>	API Gateway 指派給 API 請求的 ID。
<code>\$context.requestOverride.header.header_name</code>	請求標題會覆寫。如果此參數已經定義，它包含要使用的標頭 (而不是在 Integration Request (整合請求) 窗格中定義的 HTTP Headers (HTTP 標頭))。如需詳細資訊，請參閱 <a href="#">使用對應範本來覆寫 API 請求和回應參數和狀態碼 (p. 251)</a> 。
<code>\$context.requestOverride.path.path_name</code>	請求路徑會覆寫。如果此參數已經定義，它包含要使用的請求路徑 (而不是在 Integration Request (整合請求) 窗格中定義的 URL Path Parameters (URL 路徑參數))。如需詳細資訊，請參閱 <a href="#">使用對應範本來覆寫 API 請求和回應參數和狀態碼 (p. 251)</a> 。
<code>\$context.requestOverride.querystring.querystring_name</code>	請求查詢字串會覆寫。如果此參數已經定義，它包含要使用的請求查詢字串 (而不是在 Integration Request (整合請求) 窗格中定義的 URL Query String (URL 查詢字串))。如需詳細資訊，請參閱 <a href="#">使用對應範本來覆寫 API 請求和回應參數和狀態碼 (p. 251)</a> 。
<code>\$context.responseOverride.header.header_name</code>	回應標題會覆寫。如果此參數已經定義，它包含要傳回的標頭 (而不是在 Integration Response (整合回應) 窗格中定義為 Default mapping (預設映射) 的 Response header (回應標頭))。如需詳細資訊，請參閱 <a href="#">使用對應範本來覆寫 API 請求和回應參數和狀態碼 (p. 251)</a> 。
<code>\$context.responseOverride.status</code>	回應狀態碼會覆寫。如果此參數已經定義，它包含要傳回的狀態碼 (而不是在 Integration Response (整合回應) 窗格中定義為 Default mapping (預設映射) 的 Method response status (方法回應狀態))。如需詳細資訊，請參閱 <a href="#">使用對應範本來覆寫 API 請求和回應參數和狀態碼 (p. 251)</a> 。
<code>\$context.requestTime</code>	CLF 格式化請求時間 (dd/MMM/yyyy:HH:mm:ss +-hhmm)。
<code>\$context.requestTimeEpoch</code>	Epoch 格式化請求時間。
<code>\$context.resourceId</code>	API Gateway 指派給您資源的 ID。
<code>\$context.resourcePath</code>	您資源的路徑。例如，對於非代理請求 URI <code>https://rest-api-id.execute-api.{region}.amazonaws.com/{stage}/root/child</code> ， <code>\$context.resourcePath</code> 值是 <code>/root/child</code> 。如需詳細資訊，請參閱 <a href="#">教學：建置具有 HTTP 非代理整合的 API (p. 52)</a> 。
<code>\$context.stage</code>	API 請求的部署階段 (例如，Beta 或 Prod)。
<code>\$context.wafResponseCode</code>	從 AWS WAF 收到的回應：WAF_ALLOW 或 WAF_BLOCK。若該階段與 web ACL 不相關聯，將不會設定此值。如需詳細資訊，請參閱 <a href="#">the section called “使用 AWS WAF 來保護您的 API 避免受到常見的網路攻擊” (p. 396)</a> 。

參數	描述
\$context.webaclArn	Web ACL 的完整 ARN，用來決定是否允許或封鎖請求。若該階段與 web ACL 不相關聯，將不會設定此值。如需詳細資訊，請參閱 <a href="#">the section called “使用 AWS WAF 來保護您的 API 避免受到常見的網路攻擊” (p. 396)</a> 。
\$context.xrayTraceId	X-Ray 追蹤的追蹤 ID。如需詳細資訊，請參閱 <a href="#">the section called “設定 AWS X-Ray” (p. 519)</a> 。

## \$context 變數範本範例

如果您的 API 方法將結構化資料傳遞到需要資料採用特定格式的後端，建議您使用 \$context 變數。

以下範例顯示一個映射範本，其會將傳入的 \$context 變數映射至整合請求承載中名稱稍有不同的後端變數：

**Note**

請注意，其中一個變數是 API 金鑰。此範例假設方法已啟用「需要 API 金鑰」。

```
{
    "stage" : "$context.stage",
    "request_id" : "$context.requestId",
    "api_id" : "$context.apiId",
    "resource_path" : "$context.resourcePath",
    "resource_id" : "$context.resourceId",
    "http_method" : "$context.httpMethod",
    "source_ip" : "$context.identity.sourceIp",
    "user-agent" : "$context.identity.userAgent",
    "account_id" : "$context.identity.accountId",
    "api_key" : "$context.identity.apiKey",
    "caller" : "$context.identity.caller",
    "user" : "$context.identity.user",
    "user_arn" : "$context.identity.userArn"
}
```

## 僅適用於 CloudWatch 存取記錄的 \$context 變數

以下 \$context 變數僅適用於 CloudWatch 存取記錄。如需詳細資訊，請參閱[the section called “設定 CloudWatch API 記錄” \(p. 472\)](#)。(若為 WebSocket API，請參閱[the section called “使用 CloudWatch 監控 WebSocket API 執行” \(p. 568\)](#)。)

參數	描述
\$context.authorizer.integrationLatency	授權方延遲 (以毫秒為單位)。
\$context.integrationLatency	整合延遲 (以毫秒為單位)。
\$context.integrationStatus	整合回應狀態。
\$context.responseLatency	回應延遲 (以毫秒為單位)。
\$context.responseLength	回應承載長度。
\$context.status	方法回應狀態。

## \$input 變數

\$input 變數代表映射範本要處理的方法請求承載和參數。它提供四個函數：

變數和函數	描述
\$input.body	傳回原始請求承載做為字串。
\$input.json(x)	此函數會評估 JSONPath 表達式，並傳回結果作為 JSON 字串。  例如，\$input.json('\$.pets') 會傳回代表 pets 結構的 JSON 字串。  如需 JSONPath 的詳細資訊，請參閱 <a href="#">JSONPath</a> 或 <a href="#">適用於 Java 的 JSONPath</a> 。
\$input.params()	傳回所有請求參數的映射。
\$input.params(x)	從路徑、查詢字串或標頭值（以此順序搜尋）傳回方法請求參數值，而參數名稱字串為 x。
\$input.path(x)	採用 JSONPath 表達式字串 (x)，並傳回結果的 JSON 物件呈現。這可讓您存取和運用 <a href="#">Apache Velocity 範本語言 (VTL)</a> 中原生承載的元素。  例如，如果表達式 \$input.path('\$.pets') 傳回如下物件：
	<pre>[   {     "id": 1,     "type": "dog",     "price": 249.99   },   {     "id": 2,     "type": "cat",     "price": 124.99   },   {     "id": 3,     "type": "fish",     "price": 0.99   }]</pre> \$input.path('\$.pets').count() 會傳回 "3"。  如需 JSONPath 的詳細資訊，請參閱 <a href="#">JSONPath</a> 或 <a href="#">適用於 Java 的 JSONPath</a> 。

## \$input 變數範本範例

### 參數對應範本範例

下列參數對應範例會透過 JSON 承載將所有參數（包含 path、querystring 和 header）傳遞至整合端點

```
#set($allParams = $input.params())
{
    "params" : {
        #foreach($type in $allParams.keySet())
        #set($params = $allParams.get($type))
        "$type" : {
            #foreach($paramName in $params.keySet())
            "$paramName" : "$util.escapeJavaScript($params.get($paramName))"
            #if($foreach.hasNext),#end
            #end
        }
        #if($foreach.hasNext),#end
        #end
    }
}
```

事實上，此映射範本會輸出承載中的所有請求參數，如下所示：

```
{
    "parameters" : {
        "path" : {
            "path_name" : "path_value",
            ...
        }
        "header" : {
            "header_name" : "header_value",
            ...
        }
        "querystring" : {
            "querystring_name" : "querystring_value",
            ...
        }
    }
}
```

建議您使用 `$input` 變數來取得查詢字串和請求內文，而不論是否使用模型。也建議您將參數和承載或是承載子區段放入 Lambda 函數。下列範例示範其做法。

### 使用 `$input` 的範例 JSON 映射範本

下列範例示範如何使用映射來讀取查詢字串中的名稱，然後在元素中包含整個 POST 內文：

```
{
    "name" : "$input.params('name')",
    "body" : $input.json('$')
}
```

如果 JSON 輸入包含 JavaScript 無法剖析的非逸出字元，則可能會傳回 400 回應。套用上方的 `$util.escapeJavaScript($input.json('$'))` 可確保正確地剖析 JSON 輸入。

### 使用 `$input` 的範例映射範本

下列範例示範如何將 JSONPath 表達式傳遞給 `json()` 方法。您也可以使用後接屬性名稱的句點 (.)，來讀取請求內文物件的特定屬性：

```
{
    "name" : "$input.params('name')",
    "body" : $input.json('$.mykey')
}
```

如果方法請求承載包含 JavaScript 無法剖析的非逸出字元，您可能會取得 400 回應。在此情況下，您需要在映射範本中呼叫 `$util.escapeJavaScript()` 函數，如下所示：

```
{  
    "name" : "$input.params('name')",  
    "body" : $util.escapeJavaScript($input.json('$mykey'))  
}
```

### 使用 `$input` 的範例請求和回應

下列範例使用這三個函數：

Request Template: (請求範本：)

```
Resource: /things/{id}  
  
With input template:  
{  
    "id" : "$input.params('id')",  
    "count" : "$input.path('.things').size()",  
    "things" : $util.escapeJavaScript($input.json('.things'))  
}  
  
POST /things/abc  
{  
    "things" : {  
        "1" : {},  
        "2" : {},  
        "3" : {}  
    }  
}
```

回應：

```
{  
    "id": "abc",  
    "count": "3",  
    "things": {  
        "1": {},  
        "2": {},  
        "3": {}  
    }  
}
```

如需其他映射範例，請參閱「[針對請求與回應對應建立模型與對應範本 \(p. 244\)](#)」。

### `$stageVariables`

階段變數可以用於參數映射和映射範本，並做為方法整合中使用之 ARN 和 URL 的預留位置。如需詳細資訊，請參閱[the section called “設定 REST API 的階段變數” \(p. 475\)](#)。

語法	描述
<code>\$stageVariables.&lt;variable_name&gt;</code>	<code>&lt;variable_name&gt;</code> 代表階段變數名稱。
<code>\$stageVariables['&lt;variable_name&gt;']</code>	<code>&lt;variable_name&gt;</code> 代表任何階段變數名稱。
<code>\$(stageVariables['&lt;variable_name&gt;'])</code>	<code>&lt;variable_name&gt;</code> 代表任何階段變數名稱。

## \$util 變數

\$util 變數包含要在映射範本中使用的公用程式函數。

### Note

除非特別指定，否則預設字元集為 UTF-8。

函數	描述
<code>\$util.escapeJavaScript()</code>	<p>使用 JavaScript 字串規則來逸出字串中的字元。</p> <p><b>Note</b></p> <p>此函數會將任何一般單引號 ('') 轉換為逸出單引號 (\')。不過，逸出單引號不適用於 JSON。因此，將此函數的輸出用於 JSON 屬性時，您必須將任何逸出單引號 (\') 轉換為一般單引號 ('')。下列範例顯示這種情況：</p> <pre>\$util.escapeJavaScript(<b>data</b>).replaceAll("\\", "'")</pre>
<code>\$util.parseJson()</code>	<p>採用「字串化」JSON，並傳回結果的物件呈現。您可以使用此函數的結果，來存取和運用 Apache Velocity 範本語言 (VTL) 中原生承載的元素。例如，如果您有下列承載：</p> <pre>{"errorMessage": "{\"key1\": \"var1\", \"key2\": {\"arr\": [1, 2, 3]}}"}</pre> <p>並使用下列映射範本</p> <pre>#set (\$errorMessageObj = \$util.parseJson(\$input.path('\$.errorMessage'))) {     "errorMessageObjKey2ArrVal" :     \$errorMessageObj.key2.arr[0] }</pre> <p>您將會收到下列輸出：</p> <pre>{     "errorMessageObjKey2ArrVal" : 1 }</pre>
<code>\$util.urlEncode()</code>	將字串轉換為 "application/x-www-form-urlencoded" 格式。
<code>\$util.urlDecode()</code>	解碼 "application/x-www-form-urlencoded" 字串。
<code>\$util.base64Encode()</code>	將資料編碼為 base64 編碼字串。
<code>\$util.base64Decode()</code>	解碼 base64 編碼字串中的資料。

## 支援 API Gateway 中的二進位承載

在 API Gateway 中，API 請求與回應可以有文字或二進位承載。文字承載是 UTF-8 編碼的 JSON 字串，而二進位承載是文字承載以外的任何項目。例如，二進位承載可以是 JPEG 檔案、GZip 檔案或 XML 檔案。

根據預設，API Gateway 會將訊息本文視為文字承載，並套用任何預先設定的對應範本，來轉換 JSON 字串。如果未指定對應範本，API Gateway 可以將文字承載傳遞至整合端點或從中傳出而不進行任何修改，但前提是已在 API 方法上啟用傳遞行為。對於二進位承載，API Gateway 只會依現狀傳遞訊息。

若要讓 API Gateway 傳遞二進位承載，您可以將媒體類型新增至 RestApi 資源的 `binaryMediaTypes` 清單，或設定 `Integration` 與 `IntegrationResponse` 資源上的 `contentHandling` 屬性。`contentHandling` 值可以是 `CONVERT_TO_BINARY`、`CONVERT_TO_TEXT` 或未定義。根據 `contentHandling` 值，以及回應的 `Content-Type` 標頭或傳入請求的 `Accept` 標頭是否符合 `binaryMediaTypes` 清單中的項目，API Gateway 可以將原始二進位位元組編碼為 Base64 編碼字串、將 Base64 編碼字串解碼回其原始位元組，或傳遞本文而不進行任何修改。

您必須遵循下列方式設定 API，以在 API Gateway 中支援 API 的二進位承載：

- 將所需的二進位媒體類型新增至 RestApi 資源中的 `binaryMediaTypes` 清單。如果未定義此屬性與 `contentHandling` 屬性，則會將承載當作 UTF-8 編碼的 JSON 字串來處理。
- 將 `Integration` 資源的 `contentHandling` 屬性設定為 `CONVERT_TO_BINARY`，讓請求承載可從 Base64 編碼字串轉換成其二進位 Blob；或將屬性設定為 `CONVERT_TO_TEXT`，讓請求承載可從二進位 Blob 轉換成 Base64 編碼字串。如果未定義此屬性，API Gateway 會傳遞承載而不進行任何修改。當 `Content-Type` 標頭值符合其中一個 `binaryMediaTypes` 項目，而且同時啟用了 API 的 [傳遞行為 \(p. 273\)](#) 時，就會發生此情況。
- 將 `IntegrationResponse` 資源的 `contentHandling` 屬性設定為 `CONVERT_TO_BINARY`，讓回應承載可從 Base64 編碼字串轉換成其二進位 Blob；或將屬性設定為 `CONVERT_TO_TEXT`，讓回應承載可從二進位 Blob 轉換成 Base64 編碼字串。如果未定義 `contentHandling`，而且回應的 `Content-Type` 標頭與原始請求的 `Accept` 標頭符合 `binaryMediaTypes` 清單的項目，API Gateway 會傳遞本文。當 `Content-Type` 標頭與 `Accept` 標頭相同時，就會發生此情況；否則，API Gateway 會將回應本文轉換成 `Accept` 標頭中指定的類型。

### 主題

- [API Gateway 中的內容類型轉換 \(p. 283\)](#)
- [使用 API Gateway 主控台啟用二進位支援 \(p. 285\)](#)
- [使用 API Gateway REST API 啟用二進位支援 \(p. 289\)](#)
- [匯入與匯出內容編碼 \(p. 292\)](#)
- [二進位支援的範例 \(p. 293\)](#)

## API Gateway 中的內容類型轉換

下表顯示 API Gateway 如何轉換請求的 `Content-Type` 標頭、RestApi 資源的 `binaryMediaTypes` 清單與 Integration 資源的 `contentHandling` 屬性值之特定組態的請求承載。

### API Gateway 中的 API 請求內容類型轉換

方法請求承載	請求 Content-Type 標頭	<code>binaryMediaType</code>	<code>contentHandling</code>	整合請求承載
文字資料	任何資料類型	未定義	未定義	UTF8 編碼字串
文字資料	任何資料類型	未定義	<code>CONVERT_TO_BINARY</code>	Base64 解碼的二進位 Blob

方法請求承載	請求 Content-Type 標頭	<code>binaryMediaType</code>	<code>contentHandling</code>	整合請求承載
文字資料	任何資料類型	未定義	CONVERT_TO_TEXT	UTF8 編碼字串
文字資料	文字資料類型	設定相符媒體類型	未定義	文字資料
文字資料	文字資料類型	設定相符媒體類型	CONVERT_TO_BINARY	Base64 解碼的二進位 Blob
文字資料	文字資料類型	設定相符媒體類型	CONVERT_TO_TEXT	文字資料
二進位資料	二進位資料類型	設定相符媒體類型	未定義	二進位資料
二進位資料	二進位資料類型	設定相符媒體類型	CONVERT_TO_BINARY	二進位資料
二進位資料	二進位資料類型	設定相符媒體類型	CONVERT_TO_TEXT	Base64 編碼字串

下表顯示 API Gateway 如何轉換請求的 `Accept` 標頭、[RestAPI](#) 資源的 `binaryMediaTypes` 清單與 [IntegrationResponse](#) 資源的 `contentHandling` 屬性值之特定組態的回應承載。

#### API Gateway 回應內容類型轉換

整合回應承載	請求 Accept 標頭	<code>binaryMediaType</code>	<code>contentHandling</code>	方法回應承載
文字或二進位資料	文字類型	未定義	未定義	UTF8 編碼字串
文字或二進位資料	文字類型	未定義	CONVERT_TO_BINARY	Base64 解碼的 Blob
文字或二進位資料	文字類型	未定義	CONVERT_TO_TEXT	UTF8 編碼字串
文字資料	文字類型	設定相符媒體類型	未定義	文字資料
文字資料	文字類型	設定相符媒體類型	CONVERT_TO_BINARY	Base64 解碼的 Blob
文字資料	文字類型	設定相符媒體類型	CONVERT_TO_TEXT	UTF8 編碼字串
文字資料	二進位類型	設定相符媒體類型	未定義	Base64 解碼的 Blob
文字資料	二進位類型	設定相符媒體類型	CONVERT_TO_BINARY	Base64 解碼的 Blob
文字資料	二進位類型	設定相符媒體類型	CONVERT_TO_TEXT	UTF8 編碼字串
二進位資料	文字類型	設定相符媒體類型	未定義	Base64 編碼字串
二進位資料	文字類型	設定相符媒體類型	CONVERT_TO_BINARY	二進位資料
二進位資料	文字類型	設定相符媒體類型	CONVERT_TO_TEXT	Base64 編碼字串
二進位資料	二進位類型	設定相符媒體類型	未定義	二進位資料
二進位資料	二進位類型	設定相符媒體類型	CONVERT_TO_BINARY	二進位資料
二進位資料	二進位類型	設定相符媒體類型	CONVERT_TO_TEXT	Base64 編碼字串

Tip

當請求在其 `Accept` 標頭中包含多個媒體類型時，API Gateway 只會採用第一個 `Accept` 媒體類型。如果您無法控制 `Accept` 媒體類型的順序，而且二進位內容的媒體類型不是清單中的第一個類型，您可以在 API 的 `Accept` 清單中新增第一個 `binaryMediaTypes` 媒體類型，API Gateway 將會傳回二進位格式的內容。例如，若要在瀏覽器中使用 `<img>` 元素來傳送 JPEG 檔案，瀏覽器可能會在請求中傳送 `Accept:image/webp,image/*,*/*;q=0.8`。透過將 `image/webp` 新增至 `binaryMediaTypes` 清單，端點就會收到二進位格式的 JPEG 檔案。

將文字承載轉換成二進位 Blob 時，API Gateway 會假設文字資料是 Base64 編碼字串，並將二進位資料輸出為 Base64 解碼的 Blob。如果轉換失敗，它會傳回 500 回應，表示 API 組態錯誤。您不會為這類轉換提供對應範本，但您必須在 API 上啟用 [傳遞行為 \(p. 273\)](#)。

將二進位承載轉換成文字字串時，API Gateway 一律會在二進位資料上套用 Base64 編碼。您可以為這類承載定義對應範本，但只能透過 `$input.body` 存取對應範本中的 Base64 編碼字串，如下列範例對應範本摘要所示。

```
{  
    "data": "$input.body"  
}
```

若要傳遞二進位承載而不進行任何修改，您必須在 API 上啟用 [傳遞行為 \(p. 273\)](#)。

## 使用 API Gateway 主控台啟用二進位支援

本節說明如何使用 API Gateway 主控台來啟用二進位支援。舉例來說，我們使用與 Amazon S3 整合的 API。我們將重點放在設定支援的媒體類型，以及指定應該如何處理承載的作業上。如需如何建立與 Amazon S3 整合之 API 的詳細資訊，請參閱教學：[在 API Gateway 中建立 REST API 做為 Amazon S3 代理 \(p. 104\)](#)。

### 使用 API Gateway 主控台啟用二進位支援

1. 設定 API 的二進位媒體類型：

- a. 建立新的 API 或選擇現有的 API。在此範例中，我們將 API 命名為 `FileMan`。
- b. 在主導覽面板中所選取的 API 下，選擇 `Settings` (設定)。
- c. 在 `Settings` (設定) 窗格的 `Binary Media Types` (二進位媒體類型) 區段中，選擇 `Add Binary Media Type` (新增二進位媒體類型)。
- d. 在輸入文字欄位中，輸入必要的媒體類型，例如 `image/png`。如果需要，請重複此步驟來新增更多媒體類型。
- e. 選擇 `Save Changes` (儲存變更)。

## Settings

Configure settings for your API deployments

### API Key Source

Choose the source of your API Keys from incoming requests. Configure deployments to receive API keys from the X-APIKEY\_HEADER or from a Custom Authorizer

API Key Source

### Content Encoding

Allow compression of response bodies based on client's Accept-Encoding header. Compression can be customized to be enabled above a certain threshold response body size. The following compression types are supported: gzip, deflate, and identity.

Content Encoding enabled

### Binary Media Types

You can configure binary support for your API by specifying which media types should be treated as binary types. API Gateway will look at the Content-Type and Accept HTTP headers to decide how to handle the body.

## 2. 設定如何處理 API 方法的訊息承載：

- a. 在 API 中建立新的資源或選擇現有的資源。在此範例中，我們使用 /{folder}/{item} 資源。
- b. 在資源上建立新的方法或選擇現有的方法。舉例來說，我們使用與 Amazon S3 中 Object GET 動作整合的 GET /{folder}/{item} 方法。
- c. 在 Content Handling (內容處理) 中，選擇一個選項。

## /{{folder}}/{{item}} - GET - Setup

Choose the integration point for your new method.

Integration type  Lambda Function i

HTTP i

Mock i

AWS Service i

AWS Region

AWS Service

AWS Subdomain

HTTP method

Action Type  Use action name

Use path override

Path override (optional)

Execution role

i

Content Handling

**Save**

如果您不想要在用戶端與後端接受相同的二進位格式時轉換本文，請選擇 Passthrough (傳遞)。例如，當後端需要二進位請求承載以 JSON 屬性傳入時，選擇 Convert to text (if needed) (轉換成為文字 (如果需要))，以將二進位本文轉換成 Base64 編碼字串。此外，當用戶端提交 Base64 編碼字串且後端需要原始二進位格式時，或是當端點傳回 Base64 編碼字串且用戶端只接受二進位輸出時，選擇 Convert to binary (if needed) (轉換成二進位 (如果需要))。

- d. 在整合請求中保留傳入請求的 Accept 標頭。如果您將 contentHandling 設定為 passthrough 並想要在執行階段覆寫該設定，則應該這麼做。

▼ HTTP Headers

Name	Mapped from ⓘ	Caching	
Accept	method.request.header.Accept	<input type="checkbox"/>	 
Content-Type	method.request.header.Content-Type	<input type="checkbox"/>	 
<a href="#">+ Add header</a>			

- e. 在請求本文中啟用傳遞行為。

▼ Body Mapping Templates

Request body passthrough  When no template matches the request Content-Type header ⓘ

When there are no templates defined (recommended) ⓘ

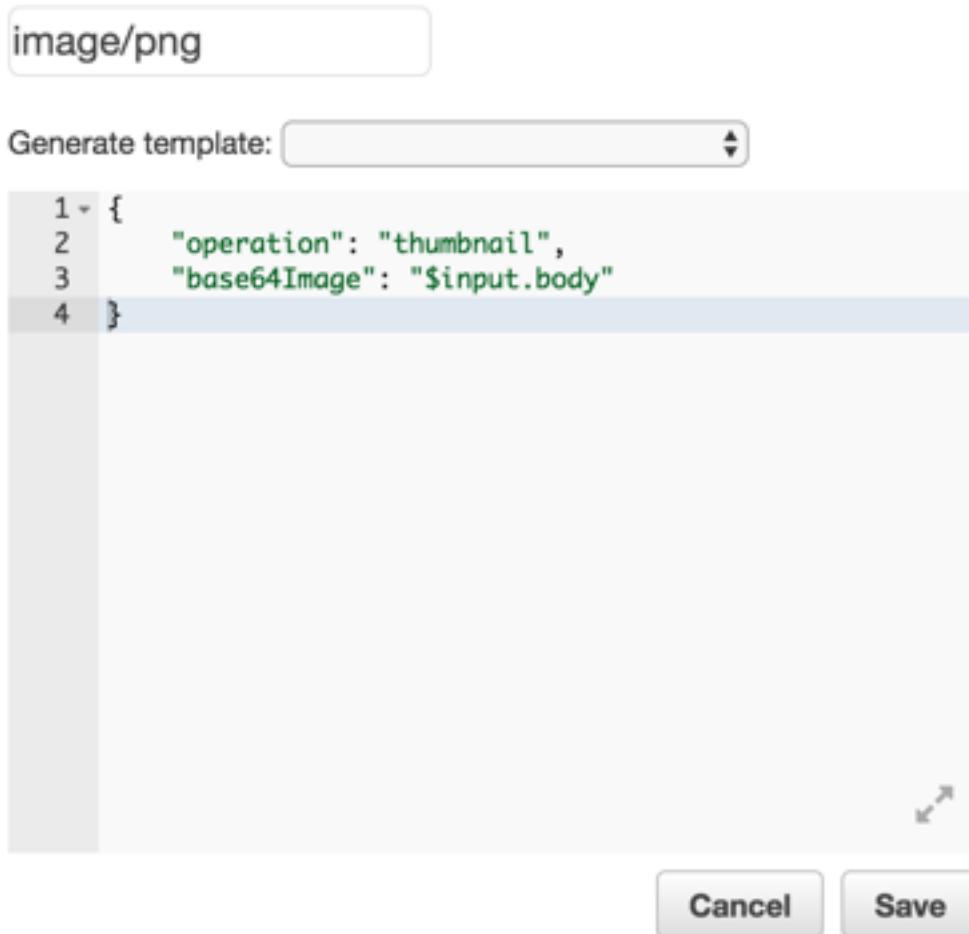
Never ⓘ

Content-Type

image/png

[+ Add mapping template](#)

- f. 若要轉換成文字，請定義對應範本，將 Base64 編碼的二進位資料設為必要的格式。



此對應範本的格式取決於輸入的端點需求。

## 使用 API Gateway REST API 啟用二進位支援

下列作業示範如何使用 API Gateway REST API 呼叫來啟用二進位支援。

### 主題

- [將支援的二進位媒體類型新增與更新至 API \(p. 289\)](#)
- [設定請求承載轉換 \(p. 290\)](#)
- [設定回應承載轉換 \(p. 290\)](#)
- [將二進位資料轉換成文字資料 \(p. 291\)](#)
- [將文字資料轉換成二進位承載 \(p. 291\)](#)
- [傳遞二進位承載 \(p. 292\)](#)

### 將支援的二進位媒體類型新增與更新至 API

若要讓 API Gateway 支援新的二進位媒體類型，您必須將二進位媒體類型新增至 RestApi 資源的 binaryMediaTypes 清單。例如，若要讓 API Gateway 處理 JPEG 影像，請對 RestApi 資源提交 PATCH 請求：

```
PATCH /restapis/<restapi_id>

{
  "patchOperations" : [ {
    "op" : "add",
    "path" : "/binaryMediaTypes/image-1jpeg"
  }
]
}
```

image/jpeg 的 MIME 類型規格是 path 屬性值的一部分，因此會逸出為 image-1jpeg。

若要更新支援的二進位媒體類型，請從 binaryMediaTypes 資源的 RestApi 清單中取代或移除媒體類型。例如，若要將二進位支援從 JPEG 檔案變更為原始位元組，請對 PATCH 資源提交 RestApi 請求，如下所示。

```
PATCH /restapis/<restapi_id>

{
  "patchOperations" : [ {
    "op" : "replace",
    "path" : "/binaryMediaTypes/image-1jpeg",
    "value" : "application/octet-stream"
  },
  {
    "op" : "remove",
    "path" : "/binaryMediaTypes/image-1jpeg"
  ]
}
```

## 設定請求承載轉換

如果端點需要二進位輸入，請將 contentHandling 資源的 Integration 屬性設定為 CONVERT\_TO\_BINARY。若要執行這項操作，請提交 PATCH 請求，如下所示：

```
PATCH /restapis/<restapi_id>/resources/<resource_id>/methods/<http_method>/integration

{
  "patchOperations" : [ {
    "op" : "replace",
    "path" : "/contentHandling",
    "value" : "CONVERT_TO_BINARY"
  }
}
```

## 設定回應承載轉換

如果用戶端接受二進位 Blob 格式的結果，而不是從端點傳回的 Base64 編碼承載，請透過提交 contentHandling 請求，將 IntegrationResponse 資源的 CONVERT\_TO\_BINARY 屬性設定為 PATCH，如下所示：

```
PATCH /restapis/<restapi_id>/resources/<resource_id>/methods/<http_method>/integration/
responses/<status_code>

{
  "patchOperations" : [ {
    "op" : "replace",
    "path" : "/contentHandling",
    "value" : "CONVERT_TO_BINARY"
  }
]
```

```
    }]
}
```

## 將二進位資料轉換成文字資料

若要透過 API Gateway 將二進位資料當做輸入的 JSON 屬性傳送至 AWS Lambda 或 Kinesis，請執行下列操作：

1. 將 `application/octet-stream` 的新二進位媒體類型新增至 API 的 `binaryMediaTypes` 清單，以啟用 API 的二進位承載支援。

```
PATCH /restapis/<restapi_id>

{
  "patchOperations" : [ {
    "op" : "add",
    "path" : "/binaryMediaTypes/application-octet-stream"
  }
]
}
```

2. 在 `CONVERT_TO_TEXT` 資源的 `contentHandling` 屬性上設定 `Integration`，並提供對應範本，以將二進位資料的 Base64 編碼字串指派給 JSON 屬性。在下列範例中，JSON 屬性是持有 Base64 編碼字串的 `body` 與 `$input.body`。

```
PATCH /restapis/<restapi_id>/resources/<resource_id>/methods/<http_method>/integration

{
  "patchOperations" : [
    {
      "op" : "replace",
      "path" : "/contentHandling",
      "value" : "CONVERT_TO_TEXT"
    },
    {
      "op" : "add",
      "path" : "/requestTemplates/application-octet-stream",
      "value" : "{\"body\": \"$input.body\"}"
    }
  ]
}
```

## 將文字資料轉換成二進位承載

假設 Lambda 函數會傳回 Base64 編碼字串格式的影像檔。若要透過 API Gateway 將此二進位輸出傳遞給用戶端，請執行下列操作：

1. 新增 `binaryMediaTypes` 的二進位媒體類型 (如果尚未在清單中)，以更新 API 的 `application/octet-stream` 清單。

```
PATCH /restapis/<restapi_id>

{
  "patchOperations" : [ {
    "op" : "add",
    "path" : "/binaryMediaTypes/application-octet-stream",
  }
]
```

2. 將 contentHandling 資源上的 Integration 屬性設定為 CONVERT\_TO\_BINARY。請勿定義對應範本。當您未定義對應範本時，API Gateway 可呼叫傳遞範本，將 Base64 解碼的二進位 Blob 做為影像檔傳回至用戶端。

```
PATCH /restapis/<restapi_id>/resources/<resource_id>/methods/<http_method>/integration/responses/<status_code>

{
  "patchOperations" : [
    {
      "op" : "replace",
      "path" : "/contentHandling",
      "value" : "CONVERT_TO_BINARY"
    }
  ]
}
```

## 傳遞二進位承載

若要使用 API Gateway 將影像存放在 Amazon S3 儲存貯體中，請執行下列操作：

1. 新增 binaryMediaTypes 的二進位媒體類型 (如果尚未在清單中)，以更新 API 的 application/octet-stream 清單。

```
PATCH /restapis/<restapi_id>

{
  "patchOperations" : [ {
    "op" : "add",
    "path" : "/binaryMediaTypes/application~loctet~stream"
  }
]
```

2. 在 contentHandling 資源的 Integration 屬性上設定 CONVERT\_TO\_BINARY。將 WHEN\_NO\_MATCH 設定為 passthroughBehavior 屬性值，而不需要定義對應範本。這可讓 API Gateway 呼叫傳遞範本。

```
PATCH /restapis/<restapi_id>/resources/<resource_id>/methods/<http_method>/integration

{
  "patchOperations" : [
    {
      "op" : "replace",
      "path" : "/contentHandling",
      "value" : "CONVERT_TO_BINARY"
    },
    {
      "op" : "replace",
      "path" : "/passthroughBehaviors",
      "value" : "WHEN_NO_MATCH"
    }
  ]
}
```

## 匯入與匯出內容編碼

若要匯入 RestApi 上的 binaryMediaTypes 清單，請使用 API OpenAPI 定義檔的下列 API Gateway 延伸。此延伸也可用來匯出 API 設定。

- [x-amazon-apigateway-binary-media-types 屬性 \(p. 538\)](#)

若要匯入與匯出 Integration 或 IntegrationResponse 資源上的 contentHandling 屬性值，請使用 OpenAPI 定義的下列 API Gateway 延伸：

- [x-amazon-apigateway-integration 物件 \(p. 542\)](#)
- [x-amazon-apigateway-integration.response 物件 \(p. 546\)](#)

## 二進位支援的範例

下列範例示範如何透過 API Gateway API 存取 Amazon S3 或 AWS Lambda 中的二進位檔案。OpenAPI 檔案中有此範例 API。此程式碼範例使用 API Gateway REST API 呼叫。

### 主題

- [透過 API Gateway API 存取 Amazon S3 中的二進位檔案 \(p. 293\)](#)
- [使用 API Gateway API 存取 Lambda 中的二進位檔案 \(p. 298\)](#)

## 透過 API Gateway API 存取 Amazon S3 中的二進位檔案

下列範例示範如何使用 OpenAPI 檔案來存取 Amazon S3 中的影像、如何從 Amazon S3 下載影像，以及如何將影像上傳至 Amazon S3。

### 主題

- [存取 Amazon S3 中影像之範例 API 的 OpenAPI 檔案 \(p. 293\)](#)
- [從 Amazon S3 下載影像 \(p. 297\)](#)
- [將影像上傳至 Amazon S3 \(p. 298\)](#)

## 存取 Amazon S3 中影像之範例 API 的 OpenAPI 檔案

下列 OpenAPI 檔案顯示一個範例 API，說明如何從 Amazon S3 下載影像檔，以及如何將影像檔上傳至 Amazon S3。此 API 會公開用於下載及上傳指定影像檔的 GET /s3?key={file-name} 與 PUT /s3?key={file-name} 方法。GET 方法會在「200 OK」回應中，將 Base64 編碼字串格式的影像檔當作 JSON 輸出的一部分傳回，後面接著所提供的對應範本。PUT 方法接受原始二進位 Blob 作為輸入，並傳回「200 OK」回應與空的承載。

### OpenAPI 3.0

```
{  
  "openapi": "3.0.0",  
  "info": {  
    "version": "2016-10-21T17:26:28Z",  
    "title": "ApiName"  
  },  
  "paths": {  
    "/s3": {  
      "get": {  
        "parameters": [  
          {  
            "name": "key",  
            "in": "query",  
            "required": false,  
            "schema": {  
              "type": "string"  
            }  
          }  
        ]  
      }  
    }  
  }  
}
```

```
        }
    ],
    "responses": {
        "200": {
            "description": "200 response",
            "content": {
                "application/json": {
                    "schema": {
                        "$ref": "#/components/schemas/Empty"
                    }
                }
            }
        },
        "500": {
            "description": "500 response"
        }
    },
    "x-amazon-apigateway-integration": {
        "credentials": "arn:aws:iam::123456789012:role/binarySupportRole",
        "responses": {
            "default": {
                "statusCode": "500"
            },
            "2\\\d{2}": {
                "statusCode": "200"
            }
        },
        "requestParameters": {
            "integration.request.path.key": "method.request.querystring.key"
        },
        "uri": "arn:aws:apigateway:us-west-2:s3:path/{key}",
        "passthroughBehavior": "when_no_match",
        "httpMethod": "GET",
        "type": "aws"
    }
},
"put": {
    "parameters": [
        {
            "name": "key",
            "in": "query",
            "required": false,
            "schema": {
                "type": "string"
            }
        }
    ],
    "responses": {
        "200": {
            "description": "200 response",
            "content": {
                "application/json": {
                    "schema": {
                        "$ref": "#/components/schemas/Empty"
                    }
                },
                "application/octet-stream": {
                    "schema": {
                        "$ref": "#/components/schemas/Empty"
                    }
                }
            }
        },
        "500": {
            "description": "500 response"
        }
    }
}
```

```
        },
        "x-amazon-apigateway-integration": {
            "credentials": "arn:aws:iam::123456789012:role/binarySupportRole",
            "responses": {
                "default": {
                    "statusCode": "500"
                },
                "2\\d{2)": {
                    "statusCode": "200"
                }
            },
            "requestParameters": {
                "integration.request.path.key": "method.request.querystring.key"
            },
            "uri": "arn:aws:apigateway:us-west-2:s3:path/{key}",
            "passthroughBehavior": "when_no_match",
            "httpMethod": "PUT",
            "type": "aws",
            "contentHandling": "CONVERT_TO_BINARY"
        }
    }
},
"x-amazon-apigateway-binary-media-types": [
    "application/octet-stream",
    "image/jpeg"
],
"servers": [
    {
        "url": "https://abcdefghi.execute-api.us-east-1.amazonaws.com/{basePath}",
        "variables": {
            "basePath": {
                "default": "/v1"
            }
        }
    }
],
"components": {
    "schemas": {
        "Empty": {
            "type": "object",
            "title": "Empty Schema"
        }
    }
}
}
```

## OpenAPI 2.0

```
{
    "swagger": "2.0",
    "info": {
        "version": "2016-10-21T17:26:28Z",
        "title": "ApiName"
    },
    "host": "abcdefghi.execute-api.us-east-1.amazonaws.com",
    "basePath": "/v1",
    "schemes": [
        "https"
    ],
    "paths": {
        "/s3": {
            "get": {
                "produces": [
                    "application/json"

```

```
],
"parameters": [
  {
    "name": "key",
    "in": "query",
    "required": false,
    "type": "string"
  }
],
"responses": {
  "200": {
    "description": "200 response",
    "schema": {
      "$ref": "#/definitions/Empty"
    }
  },
  "500": {
    "description": "500 response"
  }
},
"x-amazon-apigateway-integration": {
  "credentials": "arn:aws:iam::123456789012:role/binarySupportRole",
  "responses": {
    "default": {
      "statusCode": "500"
    },
    "2\ \d{2}": {
      "statusCode": "200"
    }
  },
  "requestParameters": {
    "integration.request.path.key": "method.request.querystring.key"
  },
  "uri": "arn:aws:apigateway:us-west-2:s3:path/{key}",
  "passthroughBehavior": "when_no_match",
  "httpMethod": "GET",
  "type": "aws"
}
},
"put": {
  "produces": [
    "application/json", "application/octet-stream"
  ],
  "parameters": [
    {
      "name": "key",
      "in": "query",
      "required": false,
      "type": "string"
    }
  ],
  "responses": {
    "200": {
      "description": "200 response",
      "schema": {
        "$ref": "#/definitions/Empty"
      }
    },
    "500": {
      "description": "500 response"
    }
  },
  "x-amazon-apigateway-integration": {
    "credentials": "arn:aws:iam::123456789012:role/binarySupportRole",
    "responses": {
      "default": {
        "statusCode": "500"
      }
    }
  }
}
```

```
        },
        "2\\d{2}": {
            "statusCode": "200"
        }
    },
    "requestParameters": {
        "integration.request.path.key": "method.request.querystring.key"
    },
    "uri": "arn:aws:apigateway:us-west-2:s3:path/{key}",
    "passthroughBehavior": "when_no_match",
    "httpMethod": "PUT",
    "type": "aws",
    "contentHandling" : "CONVERT_TO_BINARY"
}
}
},
"x-amazon-apigateway-binary-media-types" : ["application/octet-stream", "image/jpeg"],
"definitions": {
    "Empty": {
        "type": "object",
        "title": "Empty Schema"
    }
}
}
```

## 從 Amazon S3 下載影像

若要從 Amazon S3 下載二進位 Blob 格式的影像檔 (image.jpg)：

```
GET /v1/s3?key=image.jpg HTTP/1.1
Host: abcdefghi.execute-api.us-east-1.amazonaws.com
Content-Type: application/json
Accept: application/octet-stream
```

成功回應如下所示：

```
200 OK HTTP/1.1
[raw bytes]
```

由於 Accept 標頭已設定為 application/octet-stream 的二進位媒體類型，而且啟用 API 的二進位支援，因此會傳回原始位元組。

或者，若要下載 Base64 編碼字串格式的影像檔 (image.jpg) 並格式化為 JSON 屬性，請從 Amazon S3 將回應範本新增至類似此範例的 200 整合回應，如下列粗體 OpenAPI 定義區塊所示：

```
"x-amazon-apigateway-integration": {
    "credentials": "arn:aws:iam::123456789012:role/binarySupportRole",
    "responses": {
        "default": {
            "statusCode": "500"
        },
        "2\\d{2}": {
            "statusCode": "200",
            "responseTemplates": {
                "application/json": "{\n        \"image\": \"$input.body\"\n    }"
            }
        }
    }
}
```

}

要下載影像檔的請求如下所示：

```
GET /v1/s3?key=image.jpg HTTP/1.1
Host: abcdefghi.execute-api.us-east-1.amazonaws.com
Content-Type: application/json
Accept: application/json
```

成功回應如下所示：

```
200 OK HTTP/1.1

{
    "image": "W3JhdyBieXRlc10="
}
```

### 將影像上傳至 Amazon S3

若要將二進位 Blob 格式的影像檔 (image.jpg) 上傳至 Amazon S3：

```
PUT /v1/s3?key=image.jpg HTTP/1.1
Host: abcdefghi.execute-api.us-east-1.amazonaws.com
Content-Type: application/octet-stream
Accept: application/json

[raw bytes]
```

成功回應如下所示：

```
200 OK HTTP/1.1
```

若要將 Base64 編碼字串格式的影像檔 (image.jpg) 上傳至 Amazon S3：

```
PUT /v1/s3?key=image.jpg HTTP/1.1
Host: abcdefghi.execute-api.us-east-1.amazonaws.com
Content-Type: application/json
Accept: application/json

W3JhdyBieXRlc10=
```

請注意，輸入承載必須是 Base64 編碼字串，因為 Content-Type 標頭值已設定為 application/json。  
成功回應如下所示：

```
200 OK HTTP/1.1
```

### 使用 API Gateway API 存取 Lambda 中的二進位檔案

下列範例示範如何透過 API Gateway API 存取 AWS Lambda 中的二進位檔案。OpenAPI 檔案中有此範例 API。此程式碼範例使用 API Gateway REST API 呼叫。

#### 主題

- [存取 Lambda 中影像之範例 API 的 OpenAPI 檔案 \(p. 299\)](#)
- [從 Lambda 下載影像 \(p. 302\)](#)
- [將影像上傳至 Lambda \(p. 303\)](#)

## 存取 Lambda 中影像之範例 API 的 OpenAPI 檔案

下列 OpenAPI 檔案顯示一個範例 API，說明如何從 Lambda 下載影像檔，以及如何將影像檔上傳至 Lambda。

OpenAPI 3.0

```
{
  "openapi": "3.0.0",
  "info": {
    "version": "2016-10-21T17:26:28Z",
    "title": "ApiName"
  },
  "paths": {
    "/lambda": {
      "get": {
        "parameters": [
          {
            "name": "key",
            "in": "query",
            "required": false,
            "schema": {
              "type": "string"
            }
          }
        ],
        "responses": {
          "200": {
            "description": "200 response",
            "content": {
              "application/json": {
                "schema": {
                  "$ref": "#/components/schemas/Empty"
                }
              }
            }
          },
          "500": {
            "description": "500 response"
          }
        }
      },
      "x-amazon-apigateway-integration": {
        "uri": "arn:aws:apigateway:us-east-1:lambda:path/2015-03-31/functions/arn:aws:lambda:us-east-1:123456789012:function:image/invocations",
        "type": "AWS",
        "credentials": "arn:aws:iam::123456789012:role/Lambda",
        "httpMethod": "POST",
        "requestTemplates": {
          "application/json": "{\n    \"imageKey\": \"$input.params('key')\"\n}"
        },
        "responses": {
          "default": {
            "statusCode": "500"
          },
          "2\d{2}": {
            "statusCode": "200",
            "responseTemplates": {
              "application/json": "{\n    \"image\": \"$input.body\"\n}"
            }
          }
        }
      }
    },
    "put": {
      "parameters": [
        {
          "name": "body",
          "in": "body",
          "required": true,
          "schema": {
            "type": "string"
          }
        }
      ]
    }
  }
}
```

```
{  
    "name": "key",  
    "in": "query",  
    "required": false,  
    "schema": {  
        "type": "string"  
    }  
},  
],  
"responses": {  
    "200": {  
        "description": "200 response",  
        "content": {  
            "application/json": {  
                "schema": {  
                    "$ref": "#/components/schemas/Empty"  
                }  
            },  
            "application/octet-stream": {  
                "schema": {  
                    "$ref": "#/components/schemas/Empty"  
                }  
            }  
        }  
    },  
    "500": {  
        "description": "500 response"  
    }  
},  
"x-amazon-apigateway-integration": {  
    "uri": "arn:aws:apigateway:us-east-1:lambda:path/2015-03-31/functions/  
arn:aws:lambda:us-east-1:123456789012:function:image/invocations",  
    "type": "AWS",  
    "credentials": "arn:aws:iam::123456789012:role/Lambda",  
    "httpMethod": "POST",  
    "contentHandling": "CONVERT_TO_TEXT",  
    "requestTemplates": {  
        "application/json": "{\n            \"imageKey\": \"$input.params('key')\",  
            \"image\": \"$input.body\"\n        }"  
    },  
    "responses": {  
        "default": {  
            "statusCode": "500"  
        },  
        "2\d{2}": {  
            "statusCode": "200"  
        }  
    }  
},  
},  
"x-amazon-apigateway-binary-media-types": [  
    "application/octet-stream",  
    "image/jpeg"  
],  
"servers": [  
    {  
        "url": "https://abcdefghi.execute-api.us-east-1.amazonaws.com/{basePath}",  
        "variables": {  
            "basePath": {  
                "default": "/v1"  
            }  
        }  
    }  
],
```

```
    "components": {
        "schemas": {
            "Empty": {
                "type": "object",
                "title": "Empty Schema"
            }
        }
    }
```

## OpenAPI 2.0

```
{
    "swagger": "2.0",
    "info": {
        "version": "2016-10-21T17:26:28Z",
        "title": "ApiName"
    },
    "host": "abcdefghi.execute-api.us-east-1.amazonaws.com",
    "basePath": "/v1",
    "schemes": [
        "https"
    ],
    "paths": {
        "/lambda": {
            "get": {
                "produces": [
                    "application/json"
                ],
                "parameters": [
                    {
                        "name": "key",
                        "in": "query",
                        "required": false,
                        "type": "string"
                    }
                ],
                "responses": {
                    "200": {
                        "description": "200 response",
                        "schema": {
                            "$ref": "#/definitions/Empty"
                        }
                    },
                    "500": {
                        "description": "500 response"
                    }
                }
            },
            "x-amazon-apigateway-integration": {
                "uri": "arn:aws:apigateway:us-east-1:lambda:path/2015-03-31/functions/arn:aws:lambda:us-east-1:123456789012:function:image/invocations",
                "type": "AWS",
                "credentials": "arn:aws:iam::123456789012:role/Lambda",
                "httpMethod": "POST",
                "requestTemplates": {
                    "application/json": "{\n            \"imageKey\": \"$input.params('key')\"\n        }"
                },
                "responses": {
                    "default": {
                        "statusCode": "500"
                    },
                    "2\\d{2}": {
                        "statusCode": "200",
                        "responseTemplates": {
                            "application/json": "{\n            \"image\": \"$input.body\"\n        }"
                        }
                    }
                }
            }
        }
    }
}
```

```
        }
    }
},
"put": {
    "produces": [
        "application/json", "application/octet-stream"
    ],
    "parameters": [
        {
            "name": "key",
            "in": "query",
            "required": false,
            "type": "string"
        }
    ],
    "responses": {
        "200": {
            "description": "200 response",
            "schema": {
                "$ref": "#/definitions/Empty"
            }
        },
        "500": {
            "description": "500 response"
        }
    },
    "x-amazon-apigateway-integration": {
        "uri": "arn:aws:apigateway:us-east-1:lambda:path/2015-03-31/functions/arn:aws:lambda:us-east-1:123456789012:function:image/invocations",
        "type": "AWS",
        "credentials": "arn:aws:iam::123456789012:role/Lambda",
        "httpMethod": "POST",
        "contentHandling": "CONVERT_TO_TEXT",
        "requestTemplates": {
            "application/json": "{$input.params('key')}\nimageKey": \"$input.params('key')\", \"image\": \"$input.body\"\n"
        },
        "responses": {
            "default": {
                "statusCode": "500"
            },
            "200": {
                "statusCode": "200"
            }
        }
    }
},
"x-amazon-apigateway-binary-media-types" : ["application/octet-stream", "image/jpeg"],
"definitions": {
    "Empty": {
        "type": "object",
        "title": "Empty Schema"
    }
}
}
```

## 從 Lambda 下載影像

若要從 Lambda 下載二進位 Blob 格式的影像檔 (image.jpg)：

```
GET /v1/lambda?key=image.jpg HTTP/1.1
Host: abcdefghi.execute-api.us-east-1.amazonaws.com
Content-Type: application/json
Accept: application/octet-stream
```

成功回應如下所示：

```
200 OK HTTP/1.1
[raw bytes]
```

若要下載 Base64 編碼字串格式的影像檔 (image.jpg) 並格式化為 JSON 屬性，請從 Lambda：

```
GET /v1/lambda?key=image.jpg HTTP/1.1
Host: abcdefghi.execute-api.us-east-1.amazonaws.com
Content-Type: application/json
Accept: application/json
```

成功回應如下所示：

```
200 OK HTTP/1.1
{
  "image": "W3JhdBieXRlc10="
}
```

## 將影像上傳至 Lambda

若要將二進位 Blob 格式的影像檔 (image.jpg) 上傳至 Lambda：

```
PUT /v1/lambda?key=image.jpg HTTP/1.1
Host: abcdefghi.execute-api.us-east-1.amazonaws.com
Content-Type: application/octet-stream
Accept: application/json

[raw bytes]
```

成功回應如下所示：

```
200 OK
```

若要將 Base64 編碼字串格式的影像檔 (image.jpg) 上傳至 Lambda：

```
PUT /v1/lambda?key=image.jpg HTTP/1.1
Host: abcdefghi.execute-api.us-east-1.amazonaws.com
Content-Type: application/json
Accept: application/json

W3JhdBieXRlc10=
```

成功回應如下所示：

```
200 OK
```

## 啟用 API 的承載壓縮功能

API Gateway 可讓您的用戶端對使用其中一個[支援的內容編碼 \(p. 305\)](#)壓縮的承載呼叫 API。API Gateway 預設支援方法請求承載的解壓縮功能。不過，您必須設定 API，才能啟用方法回應承載的壓縮功能。

若要啟用 API 的壓縮功能，請在建立 API 時，或在建立 API 之後，將 `minimumCompressionSize` 屬性設定為介於 0 到 10485760 (一千萬個位元組) 之間的非負整數。若要停用 API 的壓縮功能，請將 `minimumCompressionSize` 設定為 Null 或將其完全移除。您可以使用 API Gateway 主控台、AWS CLI 或 API Gateway REST API，來啟用或停用 API 的壓縮功能。

如果您想要對任何大小的承載套用壓縮功能，請將 `minimumCompressionSize` 值設定為零。不過，壓縮大小很小的資料實際上可能會增加最終資料大小。此外，在 API Gateway 壓縮與在用戶端解壓縮可能會增加整體延遲，而需要更多的運算時間。您應該對 API 執行測試案例來決定最佳值。

用戶端可以提交已壓縮承載並具有適當 Content-Encoding 標頭的 API 請求，讓 API Gateway 解壓縮方法請求承載並套用適用的對應範本，再將請求傳遞到整合端點。啟用壓縮功能並部署 API 之後，若在方法請求中指定適當的 Accept-Encoding 標頭，用戶端就會收到已壓縮承載的 API 回應。

當整合端點預期並傳回未壓縮的 JSON 承載時，針對未壓縮 JSON 承載設定的任何對應範本都適用於壓縮的承載。對於壓縮的方法請求承載，API Gateway 會解壓縮承載、套用對應範本，然後將對應的請求傳遞到整合端點。對於未壓縮的整合回應承載，API Gateway 會套用對應範本、壓縮對應的承載，然後將壓縮的承載傳回用戶端。

### 主題

- [啟用 API 的承載壓縮功能 \(p. 304\)](#)
- [對壓縮的承載呼叫 API 方法 \(p. 305\)](#)
- [接收已壓縮承載的 API 回應 \(p. 306\)](#)

## 啟用 API 的承載壓縮功能

您可以使用 API Gateway 主控台、AWS CLI 或 AWS 開發套件為 API 啟用壓縮。

針對現有的 API，您必須在啟用壓縮後部署 API，以使變更生效。針對新的 API，您可以在 API 設定完成後部署 API。

### 主題

- [使用 API Gateway 主控台啟用 API 的承載壓縮功能 \(p. 304\)](#)
- [使用 AWS CLI 啟用 API 的承載壓縮功能 \(p. 305\)](#)
- [API Gateway 支援的內容編碼 \(p. 305\)](#)

## 使用 API Gateway 主控台啟用 API 的承載壓縮功能

下列程序說明如何啟用 API 的承載壓縮功能。

### 使用 API Gateway 主控台啟用承載壓縮功能

1. 登入 API Gateway 主控台。
2. 選擇現有的 API 或建立新的 API。
3. 在主導覽窗格中，在您選擇或建立的 API 下選擇 Settings (設定)。
4. 在 Settings (設定) 窗格的 Content Encoding (內容編碼) 區段下，選取 Content Encoding enabled (已啟用內容編碼) 選項來啟用承載壓縮功能。在 Minimum body size required for compression (壓縮所需的內文大小下限) 旁邊，輸入壓縮大小下限的數值 (位元組)。若要停用壓縮功能，請清除 Content Encoding enabled (已啟用內容編碼) 選項。
5. 選擇 Save Changes (儲存變更)。

## 使用 AWS CLI 啟用 API 的承載壓縮功能

若要使用 AWS CLI 來建立新的 API 並啟用壓縮功能，請呼叫 [create-rest-api](#) 命令，如下所示：

```
aws apigateway create-rest-api \
--name "My test API" \
--minimum-compression-size 0
```

若要使用 AWS CLI 來啟用現有 API 的壓縮功能，請呼叫 [update-rest-api](#) 命令，如下所示：

```
aws apigateway update-rest-api \
--rest-api-id 1234567890 \
--patch-operations op=replace,path=/minimumCompressionSize,value=0
```

`minimumCompressionSize` 屬性擁有介於 0 和 10485760 的非負數整數值 (1 千萬個位元組)。它可衡量壓縮閾值。如果承載大小小於這個值，則不會對承載進行壓縮或解壓縮。因此，請將此值設為零以允許任何承載大小使用壓縮功能。

若要使用 AWS CLI 來停用壓縮功能，請呼叫 [update-rest-api](#) 命令，如下所示：

```
aws apigateway update-rest-api \
--rest-api-id 1234567890 \
--patch-operations op=replace,path=/minimumCompressionSize,value=
```

您也可以將 `value` 設定為空白字串 ""，或在先前的呼叫中徹底省略 `value` 屬性。

## API Gateway 支援的內容編碼

API Gateway 支援下列內容編碼：

- deflate
- gzip
- identity

根據 [RFC 7231](#) 規格，API Gateway 也支援下列 `Accept-Encoding` 標頭格式：

- `Accept-Encoding: deflate,gzip`
- `Accept-Encoding:`
- `Accept-Encoding:*`
- `Accept-Encoding: deflate;q=0.5,gzip=1.0`
- `Accept-Encoding: gzip;q=1.0,identity;q=0.5,*;q=0`

## 對壓縮的承載呼叫 API 方法

若要提出已壓縮承載的 API 請求，用戶端必須使用其中一個[支援的內容編碼 \(p. 305\)](#)來設定 `Content-Encoding` 標頭。

假設您是 API 用戶端並想要呼叫 PetStore API 方法 (`POST /pets`)。請勿使用下列 JSON 輸出呼叫方法：

```
POST /pets
Host: {petstore-api-id}.execute-api.{region}.amazonaws.com
Content-Length: ...

{
    "type": "dog",
```

```
    "price": 249.99
}
```

您可以改為對使用 GZIP 編碼壓縮的相同承載呼叫方法：

```
POST /pets
Host: {petstore-api-id}.execute-api.{region}.amazonaws.com
Content-Encoding:gzip
Content-Length: ...

####RPP*#, HU#RPJ#OW##e&###L, #,-y#j
```

當 API Gateway 收到請求時，它會確認指定的內容編碼是否受到支援。然後，它會嘗試使用指定的內容編碼解壓縮。如果解壓縮成功，則會將請求發送到整合端點。如果指定的編碼不受支援或提供的承載未使用指定的編碼壓縮，API Gateway 會傳回 415 Unsupported Media Type 錯誤回應。如果在識別 API 與階段之前的解壓縮階段初期發生錯誤，該錯誤不會記錄到 CloudWatch Logs。

## 接收已壓縮承載的 API 回應

對已啟用壓縮功能的 API 提出請求時，用戶端可以透過指定 Accept-Encoding 標頭與[支援的內容編碼 \(p. 305\)](#)，選擇接收特定格式的壓縮回應承載。

只有符合下列條件時，API Gateway 才會壓縮回應承載：

- 傳入請求具有 Accept-Encoding 標頭以及支援的內容編碼與格式。

### Note

如果未設定標頭，預設值為 \*，如 [RFC 7231](#) 中所定義。在此情況下，API Gateway 不會壓縮承載。某些瀏覽器或用戶端可能會自動將 Accept-Encoding (例如 Accept-Encoding:gzip, deflate, br) 新增至已啟用壓縮功能的請求。這會觸發 API Gateway 的承載壓縮功能。若未明確指定支援的 Accept-Encoding 標頭值，API Gateway 就不會壓縮承載。

- 在 API 上設定 minimumCompressionSize 以啟用壓縮功能。
- 整合回應沒有 Content-Encoding 標頭。
- 整合回應承載的大小在套用適用的映射範本之後，大於或等於指定的 minimumCompressionSize 值。

API Gateway 會套用針對整合回應設定的任何映射範本，再壓縮承載。如果整合回應包含 Content-Encoding 標頭，API Gateway 會假設整合回應承載已壓縮並略過壓縮處理。

以 PetStore API 為例，其請求如下：

```
GET /pets
Host: {petstore-api-id}.execute-api.{region}.amazonaws.com
Accept: application/json
```

後端會以類似如下的未壓縮 JSON 承載來回應請求：

```
200 OK
[
  {
    "id": 1,
    "type": "dog",
    "price": 249.99
  },
  {
    "id": 2,
    "type": "cat",
```

```
        "price": 124.99
    },
    [
        {
            "id": 3,
            "type": "fish",
            "price": 0.99
        }
    ]
]
```

若要接收此輸出作為壓縮的承載，您的 API 用戶端可以提交請求，如下所示：

```
GET /pets
Host: {petstore-api-id}.execute-api.{region}.amazonaws.com
Accept-Encoding:gzip
```

用戶端會收到具有 Content-Encoding 標頭與 GZIP 編碼承載的回應，如下所示：

```
200 OK
Content-Encoding:gzip
...
###RP#
JV
#:P^IeA*****+(#L #X#YZ#ku0L0B7!9##C##&####Y##a##^#X
```

壓縮回應承載之後，只有壓縮的資料大小會計入數據傳輸費。

## 在 API Gateway 中啟用請求驗證

您可以先設定 API Gateway 執行 API 請求的基本驗證，再繼續進行整合請求。驗證失敗時，API Gateway 會立即讓請求失敗、將 400 錯誤回應傳回給發起人，並在 CloudWatch Logs 中發佈驗證結果。這樣可減少不必要的後端呼叫。更重要的是，它可讓您專注於應用程式特定的驗證努力。

### 主題

- [API Gateway 中的基本請求驗證概觀 \(p. 307\)](#)
- [在 API Gateway 中設定基本請求驗證 \(p. 308\)](#)
- [在 API Gateway 中測試基本請求驗證 \(p. 313\)](#)
- [具有基本請求驗證之範例 API 的 OpenAPI 定義 \(p. 316\)](#)

## API Gateway 中的基本請求驗證概觀

API Gateway 可以執行基本驗證。這可讓您 (API 開發人員) 專注於後端中的應用程式特定深入驗證。針對基本驗證，API Gateway 會驗證下列任一或兩個條件：

- 會包含 URI 中的必要請求參數、查詢字串以及傳入請求的標頭，而且不是空白。
- 適用的請求承載，會遵守在方法中設定的 [JSON 結構描述](#)請求模型 (p. 250)。

若要啟用基本驗證，您需要在[請求驗證程式](#)中指定驗證規則、將驗證程式新增至 API 的[請求驗證程式映射](#)，並將驗證程式指派給個別 API 方法。

### Note

請求內文驗證和[請求內文傳遞 \(p. 273\)](#)是兩個不同的問題。如果因模型結構描述不相符而無法驗證請求承載，則您可以選擇傳遞或封鎖原始承載。例如，當您使用 application/json 媒體類型的對應範本來啟用請求驗證時，建議您將 XML 承載傳遞至後端，即使啟用的請求驗證將會失敗也是一

樣。如果您預期未來支援方法上的 XML 承載，則可能是這種情況。若要讓具有 XML 承載的請求失敗，您必須明確選擇內容傳遞行為的 NEVER 選項。

## 在 API Gateway 中設定基本請求驗證

您可以在 API 的 OpenAPI 定義檔中設定請求驗證程式，然後將 OpenAPI 定義匯入 API Gateway 中。您也可以透過下列方式設定它們：在 API Gateway 主控台中，或透過呼叫 API Gateway REST API、AWS CLI 或其中一個 AWS 開發套件。在這裡，我們示範如何使用 OpenAPI 檔案、在主控台中以及使用 API Gateway REST API 來執行這項操作。

### 主題

- [匯入 OpenAPI 定義來設定基本請求驗證 \(p. 308\)](#)
- [使用 API Gateway REST API 設定請求驗證程式 \(p. 311\)](#)
- [使用 API Gateway 主控台設定基本請求驗證 \(p. 312\)](#)

## 匯入 OpenAPI 定義來設定基本請求驗證

下列步驟說明如何匯入 OpenAPI 檔案來啟用基本請求驗證。

### 將 OpenAPI 檔案匯入至 API Gateway 以啟用請求驗證

1. 在 API 層級的 [x-amazon-apigateway-request-validation 物件 \(p. 549\)](#) 對應中指定一組 [x-amazon-apigateway-request-validation.requestValidator 物件 \(p. 550\)](#) 物件，以在 OpenAPI 中宣告請求驗證程式。例如，範例 API OpenAPI 檔案 (p. 316) 包含將驗證程式名稱做為金鑰的 [x-amazon-apigateway-request-validation](#) 對應。

#### OpenAPI 3.0

```
{  
  "openapi": "3.0.0",  
  "info": {  
    "title": "ReqValidation Sample",  
    "version": "1.0.0"  
  },  
  "servers": [  
    {  
      "url": "/{basePath}",  
      "variables": {  
        "basePath": {  
          "default": "/v1"  
        }  
      }  
    }  
  ],  
  "x-amazon-apigateway-request-validation": {  
    "all": {  
      "validateRequestBody": true,  
      "validateRequestParameters": true  
    },  
    "params-only": {  
      "validateRequestBody": false,  
      "validateRequestParameters": true  
    }  
  }  
}
```

#### OpenAPI 2.0

```
{
```

```
    "swagger": "2.0",
    "info": {
        "title": "ReqValidation Sample",
        "version": "1.0.0"
    },
    "schemes": [
        "https"
    ],
    "basePath": "/v1",
    "produces": [
        "application/json"
    ],
    "x-amazon-apigateway-request-validation": {
        "all": {
            "validateRequestBody": true,
            "validateRequestParameters": true
        },
        "params-only": {
            "validateRequestBody": false,
            "validateRequestParameters": true
        }
    },
    ...
}
```

在 API 或方法上啟用驗證程式時，您可以選取驗證程式名稱，如下列步驟所示。

2. 若要在 API 的所有方法上啟用請求驗證程式，請在 OpenAPI 定義檔的 API 層級指定 [x-amazon-apigateway-request-validator 屬性 \(p. 548\)](#) 屬性。若要在個別方法上啟用請求驗證程式，請在方法層級指定 `x-amazon-apigateway-request-validator` 屬性。例如，除非特別覆寫，否則下列 `x-amazon-apigateway-request-validator` 屬性會在所有 API 方法上啟用 `params-only` 驗證程式。

#### OpenAPI 3.0

```
{
    "openapi": "3.0.0",
    "info": {
        "title": "ReqValidation Sample",
        "version": "1.0.0"
    },
    "servers": [
        {
            "url": "/{basePath}",
            "variables": {
                "basePath": {
                    "default": "/v1"
                }
            }
        }
    ],
    "x-amazon-apigateway-request-validator": "params-only",
    ...
}
```

#### OpenAPI 2.0

```
{
    "swagger": "2.0",
    "info": {
        "title": "ReqValidation Sample",
        "version": "1.0.0"
```

```
},
"schemes": [
  "https"
],
"basePath": "/v1",
"produces": [
  "application/json"
],
...
"x-amazon-apigateway-request-validator" : "params-only",
...
}
```

若要在個別方法上啟用請求驗證程式，請在方法層級指定 `x-amazon-apigateway-request-validator` 屬性。例如，下列 `x-amazon-apigateway-request-validator` 屬性會在 `all` 方法上啟用 POST /validation 驗證程式。這會覆寫繼承自 API 的 `params-only` 驗證程式。

#### OpenAPI 3.0

```
{
  "openapi": "3.0.0",
  "info": {
    "title": "ReqValidation Sample",
    "version": "1.0.0"
  },
  "servers": [
    {
      "url": "/{basePath}",
      "variables": {
        "basePath": {
          "default": "/v1"
        }
      }
    }
  ],
  "paths": {
    "/validation": {
      "post": {
        "x-amazon-apigateway-request-validator": "all",
        ...
      }
    }
  }
}
```

#### OpenAPI 2.0

```
{
  "swagger": "2.0",
  "info": {
    "title": "ReqValidation Sample",
    "version": "1.0.0"
  },
  "schemes": [
    "https"
  ],
  "basePath": "/v1",
  "produces": [
    "application/json"
  ],
  ...
}
```

```
"paths": {
    "/validation": {
        "post": {
            "x-amazon-apigateway-request-validator" : "all",
            ...
        },
        ...
    }
}
...
```

3. 在 API Gateway 中，匯入此範例 OpenAPI 定義 (p. 316)，以建立已啟用請求驗證程式的 API：

```
POST /restapis?mode=import&failonwarning=true HTTP/1.1
Content-Type: application/json
Host: apigateway.us-east-1.amazonaws.com
X-Amz-Date: 20170306T234936Z
Authorization: AWS4-HMAC-SHA256 Credential={access_key_ID}/20170306/us-east-1/
apigateway/aws4_request, SignedHeaders=content-length;content-type;host;x-amz-date,
Signature={sig4_hash}
```

*Copy the JSON object from this sample OpenAPI definition (p. 316) and paste it here.*

4. 將新建立的 API (fjd6crafxc) 部署至指定的階段 (testStage)。

```
POST /restapis/fjd6crafxc/deployments HTTP/1.1
Content-Type: application/json
Host: apigateway.us-east-1.amazonaws.com
X-Amz-Date: 20170306T234936Z
Authorization: AWS4-HMAC-SHA256 Credential={access_key_ID}/20170306/us-east-1/
apigateway/aws4_request, SignedHeaders=content-length;content-type;host;x-amz-date,
Signature={sig4_hash}

{
    "stageName" : "testStage",
    "stageDescription" : "Test stage",
    "description" : "First deployment",
    "cacheClusterEnabled" : "false"
}
```

如需如何使用 API Gateway REST API 測試請求驗證的說明，請參閱[使用 API Gateway REST API 測試基本請求驗證 \(p. 313\)](#)。如需如何使用 API Gateway 主控台進行測試的說明，請參閱[使用 API Gateway 主控台測試基本請求驗證 \(p. 316\)](#)。

## 使用 API Gateway REST API 設定請求驗證程式

在 API Gateway REST API 中，請求驗證程式是透過 [RequestValidator](#) 資源所呈現。若要讓 API 支援與範例 API (p. 316) 相同的請求驗證程式，請將 `params-only` 做為金鑰的僅限參數驗證程式新增至 `RequestValidators` 集合，並新增 `all` 做為其金鑰的完整驗證程式。

### 使用 API Gateway REST API 啟用基本請求驗證

假設您的 API 類似範例 API (p. 316)，但尚未設定請求驗證程式。如果您的 API 已啟用請求驗證程式，請呼叫適當的 `requestvalidator:update` 或 `method:put` 動作，而非 `requestvalidator:create` 或 `method:put`。

1. 若要設定 `params-only` 請求驗證程式，請呼叫 `requestvalidator:create` 動作，如下所示：

```
POST /restapis/restapi-id/requestvalidators HTTP/1.1
```

```
Content-Type: application/json
Host: apigateway.region.amazonaws.com
X-Amz-Date: 20170223T172652Z
Authorization: AWS4-HMAC-SHA256 Credential={access_key_ID}/20170223/region/apigateway/
aws4_request, SignedHeaders=content-type;host;x-amz-date, Signature={sig4_hash}

{
    "name" : "params-only",
    "validateRequestBody" : "false",
    "validateRequestParameters" : "true"
}
```

2. 若要設定 all 請求驗證程式，請呼叫 `requestvalidator:create` 動作，如下所示：

```
POST /restapis/restapi-id/requestvalidators HTTP/1.1
Content-Type: application/json
Host: apigateway.region.amazonaws.com
X-Amz-Date: 20170223T172652Z
Authorization: AWS4-HMAC-SHA256 Credential={access_key_ID}/20170223/region/apigateway/
aws4_request, SignedHeaders=content-type;host;x-amz-date, Signature={sig4_hash}

{
    "name" : "all",
    "validateRequestBody" : "true",
    "validateRequestParameters" : "true"
}
```

如果上述驗證程式金鑰已存在於 RequestValidators 映射中，請呼叫 `requestvalidator:update` 動作，而不要重設驗證規則。

3. 若要將 all 請求驗證程式套用至 POST 方法，請呼叫 `method:put` 以啟用指定的驗證程式 (由 `requestValidatorId` 屬性所識別) 或呼叫 `method:update` 以更新已啟用的驗證程式。

```
PUT /restapis/restapi-id/resources/resource-id/methods/POST HTTP/1.1
Content-Type: application/json
Host: apigateway.region.amazonaws.com
X-Amz-Date: 20170223T172652Z
Authorization: AWS4-HMAC-SHA256 Credential={access_key_ID}/20170223/region/apigateway/
aws4_request, SignedHeaders=content-type;host;x-amz-date, Signature={sig4_hash}

{
    "authorizationType" : "NONE",
    ...
    "requestValidatorId" : "all"
}
```

## 使用 API Gateway 主控台設定基本請求驗證

API Gateway 主控台可讓您使用三種驗證程式中的其中一種，以在方法上設定基本請求驗證：

- Validate body (驗證內文)：這是僅限內文驗證程式。
- Validate query string parameters and headers (驗證查詢字串參數和標頭)：這是僅限參數驗證程式。
- Validate body, query string parameters, and headers (驗證內文、查詢字串參數和標頭)：此驗證程式適用於內文和參數驗證。

當您選擇上述其中一種驗證程式以在 API 方法上啟用它時，如果尚未將驗證程式新增至 API 的驗證程式映射，則 API Gateway 主控台會將驗證程式新增至 API 的 `RequestValidators` 映射。

### 在方法上啟用請求驗證程式

1. 如果尚未登入，請登入 API Gateway 主控台。
2. 建立新的 API，或選擇現有 API。
3. 建立 API 的新資源，或選擇 API 的現有資源。
4. 建立資源的新方法，或選擇資源的現有方法。
5. 選擇 Method Request (方法請求)。
6. 選擇 Settings (設定) 下 Request Validator (請求驗證程式) 的鉛筆圖示。
7. 從 Request Validator (請求驗證程式) 下拉式清單中選擇 Validate body、Validate query string parameters and headers 或 Validate body, query string parameters, and headers，然後選擇核取記號圖示來儲存您的選擇。

若要在主控台中測試和使用請求驗證程式，請遵循「[使用 API Gateway 主控台測試基本請求驗證 \(p. 316\)](#)」中的說明。

## 在 API Gateway 中測試基本請求驗證

請選擇下列其中一個主題，來取得針對範例 API (p. 316) 測試基本請求驗證的說明。

### 主題

- [使用 API Gateway REST API 測試基本請求驗證 \(p. 313\)](#)
- [使用 API Gateway 主控台測試基本請求驗證 \(p. 316\)](#)

### 使用 API Gateway REST API 測試基本請求驗證

若要查看已部署 API 的呼叫 URL，您可以從階段匯出 API，並務必包含 Accept: application/json 或 Accept: application/yaml 標頭：

```
GET /restapis/fjd6crafxc/stages/testStage/exports/swagger?extensions=validators HTTP/1.1
Accept: application/json
Content-Type: application/json
Host: apigateway.us-east-1.amazonaws.com
X-Amz-Date: 20170306T234936Z
Authorization: AWS4-HMAC-SHA256 Credential={access_key_ID}/20170306/us-east-1/
apigateway/aws4_request, SignedHeaders=content-length;content-type;host;x-amz-date,
Signature={sig4_hash}
```

如果您不想要下載與請求驗證相關的 OpenAPI 規格，則可以忽略 ?extensions=validators 查詢參數。

### 使用 API Gateway REST API 呼叫測試請求驗證

1. 呼叫 GET /validation?q1=cat。

```
GET /testStage/validation?q1=cat HTTP/1.1
Host: fjd6crafxc.execute-api.us-east-1.amazonaws.com
Content-Type: application/json
Accept: application/json
```

因為必要參數 q1 已設定且不是空白，所以請求會通過驗證。API Gateway 傳回下列 200 OK 回應：

```
[{"id": 1, "type": "cat", "price": 249.99}, {"id": 2, "type": "cat", "price": 124.99}, {"id": 3, "type": "cat", "price": 0.99}]
```

2. 呼叫 GET /validation。

```
GET /testStage/validation HTTP/1.1
Host: fjd6crafxc.execute-api.us-east-1.amazonaws.com
Content-Type: application/json
Accept: application/json
```

因為未設定必要參數 q1，所以請求無法通過驗證。API Gateway 傳回下列 400 Bad Request 回應：

```
{"message": "Missing required request parameters: [q1]"}
```

3. 呼叫 GET /validation?q1=。

```
GET /testStage/validation?q1= HTTP/1.1
Host: fjd6crafxc.execute-api.us-east-1.amazonaws.com
Content-Type: application/json
Accept: application/json
```

因為必要參數 q1 空白，所以請求無法通過驗證。API Gateway 傳回與上述範例相同的 400 Bad Request 回應。

4. 呼叫 POST /validation。

```
POST /testStage/validation HTTP/1.1
Host: fjd6crafxc.execute-api.us-east-1.amazonaws.com
Content-Type: application/json
Accept: application/json
h1: v1
```

```
{  
    "name" : "Marco",  
    "type" : "dog",  
    "price" : 260  
}
```

因為必要標頭參數 h1 已設定且不是空白，而且承載格式遵守 RequestDataModel 必要屬性和相關聯的限制，所以請求會傳送驗證。API Gateway 傳回下列成功回應。

```
{  
    "pet": {  
        "name": "Marco",  
        "type": "dog",  
        "price": 260  
    },  
    "message": "success"  
}
```

5. 呼叫 POST /validation，而不需要指定 h1 標頭或將其值設定為空白。

```
POST /testStage/validation HTTP/1.1  
Host: fjd6crafxc.execute-api.us-east-1.amazonaws.com  
Content-Type: application/json  
Accept: application/json  
  
{  
    "name" : "Marco",  
    "type" : "dog",  
    "price" : 260  
}
```

因為必要標頭參數 h1 遺失或設定為空白，所以請求無法通過驗證。API Gateway 傳回下列 400 Bad Request 回應：

```
{  
    "message": "Missing required request parameters: [h1]"  
}
```

6. 呼叫 POST /validation，並將承載的 type 屬性設定為 bird。

```
POST /testStage/validation HTTP/1.1  
Host: fjd6crafxc.execute-api.us-east-1.amazonaws.com  
Content-Type: application/json  
Accept: application/json  
X-Amz-Date: 20170309T000215Z  
h1: v1  
  
{  
    "name" : "Molly",  
    "type" : "bird",  
    "price" : 269  
}
```

因為 type 屬性值不是列舉 ["dog", "cat", "fish"] 的成員，所以請求無法通過驗證。API Gateway 傳回下列 400 Bad Request 回應：

```
{  
    "message": "Invalid request body"  
}
```

將 price 設定為 501 會違反屬性限制。這會讓驗證失敗，並傳回相同的 400 Bad Request 回應。

## 使用 API Gateway 主控台測試基本請求驗證

下列步驟說明如何在 API Gateway 主控台中測試基本請求驗證。

### 在 API Gateway 主控台中使用 TestInvoke 測試方法上的請求驗證

登入 API Gateway 主控台時，請執行下列操作：

1. 選擇您已設定請求驗證程式對應之 API 的 Resources (資源)。
2. 選擇方法，而您已使用所指定的請求驗證程式啟用此方法的請求驗證。
3. 在 Method Execution (方法執行) 的 Client (用戶端) 方塊中，選擇 Test (測試)。
4. 試用必要請求參數或適用內文的不同值，然後選擇 Test (測試) 以查看回應。

方法呼叫通過驗證時，您應該會收到預期回應。如果驗證失敗，則會在承載的格式不正確時傳回下列錯誤訊息：

```
{  
  "message": "Invalid request body"  
}
```

如果請求參數無效，則會傳回下列錯誤訊息：

```
{  
  "message": "Missing required request parameters: [p1]"  
}
```

## 具有基本請求驗證之範例 API 的 OpenAPI 定義

下列 OpenAPI 定義會定義已啟用請求驗證的範例 API。API 是 PetStore API 的子集。它會公開 POST 方法以將寵物新增至 pets 集合，並公開 GET 方法以依指定的類型來查詢寵物。

在 API 層級的 x-amazon-apigateway-request-validation 對應中，已宣告兩種請求驗證程式。params-only 驗證程式已在 API 上予以啟用，並且透過 GET 方法繼承。此驗證程式可讓 API Gateway 驗證傳入請求中已包含不是空白的必要查詢參數 (q1)。all 驗證程式已在 POST 方法上予以啟用。此驗證程式會驗證已設定不是空白的必要標頭參數 (h1)。它還會驗證承載格式遵守指定的 RequestBodyModel 結構描述。此模型需要輸入 JSON 物件包含 name、type 和 price 屬性。name 屬性可以是任意字串、type 必須是其中一個指定的列舉欄位 (["dog", "cat", "fish"])，而 price 的範圍必須是 25 到 500。id 不是必要參數。

如需此 API 行為的詳細資訊，請參閱「[在 API Gateway 中啟用請求驗證 \(p. 307\)](#)」。

OpenAPI 2.0

```
{  
  "swagger": "2.0",  
  "info": {  
    "title": "ReqValidators Sample",  
    "version": "1.0.0"  
  },  
  "schemes": [  
    "https"  
  ],
```

```
"basePath": "/v1",
"produces": [
    "application/json"
],
"x-amazon-apigateway-requestValidators": {
    "all" : {
        "validateRequestBody" : true,
        "validateRequestParameters" : true
    },
    "params-only" : {
        "validateRequestBody" : false,
        "validateRequestParameters" : true
    }
},
"x-amazon-apigateway-request-validator" : "params-only",
"paths": {
    "/validation": {
        "post": {
            "x-amazon-apigateway-request-validator" : "all",
            "parameters": [
                {
                    "in": "header",
                    "name": "h1",
                    "required": true
                },
                {
                    "in": "body",
                    "name": "RequestBodyModel",
                    "required": true,
                    "schema": {
                        "$ref": "#/definitions/RequestBodyModel"
                    }
                }
            ],
            "responses": {
                "200": {
                    "schema": {
                        "type": "array",
                        "items": {
                            "$ref": "#/definitions/Error"
                        }
                    },
                    "headers" : {
                        "test-method-response-header" : {
                            "type" : "string"
                        }
                    }
                },
                "security" : [
                    "api_key" : []
                ],
                "x-amazon-apigateway-auth" : {
                    "type" : "none"
                },
                "x-amazon-apigateway-integration" : {
                    "type" : "http",
                    "uri" : "http://petstore-demo-endpoint.execute-api.com/petstore/pets",
                    "httpMethod" : "POST",
                    "requestParameters": {
                        "integration.request.header.custom_h1": "method.request.header.h1"
                    },
                    "responses" : {
                        "2\\d{2}" : {
                            "statusCode" : "200"
                        }
                    }
                }
            }
        }
    }
}
```

```
"default" : {
    "statusCode" : "400",
    "responseParameters" : {
        "method.response.header.test-method-response-header" : "'static value'"
    },
    "responseTemplates" : {
        "application/json" : "json 400 response template",
        "application/xml" : "xml 400 response template"
    }
}
},
"get": {
    "parameters": [
        {
            "name": "q1",
            "in": "query",
            "required": true
        }
    ],
    "responses": {
        "200": {
            "schema": {
                "type": "array",
                "items": {
                    "$ref": "#/definitions/Error"
                }
            },
            "headers" : {
                "test-method-response-header" : {
                    "type" : "string"
                }
            }
        },
        "security" : [
            {
                "api_key" : []
            }
        ],
        "x-amazon-apigateway-auth" : {
            "type" : "none"
        },
        "x-amazon-apigateway-integration" : {
            "type" : "http",
            "uri" : "http://petstore-demo-endpoint.execute-api.com/petstore/pets",
            "httpMethod" : "GET",
            "requestParameters": {
                "integration.request.querystring.type": "method.request.querystring.q1"
            },
            "responses" : {
                "2\d{2}" : {
                    "statusCode" : "200"
                },
                "default" : {
                    "statusCode" : "400",
                    "responseParameters" : {
                        "method.response.header.test-method-response-header" : "'static value'"
                    },
                    "responseTemplates" : {
                        "application/json" : "json 400 response template",
                        "application/xml" : "xml 400 response template"
                    }
                }
            }
        }
    }
}
```

```
        },
    },
    "definitions": {
        "RequestBodyModel": {
            "type": "object",
            "properties": {
                "id": { "type": "integer" },
                "type": { "type": "string", "enum": ["dog", "cat", "fish"] },
                "name": { "type": "string" },
                "price": { "type": "number", "minimum": 25, "maximum": 500 }
            },
            "required": [ "type", "name", "price" ]
        },
        "Error": {
            "type": "object",
            "properties": {
            }
        }
    }
}
```

## 將 REST API 匯入 API Gateway

您可使用 API Gateway 匯入 API 功能，將 REST API 從外部定義檔匯入 API Gateway。目前，匯入 API 功能支援 [OpenAPI v2.0](#) 和 [OpenAPI v3.0](#) 定義檔。您可以用新的定義覆寫 API 來更新 API，或與現有的 API 合併定義。您可以在請求 URL 中使用 mode 查詢參數來指定選項。

如需從 API Gateway 主控台使用「匯入 API」功能的教學課程，請參閱[教學課程：匯入範例來建立 REST API \(p. 40\)](#)。

### 主題

- [將邊緣最佳化的 API 匯入 API Gateway \(p. 319\)](#)
- [將區域 API 匯入 API Gateway \(p. 320\)](#)
- [匯入 OpenAPI 檔案以更新現有的 API 定義 \(p. 321\)](#)
- [設定 OpenAPI basePath 屬性 \(p. 322\)](#)
- [匯入期間的錯誤與警告 \(p. 324\)](#)

## 將邊緣最佳化的 API 匯入 API Gateway

您可以匯入 API OpenAPI 定義檔來建立新的邊緣最佳化 API，方法是指定 EDGE 端點類型做為匯入操作的額外輸入 (除了 OpenAPI 檔案之外)。您可以使用 API Gateway 主控台、AWS CLI 或 AWS 開發套件這麼做。

如需從 API Gateway 主控台使用「匯入 API」功能的教學課程，請參閱[教學課程：匯入範例來建立 REST API \(p. 40\)](#)。

### 主題

- [使用 API Gateway 主控台匯入邊緣最佳化的 API \(p. 319\)](#)
- [使用 AWS CLI 匯入邊緣最佳化的 API \(p. 320\)](#)

## 使用 API Gateway 主控台匯入邊緣最佳化的 API

若要使用 API Gateway 主控台匯入邊緣最佳化的 API，請按以下操作：

1. 登入 API Gateway 主控台，然後選擇 + Create API (+ 建立 API)。

2. 選取 Create new API (建立新的 API) 下的 Import from OpenAPI (從 OpenAPI 匯入) 選項。
3. 複製 API OpenAPI 定義並將它貼入程式碼編輯器，或選擇 Select OpenAPI File (選取 OpenAPI 檔案) 從本機磁碟載入 OpenAPI 檔案。
4. 在 Settings (設定) 下，針對 Endpoint Type (端點類型)，選擇 Edge optimized。
5. 選擇 Import (匯入) 開始匯入 OpenAPI 定義。

## 使用 AWS CLI 匯入邊緣最佳化的 API

若要透過 AWS CLI 從 OpenAPI 定義檔匯入 API 來建立新的邊緣最佳化 API，請使用 `import-rest-api` 命令，如下所示：

```
aws apigateway import-rest-api \
--fail-on-warnings \
--body 'file://path/to/API_OpenAPI_template.json'
```

或將 `endpointConfigurationTypes` 查詢字串參數明確指定為 EDGE：

```
aws apigateway import-rest-api \
--endpointConfigurationTypes 'EDGE' \
--fail-on-warnings \
--body 'file://path/to/API_OpenAPI_template.json'
```

## 將區域 API 匯入 API Gateway

當您匯入 API 時，可以選擇 API 的區域端點組態。您可以使用 API Gateway 主控台、AWS CLI 或 AWS 開發套件。

當您匯出 API 時，API 端點組態不會包含在匯出的 API 定義中。

如需從 API Gateway 主控台使用「匯入 API」功能的教學課程，請參閱[教學課程：匯入範例來建立 REST API \(p. 40\)](#)。

### 主題

- [使用 API Gateway 主控台匯入區域 API \(p. 320\)](#)
- [使用 AWS CLI 匯入區域 API \(p. 320\)](#)

## 使用 API Gateway 主控台匯入區域 API

若要使用 API Gateway 主控台匯入區域端點的 API，請執行下列動作：

1. 登入 API Gateway 主控台，然後選擇 Create API (建立 API)。
2. 選取 Create new API (建立新的 API) 下的 Import from OpenAPI (從 OpenAPI 匯入) 選項。
3. 複製 API OpenAPI 定義並將它貼入程式碼編輯器，或選擇 Select OpenAPI File (選取 OpenAPI 檔案) 從本機磁碟載入 OpenAPI 檔案。
4. 在 Settings (設定) 下，針對 Endpoint Type (端點類型)，選擇 Regional。
5. 選擇 Import (匯入) 開始匯入 OpenAPI 定義。

## 使用 AWS CLI 匯入區域 API

若要使用 AWS CLI 從 OpenAPI 定義檔匯入 API，請使用 `import-rest-api` 命令：

```
aws apigateway import-rest-api \
```

```
--endpointConfigurationTypes 'REGIONAL' \
--fail-on-warnings \
--body 'file://path/to/API_OpenAPI_template.json'
```

## 匯入 OpenAPI 檔案以更新現有的 API 定義

您可以匯入 API 定義只更新現有的 API，無需變更其端點組態，以及階段與階段變數，或參考 API 金鑰。

匯入到更新操作可在兩種模式下進行：合併或覆寫。

當 API (A) 合併到另一個 (B)，如果兩個 API 不共用任何衝突的定義，則產生的 API 會保留 A 和 B 兩者的定義。發生衝突時，合併 API (A) 之方法定義會覆寫被合併 API (B) 的對應方法定義。例如，假設 B 已宣告以下列方法傳回 200 和 206 回應：

```
GET /a
POST /a
```

而 A 壓告以下列方法傳回 200 和 400 回應：

```
GET /a
```

當 A 合併到 B，產生的 API 會產出以下方法：

```
GET /a
```

這會傳回 200 和 400 回應，以及

```
POST /a
```

這會傳回 200 和 206 回應。

合併 API 適用於您已將外部 API 定義分解成多個更小的組件，而且一次只想要套用其中一個組件的變更時。例如，如果多個小組負責 API 的不同組件並有不同速率的變更時，則可能會出現此情況。在此模式下，現有 API 中的項目若在已匯入的定義中沒有特別定義，則會保持不變。

當 API (A) 覆寫另一個 API (B)，產生的 API 會採用覆寫的 API (A) 之定義。覆寫 API 適用於外部 API 定義包含完整的 API 定義時。在此模式下，現有 API 中的項目若在已匯入的定義中沒有特別定義，則會遭到刪除。

若要合併 API，請將 PUT 請求提交給 `https://apigateway.<region>.amazonaws.com/restapis/<restapi_id>?mode=merge`。`restapi_id` 路徑參數值指定要與所提供的 API 定義合併的 API。

下列程式碼片段顯示 PUT 請求範例，其中會將 JSON 中的 OpenAPI API 定義做為承載與 API Gateway 中已指定的 API 合併。

```
PUT /restapis/<restapi_id>?mode=merge
Host:apigateway.<region>.amazonaws.com
Content-Type: application/json
Content-Length: ...
```

*An OpenAPI API definition in JSON (p. 100)*

合併更新操作可將兩個完整的 API 定義合併在一起。對於小型累加變更，您可以使用 [資源更新](#) 操作。

若要覆寫 API，請將 PUT 請求提交給 `https://apigateway.<region>.amazonaws.com/restapis/<restapi_id>?mode=overwrite`。`restapi_id` 路徑參數指定要使用所提供的 API 定義覆寫的 API。

下列程式碼片段顯示使用 JSON 格式 OpenAPI 定義的承載覆寫請求的範例：

```
PUT /restapis/<restapi_id>?mode=overwrite
Host:apigateway.<region>.amazonaws.com
Content-Type: application/json
Content-Length: ...
```

*An OpenAPI API definition in JSON (p. 100)*

未指定 mode 查詢參數時，會假設使用合併。

Note

PUT 操作為等幂，但不可部分完成。這表示如果在處理到一半時發生系統錯誤，API 最後可能會是錯誤狀態。不過，重複此操作會將 API 放到相同的最終狀態，就像是第一個操作已成功。

## 設定 OpenAPI basePath 屬性

在 [OpenAPI 2.0](#) 中，您可以使用 `basePath` 屬性，來提供 `paths` 屬性中所定義之每個路徑之前的一或多個路徑部分。由於 API Gateway 表達資源路徑的方式有幾種，因此匯入 API 功能會在匯入期間提供下列選項來解譯 `basePath` 屬性：`ignore`、`prepend` 和 `split`。

在 [OpenAPI 3.0](#) 中，`basePath` 不再是頂層屬性。反之，API Gateway 會使用 [伺服器變數](#) 做為慣例。匯入 API 功能會提供相同選項，以在匯入期間解譯基本路徑。基本路徑會依下列所述進行識別：

- 如果 API 未包含任何 `basePath` 變數，則匯入 API 功能會檢查 `server.url` 字串，以查看其是否包含超過 "/" 的路徑。若是，該路徑會用作基本路徑。
- 如果 API 只包含一個 `basePath` 變數，則匯入 API 功能會使用它，做為基本路徑，即使未在 `server.url` 中參考它也一樣。
- 如果 API 包含多個 `basePath` 變數，則匯入 API 功能只會使用第一個變數，做為基本路徑。

### 忽略

如果 OpenAPI 檔案具有 `/a/b/c` 的 `basePath` 值，而且 `paths` 屬性包含 `/e` 與 `/f`，則下列 POST 或 PUT 請求：

```
POST /restapis?mode=import&basepath=ignore
```

```
PUT /restapis/<api_id>?basepath=ignore
```

會導致 API 中的下列資源：

- /
- /e
- /f

結果是將 `basePath` 視為不存在，並提供與主機相關之所有已宣告的 API 資源。例如，當您有自訂網域名稱，其 API 映射未包含 Base Path (基底路徑) 以及參考您生產階段的 Stage (階段) 值，就可以使用此選項。

### Note

API Gateway 會自動為您建立根資源，即使您的定義檔中沒有明確宣告也一樣。

未指定時，`basePath` 預設會採用 `ignore`。

### 前綴

如果 OpenAPI 檔案具有 `/a/b/c` 的 `basePath` 值，而且 `paths` 屬性包含 `/e` 與 `/f`，則下列 POST 或 PUT 請求：

```
POST /restapis?mode=import&basepath=prepend
```

```
PUT /restapis/api_id?basepath=prepend
```

會導致 API 中的下列資源：

- `/`
- `/a`
- `/a/b`
- `/a/b/c`
- `/a/b/c/e`
- `/a/b/c/f`

結果是將 `basePath` 視為指定其他資源 (不含方法)，並將其新增至已宣告的資源集。例如，當不同小組負責 API 的不同組件，而且 `basePath` 可能參考每個小組之 API 組件的路徑位置時，就可以使用此選項。

### Note

API Gateway 會自動為您建立中繼資源，即使您的定義中沒有明確宣告也一樣。

### 分割

如果 OpenAPI 檔案具有 `/a/b/c` 的 `basePath` 值，而且 `paths` 屬性包含 `/e` 與 `/f`，則下列 POST 或 PUT 請求：

```
POST /restapis?mode=import&basepath=split
```

```
PUT /restapis/api_id?basepath=split
```

會導致 API 中的下列資源：

- `/`
- `/b`
- `/b/c`
- `/b/c/e`
- `/b/c/f`

結果是將最上層的路徑部分 `/a` 視為每個資源路徑的開頭，並在 API 本身內建立其他資源 (不含方法)。例如，當 `a` 是您要公開為 API 一部分的階段名稱時，就可以使用此選項。

## 匯入期間的錯誤與警告

### 匯入期間的錯誤

匯入期間，OpenAPI 文件無效等重大問題可能會產生錯誤。這些錯誤會在失敗回應中以例外狀況 (例如 `BadRequestException`) 傳回。發生錯誤時，會捨棄新的 API 定義，而且不會對現有的 API 進行變更。

### 匯入期間的警告

匯入期間，遺失模型參考等次要問題可能會產生警告。如果發生警告，並將 `failonwarnings=false` 查詢表達式附加至請求 URL，操作將會繼續。否則會復原更新。`failonwarnings` 預設會設定為 `false`。在此類情況下，警告會在所產生的 `RestApi` 資源中的欄位回傳。否則，警告會以例外狀況中的訊息傳回。

# 在 API Gateway 中控制和管理 REST API 的存取

API Gateway 支援多種機制來控制和管理對 API 的存取。

以下機制可用於身份驗證和授權：

- 資源政策可讓您建立以資源為基礎的政策，以允許或拒絕從指定的來源 IP 地址或 VPC 端點叫用您的 API 和方法。如需詳細資訊，請參閱[the section called “使用 API Gateway 資源政策” \(p. 324\)](#)。
- 標準 AWS IAM 角色和政策提供彈性且強大的存取控制，可套用至整個 API 或個別方法。IAM 角色和政策可用來控制誰可以建立和管理您的 API，以及誰可以叫用您的 API。如需詳細資訊，請參閱[the section called “使用 IAM 許可” \(p. 333\)](#)。
- IAM 標籤可搭配 IAM 政策一起用來控制存取。如需詳細資訊，請參閱[the section called “標記型存取控制” \(p. 623\)](#)。
- Lambda 授權方是 Lambda 函數，其可使用承載字符身份驗證以及標頭、路徑、查詢字串、階段變數或上下文變數請求參數所述的資訊，控制 REST API 方法的存取。Lambda 授權方用來控制誰可以叫用 REST API 方法。如需詳細資訊，請參閱[the section called “使用 Lambda 授權方” \(p. 348\)](#)。
- Amazon Cognito 使用者集區可讓您為 REST API 建立可自訂的身份驗證和授權解決方案。Amazon Cognito 使用者集區用來控制誰可以叫用 REST API 方法。如需詳細資訊，請參閱[the section called “針對 REST API 使用 Cognito 使用者集區做為授權方” \(p. 363\)](#)。

以下機制可用於執行其他與存取控制相關的任務：

- 跨來源資源共享 (CORS) 可讓您控制 REST API 回應跨網域資源請求的方式。如需詳細資訊，請參閱 [the section called “為 REST API 資源啟用 CORS” \(p. 371\)](#)。
- 用戶端 SSL 憑證可用來驗證對後端系統的 HTTP 請求是否來自 API Gateway。如需詳細資訊，請參閱 [the section called “使用用戶端 SSL 憑證” \(p. 377\)](#)。
- AWS WAF 可以用來保護您的 API Gateway API 避免受到常見的網路攻擊。如需詳細資訊，請參閱 [the section called “使用 AWS WAF 來保護您的 API 避免受到常見的網路攻擊” \(p. 396\)](#)。

以下機制可用於追蹤和限制您已授與授權用戶端之存取：

- 用量計劃可讓您將 API 金鑰提供給您的客戶 —，然後為每個 API 金鑰追蹤和限制 API 階段和方法的用量。如需詳細資訊，請參閱 [the section called “使用 API 金鑰的用量計劃” \(p. 397\)](#)。

## 使用 Amazon API Gateway 資源政策控制存取 API

Amazon API Gateway 資源政策為連接到 API 的 JSON 政策文件，以控制指定的委託人 (通常是 IAM 使用者或角色) 是否可以叫用 API。您可以使用 API Gateway 資源政策，允許您的 API 由以下來源安全地呼叫：

- 指定 AWS 帳戶的使用者
- 指定來源 IP 地址範圍或 CIDR 區塊
- 指定虛擬私有雲端 (VPC) 或 VPC 端點 (在任何帳戶)

您可以在 API Gateway 中使用所有 API 端點類型的資源政策：私有、邊緣最佳化和區域。

您可以使用 AWS 主控台、AWS CLI 或 AWS SDK 將資源政策連接至 API。

API Gateway 資源政策不同於 IAM 政策。IAM 政策連接到 IAM 實體 (使用者、群組或角色) 並且定義這些實體能夠在哪些資源上執行哪些動作。API Gateway 資源政策則是連接到資源。如需更多以身分為基礎 (IAM) 政策和資源政策間差異之詳細討論的更多資訊，請參閱[以身分為基礎的政策和以資源為基礎的政策](#)。

您可以同時使用 API Gateway 資源政策與 IAM 政策。

#### 主題

- [Amazon API Gateway 的存取政策語言概觀 \(p. 325\)](#)
- [Amazon API Gateway 資源政策如何影響授權工作流程 \(p. 326\)](#)
- [API Gateway 資源政策範例 \(p. 329\)](#)
- [建立一個 API Gateway 資源政策並將其連接到 API \(p. 331\)](#)
- [可在 API Gateway 資源政策中使用的 AWS 條件金鑰 \(p. 332\)](#)

## Amazon API Gateway 的存取政策語言概觀

此頁面說明 Amazon API Gateway 資源政策中使用的基本元素。

資源政策是使用與 IAM 政策相同的語法來進行指定。如需完整政策語言資訊，請參閱 IAM User Guide 中的[IAM 政策概觀與 AWS Identity and Access Management 政策參考](#)。

如需有關 AWS 服務如何決定是否應該允許或拒絕特定要求的詳細資訊，請參閱[決定是否允許或拒絕要求](#)。

### 存取政策中的常用元素

就其最基本意義而言，資源政策包含下列元素：

- 資源 – API 是您可允許或拒絕許可的 Amazon API Gateway 資源。在政策中，您可以使用 Amazon Resource Name (ARN) 來識別資源。  
如需 Resource 元素的格式，請參閱「[在 API Gateway 中執行 API 之許可的資源格式 \(p. 341\)](#)」。
- 動作 – Amazon API Gateway 會針對每項資源支援一組操作。您可使用動作關鍵字識別要允許 (或拒絕) 的資源操作。

例如，`apigateway:invoke` 許可讓使用者在使用者請求時有權叫用 API。

如需 Action 元素的格式，請參閱「[在 API Gateway 中執行 API 之許可的動作格式 \(p. 341\)](#)」。

- 效果 – 當使用者要求特定動作時會有什麼效果—這可以是 Allow 或 Deny。您也可以明確拒絕存取資源，您可能會這樣做，以確保使用者無法存取資源，即使另有其他政策授予存取。

#### Note

「隱含拒絕」與「預設拒絕」是相同的。

「隱含拒絕」與「明確拒絕」不同。如需詳細資訊，請參閱[預設拒絕與明確拒絕間的差異](#)。

- 委託人 – 允許存取帳戶明細中動作與資源的帳戶或使用者。在資源政策中，委託人是身為此許可收件人的 IAM 使用者或帳戶。

下列範例資源政策會顯示前面的常用政策元素。此政策會將指定##中指定 *account-id* 下 API 的存取權，授與其來源 IP 地址是在地址區塊 *123.4.5.6/24* 中的任何使用者。如果使用者的來源 IP 不在範圍內，則政策會拒絕所有對 API 的存取。

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Principal": "*",  
            "Action": "execute-api:Invoke",  
            "Resource": "arn:aws:execute-api:region:account-id:*"  
        },  
        {  
            "Effect": "Deny",  
            "Principal": "*",  
            "Action": "execute-api:Invoke",  
            "Resource": "arn:aws:execute-api:region:account-id:*",  
            "Condition": {  
                "NotIpAddress": {  
                    "aws:SourceIp": "123.4.5.6/24"  
                }  
            }  
        }  
    ]  
}
```

## Amazon API Gateway 資源政策如何影響授權工作流程

API Gateway 評估連接到您 API 的資源政策時，其結果會受到您為 API 定義的身份驗證類型所影響，如下節流程圖所示。

### 主題

- [僅 API Gateway 資源政策 \(p. 326\)](#)
- [Lambda 授權方和資源政策 \(p. 327\)](#)
- [IAM 身份驗證和資源政策 \(p. 327\)](#)
- [Amazon Cognito 身份驗證和資源政策 \(p. 328\)](#)
- [政策評估結果表 \(p. 328\)](#)

## 僅 API Gateway 資源政策

在此工作流程中，API Gateway 資源政策連接到 API，但未替 API 定義身份驗證類型。評估政策涉及根據呼叫者的傳入條件來尋求明確允許。隱含拒絕或任何明確拒絕會導致拒絕呼叫者。

以下是這種資源政策的範例。

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Principal": "*",  
            "Action": "execute-api:Invoke",  
            "Resource": "arn:aws:execute-api:region:account-id:api-id/*",  
            "Condition": {  
                "IpAddress": {  
                    "aws:SourceIp": "123.4.5.6/24"  
                }  
            }  
        }  
    ]  
}
```

```
        "aws:SourceIp": ["192.0.2.0/24", "198.51.100.0/24"]  
    }  
}  
]  
}
```

## Lambda 授權方和資源政策

在此工作流程中，除了資源政策，也為 API 設定 Lambda 授權方。資源政策將以兩個階段評估。呼叫 Lambda 授權方之前，API Gateway 會先評估政策，並檢查是否有任何明確拒絕。若有找到，會立即拒絕呼叫者存取。否則，會呼叫 Lambda 授權方，它會傳回政策文件 (p. 358)，搭配資源政策一起評估。結果會根據以下表 A (p. 328) 來判斷。

以下資源政策範例僅允許來自 VPC 端點 ID 為 *vpce-1a2b3c4d* 之 VPC 端點的呼叫。在預先授權評估期間，只會允許來自以下指定之 VPC 端點的呼叫前進並評估 Lambda 授權方。將封鎖所有剩餘的呼叫。

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Deny",  
            "Principal": "*",  
            "Action": "execute-api:Invoke",  
            "Resource": [  
                "arn:aws:execute-api:region:account-id:api-id/"  
            ],  
            "Condition": {  
                "StringNotEquals": {  
                    "aws:SourceVpce": "vpce-1a2b3c4d"  
                }  
            }  
        }  
    ]  
}
```

## IAM 身份驗證和資源政策

在此工作流程中，除了資源政策，也為 API 設定 IAM 身份驗證。使用 IAM 服務驗證使用者之後，連接到 IAM 使用者的政策會和資源政策一起評估。結果會根據呼叫者是否與 API 擁有者位於相同的帳戶，或是位於不同的 AWS 帳戶而有所不同。如果呼叫者和 API 擁有者來自不同的帳戶，IAM 使用者政策和資源政策都需明確允許呼叫者繼續。(請參閱以下表 B (p. 328))。反之，如果呼叫者和 API 擁有者位於相同的帳戶，則使用者政策或資源政策必須明確允許呼叫者繼續。(請參閱以下表 A (p. 328))。

以下是跨帳戶資源政策的範例。假設 IAM 使用者政策包含允許，此資源政策將僅允許來自 VPC ID 是 *vpc-2f09a348* 之 VPC 的呼叫。(請參閱以下表 B (p. 328))。

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Principal": "",  
            "Action": "execute-api:Invoke",  
            "Resource": [  
                "arn:aws:execute-api:region:account-id:api-id/"  
            ],  
            "Condition": {  
                "StringEquals": {  
                    "aws:SourceVpc": "vpc-2f09a348"  
                }  
            }  
        }  
    ]  
}
```

```
        "StringEquals": {
            "aws:SourceVpc": "vpc-2f09a348"
        }
    }
}
```

## Amazon Cognito 身份驗證和資源政策

在此工作流程中，除了資源政策，還會為 API 設定 Amazon Cognito 使用者集區 (p. 363)。API Gateway 會先嘗試透過 Amazon Cognito 驗證呼叫者。這通常是透過呼叫者提供的 JWT 字符來執行。如果身份驗證成功，則會獨立評估資源政策，並需要明確允許。拒絕或既不允許也不拒絕會產生拒絕。以下是可搭配 Amazon Cognito User Pools 使用的資源政策範例。

以下是只允許來自所指定來源 IP 之呼叫的資源政策範例，其中假設 Amazon Cognito 身份驗證字符包含允許。(請參閱以下表 A (p. 328))。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": "*",
            "Action": "execute-api:Invoke",
            "Resource": "arn:aws:execute-api:region:account-id:api-id/",
            "Condition": {
                "IpAddress": {
                    "aws:SourceIp": ["192.0.2.0/24", "198.51.100.0/24"]
                }
            }
        }
    ]
}
```

## 政策評估結果表

表 A 列出對 API Gateway API 的存取是由 IAM 政策 (或 Lambda 或 Amazon Cognito User Pools 授權方) 以及 API Gateway 資源政策控制 (這兩者位於相同的 AWS 帳戶中)，所產生的行為。

表 A：帳戶 A 呼叫帳戶 A 擁有的 API

IAM 使用者政策 (或 Lambda 或 Amazon Cognito User Pools 授權方)	API Gateway 資源政策	產生行為
允許	允許	允許
允許	不允許也不拒絕	Allow
Allow	拒絕	明確拒絕
不允許也不拒絕	Allow	允許
不允許也不拒絕	不允許也不拒絕	隱含拒絕
不允許也不拒絕	拒絕	明確拒絕
拒絕	Allow	明確拒絕

IAM 使用者政策 (或 Lambda 或 Amazon Cognito User Pools 授權方)	API Gateway 資源政策	產生行為
拒絕	不允許也不拒絕	明確拒絕
拒絕	拒絕	明確拒絕

表 B 列出對 API Gateway API 的存取是由 IAM 政策 (或 Lambda 或 Amazon Cognito User Pools 授權方) 以及 API Gateway 資源政策控制 (這兩者位於不同的 AWS 帳戶中)，所產生的行為。如果其中一個無訊息 (不允許也不拒絕)，則會拒絕跨帳戶存取。這是因為跨帳戶存取需要資源政策和 IAM 政策 (或 Lambda 或 Amazon Cognito User Pools 授權方) 明確地授與存取權。

表 B：帳戶 B 呼叫帳戶 A 擁有的 API

IAM 使用者政策 (或 Lambda 或 Amazon Cognito User Pools 授權方)	API Gateway 資源政策	產生行為
允許	允許	允許
允許	不允許也不拒絕	隱含拒絕
Allow	拒絕	明確拒絕
不允許也不拒絕	Allow	隱含拒絕
不允許也不拒絕	不允許也不拒絕	隱含拒絕
不允許也不拒絕	拒絕	明確拒絕
拒絕	Allow	明確拒絕
拒絕	不允許也不拒絕	明確拒絕
拒絕	拒絕	明確拒絕

## API Gateway 資源政策範例

本頁面顯示 API Gateway 資源政策之一般使用案例的一些範例。這些政策使用資源值中的 **account-id** 與 **api-id** 字串。若要測試這些政策，您需要將這些字串取代為您的帳戶 ID 和 API ID。如需存取原則語言的詳細資訊，請參閱「[Amazon API Gateway 的存取政策語言概觀 \(p. 325\)](#)」。

### 主題

- 範例：允許另一個 AWS 帳戶中的使用者使用一個 API (p. 329)
- 範例：拒絕根據來源 IP 地址或範圍的 API 流量 (p. 330)
- 範例：允許以來源 VPC 或 VPC 端點為依據的私有 API 流量 (p. 330)

### 範例：允許另一個 AWS 帳戶中的使用者使用一個 API

以下範例資源政策會透過 [簽章版本 4 \(SigV4\)](#) 通訊協定，將某個 AWS 帳戶的 API 存取權授與不同 AWS 帳戶中的兩個使用者。具體而言，Alice 和 **account-id-2** 所識別之 AWS 帳戶的根使用者獲授與 `execute-api:Invoke` 動作，以在 `pets` 資源 (API) 上執行 GET 動作，而此資源位於 **account-id-1** 所識別的 AWS 帳戶中。

```
{
```

```
"Version": "2012-10-17",
"Statement": [
    {
        "Effect": "Allow",
        "Principal": {
            "AWS": [
                "arn:aws:iam::account-id:user/Alice",
                "account-id-2"
            ]
        },
        "Action": "execute-api:Invoke",
        "Resource": [
            "arn:aws:execute-api:region:account-id-1:api-id/stage/GET/pets"
        ]
    }
]
```

## 範例：拒絕根據來源 IP 地址或範圍的 API 流量

以下範例資源政策是「封鎖清單」政策，其會拒絕（封鎖）從兩個指定的來源 IP 位址區塊傳入 API 流量。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": "*",
            "Action": "execute-api:Invoke",
            "Resource": [
                "arn:aws:execute-api:region:account-id:api-id/*"
            ]
        },
        {
            "Effect": "Deny",
            "Principal": "*",
            "Action": "execute-api:Invoke",
            "Resource": [
                "arn:aws:execute-api:region:account-id:api-id/*"
            ],
            "Condition": {
                "IpAddress": {
                    "aws:SourceIp": ["192.0.2.0/24", "198.51.100.0/24"]
                }
            }
        }
    ]
}
```

## 範例：允許以來源 VPC 或 VPC 端點為依據的私有 API 流量

以下範例資源政策僅允許來自指定的虛擬私有雲端（VPC）或 VPC 端點的流量傳入私有 API。

此範例資源政策會指定來源 VPC：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": "*",
            "Action": "execute-api:Invoke",
            "Resource": [
                "arn:aws:execute-api:region:account-id:api-id/*"
            ]
        }
    ]
}
```

```
        "Resource": [
            "arn:aws:execute-api:region:account-id:api-id/*"
        ]
    },
{
    "Effect": "Deny",
    "Principal": "*",
    "Action": "execute-api:Invoke",
    "Resource": [
        "arn:aws:execute-api:region:account-id:api-id/*"
    ],
    "Condition" : {
        "StringNotEquals": {
            "aws:SourceVpc": "vpc-1a2b3c4d"
        }
    }
}
]
```

此範例資源政策會指定來源 VPC 端點：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": "*",
            "Action": "execute-api:Invoke",
            "Resource": [
                "arn:aws:execute-api:region:account-id:api-id/*"
            ]
        },
        {
            "Effect": "Deny",
            "Principal": "*",
            "Action": "execute-api:Invoke",
            "Resource": [
                "arn:aws:execute-api:region:account-id:api-id/*"
            ],
            "Condition" : {
                "StringNotEquals": {
                    "aws:SourceVpc": "vpce-1a2b3c4d"
                }
            }
        }
    ]
}
```

## 建立一個 API Gateway 資源政策並將其連接到 API

為了讓使用者透過呼叫 API 執行服務來存取您的 API，您必須建立 API Gateway 資源政策，其會控制對 API Gateway 資源的存取權，並將政策連接到 API。

### Important

若要更新 API Gateway 資源政策，除了 `apigateway:PATCH` 許可，您還需要具備 `apigateway:UpdateRestApiPolicy` 許可。

您可以在 API 建立時將資源政策連接到 API，也可以之後再進行連接。請注意，對於私有 API 而言，在您將資源政策連接至私有 API 之前，所有對 API 的呼叫都將會失敗。

### Important

如果您在 API 建立後更新資源政策，您會需要部署 API 以在您連接更新的政策後傳播變更。單獨更新或儲存政策不會變更 API 的執行階段行為。如需部署 API 的詳細資訊，請參閱 [在 Amazon API Gateway 中部署 REST API \(p. 457\)](#)。

存取可透過 IAM 條件元素來進行控制，包括 AWS 帳戶、來源 VPC、來源 VPC 端點或 IP 範圍的條件。如果政策中的 Principal 是設定為 "\*"，其他授權類型可與資源政策搭配使用。如果 Principal 是設定為 "AWS"，所有未透過 AWS\_IAM 授權保護之資源的授權（包含不安全的資源）將會失敗。

以下各節說明如何建立自己的 API Gateway 資源政策並將其連接到您的 API。將在政策中套用許可的政策連接到 API 中的方法。

### Important

如果您使用 API Gateway 主控台來將資源政策連接到已部署的 API，或者如果要更新現有的資源政策，您將需要在主控台中重新部署 API，以讓變更生效。

#### 主題

- [連接 API Gateway 資源政策 \(主控台\) \(p. 332\)](#)
- [連接 API Gateway 資源政策 \(AWS CLI\) \(p. 332\)](#)

## 連接 API Gateway 資源政策 (主控台)

您可以使用 AWS 管理主控台將資源政策連接到 API Gateway API。

### 將資源政策連接至 API Gateway API

1. 登入位於 <https://console.aws.amazon.com/apigateway> 的 API Gateway 主控台。
2. 選擇 API 名稱。
3. 在左側導覽窗格中選擇 Resource Policy (資源政策)。
4. 如有需要，選擇其中一個範例。在範例政策中，預留位置位於雙大括號中 ("{{*placeholder*}}")。將每個預留位置（包括大括號）取代為所需的資訊。

如果不使用其中一個範例，請輸入您的資源政策。

5. 選擇 Save (儲存)。

如果先前已在 API Gateway 主控台中部署該 API，您將需要為資源政策重新部署 API，才能生效。

## 連接 API Gateway 資源政策 (AWS CLI)

若要使用 AWS CLI 來建立新的 API，並將資源政策連接至其中，請呼叫 `create-rest-api` 命令，如下所示：

```
aws apigateway create-rest-api \
--name "api-name" \
--policy "{\"jsonEscapedPolicyDocument}\""
```

若要使用 AWS CLI 來將資源政策連接至現有 API，請呼叫 `update-rest-api` 命令，如下所示：

```
aws apigateway update-rest-api \
--rest-api-id api-id \
--patch-operations op=replace,path=/policy,value='{\"jsonEscapedPolicyDocument}\"'
```

## 可在 API Gateway 資源政策中使用的 AWS 條件金鑰

下表包含 AWS 條件金鑰的完整清單，這些金鑰可在每個授權類型 API Gateway 中的 API 資源政策中使用。

如需有關 AWS 條件金鑰的詳細資訊，請參閱 [AWS 全域條件內容金鑰](#)。

#### 條件金鑰表格

條件金鑰	條件	需要 AuthN？	授權類型
aws:CurrentTime	無	否	全部
aws:EpochTime	無	否	全部
aws:TokenIssueTime	金鑰僅在使用臨時安全登入資料簽署的要求中才會出現。	是	IAM
aws:MultiFactorAuthP	金鑰僅在使用臨時安全登入資料簽署的要求中才會出現。	是	IAM
aws:MultiFactorAuthA	金鑰僅在 MFA 存在於請求時才會出現。	是	IAM
aws:PrincipalType	無	是	IAM
aws:Referer	金鑰僅在 HTTP 標頭中的呼叫者提供值時才會出現。	否	全部
aws:SecureTransport	無	否	全部
aws:SourceArn	無	否	全部
aws:SourceIp	無	否	全部
aws:SourceVpc	此金鑰只能用於私有 API。	否	全部
aws:SourceVpce	此金鑰只能用於私有 API。	否	全部
aws:UserAgent	金鑰僅在 HTTP 標頭中的呼叫者提供值時才會出現。	否	全部
aws:userid	無	是	IAM
aws:username	無	是	IAM

## 使用 IAM 許可控制 API 的存取

您可以使用 [IAM 許可](#)，透過控制下列兩個 API Gateway 元件處理的存取，來控制 Amazon API Gateway API 的存取：

- 若要在 API Gateway 中建立、部署及管理 API，您必須授予 API 開發人員許可來執行 API Gateway API 管理元件支援的必要動作。
- 若要呼叫已部署的 API 或重新整理 API 快取，您必須授予 API 發起人許可來執行 API Gateway API 執行元件支援的必要 IAM 動作。

這兩個處理的存取控制需要不同的許可模型，以下將進行說明。

## 建立與管理 API 的 API Gateway 許可模型

若要讓 API 開發人員在 API Gateway 中建立及管理 API，您必須建立 IAM 許可政策，允許指定的 API 開發人員建立、更新、部署、檢視或刪除必要的 API 實體。您可以將許可政策連接到代表開發人員的 IAM 使用者、包含使用者的 IAM 群組或使用者所擔任的 IAM 角色。

在此 IAM 政策文件中，IAM Resource 元素包含 API Gateway API 實體清單，其中包括 API Gateway 資源與 API Gateway 連結關係。IAM Action 元素包含必要的 API Gateway API 管理動作。這些動作是以 apigateway:*HTTP\_VERB* 格式宣告，其中 apigateway 指定 API Gateway 的基礎 API 管理元件，而 *HTTP\_VERB* 代表 API Gateway 支援的 HTTP 動詞。

如需如何使用此許可模型的詳細資訊，請參閱「[控制管理 API 的存取 \(p. 335\)](#)」。

## 呼叫 API 的 API Gateway 許可模型

若要讓 API 發起人呼叫 API 或重新整理其快取，您必須建立 IAM 政策，允許已啟用 IAM 使用者身分驗證的指定 API 發起人呼叫 API 方法。API 開發人員可將方法的 authorizationType 屬性設定為 AWS\_IAM，以要求發起人提交 IAM 使用者的存取金鑰進行身分驗證。然後，您可以將政策連接到代表 API 發起人的 IAM 使用者、包含使用者的 IAM 群組或使用者所擔任的 IAM 角色。

在此 IAM 許可政策陳述式中，IAM Resource 元素包含指定 HTTP 動詞與 API Gateway 資源路徑所識別的已部署 API 方法清單。IAM Action 元素包含必要的 API Gateway API 執行動作。這些動作包含 execute-api:Invoke 或 execute-api:InvalidateCache，其中 execute-api 指定 API Gateway 的基礎 API 執行元件。

如需如何使用此許可模型的詳細資訊，請參閱「[控制呼叫 API 的存取 \(p. 339\)](#)」。

當 API 與後端 AWS 服務（例如 AWS Lambda）整合時，API Gateway 也必須具備許可，才能代表 API 發起人存取整合的 AWS 資源（例如呼叫 Lambda 函數）。要授與這些許可，請建立適用於 API Gateway 的 AWS 服務類型的 IAM 角色。當您 在 IAM 管理主控台建立此角色，這個產生的角色會包含下列 IAM 信任政策，其中宣告 API Gateway 是允許擔任角色的受信任實體：

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

如果您藉由呼叫 [create-role](#) 的 CLI 命令或是藉由對應的開發套件方法來建立 IAM 角色，則您必須將 assume-role-policy-document 的輸入參數提供給上述政策。請勿嘗試在 IAM 管理主控台或是呼叫 AWS CLI [create-policy](#) 命令或對應的開發套件方法直接建立這種政策。

要讓 API Gateway 呼叫整合的 AWS 服務，您也必須連接到適用此角色的 IAM 許可政策以呼叫整合的 AWS 服務。例如，要呼叫 Lambda 函數，您必須在 IAM 角色中納入下列 IAM 許可政策：

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": "lambda:InvokeFunction",  
            "Resource": "arn:aws:lambda:us-east-1:123456789012:function:myLambda"  
        }  
    ]  
}
```

```
        "Action": "lambda:InvokeFunction",
        "Resource": "*"
    }
}
```

請注意，Lambda 支援合併信任與許可政策的資源類型存取政策。使用 API Gateway 主控台整合 API 與 Lambda 函數時，不會請您明確設定此 IAM 角色，因為主控台會在您的同意下，為您設定 Lambda 函數的資源型許可。

#### Note

若要制定 AWS 服務的存取控制，您可以使用發起人類型許可模型，其中許可政策會直接連接到發起人的 IAM 使用者或群組；或使用角色類型許可模型，其中許可政策會連接到 API Gateway 可擔任的 IAM 角色。這兩個模型的許可政策可能有所不同。例如，發起人類型政策會封鎖存取，而角色類型政策則會允許存取。您可以利用這點，要求 IAM 使用者只透過 API Gateway API 存取 AWS 服務。

## 控制管理 API 的存取

在本節中，您將了解如何撰寫 IAM 政策陳述式，以控制誰可以或無法在 API Gateway 中建立、部署及更新 API。您也將尋找政策陳述式參考，包括與 API 管理服務相關之 Action 與 Resource 欄位的格式。

### 控制誰可以使用 IAM 政策建立及管理 API Gateway API

若要控制誰可以或無法使用 API Gateway 的 API 管理服務來建立、部署及更新您的 API，請以必要的許可建立 IAM 政策文件，如下列政策範本所示：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Permission",
            "Action": [
                "apigateway:HTTP_VERB"
            ],
            "Resource": [
                "arn:aws:apigateway:region::resource1-path",
                "arn:aws:apigateway:region::resource2-path",
                ...
            ]
        }
    ]
}
```

在本例中，*Permission* 可以是 Allow 或 Deny，這會分別授予或撤銷政策陳述式所規定的存取權限。如需詳細資訊，請參閱 [AWS IAM 許可](#)。

*HTTP\_VERB* 可以是任何 API Gateway 支援的 HTTP 動詞 (p. 336)。\* 可用來表示任何 HTTP 動作。

Resource 包含受影響 API 實體的 ARN 清單，包括 RestApi、Resource、Method、Integration、DocumentationPart、Model、Authorizer、UsagePlan 等。如需詳細資訊，請參閱 [在 API Gateway 中管理 API 之許可的資源格式 \(p. 337\)](#)。

透過合併不同的政策陳述式，您可以自訂個別使用者、群組或角色的存取許可，來存取選取的 API 實體，並針對這些實體執行指定的動作。例如，您可以在 IAM 政策中包含下列陳述式，以授予文件團隊許可，讓他們可以建立、發佈、更新及刪除指定 API 的文件組件，以及檢視 API 實體。

```
{
```

```
"Version": "2012-10-17",
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "apigateway:GET"
    ],
    "Resource": [
      "arn:aws:apigateway:region::/restapis/api-id/*"
    ]
  },
  {
    "Effect": "Allow",
    "Action": [
      "apigateway:POST",
      "apigateway:PATCH",
      "apigateway:DELETE"
    ],
    "Resource": [
      "arn:aws:apigateway:region::/restapis/api-id/documentation/*"
    ]
  }
]
```

對於負責所有操作的 API 核心開發團隊，您可以包含下列 IAM 政策中的陳述式，授予團隊更廣泛的存取許可。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "apigateway:/*"
      ],
      "Resource": [
        "arn:aws:apigateway:/*::/*"
      ]
    }
  ]
}
```

## 在 API Gateway 中管理 API 之 IAM 政策的陳述式參考

下列資訊說明 IAM 政策陳述式中用來授予或撤銷管理 API Gateway API 實體許可的 Action 與 Resource 元素格式。

### 在 API Gateway 中管理 API 之許可的動作格式

API 管理 Action 表達式具有下列一般格式：

```
apigateway:action
```

其中 **action** 是下列其中一個 API Gateway 動作：

- \*，表示下列所有動作。
- GET，用來取得資源的相關資訊。
- POST，主要用來建立子資源。
- PUT，主要用來更新資源（也可用來建立子資源，但不建議）。

- DELETE，用來刪除資源。
- PATCH，可用來更新資源。

Action 表達式的一些範例包括：

- **apigateway:\***，表示所有 API Gateway 動作。
- **apigateway:GET**，只表示 API Gateway 中的 GET 動作。

## 在 API Gateway 中管理 API 之許可的資源格式

API 管理 Resource 表達式具有下列一般格式：

```
arn:aws:apigateway:region::resource-path-specifier
```

其中 *region* 是目標 AWS 區域 (例如 **us-east-1** 或表示所有支援 AWS 區域的 **\***)，而 *resource-path-specifier* 是目標資源的路徑。

一些範例資源表達式包括：

- **arn:aws:apigateway:*region*::/restapis/\***，表示 AWS 區域 *region* 中的所有資源、方法、模型與階段。
- **arn:aws:apigateway:*region*::/restapis/*api-id*/\***，表示 AWS 區域 *region* 中識別符為 *api-id* 之 API 的所有資源、方法、模型與階段。
- **arn:aws:apigateway:*region*::/restapis/*api-id*/resources/*resource-id*/\***，表示識別符為 *resource-id* 之資源中的所有資源與方法，該資源位於 AWS 區域 *region* 之識別符為 *api-id* 的 API 中。
- **arn:aws:apigateway:*region*::/restapis/*api-id*/resources/*resource-id*/methods/\***，表示識別符為 *resource-id* 之資源中的所有方法，該資源位於 AWS 區域 *region* 之識別符為 *api-id* 的 API 中。
- **arn:aws:apigateway:*region*::/restapis/*api-id*/resources/*resource-id*/methods/GET**，只表示識別符為 *resource-id* 之資源中的 GET 方法，該資源位於 AWS 區域 *region* 之識別符為 *api-id* 的 API 中。
- **arn:aws:apigateway:*region*::/restapis/*api-id*/models/\***，表示 AWS 區域 *region* 中識別符為 *api-id* 之 API 的所有模型。
- **arn:aws:apigateway:*region*::/restapis/*api-id*/models/*model-name***，表示名稱為 *model-name* 的模型，該模型位於 AWS 區域 *region* 之識別符為 *api-id* 的 API 中。
- **arn:aws:apigateway:*region*::/restapis/*api-id*/stages/\***，表示 AWS 區域 *region* 中識別符為 *api-id* 之 API 的所有階段。
- **arn:aws:apigateway:*region*::/restapis/*api-id*/stages/*stage-name***，只表示名稱為 *stage-name* 的階段，該模型位於 AWS 區域 *region* 之識別符為 *api-id* 的 API 中。

## 控制對 API 的跨帳戶存取權

您可以透過建立 IAM 許可政策以控制誰可以或不可以建立、更新、部署、檢視或刪除 API 實體來管理 API 的存取權。將政策連接到代表您的使用者的 IAM 使用者、包含使用者的 IAM 群組或使用者所擔任的 IAM 角色。

在為 API 建立的 IAM 政策中，您可以使用 Condition 元素，以僅允許存取特定 Lambda 整合或授權方。

Condition 區塊使用布林條件運算子，以符合在請求中的值適用的政策條件。StringXXX 條件運算子同時適用於 AWS 整合 (其中值應為 Lambda 函數 ARN) 和 Http 整合 (其中值應為 Http URI)。下列 StringXXX 條件運算值受支

援 : `StringEquals`、`StringNotEquals`、`StringEqualsIgnoreCase`、`StringNotEqualsIgnoreCase`、`StringLike`。如需詳細資訊，請參閱《IAM 使用者指南》中的[字串條件運算子](#)。

## 對於跨帳戶 Lambda 授權方的 IAM 政策

此為 IAM 政策的範例，其可控制跨帳戶 Lambda 授權方函數：

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "apigateway:POST"  
            ],  
            "Resource": [  
                "arn:aws:apigateway:[region]::/restapis/restapi_id/authorizers"  
            ],  
            //Create Authorizer operation is allowed only with the following Lambda  
            function  
            "Condition": {  
                "StringEquals": {  
                    "apigateway:AuthorizerUri":  
                    "arn:aws:apigateway:region:lambda:path/2015-03-31/functions/  
                    arn:aws:lambda:region:123456789012:function:example/invocations"  
                }  
            }  
        ]  
    }  
}
```

## 對於跨帳戶 Lambda 整合的 IAM 政策

使用跨帳戶整合，以限制對一些特定資源的操作（例如，特定 Lambda 函數的 `put-integration`），`Condition` 元素可新增到政策，以指定哪些資源（Lambda 函數）會受到影響。

此為 IAM 政策的範例，其可控制跨帳戶 Lambda 整合函數：

要授與另一個 AWS 帳戶許可呼叫 `integration:put` 或 `put-integration`，以在 API 中設定 Lambda 整合，您可以在 IAM 政策中包含下列陳述式。

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "apigateway:PUT"  
            ],  
            "Resource": [  
                "arn:aws:apigateway:api-region::/restApis/api-id/resources/resource-id/  
                methods/GET/integration"  
            ],  
            //PutIntegration is only valid with the following Lambda function  
            "Condition": {  
                "StringEquals": {  
                    "apigateway:IntegrationUri": "arn:aws:lambda:region:account-  
                    id:function:lambda-function-name"  
                }  
            }  
        ]  
    }  
}
```

}

## 允許其他帳戶來管理匯入 OpenAPI 檔案時使用的 Lambda 函數

要授與另一個 AWS 帳戶許可來呼叫 `restapi:import` 或 `import-rest-api` 以匯入 OpenAPI 檔案，您可以在 IAM 政策中包含下列陳述式。

在下列 Condition 陳述式中，字串 "lambda:path/2015-03-31/functions/arn:aws:lambda:**us-east-1:account-id:function:lambda-function-name**" 為 Lambda 函數的完整 ARN。

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "apigateway:POST"  
            ],  
            "Resource": "arn:aws:apigateway:*:::restapis",  
            "Condition": {  
                "StringLike": {  
                    "apigateway:IntegrationUri": [  
                        "arn:aws:apigateway:apigateway-region:lambda:path/2015-03-31/  
functions/arn:aws:lambda:lambda-region:account-id:function:lambda-function-name/  
invocations"  
                    ]  
                }  
            }  
        },  
        {  
            "Effect": "Allow",  
            "Action": [  
                "apigateway:POST"  
            ],  
            "Resource": "arn:aws:apigateway:*:::restapis",  
            "Condition": {  
                "StringLike": {  
                    "apigateway:AuthorizerUri": [  
                        "arn:aws:apigateway:apigateway-region:lambda:path/2015-03-31/  
functions/arn:aws:lambda:lambda-region:account-id:function:lambda-function-name/  
invocations"  
                    ]  
                }  
            }  
        }  
    ]  
}
```

## 控制呼叫 API 的存取

在本節中，您將了解如何撰寫 IAM 政策陳述式，以控制誰可以或無法在 API Gateway 中呼叫已部署的 API。在本例中，您也將尋找政策陳述式參考，包括與 API 執行服務相關之 Action 與 Resource 欄位的格式。

### Note

您也應研究 [the section called “資源政策如何影響授權工作流程” \(p. 326\)](#)中的 IAM 一節。

## 控制誰可以使用 IAM 政策呼叫 API Gateway API 方法

若要控制誰可以或無法使用 IAM 許可呼叫已部署的 API，請以必要的許可建立 IAM 政策文件。這類政策文件的範本如下所示。

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Permission",  
            "Action": [  
                "execute-api:Execution-operation"  
            ],  
            "Resource": [  
                "arn:aws:execute-api:region:account-id:api-id/stage/METHOD_HTTP_VERB/Resource-path"  
            ]  
        }  
    ]  
}
```

在本例中，*Permission* 將根據您要授予或撤銷內含許可，取代為 Allow 或 Deny。*Execution-operation* 將取代為 API 執行服務支援的操作。*METHOD\_HTTP\_VERB* 代表指定資源支援的 HTTP 動詞。*Resource-path* 是支援上述 *METHOD\_HTTP-VERB* 之已部署 API *Resource* 執行個體的 URL 路徑預留位置。如需詳細資訊，請參閱在 API Gateway 中執行 API 之 IAM 政策的陳述式參考 (p. 341)。

#### Note

若要讓 IAM 政策生效，您必須將方法的 *authorizationType* 屬性設定為 AWS\_IAM，以啟用 API 方法上的 IAM 身份驗證。不這樣做會使這些 API 方法可供公開存取。

例如，若要授予使用者檢視指定 API 所公開之寵物清單的許可，但拒絕使用者將寵物新增至清單的許可，您可以在 IAM 政策中包含下列陳述式：

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "execute-api:Invoke"  
            ],  
            "Resource": [  
                "arn:aws:execute-api:us-east-1:account-id:api-id/*/GET/pets"  
            ]  
        },  
        {  
            "Effect": "Deny",  
            "Action": [  
                "execute-api:Invoke"  
            ],  
            "Resource": [  
                "arn:aws:execute-api:us-east-1:account-id:api-id/*/POST/pets"  
            ]  
        }  
    ]  
}
```

若要授予使用者許可，讓他們可以檢視設定為 GET /pets/{petId} 之 API 所公開的特定寵物，您可以在 IAM 政策中包含以下陳述式：

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "execute-api:Invoke"  
            ]  
        }  
    ]  
}
```

```
        ],
        "Resource": [
            "arn:aws:execute-api:us-east-1:account-id:api-id/*/GET/pets/a1b2"
        ]
    }
}
```

對於測試 API 的開發人員團隊，您可以在 IAM 政策中包含建立下列陳述式，允許任何開發人員團隊在 test 階段中呼叫任何 API 之任何資源上的任何方法。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "execute-api:Invoke",
                "execute-api:InvalidateCache"
            ],
            "Resource": [
                "arn:aws:execute-api:*:*:*/test/*"
            ]
        }
    ]
}
```

## 在 API Gateway 中執行 API 之 IAM 政策的陳述式參考

下列資訊說明執行 API 的存取許可之 IAM 政策陳述式的動作與資源格式。

### 在 API Gateway 中執行 API 之許可的動作格式

API 執行 Action 表達式具有下列一般格式：

```
execute-api:action
```

其中 **action** 是可用的 API 執行動作：

- \*, 表示下列所有動作。
- Invoke，在用戶端請求下用來呼叫 API。
- InvalidateCache，在用戶端請求下用來使 API 快取失效。

### 在 API Gateway 中執行 API 之許可的資源格式

API 執行 Resource 表達式具有下列一般格式：

```
arn:aws:execute-api:region:account-id:api-id/stage-name/HTTP-VERB/resource-path-specifier
```

其中：

- region** 是對應到方法之已部署 API 的 AWS 區域 (例如 **us-east-1** 或表示所有 AWS 區域的 \*)。
- account-id** 是 REST API 擁有者之 12 位數的 AWS 帳戶 ID。
- api-id** 是 API Gateway 已指派給方法之 API 的識別符。(\*) 可用於所有 API，無論 API 的識別符為何)。
- stage-name** 是與方法相關聯之階段的名稱 (\* 可用於所有階段，無論階段的名稱為何)。
- HTTP-VERB** 是方法的 HTTP 動詞。它可以是下列其中一項：GET、POST、PUT、DELETE、PATCH。
- resource-path-specifier** 是所需方法的路徑 (\* 可用於所有路徑)。

一些範例資源表達式包括：

- **arn:aws:execute-api:\*:::\***，表示任何 AWS 區域中任何 API 之任何階段的任何資源路徑。(這相當於 \*)。
- **arn:aws:execute-api:us-east-1:::\***，表示 AWS 區域 us-east-1 中任何 API 之任何階段的任何資源路徑。
- **arn:aws:execute-api:us-east-1:::*api-id*/\***，表示 AWS 區域 us-east-1 中識別符為 *api-id* 的 API 之任何階段的任何資源路徑。
- **arn:aws:execute-api:us-east-1:::*api-id*/test/\***，表示 AWS 區域 us-east-1 中識別符為 *api-id* 的 API 之 test 階段的資源路徑。
- **arn:aws:execute-api:us-east-1:::*api-id*/test/\*/*mydemoresource*/\***，表示延著路徑 *mydemoresource* 的任何資源路徑，這是 AWS 區域 us-east-1 中識別符為 *api-id* 的 API 之 test 階段的任何 HTTP 方法路徑。
- **arn:aws:execute-api:us-east-1:::*api-id*/test/GET/*mydemoresource*/\***，表示延著路徑 *mydemoresource* 之任何資源路徑下的 GET 方法，其位於 AWS 區域 us-east-1 之識別符為 *api-id* 的 API 階段 test 中。

## 管理 API Gateway API 的 IAM 政策範例

下列範例政策文件示範設定在 API Gateway 中管理 API 資源存取許可的各種使用案例。如需許可模型與其他背景資訊，請參閱「[控制誰可以使用 IAM 政策建立及管理 API Gateway API \(p. 335\)](#)」。

### 主題

- [簡易讀取許可 \(p. 342\)](#)
- [任何 API 的唯讀許可 \(p. 343\)](#)
- [任何 API Gateway 資源的完整存取許可 \(p. 344\)](#)
- [管理 API 階段的完整存取許可 \(p. 344\)](#)
- [防止指定的使用者刪除任何 API 資源 \(p. 345\)](#)

### 簡易讀取許可

下列政策陳述式提供使用者許可，以取得 AWS 區域 us-east-1 之識別符為 a123456789 的 API 中所有資源、方法、模型與階段的相關資訊：

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "apigateway:GET"  
            ],  
            "Resource": [  
                "arn:aws:apigateway:us-east-1::/restapis/a123456789/*"  
            ]  
        }  
    ]  
}
```

下列範例政策陳述式提供 IAM 使用者許可，以列出任何區域中之所有資源、方法、模型與階段的資訊。使用者也具備許可，能夠執行 AWS 區域 us-east-1 中識別符為 a123456789 之 API 的所有可用 API Gateway 動作：

```
{
```

```
"Version": "2012-10-17",
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "apigateway:GET"
    ],
    "Resource": [
      "arn:aws:apigateway:*:*/restapis/*"
    ]
  },
  {
    "Effect": "Allow",
    "Action": [
      "apigateway:)"
    ],
    "Resource": [
      "arn:aws:apigateway:us-east-1::/restapis/a123456789/*"
    ]
  }
]
```

## 任何 API 的唯讀許可

下列政策文件將允許連接的實體 (使用者、群組或角色) 擷取發起人之 AWS 帳戶的任何 API。這包括 API 的任何子資源，例如方法、整合等。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Stmt1467321237000",
      "Effect": "Deny",
      "Action": [
        "apigateway:POST",
        "apigateway:PUT",
        "apigateway:PATCH",
        "apigateway:DELETE"
      ],
      "Resource": [
        "arn:aws:apigateway:us-east-1::/*"
      ]
    },
    {
      "Sid": "Stmt1467321341000",
      "Effect": "Deny",
      "Action": [
        "apigateway:GET"
      ],
      "Resource": [
        "arn:aws:apigateway:us-east-1::/",
        "arn:aws:apigateway:us-east-1::/account",
        "arn:aws:apigateway:us-east-1::/clientcertificates",
        "arn:aws:apigateway:us-east-1::/domainnames",
        "arn:aws:apigateway:us-east-1::/apikeys"
      ]
    },
    {
      "Sid": "Stmt1467321344000",
      "Effect": "Allow",
      "Action": [
        "apigateway:GET"
      ],
    }
]
```

```
        "Resource": [
            "arn:aws:apigateway:us-east-1::/restapis/*"
        ]
    }
}
```

第一個 Deny 陳述式明確禁止在 API Gateway 的任何資源上進行任何 POST、PUT、PATCH、DELETE 呼叫。這可確保這類許可不會被其他也連接到發起人的政策文件所覆寫。第二個 Deny 陳述式會防止發起人查詢根 (/) 資源、帳戶資訊 (/account)、用戶端憑證 (/clientcertificates)、自訂網域名稱 (/domainnames) 與 API 金鑰 (/apikeys)。三個陳述式合起來可確保發起人只能查詢 API 相關的資源。這在您不希望測試人員修改任何程式碼的 API 測試中可能很有用。

若要限制指定 API 的上述唯讀存取，請以下列內容取代 Resource 陳述式的 Allow 屬性：

```
"Resource": [ "arn:aws:apigateway:us-east-1::/restapis/restapi_id1/*",
    "arn:aws:apigateway:us-east-1::/restapis/restapi_id2/*" ]
```

## 任何 API Gateway 資源的完整存取許可

下列範例政策文件會將完整存取許可授予 AWS 帳戶的任何 API Gateway 資源。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Stmt1467321765000",
            "Effect": "Allow",
            "Action": [
                "apigateway:/*"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

一般而言，您應該避免使用這種廣泛且開放的存取政策。但對於您的 API 開發核心團隊而言則可能必須這麼做，如此一來，他們才可以建立、部署、更新及刪除任何 API Gateway 資源。

## 管理 API 階段的完整存取許可

下列範例政策文件會授予發起人 AWS 帳戶中任何 API 之階段相關資源的完整存取許可。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "apigateway:/*"
            ],
            "Resource": [
                "arn:aws:apigateway:us-east-1::/restapis/*/stages",
                "arn:aws:apigateway:us-east-1::/restapis/*/*"
            ]
        }
    ]
}
```

}

上述政策文件授予 `stages` 集合與任何內含 `stage` 資源的完整存取許可，但前提是沒有授予更多存取的其他政策已連接到發起人。否則，您必須明確拒絕所有其他存取。

使用上述政策時，發起人必須事先查明 REST API 的識別符，因為使用者無法呼叫 GET /restapis 來查詢可用的 API。此外，如果 `arn:aws:apigateway:us-east-1::/restapis/*/stages` 清單中未指定 `Resource`，階段資源會變成無法存取。在這種情況下，發起人將無法建立階段，也無法取得現有的階段，但只要階段名稱已知，則仍然可以檢視、更新或刪除階段。

若要授予特定 API 階段的許可，只要以 `restapis/restapi_id` 取代 `Resource` 規格的 `restapis/*` 部分，其中 `restapi_id` 是相關 API 的識別符。

## 防止指定的使用者刪除任何 API 資源

下列範例 IAM 政策文件會防止指定的使用者刪除 API Gateway 中的任何 API 資源。

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "Stmt1467331998000",  
            "Effect": "Allow",  
            "Action": [  
                "apigateway:GET",  
                "apigateway:PATCH",  
                "apigateway:POST",  
                "apigateway:PUT"  
            ],  
            "Resource": [  
                "arn:aws:apigateway:us-east-1::/restapis/*"  
            ]  
        },  
        {  
            "Sid": "Stmt1467332141000",  
            "Effect": "Allow",  
            "Action": [  
                "apigateway:DELETE"  
            ],  
            "Condition": {  
                "StringNotLike": {  
                    "aws:username": "johndoe"  
                }  
            },  
            "Resource": [  
                "arn:aws:apigateway:us-east-1::/restapis/*"  
            ]  
        }  
    ]  
}
```

此 IAM 政策會授予完整存取許可，以建立、部署、更新及刪除連接使用者、群組或角色的 API，但不包括無法刪除任何 API 資源的指定使用者 (`johndoe`)。它假設沒有其他任何授予根目錄、API 金鑰、用戶端憑證或自訂網域名稱 Allow 許可的政策文件已連接到發起人。

若要防止指定的使用者刪除特定 API Gateway 資源 (例如特定 API 或 API 的資源)，請將上述 `Resource` 規格取代為：

```
"Resource": ["arn:aws:apigateway:us-east-1::/restapis/restapi_id_1",  
             "arn:aws:apigateway:us-east-1::/restapis/restapi_id_2/resources"]
```

## API 執行許可的 IAM 政策範例

如需許可模型與其他背景資訊，請參閱「[控制呼叫 API 的存取 \(p. 339\)](#)」。

下列政策陳述式會授予使用者許可，以呼叫識別符為 a123456789 之 API 的階段 test 中沿著路徑 mydemoresource 的任何 POST 方法 (假設相應的 API 已部署至 AWS 區域 us-east-1)：

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "execute-api:Invoke"  
            ],  
            "Resource": [  
                "arn:aws:execute-api:us-east-1:*:a123456789/test/POST/mydemoresource/*"  
            ]  
        }  
    ]  
}
```

下列範例政策陳述式會提供使用者許可，在已部署識別符為 a123456789 之 API 的 AWS 區域中，呼叫對應 API 之任何階段中資源路徑 petstorewalkthrough/pets 上的任何方法：

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "execute-api:Invoke"  
            ],  
            "Resource": [  
                "arn:aws:execute-api:*:*:a123456789/*/*/petstorewalkthrough/pets"  
            ]  
        }  
    ]  
}
```

## 建立政策並連接到 IAM 使用者

若要讓使用者呼叫 API 管理服務或 API 執行服務，您必須為 IAM 使用者建立 IAM 政策，以控制 API Gateway 實體的存取，然後將政策連接到 IAM 使用者。下列步驟說明如何建立您的 IAM 政策。

### 建立您自己的 IAM 政策

1. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.
2. 選擇 Policies (政策)，然後選擇 Create Policy (建立政策)。出現 Get Started (開始使用) 按鈕時，選擇它，然後選擇 Create Policy (建立政策)。
3. 在 Create Your Own Policy (建立您自己的政策) 旁邊，選擇 Select (選取)。
4. 針對 Policy Name (政策名稱)，輸入您稍後容易參考的任何值。(選擇性) 在 Description (說明) 中輸入描述性文字。
5. 針對 Policy Document (政策文件)，使用下列格式輸入政策文件，然後選擇 Create Policy (建立政策)：

```
{  
    "Version": "2012-10-17",
```

```
"Statement" : [
    {
        "Effect" : "Allow",
        "Action" : [
            "action-statement"
        ],
        "Resource" : [
            "resource-statement"
        ]
    },
    {
        "Effect" : "Allow",
        "Action" : [
            "action-statement"
        ],
        "Resource" : [
            "resource-statement"
        ]
    }
]
```

在此陳述式中，視需要替代 *action-statement* 與 *resource-statement*，並新增其他陳述式，以指定要讓 IAM 使用者管理的 API Gateway 實體、IAM 使用者可以呼叫的 API 方法，或指定兩者。根據預設，除非有明確對應的 Allow 陳述式，否則 IAM 使用者沒有許可。

6. 若要為使用者啟用政策，請選擇 Users (使用者)。
7. 選擇要連接政策的 IAM 使用者。

您剛建立 IAM 政策。在您將它連接到 IAM 使用者、包含使用者的 IAM 群組或使用者所擔任的 IAM 角色之前，它不會有任何作用。

#### 將 IAM 政策連接到 IAM 使用者

1. 針對已選擇的使用者，選擇 Permissions (許可) 標籤，然後選擇 Attach Policy (連接政策)。
2. 在 Grant permissions (授予許可) 下，選擇 Attach existing policies directly (直接連接現有政策)。
3. 從顯示的清單中選擇剛建立的政策文件，然後選擇 Next: Review (下一步：檢閱)。
4. 在 Permissions summary (許可摘要) 下，選擇 Add permissions (新增許可)。

或者，您可以將使用者新增至 IAM 群組（如果使用者還不是成員），並將政策文件新增至群組，讓連接的政策適用於所有群組成員。這有助於管理及更新 IAM 使用者群組的政策組態。在下一節中，我們重點說明如何將政策連接到 IAM 群組（假設您已建立群組並將使用者新增至群組）。

#### 將 IAM 政策文件連接到 IAM 群組

1. 從主要導覽窗格中，選擇 Groups (群組)。
2. 在所選擇的群組下，選擇 Permissions (許可) 標籤。
3. 選擇 Attach policy (連接政策)。
4. 選擇您之前建立的政策文件，然後選擇 Attach policy (連接政策)。

若要讓 API Gateway 代表您呼叫其他 AWS 服務，請建立 Amazon API Gateway 類型的 IAM 角色。

#### 建立 Amazon API Gateway 類型的角色

1. 從主要導覽窗格中，選擇 Roles (角色)。
2. 選擇 Create New Role (建立新角色)。
3. 針對 Role name (角色名稱) 輸入名稱，然後選擇 Next Step (下一步)。

4. 在 Select Role Type (選取角色類型) 下的 AWS Service Roles (AWS 服務角色) 中，選擇 Amazon API Gateway 旁的 Select (選取)。
5. 在 Attach Policy (連接政策) 下，選擇可用的受管 IAM 許可政策 (例如，如果您想要讓 API Gateway 在 CloudWatch 中記錄指標，請選擇 AmazonAPIGatewayPushToCloudWatchLog)，然後選擇 Next Step (下一步)。
6. 在 Trusted Entities (信任的實體) 下，確認 apigateway.amazonaws.com 列為項目，然後選擇 Create Role (建立角色)。
7. 在新建立的角色中，選擇 Permissions (許可) 標籤，然後選擇 Attach Policy (連接政策)。
8. 選擇之前建立的自訂 IAM 政策文件，然後選擇 Attach Policy (連接政策)。

## 使用標籤來控制對 API Gateway 中的 REST API 的存取

您可以在 IAM 政策中使用標籤型存取控制，以微調對 REST API 的存取許可。

如需詳細資訊，請參閱[the section called “標記型存取控制” \(p. 623\)](#)。

## 使用 API Gateway Lambda 授權方

Lambda 授權方 (先前稱為自訂授權方) 是一種 API Gateway 功能，其會使用 Lambda 函數控制對 API 的存取。

如果您想要實作自訂授權方機制，Lambda 授權方很有用，而此機制會使用承載符記身份驗證策略(例如 OAuth 或 SAML)，或使用請求參數來判定發起人的身分。

當用戶端對您的其中一個 API 方法發出請求時，API Gateway 會呼叫您的 Lambda 授權方，其會採取發起人的身分做為輸入，並傳回 IAM 政策做為輸出。

有兩種類型的 Lambda 授權方：

- 符記型 Lambda 授權方 (也稱為 TOKEN 授權方) 會以承載符記接收發起人的身分，例如 JSON Web 符記 (JWT) 或 OAuth 符記。
- 請求參數型 Lambda 授權方 (也稱為 REQUEST 授權方) 會以標頭、查詢字串參數、[stageVariables \(p. 281\)](#) 和 [\\$context \(p. 274\)](#) 變數的組合方式接收發起人的身分。

對於 WebSocket API，只支援請求參數型授權方。

您可以使用與您在 Lambda 授權方函數中建立之函數不同，來自 AWS 帳戶的 AWS Lambda 參數。如需詳細資訊，請參閱[the section called “設定跨帳戶 Lambda 授權方” \(p. 362\)](#)。

如果您目前已使用 Amazon Cognito User Pools 來管理您的使用者帳戶，或如果您考慮這樣做，則可能想要考慮對您的 API 使用 [Amazon Cognito 使用者集區做為授權方 \(p. 363\)](#)，而不是 Lambda 授權方。

對於 WebSocket API，只支援請求參數型授權方。

若要查看 .NET Lambda 授權方的範例，請檢視[在 .NET Core 2.0 建立 API 閘道自訂授權方](#)影片。

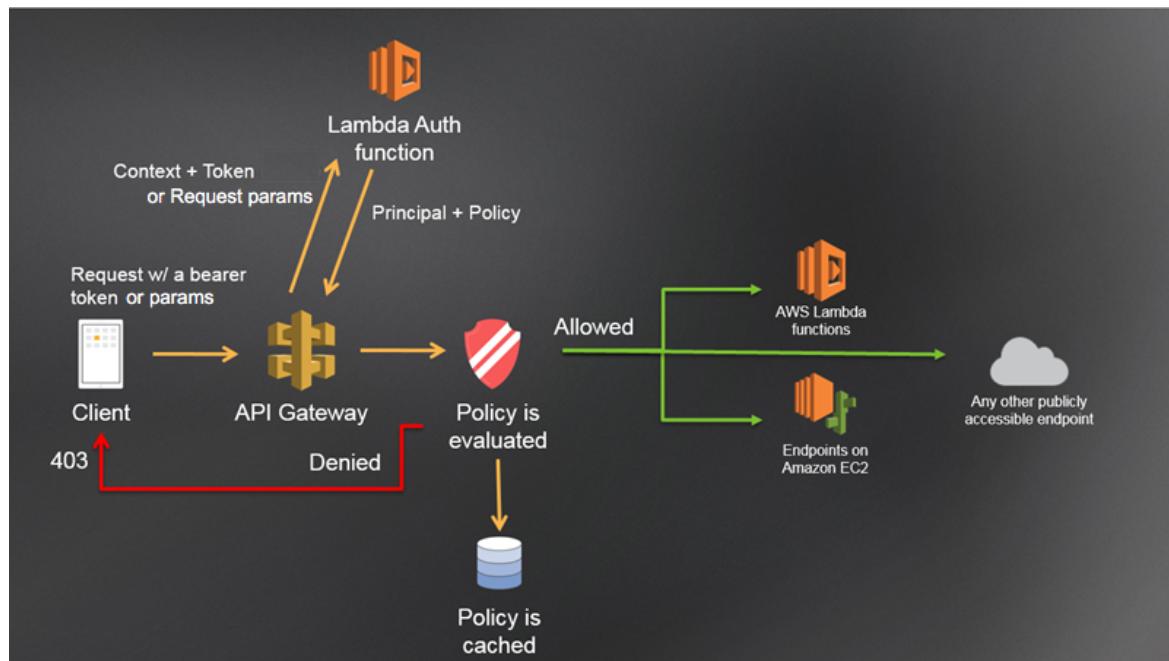
### 主題

- [Lambda 授權方授權工作流程 \(p. 349\)](#)
- [建立 API Gateway Lambda 授權方的步驟 \(p. 349\)](#)
- [在 Lambda 主控台建立 API Gateway Lambda 授權方函數 \(p. 350\)](#)
- [使用 API Gateway 主控台設定 Lambda 授權方 \(p. 355\)](#)
- [使用 AWS CLI 或 AWS 開發套件設定 Lambda 授權方 \(p. 356\)](#)

- Amazon API Gateway Lambda 授權方的輸入 (p. 357)
- Amazon API Gateway Lambda 授權方的輸出 (p. 358)
- 呼叫具有 API Gateway Lambda 授權方的 API (p. 360)
- 設定跨帳戶 Lambda 授權方 (p. 362)

## Lambda 授權方授權工作流程

下圖說明 Lambda 授權方的授權工作流程。



### API Gateway Lambda 授權工作流程

1. 用戶端會對 API Gateway API 方法呼叫一種方法，傳遞承載符記或請求參數。
2. API Gateway 會檢查是否已對方法設定 Lambda 授權方。若是，API Gateway 會呼叫 Lambda 函數。
3. Lambda 函數透過如下方式驗證發起人：
  - 呼叫 OAuth 提供者來取得 OAuth 存取符記。
  - 呼叫 SAML 提供者來取得 SAML 聲明。
  - 根據請求參數值產生 IAM 政策。
  - 從資料庫擷取登入資料。
4. 如果呼叫成功，則 Lambda 函數會傳回一個輸出物件，其中至少包含一個 IAM 政策和一個委託人識別符，來授予存取權。
5. API Gateway 會評估政策。
  - 如果拒絕存取，API Gateway 會傳回一個適當的 HTTP 狀態碼，例如 403 ACCESS\_DENIED。
  - 如果允許存取，API Gateway 會執行方法。如果在授權方設定中啟用快取，Lambda 也會快取政策，以便不需要重新叫用 API Gateway 授權方函數。

## 建立 API Gateway Lambda 授權方的步驟

若要建立 Lambda 授權方，您需要執行下列工作：

1. 在 Lambda 主控台中建立 Lambda 授權方函數，如[the section called “在 Lambda 主控台建立 Lambda 授權方” \(p. 350\)](#)所述。您可以使用其中一個藍圖範例做為起點，並視需要自訂輸入 (p. 357)和輸出 (p. 358)。
2. 將 Lambda 函數設定為 API Gateway 授權方，並將 API 方法設定為需要它，如[the section called “使用主控台設定 Lambda 授權方” \(p. 355\)](#)所述。或者，如果您需要跨帳戶 Lambda 授權方，請參閱 [the section called “設定跨帳戶 Lambda 授權方” \(p. 362\)](#)。

Note

您也可以使用 AWS CLI 或 AWS 開發套件設定授權方。請參閱 [the section called “使用 AWS CLI 或 AWS 開發套件設定 Lambda 授權方” \(p. 356\)](#)。

3. 使用 Postman 測試您的授權方，如 [the section called “呼叫具有 Lambda 授權方的 API” \(p. 360\)](#) 所述。

## 在 Lambda 主控台建立 API Gateway Lambda 授權方函數

設定 Lambda 授權方之前，您必須先建立 Lambda 函數，以實作邏輯來授權並視需要驗證發起人。Lambda 主控台會提供 Python 藍圖，您可以選擇 Use a blueprint (使用的藍圖)，並選擇 api-gateway-authorizer-python 藍圖來使用此藍圖。否則，您將想要使用 [awslabs GitHub 儲存庫](#) 的其中一個藍圖做為起點。

對於本節中的範例 Lambda 授權方函數，由於不會呼叫其他服務，您可以使用內建的 [AWSLambdaBasicExecutionRole](#)。為您自己的 API Gateway Lambda 授權方建立 Lambda 函數時，如果 Lambda 函數呼叫其他 AWS 服務，則您需要將 IAM 執行角色指派給該函數。若要建立角色，請依照 [AWS Lambda 執行角色](#) 中的指示。

### 範例：建立符記型 Lambda 授權方函數

若要建立符記型 Lambda 授權方函數，請在 Lambda 主控台中輸入以下 Node.js 8.10 程式碼，並在 API Gateway 主控台中測試它，如下所示。

1. 在 Lambda 主控台中，請選擇 Create function (建立函數)。
2. 選擇 Author from scratch (從頭開始撰寫)。
3. 輸入函數的名稱。
4. 對於 Runtime (執行時間)，請選擇 Node.js 8.10。
5. 選擇 Create function (建立函數)。
6. 複製以下程式碼並貼至程式碼編輯器。

```
// A simple token-based authorizer example to demonstrate how to use an authorization
// token
// to allow or deny a request. In this example, the caller named 'user' is allowed to
// invoke
// a request if the client-supplied token value is 'allow'. The caller is not allowed
// to invoke
// the request if the token value is 'deny'. If the token value is 'unauthorized' or an
// empty
// string, the authorizer function returns an HTTP 401 status code. For any other token
// value,
// the authorizer returns an HTTP 500 status code.
// Note that token values are case-sensitive.

exports.handler = function(event, context, callback) {
    var token = event.authorizationToken;
    switch (token) {
        case 'allow':
            callback(null, generatePolicy('user', 'Allow', event.methodArn));
            break;
```

```
        case 'deny':
            callback(null, generatePolicy('user', 'Deny', event.methodArn));
            break;
        case 'unauthorized':
            callback("Unauthorized");    // Return a 401 Unauthorized response
            break;
        default:
            callback("Error: Invalid token"); // Return a 500 Invalid token response
    }
};

// Help function to generate an IAM policy
var generatePolicy = function(principalId, effect, resource) {
    var authResponse = {};

    authResponse.principalId = principalId;
    if (effect && resource) {
        var policyDocument = {};
        policyDocument.Version = '2012-10-17';
        policyDocument.Statement = [];
        var statementOne = {};
        statementOne.Action = 'execute-api:Invoke';
        statementOne.Effect = effect;
        statementOne.Resource = resource;
        policyDocument.Statement[0] = statementOne;
        authResponse.policyDocument = policyDocument;
    }

    // Optional output with custom properties of the String, Number or Boolean type.
    authResponse.context = {
        "stringKey": "stringval",
        "numberKey": 123,
        "booleanKey": true
    };
    return authResponse;
}
```

7. 選擇 Save (儲存)。
8. 在 API Gateway 主控台，建立[簡單 API \(p. 40\)](#) (如果尚未建立的話)。
9. 從 API 清單中選擇您的 API。
10. 選擇 Authorizers (授權方)。
11. 選擇 Create New Authorizer (建立新的授權方)。
12. 輸入授權方的名稱。
13. 針對 Type (類型)，選擇 Lambda。
14. 針對 Lambda Function (Lambda 函數)，選擇已建立 Lambda 授權方函數的區域，然後從下拉式清單中選擇函數名稱。
15. 將 Lambda Invoke Role (Lambda 叫用角色) 保留空白。
16. 針對 Lambda Event Payload (Lambda 事件承載)，選擇 Token (符記)。
17. 針對 Token Source (符記來源)，輸入 **tokenHeader**。
18. 選擇 Test (測試)。
19. 針對 tokenHeader 值，輸入 **allow**。
20. 選擇 Test (測試)。

在這個範例中，當 API 收到方法請求時，API Gateway 會在 `event.authorizationToken` 屬性中將來源符記傳遞至這個 Lambda 授權方函數。Lambda 授權方函數會讀取符記，並運作如下：

- 如果符記值為 '`allow`'，授權方函數會傳回一個 `200 OK` HTTP 回應和一個如下的 IAM 政策，而且方法請求會成功：

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Action": "execute-api:Invoke",  
            "Effect": "Allow",  
            "Resource": "arn:aws:execute-api:us-east-1:123456789012:ivdtdhp7b5/ESTestInvoke-  
stage/GET/"  
        }  
    ]  
}
```

- 如果符記值為 'deny'，授權方函數會傳回一個 403 Forbidden HTTP 回應和一個如下的 Deny IAM 政策，而且方法請求會失敗：

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Action": "execute-api:Invoke",  
            "Effect": "Deny",  
            "Resource": "arn:aws:execute-api:us-east-1:123456789012:ivdtdhp7b5/ESTestInvoke-  
stage/GET/"  
        }  
    ]  
}
```

- 如果符記值為 'unauthorized' 或空字串，授權方函數會傳回 401 Unauthorized HTTP 回應，而且方法呼叫會失敗。
- 如果符記為其他值，用戶端會收到 500 Invalid token 回應，而且方法呼叫會失敗。

#### Note

在生產程式碼中，您可能需要驗證使用者，再授予授權。若是如此，您也可以在 Lambda 函數中新增驗證邏輯，方法為依照該提供者文件中的指示來呼叫身分驗證提供者。

除了傳回 IAM 政策，Lambda 授權方函數還必須傳回發起人的主體識別符。它也可以選擇地傳回 context 物件，其中包含可傳入整合後端的其他資訊。如需詳細資訊，請參閱 [Amazon API Gateway Lambda 授權方的輸出 \(p. 358\)](#)。

### 範例：建立請求型 Lambda 授權方函數

若要建立請求型 Lambda 授權方函數，請在 Lambda 主控台中輸入以下 Node.js 8.10 程式碼，並在 API Gateway 主控台中測試它，如下所示。

- 在 Lambda 主控台中，請選擇 Create function (建立函數)。
- 選擇 Author from scratch (從頭開始撰寫)。
- 輸入函數的名稱。
- 對於 Runtime (執行時間)，請選擇 Node.js 8.10。
- 選擇 Create function (建立函數)。
- 複製以下程式碼並貼至程式碼編輯器。

```
exports.handler = function(event, context, callback) {  
    console.log('Received event:', JSON.stringify(event, null, 2));
```

```
// A simple request-based authorizer example to demonstrate how to use request
// parameters to allow or deny a request. In this example, a request is
// authorized if the client-supplied HeaderAuth1 header, QueryString1
// query parameter, and stage variable of StageVar1 all match
// specified values of 'headerValue1', 'queryValue1', and 'stageValue1',
// respectively.

// Retrieve request parameters from the Lambda function input:
var headers = event.headers;
var queryStringParameters = event.queryStringParameters;
var pathParameters = event.pathParameters;
var stageVariables = event.stageVariables;

// Parse the input for the parameter values
var tmp = event.methodArn.split(':');
var apiGatewayArnTmp = tmp[5].split('/');
var awsAccountId = tmp[4];
var region = tmp[3];
var restApiId = apiGatewayArnTmp[0];
var stage = apiGatewayArnTmp[1];
var method = apiGatewayArnTmp[2];
var resource = '/'; // root resource
if (apiGatewayArnTmp[3]) {
    resource += apiGatewayArnTmp[3];
}

// Perform authorization to return the Allow policy for correct parameters and
// the 'Unauthorized' error, otherwise.
var authResponse = {};
var condition = {};
conditionIpAddress = {};

if (headers.HeaderAuth1 === "headerValue1"
    && queryStringParameters.QueryString1 === "queryValue1"
    && stageVariables.StageVar1 === "stageValue1") {
    callback(null, generateAllow('me', event.methodArn));
} else {
    callback("Unauthorized");
}
}

// Help function to generate an IAM policy
var generatePolicy = function(principalId, effect, resource) {
    // Required output:
    var authResponse = {};
    authResponse.principalId = principalId;
    if (effect && resource) {
        var policyDocument = {};
        policyDocument.Version = '2012-10-17'; // default version
        policyDocument.Statement = [];
        var statementOne = {};
        statementOne.Action = 'execute-api:Invoke'; // default action
        statementOne.Effect = effect;
        statementOne.Resource = resource;
        policyDocument.Statement[0] = statementOne;
        authResponse.policyDocument = policyDocument;
    }
    // Optional output with custom properties of the String, Number or Boolean type.
    authResponse.context = {
        "stringKey": "stringval",
        "numberKey": 123,
        "booleanKey": true
    };
    return authResponse;
}
```

```
var generateAllow = function(principalId, resource) {
    return generatePolicy(principalId, 'Allow', resource);
}

var generateDeny = function(principalId, resource) {
    return generatePolicy(principalId, 'Deny', resource);
}
```

7. 選擇 Save (儲存)。
8. 在 API Gateway 主控台，建立[簡單 API \(p. 40\)](#) (如果尚未建立的話)。
9. 從 API 清單中選擇您的 API。
10. 選擇 Authorizers (授權方)。
11. 選擇 Create New Authorizer (建立新的授權方)。
12. 輸入授權方的名稱。
13. 針對 Type (類型)，選擇 Lambda。
14. 針對 Lambda Function (Lambda 函數)，選擇已建立 Lambda 授權方函數的區域，然後從下拉式清單中選擇函數名稱。
15. 將 Lambda Invoke Role (Lambda 叫用角色) 保留空白。
16. 針對 Lambda Event Payload (Lambda 事件承載)，選擇 Request (請求)。
17. 在 Identity Sources (身分來源) 下，新增 Header (標頭) (名為 **HeaderAuth1**)、Query String (查詢字串) (名為 **queryString1**)，以及 Stage Variable (階段變數) (名為 **StageVar1**)。
18. 選擇 Test (測試)。
19. 針對 HeaderAuth1，輸入 **headerValue1**。針對 QueryString1，輸入 **queryValue1**。針對 StageVar1，輸入 **stageValue1**。
20. 選擇 Test (測試)。

在這個範例中，Lambda 授權方函數會檢查輸入參數，並運作如下：

- 如果所有必要參數符合預期值，授權方函數會傳回一個 200 OK HTTP 回應和一個如下的 IAM 政策，而且方法請求會成功：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": "execute-api:Invoke",
            "Effect": "Allow",
            "Resource": "arn:aws:execute-api:us-east-1:123456789012:ivdtdhp7b5/ESTestInvoke-
stage/GET/"
        }
    ]
}
```

- 否則，授權方函數會傳回 401 Unauthorized HTTP 回應，而且方法呼叫會失敗。

#### Note

在生產程式碼中，您可能需要驗證使用者，再授予授權。若是如此，您也可以在 Lambda 函數中新增驗證邏輯，方法為依照該提供者文件中的指示來呼叫身分驗證提供者。

除了傳回 IAM 政策，Lambda 授權方函數還必須傳回發起人的主體識別符。它也可以選擇地傳回 context 物件，其中包含可傳入整合後端的其他資訊。如需詳細資訊，請參閱[Amazon API Gateway Lambda 授權方的輸出 \(p. 358\)](#)。

## 使用 API Gateway 主控台設定 Lambda 授權方

建立 Lambda 函數並確認其運作正常之後，請使用下列步驟在 API Gateway 主控台中設定 API Gateway Lambda 授權方（先前稱作自訂授權方）。

### 使用 API Gateway 主控台設定 Lambda 授權方

1. 登入 API Gateway 主控台。
2. 建立新的 API 或選取現有的 API，然後選擇該 API 下的 Authorizers（授權方）。
3. 選擇 Create New Authorizer（建立新的授權方）。
4. 對於 Create Authorizer（建立授權方），在 Name（名稱）輸入欄位中輸入授權方名稱。
5. 對於 Type（類型），選擇 Lambda 選項。
6. 對於 Lambda Function（Lambda 函數），選擇一個區域，然後選擇在您帳戶中可用 Lambda 授權方的函數。
7. 將 Lambda Invoke Role (&LAM; 呼叫角色) 保留空白，讓 API Gateway 主控台設定資源類型政策。該政策會授予 API Gateway 許可來呼叫授權方 Lambda 函數。您也可以選擇輸入 IAM 角色的名稱，來允許 API Gateway 呼叫授權方 Lambda 函數。如需這類角色的範例，請參閱「[建立可擔任的 IAM 角色 \(p. 87\)](#)」。

如果您選擇讓 API Gateway 主控台設定資源類型政策，則會顯示 Add Permission to Lambda Function（新增 Lambda 函數的許可）對話方塊。選擇 OK（確定）。建立 Lambda 授權之後，您可以使用適當的授權字符串進行測試，以驗證其是否如預期般運作。

8. 對於 Lambda Event Payload (&LAM; 事件承載)，選擇 Token（字符）代表 TOKEN 授權方，或 Request（請求）代表 REQUEST 授權方。（這相當於將 type 屬性設定為 TOKEN 或 REQUEST）。
9. 根據上一個步驟的選擇，執行下列其中一項操作：
  - a. 對於 Token（字符）選項，執行下列操作：
    - 在 Token Source（字符來源）中，輸入標頭的名稱。API 用戶端必須包含此名稱的標頭，才能將授權字符串傳送到 Lambda 授權方。
    - 或者，在 Token Validation 輸入欄位中提供 RegEx 陳述式。API Gateway 會對此表達式執行輸入字符的初始驗證，並在成功驗證時呼叫授權方。這有助於降低支付無效字符費用的機率。
    - 針對 Authorization Caching（授權快取），根據您是否想要快取授權方所產生的授權政策，選取或清除 Enabled（已啟用）選項。啟用政策快取時，您可以選擇修改 TTL 值。將 TTL 設定為零會停用政策快取。啟用政策快取時，Token Source（字符來源）中指定的標頭名稱會成為快取金鑰。

#### Note

預設 TTL 值為 300 秒。最大值為 3600 秒，而且無法增加此限制。

- b. 對於 Request（請求）選項，執行下列操作：

- 針對 Identity Sources（身分來源），輸入所選參數類型的請求參數名稱。支援的參數類型為 Header、Query String、Stage Variable 與 Context。若要新增更多身分來源，請選擇 Add Identity Source（新增身分來源）。

API Gateway 使用指定的身分來源做為請求授權方快取金鑰。啟用快取時，只有在成功驗證執行時間存在所有指定的身分來源之後，API Gateway 才會呼叫授權方的 Lambda 函數。如果指定的身分來源遺漏、為 Null 或空白，API Gateway 會傳回 401 Unauthorized 回應，而不會呼叫授權方 Lambda 函數。

定義多個身分來源時，會使用這些來源衍生授權方的快取金鑰。變更任何快取金鑰部分都會使授權方捨棄快取的政策文件，並產生新的文件。

- 針對 Authorization Caching（授權快取），根據您是否想要快取授權方所產生的授權政策，選取或取消選取 Enabled（已啟用）選項。啟用政策快取時，您可以選擇修改 TTL 的預設值（300）。設定 TTL=0 會停用政策快取。

停用快取時，不需指定身分來源。API Gateway 不會在呼叫授權方 Lambda 函數之前執行任何驗證。

#### Note

若要啟用快取，您的授權方必須傳回適用於 API 之所有方法的政策。若要強制執行特定方法的政策，您可以將 TTL 值設定為零來停用 API 的政策快取。

10. 選擇 Create (建立)，建立所選 API 的新 Lambda 授權方。
11. 建立 API 的授權方之後，您可以選擇性地測試呼叫授權方，再對方法進行設定。

若是 TOKEN 授權方，請在 Identity token (身分字符) 輸入文字欄位中輸入有效的字符，然後選擇 Test (測試)。該字符會當做您在授權方的 Identity token source (身分字符來源) 設定中指定的標頭傳遞到 Lambda 函數。

若是 REQUEST 授權方，請輸入對應到指定身分來源的有效請求參數，然後選擇 Test (測試)。

除了使用 API Gateway 主控台，您還可以使用 AWS CLI 或適用於 API Gateway 的 AWS 開發套件來測試呼叫授權方。若要使用 AWS CLI 來執行這項操作，請參閱 [test-invoke-authorizer](#)。

#### Note

測試呼叫授權方的 test-invoke 方法執行是獨立的程序。

若要使用 API Gateway 主控台測試呼叫方法，請參閱 [使用主控台測試 REST API 方法 \(p. 498\)](#)。若要使用 AWS CLI 測試呼叫方法，請參閱 [test-invoke-method](#)。

若要測試呼叫方法與已設定的授權方，請部署 API，然後使用 cURL 或 Postman 呼叫方法，並提供必要的字符或請求參數。

下一個程序示範如何設定 API 方法來使用 Lambda 授權方。

#### 設定 API 方法使用 Lambda 授權方

1. 回到 API。建立新的方法或選擇現有的方法。如果需要，請建立新的資源。
2. 在 Method Execution (方法執行) 中，選擇 Method Request (方法請求) 連結。
3. 在 Settings (設定) 下，展開 Authorization (授權) 下拉式清單以選取您剛建立的 Lambda 授權方 (例如 myTestApiAuthorizer)，然後選擇核取記號圖示以儲存選擇。
4. (選擇性) 同樣在 Method Request (方法請求) 頁面上，如果您也想要將授權字符傳遞到後端，請選擇 Add header (新增標頭)。在 Name (名稱) 中，輸入符合您在建立 API 的 Lambda 授權方時所指定之 Token Source (字符來源) 名稱的標頭名稱。然後，選擇核取記號圖示以儲存設定。此步驟不適用於 REQUEST 授權方。
5. 選擇 Deploy API (部署 API) 將 API 部署到階段。記下 Invoke URL (呼叫 URL) 值。呼叫 API 時需要此值。針對使用階段變數的 REQUEST 授權方，您還必須定義必要的階段變數，並在 Stage Editor (階段編輯器) 中指定其值。

## 使用 AWS CLI 或 AWS 開發套件設定 Lambda 授權方

您也可以使用 AWS CLI 或 AWS 開發套件設定 API Gateway Lambda 授權方。

首先，在擁有 API 許可的帳戶中授予 IAM 委託人，來呼叫 `authorizer:create` 或 `create-authorizer`，以建立 Lambda 授權方中使用的 Lambda 函數。若要授予此許可，您需要建立下列 IAM 政策：

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {
```

```
"Effect": "Allow",
"Action": [
    "apigateway:POST"
],
"Resource": [
    "arn:aws:apigateway:region::/restapis/restapi_id/authorizers"
],
//Create Authorizer operation is allowed only with the following Lambda
function
"Condition": {
    "StringEquals": {
        "apigateway:AuthorizerUri": "arn:aws:lambda:region:account-
id:function:lambda-function-name"
    }
}
]
```

## Amazon API Gateway Lambda 授權方的輸入

針對 TOKEN 類型的 Lambda 授權方 (先前稱作自訂授權方)，您必須在設定 API 的授權方時，將自訂標頭指定為 Token Source (字符來源)。API 用戶端必須在傳入請求中以該標頭傳遞必要的授權符記。收到傳入方法請求時，API Gateway 會從自訂標頭中擷取字符。然後，它會傳遞字符做為 Lambda 函數之 event 物件的 authorizationToken 屬性，以及方法 ARN 做為 methodArn 屬性：

```
{
    "type": "TOKEN",
    "authorizationToken": "{caller-supplied-token}",
    "methodArn": "arn:aws:execute-api:{regionId}:{accountId}:{appId}/{stage}/{httpVerb}/
[resource]/[child-resources]]"
}
```

在此範例中，type 屬性會指定 TOKEN 授權方做為授權方類型。*{caller-supplied-token}* 來自用戶端請求的授權標頭。methodArn 是傳入方法請求的 ARN，並會由 API Gateway 根據 Lambda 授權方組態填入。

在上一節所示的 TOKEN 授權方 Lambda 函數範例中，*{caller-supplied-token}* 字串是 allow、deny、unauthorized 或任何其他字串值。空白字串值與 unauthorized 相同。以下示範這類輸入，該範例會在任何階段 (\*) 中 AWS 帳戶 (123456789012) 之 API (yml8tbxw7b) 的 GET 方法上取得 Allow 政策。

```
{
    "type": "TOKEN",
    "authorizationToken": "allow",
    "methodArn": "arn:aws:execute-api:us-west-2:123456789012:yml8tbxw7b/*/GET/"
}
```

針對 REQUEST 類型的 Lambda 授權方，API Gateway 會將必要的請求參數傳遞到授權方 Lambda 函數，做為 event 物件的一部分。受影響的請求參數包括標頭、路徑參數、查詢字串參數、階段變數與一些請求上下文變數。API 發起人可以設定路徑參數、標頭與查詢字串參數。API 開發人員必須在 API 部署期間設定階段變數，而 API Gateway 則會在執行時間提供請求內容。

### Note

可以將路徑參數當作傳遞到 Lambda 授權方函數的要求參數，但它們無法用作身分來源。

下列範例顯示具有代理整合之 API 方法 (REQUEST) 的 GET /request 授權方輸入：

```
{
```

```
"type": "REQUEST",
"methodArn": "arn:aws:execute-api:us-east-1:123456789012:s4x3opwd6i/test/GET/request",
"resource": "/request",
"path": "/request",
"httpMethod": "GET",
"headers": {
    "X-AMZ-Date": "20170718T062915Z",
    "Accept": "*/*",
    "HeaderAuth1": "headerValue1",
    "CloudFront-Viewer-Country": "US",
    "CloudFront-Forwarded-Proto": "https",
    "CloudFront-Is-Tablet-Viewer": "false",
    "CloudFront-Is-Mobile-Viewer": "false",
    "User-Agent": "...",
    "X-Forwarded-Proto": "https",
    "CloudFront-Is-SmartTV-Viewer": "false",
    "Host": "...execute-api.us-east-1.amazonaws.com",
    "Accept-Encoding": "gzip, deflate",
    "X-Forwarded-Port": "443",
    "X-Amzn-Trace-Id": "...",
    "Via": "...cloudfront.net (CloudFront)",
    "X-Amz-Cf-Id": "...",
    "X-Forwarded-For": "..., ...",
    "Postman-Token": "...",
    "cache-control": "no-cache",
    "CloudFront-Is-Desktop-Viewer": "true",
    "Content-Type": "application/x-www-form-urlencoded"
},
"queryStringParameters": {
    "QueryString1": "queryValue1"
},
"pathParameters": {},
"stageVariables": {
    "StageVar1": "stageValue1"
},
"requestContext": {
    "path": "/request",
    "accountId": "123456789012",
    "resourceId": "05c7jb",
    "stage": "test",
    "requestId": "...",
    "identity": {
        "apiKey": "...",
        "sourceIp": ...
    },
    "resourcePath": "/request",
    "httpMethod": "GET",
    "apiId": "s4x3opwd6i"
}
}
```

`requestContext` 是鍵值對的對應，並對應到 [\\$context \(p. 274\)](#) 變數。它的結果是與 API 相關。API Gateway 可能會將新鍵新增至對應。如需 Lambda 代理整合中 Lambda 函數輸入的詳細資訊，請參閱[代理整合之 Lambda 函數的輸入格式 \(p. 212\)](#)。

## Amazon API Gateway Lambda 授權方的輸出

Lambda 授權方函數的輸出是類似字典的物件，其中必須包含主體識別符 (`principalId`) 與列出政策陳述式的政策文件 (`policyDocument`)。此輸出也可包含由鍵值對組成的 `context` 對應。如果 API 使用用量計劃 (`apiKeySource` 設為 `AUTHORIZER`)，Lambda 授權方函數必須傳回其中一個用量計劃的 API 金鑰做為 `usageIdentifierKey` 屬性值。

以下示範此輸出。

```
{  
    "principalId": "yyyyyyyy", // The principal user identification associated with the token sent by the client.  
    "policyDocument": {  
        "Version": "2012-10-17",  
        "Statement": [  
            {  
                "Action": "execute-api:Invoke",  
                "Effect": "Allow|Deny",  
                "Resource": "arn:aws:execute-api:{regionId}:{accountId}:{appId}/{stage}/{httpVerb}/[{resource}]/[{child-resources}]"  
            }  
        ]  
    },  
    "context": {  
        "stringKey": "value",  
        "numberKey": "1",  
        "booleanKey": "true"  
    },  
    "usageIdentifierKey": "{api-key}"  
}
```

在本例中，政策陳述式會指定要允許或拒絕 (Effect) API Gateway 執行服務呼叫 (Action) 指定的 API 方法 (Resource)。您可以使用萬用字元 (\*) 來指定資源類型 (方法)。如需設定有效政策來呼叫 API 的資訊，請參閱「[在 API Gateway 中執行 API 之 IAM 政策的陳述式參考 \(p. 341\)](#)」。

針對啟用授權的方法 ARN (例如 arn:aws:execute-api:{regionId}:{accountId}:{appId}/{stage}/{httpVerb}/[{resource}]/[{child-resources}])，長度上限為 1600 個位元組。路徑參數值是在執行時間所決定的大小，可能會導致 ARN 長度超過限制。發生此情況時，API 用戶端會收到 414 Request URI too long 回應。

此外，授權方之政策陳述式輸出中所示的資源 ARN 長度目前限制為 512 個字元。因此，您不得在請求 URI 中，使用 JWT 字符長度很長的 URI。反之，您可以在請求標頭中安全傳遞此 JWT 字符。

您可以使用 principalId 變數，來存取對應範本中的 \$context.authorizer.principalId 值。如果您想要將此值傳遞到後端，這會很有用。如需更多詳細資訊，請參閱 [適用於資料模型、授權方、映射範本及 CloudWatch 存取記錄的 \\$context 變數 \(p. 274\)](#)。

您可以分別呼叫 stringKey、numberKey 或 booleanKey，來存取對應範本中 "value" 對應的 "1"、"true" 或 context 值 (例如 \$context.authorizer.stringKey、\$context.authorizer.numberKey 或 \$context.authorizer.booleanKey)。傳回的值全部都已字串化。請注意，您無法將 JSON 物件或陣列設定為 context 對應中任何金鑰的有效值。

您可以透過整合請求對應範本，使用 context 對應將快取的登入資料從授權方傳回後端。這可讓後端使用快取的登入資料來降低存取秘密金鑰的需求，並開啟每個請求的授權字符，藉此提供改善的使用者體驗。

針對 Lambda 代理整合，API Gateway 會將 context 物件從 Lambda 授權方直接傳遞到後端 Lambda 函數，以做為輸入 event 的一部分。您可以呼叫 \$event.requestContext.authorizer.key，來擷取 Lambda 函數中的 context 鍵值對。

API 階段用量計劃中，{api-key} 代表 API 金鑰。如需詳細資訊，請參閱 [the section called “使用 API 金鑰的用量計劃” \(p. 397\)](#)。

以下顯示範例 Lambda 授權方的範例輸出。此範例輸出所包含的政策陳述式，可封鎖 (Deny) 任何階段 (\*) 的 AWS 帳戶 (123456789012) 之 API (ymy8tbxw7b) 中的 GET 方法呼叫。

```
{  
    "principalId": "user",
```

```
"policyDocument": {  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Action": "execute-api:Invoke",  
            "Effect": "Deny",  
            "Resource": "arn:aws:execute-api:us-west-2:123456789012:ymy8tbxw7b/*/GET/"  
        }  
    ]  
}
```

## 呼叫具有 API Gateway Lambda 授權方的 API

設定 Lambda 授權方 (先前稱作自訂授權方) 並部署 API 之後，您應該測試啟用 Lambda 授權方的 API。為此，您需要一個 REST 用戶端，例如 cURL 或 Postman。在下列範例中，我們使用 Postman。

### Note

呼叫啟用授權方的方法時，如果 TOKEN 授權方的必要符記未設定、為 Null 或因指定的 Token validation expression (符記驗證表達式) 而失效，則 API Gateway 不會將呼叫記錄到 CloudWatch。同樣地，如果 REQUEST 授權方的任何必要身分來源未設定、為 Null 或空白，API Gateway 不會將呼叫記錄到 CloudWatch。

以下示範如何使用 Postman 來呼叫或測試上述已啟用 Lambda TOKEN 授權方的 API。如果您明確指定必要的路徑、標頭或查詢字串參數，則可以套用此方法呼叫具有 Lambda REQUEST 授權方的 API。

### 呼叫具有自訂 TOKEN 授權方的 API

1. 開啟 Postman，選擇 GET 方法，然後將 API 的 Invoke URL (呼叫 URL) 貼到相鄰的 URL 欄位中。

新增 Lambda 授權字符標頭，並將值設定為 allow。選擇 Send (傳送)。

The screenshot shows the Postman interface with the following details:

- URL:** https://<api-id>.execute-api.<region>.amazonaws.com/test
- Method:** GET
- Authorization:** A header named "Auth" is set to "allow".
- Status:** 200 OK (highlighted with an orange circle)
- Body:** The response body is displayed in JSON format, showing the following structure:

```
1  [  
2      "args": {},  
3      "headers": {  
4          "Accept": "application/json",  
5          "Host": "httpbin.org",  
6          "User-Agent": "AmazonAPIGateway_y_ _ _ _ _",  
7          "X-Amzn-Apigateway-Api-Id": " _ _ _ _ "  
8      },  
9      "origin": "54.186.57.107",  
10     "url": "http://httpbin.org/get"  
11 }
```

回應顯示 API Gateway Lambda 授權方傳回 200 OK 回應，並成功授權呼叫存取與方法整合的 HTTP 端點 (<http://httpbin.org/get>)。

2. 同樣在 Postman 中，將 Lambda 授權字符標頭值變更為 deny。選擇 Send (傳送)。

The screenshot shows the Postman interface with the following details:

- Method: GET
- URL: `https://<api-id>.execute-api.<region>.amazonaws.com/test`
- Headers tab selected.
- Auth section: Value is set to `deny`.
- Status: 403 Forbidden
- Body tab selected. Response body:

```
1 {  
2   "Message": "User is not authorized to access this resource"  
3 }
```

回應顯示 API Gateway Lambda 授權方傳回 403 Forbidden (403 禁止) 回應，而不授權呼叫存取 HTTP 端點。

3. 在 Postman 中，將 Lambda 授權字符標頭值變更為 unauthorized，然後選擇 Send (傳送)。

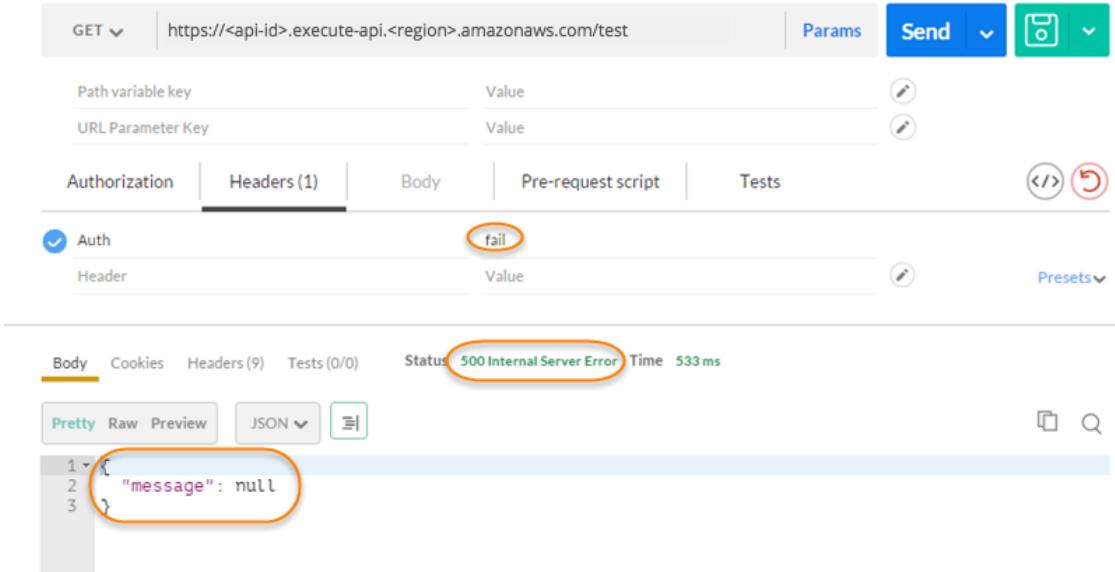
The screenshot shows the Postman interface with the following details:

- Method: GET
- URL: `https://<api-id>.execute-api.<region>.amazonaws.com/test`
- Headers tab selected.
- Auth section: Value is set to `unauthorized`.
- Status: 401 Unauthorized
- Body tab selected. Response body:

```
1 {  
2   "message": "Unauthorized"  
3 }
```

回應顯示 API Gateway 傳回 401 Unauthorized (401 未經授權) 回應，而不授權呼叫存取 HTTP 端點。

4. 現在，將 Lambda 授權字符標頭值變更為 fail。選擇 Send (傳送)。



回應顯示 API Gateway 傳回 500 Internal Server Error (500 內部伺服器錯誤) 回應，而不授權呼叫存取 HTTP 端點。

## 設定跨帳戶 Lambda 授權方

您現在也可以從不同的 AWS 帳戶使用 AWS Lambda 函數做為 API 授權方函數。每個帳戶可以位於 Amazon API Gateway 可用的任何區域。Lambda 授權方函數可以使用承載字元身份驗證策略，例如 OAuth 或 SAML。這可讓您輕鬆地集中管理和分享跨多個 API Gateway API 的中央 Lambda 授權方函數。

在本節中，我們示範如何使用 Amazon API Gateway 主控台設定跨帳戶 Lambda 授權方函數。

這些指示假設您在一個 AWS 帳戶已經有 API Gateway API，並在另一個帳戶中擁有 Lambda 授權方函數。

### 使用 API Gateway 主控台設定跨帳戶 Lambda 授權方

登入在您第一個帳戶中的 Amazon API Gateway 主控台 (即其中具有 API 的主控台)，並執行下列動作：

1. 找出您的 API，然後選擇 Authorizers (授權方)。
2. 選擇 Create New Authorizer (建立新的授權方)。
3. 針對 Create Authorizer (建立授權方)，在 Name (名稱) 輸入欄位中輸入授權方名稱。
4. 針對 Type (類型)，選擇 Lambda 選項。
5. 對於 Lambda Function (Lambda 函數)，複製/貼上您在第二個帳戶中擁有之 Lambda 授權方函數的完整 ARN。

#### Note

在 Lambda 主控台中，您可以主控台視窗的右上角找到適用於您函數的 ARN。

6. 將 Lambda Invoke Role (&LAM; 呼叫角色) 保留空白，讓 API Gateway 主控台設定資源類型政策。該政策會授予 API Gateway 許可來呼叫授權方 Lambda 函數。您也可以選擇輸入 IAM 角色的名稱，來允許 API Gateway 呼叫授權方 Lambda 函數。如需這類角色的範例，請參閱「[建立可擔任的 IAM 角色 \(p. 87\)](#)」。

如果您選擇讓 API Gateway 主控台設定資源類型政策，則會顯示 Add Permission to Lambda Function (新增 Lambda 函數的許可) 對話方塊。選擇 OK (確定)。建立 Lambda 授權之後，您可以使用適當的授權字符串進行測試，以驗證其是否如預期般運作。

7. 對於 Lambda Event Payload (&LAM; 事件承載) , 選擇 Token (字符) 代表 TOKEN 授權方 , 或 Request (請求) 代表 REQUEST 授權方。
8. 根據上一個步驟所做的選擇 , 執行下列其中一項操作 :
  - a. 對於 Token (字符) 選項 , 執行下列操作 :
    - i. 在 Token Source (字符來源) 中 , 輸入標頭的名稱。API 用戶端必須包含此名稱的標頭 , 才能將授權字符傳送到 Lambda 授權方。
    - ii. 或者 , 在 Token Validation (字元驗證) 輸入欄位中提供 RegEx 陳述式。API Gateway 會對此運算式執行輸入字元的初始驗證 , 並在成功驗證時呼叫授權方。這有助於降低支付無效字符費用的機率。
    - iii. 對於 Authorization Caching (授權快取) , 根據您是否想要快取授權方所產生的授權政策 , 選取或清除 Enabled (已啟用) 選項。啟用政策快取時 , 您可以選擇修改 TTL 的預設值 (300)。設定 TTL=0 會停用政策快取。啟用政策快取時 , Token Source (字符來源) 中指定的標頭名稱會成為快取金鑰。
  - b. 對於 Request (請求) 選項 , 執行下列操作 :
    - i. 對於 Identity Sources (身分來源) , 輸入所選參數類型的請求參數名稱。支援的參數類型為 Header、Query String、Stage Variable 與 Context。若要新增更多身分來源 , 請選擇 Add Identity Source (新增身分來源)。
9. 選擇 Create (建立) , 建立所選 API 的新 Lambda 授權方。
10. 您會看到一個快顯視窗 , 告知您將許可新增至 Lambda 函數 : 您已從另一個帳戶選取 Lambda 函數。請確保您對此函數有適當的函數政策。您可以從帳戶 **123456789012** : 接著一個 aws lambda add-permission 命令字串來執行以下 AWS CLI 命令來進行確認。
11. 將 aws lambda add-permission 命令字串複製貼上至為第二個帳戶所設定的 AWS CLI 視窗。這會授與您的第一個帳戶對第二個帳戶的 Lambda 授權方函數的存取權。
12. 在先前的步驟中的快顯視窗中 , 選擇 OK (確定)。

## 使用 Amazon Cognito User Pools 做為授權方來控制 REST API 的存取

除了使用 [IAM 角色和政策 \(p. 333\)](#) 或 [Lambda 授權方 \(p. 348\)](#) (先前稱為自訂授權方) 之外 , 您還可以使用 [Amazon Cognito 使用者集區](#) 來控制可在 Amazon API Gateway 中存取您的 API 的人。

若要搭配使用 Amazon Cognito 使用者集區與您的 API , 您必須先建立 COGNITO\_USER\_POOLS 類型的授權方 , 然後設定 API 方法使用該授權方。API 部署後 , 用戶端必須先將使用者登入使用者集區 , 取得該使用者的 [身分或存取字符](#) , 然後透過通常會設定為請求 Authorization 標頭的字符其中之一來呼叫 API 方法。API 呼叫只有在您提供有效的字符時才會成功 ; 否則 , 用戶端無權發出呼叫 , 因為用戶端沒有可獲得授權的登入資料。

身分字符是根據登入使用者宣告的身分 , 用來授權 API 呼叫。存取字符是根據指定的受存取保護資源的自訂範圍 , 用來授權 API 呼叫。如需詳細資訊 , 請參閱 [將字符用於使用者集區和資源伺服器和自訂範圍](#)。

若要為您的 API 建立和設定 Amazon Cognito 使用者集區 , 您要執行下列任務 :

- 使用 Amazon Cognito 主控台、CLI/開發套件或 API 來建立使用者集區—或使用其他 AWS 帳戶擁有的使用者集區。
- 使用 API Gateway 主控台 (CLI/開發套件) 或 API 在選擇的使用者集區中建立 API Gateway 授權方。
- 使用 API Gateway 主控台 (CLI/開發套件) 或 API 在選取的 API 方法中啟用授權方。

若要在使用者集區啟用的狀況下呼叫任何 API 方法，您的 API 用戶端要執行下列任務：

- 使用 Amazon Cognito CLI/開發套件或 API 將使用者登入所選使用者集區，並取得身分字符串或存取字符串。
- 使用用戶端特定架構呼叫已部署的 API Gateway API，並在 Authorization 標頭中提供適當的字符串。

身為 API 開發人員，您必須為您的用戶端開發人員提供使用者集區 ID、用戶端 ID，以及定義為使用者集區一部分之相關聯的用戶端密碼 (如可能)。

#### Note

為了讓使用者使用 Amazon Cognito 登入資料登入，同時取得具備 IAM 角色許可的暫時性登入資料，請使用 [Amazon Cognito 聯合身分](#)。對於每個 API 資源端點 HTTP 方法，將授權類型 Method Execution 設定為 AWS\_IAM。

我們會在本節中說明如何建立使用者集區、如何整合 API Gateway API 與使用者集區，以及如何呼叫與使用者集區整合的 API。

#### 主題

- [取得為 REST API 建立Amazon Cognito 使用者集區授權方的許可 \(p. 364\)](#)
- [為 REST API 建立 Amazon Cognito 使用者集區組態 \(p. 365\)](#)
- [整合 REST API 與 Amazon Cognito 使用者集區 \(p. 366\)](#)
- [呼叫與 Amazon Cognito 使用者集區整合的 REST API \(p. 370\)](#)
- [使用 API Gateway 主控台為 REST API 設定跨帳戶 Amazon Cognito 授權方 \(p. 370\)](#)

## 取得為 REST API 建立Amazon Cognito 使用者集區授權方的許可

若要在 Amazon Cognito 使用者集區中建立授權方，您必須有能在所選 Amazon Cognito 使用者集區中建立或更新授權方的 Allow 許可。下列 IAM 政策文件會顯示此類許可的範例：

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "apigateway:POST"  
            ],  
            "Resource": "arn:aws:apigateway:*:::/restapis/*/authorizers",  
            "Condition": {  
                "ArnLike": {  
                    "apigateway:CognitoUserPoolProviderArn": [  
                        "arn:aws:cognito-idp:us-east-1:123456789012:userpool/us-  
east-1_aD06NQmjO",  
                        "arn:aws:cognito-idp:us-east-1:234567890123:userpool/us-  
east-1_xJ1MQtPEN"  
                    ]  
                }  
            }  
        },  
        {  
    ]
```

```
"Effect": "Allow",
"Action": [
    "apigateway:PATCH"
],
"Resource": "arn:aws:apigateway:*:::restapis/*/authorizers/*",
"Condition": {
    "ArnLike": [
        "apigateway:CognitoUserPoolProviderArn": [
            "arn:aws:cognito-idp:us-east-1:123456789012:userpool/us-
east-1_aD06NQmjo",
            "arn:aws:cognito-idp:us-east-1:234567890123:userpool/us-
east-1_xJ1MQtPEN"
        ]
    }
}
}
```

確定政策連接到您的 IAM 使用者、您所屬的 IAM 群組，或您被指派的 IAM 角色。

在前一份政策文件中，`apigateway:POST` 動作是用於建立新的授權方，而 `apigateway:PATCH` 動作是用於更新現有的授權方。您可以分別覆寫 `Resource` 值的前兩個萬用字元 (\*)，將政策限制在特定區域或特定的 API。

`Condition` 子句用在這裡以限制指定使用者集區的 `Allowed` 許可。當有 `Condition` 子句時，拒絕任何不符合條件的存取使用者集區。當許可沒有 `Condition` 子句時，允許存取任何使用者集區。

設定 `Condition` 子句有下列選項：

- 您可以設定 `ArnLike` 或 `ArnEquals` 條件式表達式，只允許在指定的使用者集區中建立或更新 `COGNITO_USER_POOLS` 授權方。
- 您可以設定 `ArnNotLike` 或 `ArnNotEquals` 條件式表達式，允許在表達式中未指定的任何使用者集區中建立或更新 `COGNITO_USER_POOLS` 授權方。
- 您可以省略 `Condition` 子句，允許在任何區域之任何 AWS 帳戶的任何使用者集區中建立或更新 `COGNITO_USER_POOLS` 授權方。

如需 Amazon Resource Name (ARN) 條件式表達式的詳細資訊，請參閱 [Amazon Resource Name 條件運算子](#)。如範例中所示，`apigateway:CognitoUserPoolProviderArn` 是 `COGNITO_USER_POOLS` 使用者集區的 ARN 清單，可以或無法和 `COGNITO_USER_POOLS` 類型的 API Gateway 授權方一起使用。

## 為 REST API 建立 Amazon Cognito 使用者集區組態

整合您的 API 和使用者集區之前，您必須先在 Amazon Cognito 中建立使用者集區。如需如何建立使用者集區的相關說明，請參閱 Amazon Cognito 開發人員指南中的[設定使用者集區](#)。

### Note

請記下使用者集區 ID、用戶端 ID 和任何用戶端密碼。用戶端必須向 Amazon Cognito 提供它們以讓使用者註冊使用者集區、登入使用者集區，以及取得要包含在請求中的身分或存取字元，來呼叫以使用者集區設定的 API 方法。此外，您必須在 API Gateway 中將使用者集區設定為授權方時，指定使用者集區名稱，如下所述。

如果您使用存取字符授權 API 方法呼叫，請務必設定應用程式與使用者集區整合，以設定您想要的指定資源伺服器自訂範圍。如需詳細資訊，請參閱[為您的使用者集區定義資源伺服器](#)。

請記下已設定的資源伺服器識別符和自訂範圍名稱。您需要它們來建構 OAuth Scopes (OAuth 範圍) 的存取範圍完整名稱，這是由 `COGNITO_USER_POOLS` 授權方使用。

The screenshot shows the AWS API Gateway console. On the left, a sidebar lists various settings like General settings, Users and groups, Attributes, Policies, etc., with 'Resource servers' highlighted. The main area displays a modal window for defining a resource server named 'hamuta movies'. The modal includes fields for 'Identifier' (set to 'com.haymuta.movies') and 'Scopes'. Under 'Scopes', there are two entries: 'drama.view' and 'comedy.view'. The 'Identifier' field and the first scope entry ('drama.view') are both circled in red.

## 整合 REST API 與 Amazon Cognito 使用者集區

在 API Gateway 中建立 Amazon Cognito 使用者集區後，您必須接著建立使用使用者集區的 COGNITO\_USER\_POOLS 授權方。下列程序會逐步引導您使用 API Gateway 主控台執行此項操作的步驟。

### Important

執行下列任何程序之後，您需要部署或重新部署您的 API 以傳播變更。如需部署 API 的詳細資訊，請參閱 [在 Amazon API Gateway 中部署 REST API \(p. 457\)](#)。

### 使用 API Gateway 主控台建立 COGNITO\_USER\_POOLS 授權方

1. 在 API Gateway 中建立新的 API 或選取現有的 API。
2. 從主導覽窗格中，選擇指定 API 下的 Authorizers (授權方)。
3. 在 Authorizers (授權方) 下，選擇 Create New Authorizer (建立新的授權方)。
4. 若要設定新的授權方使用使用者集區，請執行下列操作：
  - a. 在 Name (名稱) 中輸入授權方名稱。
  - b. 選取 Cognito 選項。
  - c. 選擇 Cognito User Pool (Cognito 使用者集區) 下的區域。
  - d. 選取可用的使用者集區。您必須已在所選 Amazon Cognito 區域中建立使用者集區，它才會出現在下拉式清單中。
  - e. 針對 Token source (字符來源)，輸入 Authorization 做為標頭名稱，以傳送 Amazon Cognito 於使用者成功登入時所傳回的身分或存取字符。
  - f. (選擇性) 在 Token validation (字符驗證) 欄位中輸入規則表達式，先驗證 aud 欄位的身分字符，再使用 Amazon Cognito 授權請求。
  - g. 若要完成使用者集區與 API 的整合，請選擇 Create (建立)。
5. 建立 COGNITO\_USER\_POOLS 授權方之後，您可以提供從使用者集區佈建的身分字符，選擇性地對它進行呼叫測試。您可以呼叫 [Amazon Cognito 身分開發套件](#) 來取得此身分字符，藉此執行使用者登入。請務必使用傳回的身分字符，不要使用存取字符。

上述程序會建立使用新建立之 Amazon Cognito 使用者集區的 COGNITO\_USER\_POOLS 授權方。視您在 API 方法中啟用授權方的方式而定，您可以使用從已整合使用者集區佈建的身分字符或存取字符。下一個程序會逐步帶領您完成在 API 方法中設定授權方的各項步驟。

#### 在方法中設定 COGNITO\_USER\_POOLS 授權方

1. 選擇 (或建立) 您的 API 方法。
2. 選擇 Method Request (方法請求)。
3. 在 Settings (設定) 下，選擇 Authorization (授權) 旁的鉛筆圖示。
4. 從下拉式清單中選擇一個可用的 Amazon Cognito user pool authorizers (&COG; 使用者集區授權方)。
5. 選擇核取記號圖示以儲存設定。
6. 若要使用身分字符，請執行下列操作：
  - a. 保持 OAuth Scopes (OAuth 範圍) 選項不予指定 (NONE)。
  - b. 如有需要，請選擇 Integration Request (整合請求) 在內文對應範本中新增 \$context.authorizer.claims['*property-name*'] 或 \$context.authorizer.claims.*property-name* 表達式，將指定的身分宣告屬性從使用者集區傳送到後端。對於簡單的屬性名稱，例如 sub 或 custom-sub，兩個表示法完全相同。至於複雜的屬性名稱，例如 custom:role，則無法使用點表示法。例如，下列映射表達式會將宣告的 **標準欄位** sub 和 email 傳送到後端：

```
{  
  "context" : {  
    "sub" : "$context.authorizer.claims.sub",  
    "email" : "$context.authorizer.claims.email"  
  }  
}
```

如已在設定使用者集區時宣告自訂的宣告欄位，您可以遵循相同的模式來存取自訂欄位。下列範例會取得自訂的宣告 role 欄位：

```
{  
  "context" : {  
    "role" : "$context.authorizer.claims.role"  
  }  
}
```

如果自訂宣告欄位宣告為 custom:role，請使用下列範例來取得宣告的屬性：

```
{  
  "context" : {  
    "role" : "$context.authorizer.claims['custom:role']"  
  }  
}
```

7. 若要使用存取字符，請執行下列操作：

- a. 選擇 OAuth Scopes (OAuth 範圍) 旁的鉛筆圖示。
- b. 輸入已在 Amazon Cognito 使用者集區建立時已設定之範圍的一個或多個完整名稱。例如，在 [為 REST API 建立 Amazon Cognito 使用者集區組態 \(p. 365\)](#) 提供的範例之後，其中一個範圍是 com.hamuta.movies/drama.view。多範圍使用單一空格加以分隔。

在執行時間，如果在這個步驟中，方法內指定的任何範圍符合傳入字符宣告的範圍，則方法呼叫就會成功。否則，呼叫失敗並傳回 401 Unauthorized 回應。

- c. 選擇核取記號圖示來儲存設定。
8. 為您選擇的其他方法重複這些步驟。

使用 COGNITO\_USER\_POOLS 授權方，如果不指定 OAuth Scopes (OAuth 範圍) 選項，API Gateway 會將提供的字符視為身分字符，並使用使用者集區中的身分來驗證宣告的身分。否則，API Gateway 會將提供的字符視為存取字符，並根據方法中宣告的授權範圍來驗證字符中宣告的存取範圍。

除了使用 API Gateway 主控台之外，您也可以指定 OpenAPI 定義檔並將 API 定義匯入 API Gateway，在方法中啟用 Amazon Cognito 使用者集區。

#### 使用 OpenAPI 定義檔匯入 COGNITO\_USER\_POOLS 授權方

1. 為您的 API 建立 (或匯出) OpenAPI 定義檔。
2. 指定 COGNITO\_USER\_POOLS 授權方 (MyUserPool) JSON 定義，做為 OpenAPI 3.0 的 securitySchemes 部分或 Open API 2.0 的 securityDefinitions 部分，如下所示：

##### OpenAPI 3.0

```
"securitySchemes": {  
    "MyUserPool": {  
        "type": "apiKey",  
        "name": "Authorization",  
        "in": "header",  
        "x-amazon-apigateway-authtype": "cognito_user_pools",  
        "x-amazon-apigateway-authorizer": {  
            "type": "cognito_user_pools",  
            "providerARNs": [  
                "arn:aws:cognito-idp:{region}:{account_id}:userpool/{user_pool_id}"  
            ]  
        }  
    }  
}
```

##### OpenAPI 2.0

```
"securityDefinitions": {  
    "MyUserPool": {  
        "type": "apiKey",  
        "name": "Authorization",  
        "in": "header",  
        "x-amazon-apigateway-authtype": "cognito_user_pools",  
        "x-amazon-apigateway-authorizer": {  
            "type": "cognito_user_pools",  
            "providerARNs": [  
                "arn:aws:cognito-idp:{region}:{account_id}:userpool/{user_pool_id}"  
            ]  
        }  
    }  
}
```

3. 若要使用身分字符授權方法，請將 { "MyUserPool": [] } 新增到方法的 security 定義，如下列根資源中的 GET 方法所示。

```
"paths": {  
    "/": {  
        "get": {  
            "consumes": [  
                "application/json"  
            ],  
            "produces": [  
                "text/html"  
            ],  
            "responses": {  
                "200": {  
                    "description": "200 response",  
                    "headers": {  
                        "Content-Type": {  
                            "type": "string"  
                        }  
                    }  
                }  
            }  
        }  
    }  
}
```

```
        "type": "string"
    }
}
},
"security": [
{
"MyUserPool": []
}
],
"x-amazon-apigateway-integration": {
"type": "mock",
"responses": {
"default": {
"statusCode": "200",
"responseParameters": {
"method.response.header.Content-Type": "'text/html'"
},
},
"requestTemplates": {
"application/json": "{\"statusCode\": 200}"
},
"passthroughBehavior": "when_no_match"
}
},
...
}
```

4. 若要使用存取字符授權方法，請將上述安全性定義變更為 { "MyUserPool": [resource-server/scoped, ...] } :

```
"paths": {
"/": {
"get": {
"consumes": [
"application/json"
],
"produces": [
"text/html"
],
"responses": {
"200": {
"description": "200 response",
"headers": {
"Content-Type": {
"type": "string"
}
}
}
},
"security": [
{
"MyUserPool": ["com.hamuta.movies/drama.view", "http://my.resource.com/file.read"]
}
]
},
"x-amazon-apigateway-integration": {
"type": "mock",
"responses": {
"default": {
"statusCode": "200",
"responseParameters": {
"method.response.header.Content-Type": "'text/html'"
}
},
}
```

```
        },
        "requestTemplates": {
            "application/json": "{\"statusCode\": 200}"
        },
        "passthroughBehavior": "when_no_match"
    },
    ...
}
```

5. 如有需要，您可以使用適當的 OpenAPI 定義或延伸，設定其他的 API 組態設定。如需詳細資訊，請參閱[OpenAPI 的 API Gateway 延伸 \(p. 533\)](#)。

## 呼叫與 Amazon Cognito 使用者集區整合的 REST API

若要呼叫已設定使用者集區授權方的方法，用戶端必須執行下列操作：

- 讓使用者以使用者集區註冊。
- 讓使用者登入使用者集區。
- 從使用者集區取得已登入使用者的身分字符。
- 在 `Authorization` 標頭 (或您在建立授權方時指定的另一個標頭) 中包含身分字符。

您可以使用其中一種 AWS 開發套件來執行這些任務。例如：

- 若要使用 Android 開發套件，請參閱[設定適用於 Android 的 AWS Mobile SDK 以搭配使用者集區](#)。
- 若要使用 iOS 開發套件，請參閱[設定適用於 iOS 的 AWS Mobile SDK 以搭配使用者集區](#)。
- 若要使用 JavaScript，請參閱[在瀏覽器中設定適用於 JavaScript 的 AWS 開發套件以搭配使用者集區](#)。

下列程序概述執行這些任務的步驟。如需詳細資訊，請參閱[使用 Android 開發套件和 Amazon Cognito 使用者集區](#)和[使用適用於 iOS 的 Amazon Cognito 使用者集區](#)等部落格文章。

### 呼叫與使用者集區整合的 API

1. 將第一次使用的使用者註冊到指定的使用者集區。
2. 將使用者登入使用者集區。
3. 取得使用者的身分字符。
4. 呼叫已設定使用者集區授權方的 API 方法，並在 `Authorization` 標頭或您選擇的另一個標頭中提供未過期的字符。
5. 當字符到期時，請重複步驟 2–4。Amazon Cognito 佈建的身分字符會在一小時內到期。

如需程式碼範例，請參閱[Android Java 範例](#)和[iOS Objective-C 範例](#)。

## 使用 API Gateway 主控台為 REST API 設定跨帳戶 Amazon Cognito 授權方

您現在也可以從不同的 AWS 帳戶使用 Amazon Cognito 使用者集區做為 API 授權方函數。每個帳戶可以位於 Amazon API Gateway 可用的任何區域。Amazon Cognito 使用者集區可以使用承載字符身份驗證策略，例如 OAuth 或 SAML。這可讓您輕鬆地集中管理和分享跨多個 API Gateway API 的中央 Amazon Cognito 使用者集區授權方。

在本節中，我們示範如何使用 Amazon API Gateway 主控台設定跨帳戶 Amazon Cognito 使用者集區。

這些指示假設您在一個 AWS 帳戶已經有 API Gateway API，並在另一個帳戶中擁有 Amazon Cognito 使用者集區。

## 使用 API Gateway 主控台設定跨帳戶 Amazon Cognito 授權方

登入在您第一個帳戶中的 Amazon API Gateway 主控台 (即其中具有 API 的主控台)，並執行下列動作：

1. 找出您的 API，然後選擇 Authorizers (授權方)。
2. 選擇 Create New Authorizer (建立新的授權方)。
3. 針對 Create Authorizer (建立授權方)，在 Name (名稱) 輸入欄位中輸入授權方名稱。
4. 針對 Type (類型)，選擇 Cognito 選項。
5. 如需 Cognito 使用者集區，複製/貼上您在第二個帳戶中擁有之使用者集區的完整 ARN。

### Note

在 Amazon Cognito 主控台中，您可以一般設定窗格的 Pool ARN 欄位找到適用於您使用者集區的 ARN。

6. 在 Token Source (字符來源) 中，輸入標頭的名稱。API 用戶端必須包含此名稱的標頭，才能將授權字符串傳送到 Amazon Cognito 授權方。
7. 或者，在 Token Validation (字元驗證) 輸入欄位中提供 RegEx 陳述式。API Gateway 會對此運算式執行輸入字元的初始驗證，並在成功驗證時呼叫授權方。這有助於降低支付無效字符費用的機率。
8. 選擇 Create (建立)，建立您 API 的新 Amazon Cognito 授權方。

## 為 API Gateway REST API 資源啟用 CORS

跨來源資源共享 (CORS) 是一種瀏覽器安全功能，限制從瀏覽器中執行之指令碼啟動的跨來源 HTTP 請求。如果您的 REST API 的資源接收非簡單的跨來源 HTTP 請求，則您需要啟用 CORS 支援。

### 主題

- [決定是否啟用 CORS 支援 \(p. 371\)](#)
- [啟用 CORS 支援所代表的意義 \(p. 372\)](#)
- [測試 CORS \(p. 373\)](#)
- [使用 API Gateway 主控台在資源上啟用 CORS \(p. 374\)](#)
- [使用 API Gateway 匯入 API 在資源上啟用 CORS \(p. 376\)](#)

## 決定是否啟用 CORS 支援

跨來源 HTTP 請求是針對下列項目所提出的請求：

- 不同的網域 (例如，從 example.com 到 amazondomains.com)
- 不同的子網域 (例如，從 example.com 到 petstore.example.com)
- 不同的連接埠 (例如，從 example.com 到 example.com:10777)
- 不同的通訊協定 (例如，從 https://example.com 到 http://example.com)

跨來源 HTTP 請求可分為兩種類型：簡單請求和非簡單請求。

如果下列所有條件皆為真，則 HTTP 請求為簡單請求：

- 它是針對只允許 GET、HEAD 和 POST 請求的 API 資源所發出的。
- 如果它是 POST 方法請求，則必須包含 Origin 標頭。

- 請求承載內容類型為 `text/plain`、`multipart/form-data` 或 `application/x-www-form-urlencoded`。
- 請求不包含自訂標頭。
- [Mozilla CORS 文件中針對簡單請求](#)列出的任何其他需求。

對於簡單的跨來源 POST 方法請求，來自您資源的回應需要包括標頭 `Access-Control-Allow-Origin`，其中標頭金鑰的值會設為 '\*' (任何來源)，或設定為允許存取該資源的來源。

所有其他跨來源 HTTP 請求都是非簡單請求。如果您的 API 資源接收非簡單請求，您將需要啟用 CORS 支援。

## 啟用 CORS 支援所代表的意義

當瀏覽器接收非簡單 HTTP 請求時，[CORS 通訊協定](#)需要瀏覽器將預檢請求傳送到伺服器，並等待伺服器的核准 (或請求登入資料)，然後再傳送實際請求。預檢請求會出現在您的 API 請求上，做為 HTTP 請求，其中：

- 包含 `Origin` 標頭。
- 使用 `OPTIONS` 方法。
- 包含下列標題：
  - `Access-Control-Request-Method`
  - `Access-Control-Request-Headers`

因此，若要支援 CORS，REST API 資源需要實作 `OPTIONS` 方法，其可以至少使用擷取標準所要求的下列回應標頭來回應 `OPTIONS` 預檢請求：

- `Access-Control-Allow-Methods`
- `Access-Control-Allow-Headers`
- `Access-Control-Allow-Origin`

您啟用 CORS 支援的方式取決於 API 的整合類型。

## 針對模擬整合啟用 CORS 支援

若為模擬整合，您可以啟用 CORS，方法為建立 `OPTIONS` 方法，傳回所需的回應標頭 (具有適當的靜態值) 做為方法回應標頭。此外，每個實際啟用 CORS 的方法也必須在至少 200 個回應中傳回 `Access-Control-Allow-Origin:'request-originating server addresses'` 標頭，其中標頭金鑰的值設定為 '\*' (任何來源)，或設定為允許存取資源的來源。

## 針對 Lambda 或 HTTP 非代理整合和 AWS 服務整合啟用 CORS 支援

若為 Lambda 自訂 (非代理) 整合、HTTP 自訂 (非代理) 整合，或 AWS 服務整合，您可以設定所需的標頭，方法為使用 API Gateway 方法回應和整合回應設定。API Gateway 會建立 `OPTIONS` 方法，並嘗試將 `Access-Control-Allow-Origin` 標頭新增到現有的方法整合回應。這不一定可以運作，有時您需要手動修改整合回應，才能適當地啟用 CORS。通常，這僅表示手動修改整合回應，以傳回 `Access-Control-Allow-Origin` 標頭。

## 針對 Lambda 或 HTTP 代理整合啟用 CORS 支援

若為 Lambda 代理整合或 HTTP 代理整合，您仍然可以在 API Gateway 中設定必要的 `OPTIONS` 回應標頭。不過，您的後端負責傳回 `Access-Control-Allow-Origin` 和 `Access-Control-Allow-Headers` 標頭，因為代理整合不會傳回整合回應。

以下是設定為傳回所需 CORS 標頭的 Node.js Lambda 函數範例：

```
'use strict';

exports.handler = function(event, context) {

    var responseCode = 200;

    var response = {
        statusCode: responseCode,
        headers: {
            "x-custom-header" : "my custom header value",
            "Access-Control-Allow-Origin": "my-origin.com"
        },
        body: JSON.stringify(event)
    };

    context.succeed(response);
};
```

您可以在 [https://github.com/awslabs/serverless-application-model/blob/master/examples/2016-10-31/api\\_swagger\\_cors/index.js](https://github.com/awslabs/serverless-application-model/blob/master/examples/2016-10-31/api_swagger_cors/index.js) 找到更完整的 Node.js 範例。

以下是傳回所需 CORS 標頭的 Python 程式碼片段範例：

```
response[ "headers" ] = {
    'Content-Type': 'application/json',
    'Access-Control-Allow-Origin': '*'
}
```

以下是使用無伺服器應用程式模型 CORS (SAM) , 為 CORS 傳回必要標頭的範例，包括 AllowHeaders：

```
Globals:
  Api:
    # Allows an application running locally on port 8080 to call this API
  Cors:
    AllowMethods: "'OPTIONS,POST,GET'"
    AllowHeaders: "'Content-Type'"
    AllowOrigin: "'http://localhost:8080'"
```

以下是傳回與 SAM 範例相同之標頭的 Lambda 代理範例：

```
return {
    'statusCode': 200,
    'headers': {
        "Access-Control-Allow-Origin": "http://localhost:8080",
        "Access-Control-Allow-Headers": "Content-Type",
        "Access-Control-Allow-Methods": "OPTIONS,POST,GET"
    },
    'body': json.dumps(response)
}
```

## 測試 CORS

對於實際方法和您正在使用的來源標頭，您可以自訂以下 cURL 命令來測試 CORS：

```
curl -v -X OPTIONS -H "Access-Control-Request-Method: POST" -H "Origin: http://example.com"
https://{{restapi_id}}.execute-api.{{region}}.amazonaws.com/{{stage_name}}
```

## 使用 API Gateway 主控台在資源上啟用 CORS

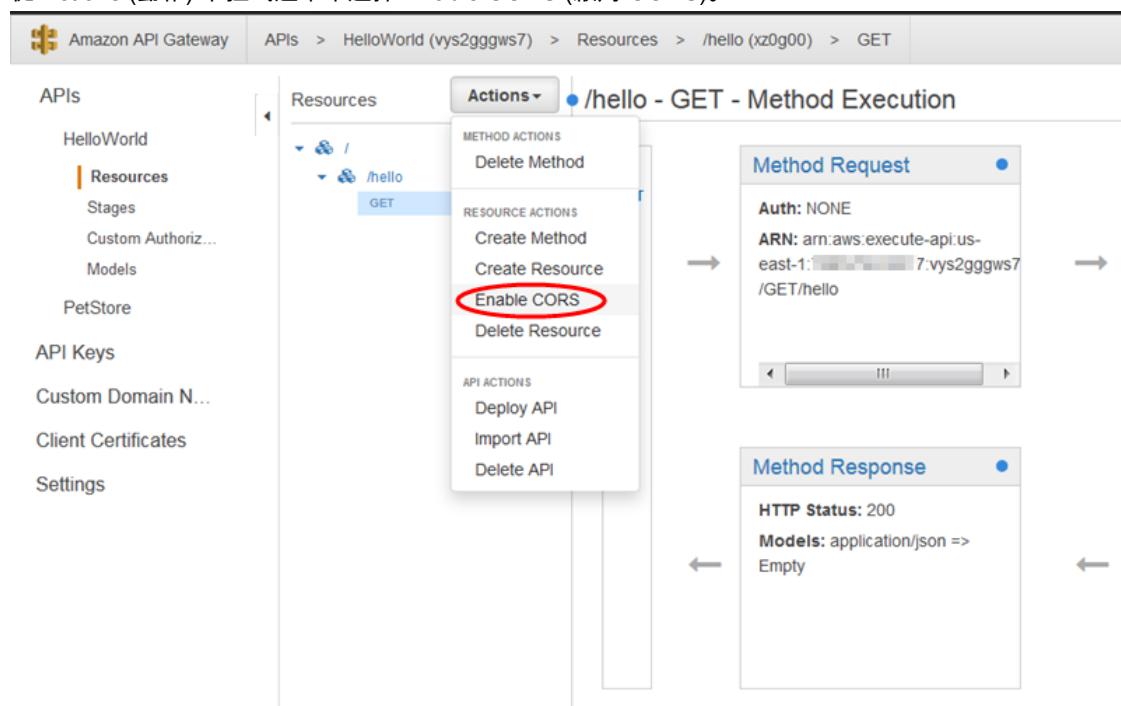
您可以使用 API Gateway 主控台，對 REST API 資源上您已建立的一個或所有方法啟用 CORS 支援。

### Important

資源可以包含子資源。對資源及其方法啟用 CORS 支援並不會對子資源及其方法遞迴啟用此支援。

### 在 REST API 資源上啟用 CORS 支援

1. 登入位於 <https://console.aws.amazon.com/apigateway> 的 API Gateway 主控台。
2. 從 APIs 清單中選擇 API。
3. 在 Resources (資源) 下，選擇一個資源。這會為資源上的所有方法啟用 CORS。  
或者，您可以在資源下選擇一個方法，只為此方法啟用 CORS。
4. 從 Actions (動作) 下拉式選單中選擇 Enable CORS (啟用 CORS)。



5. 在 Enable CORS (啟用 CORS) 表單中，執行下列操作：
  - a. 在 Access-Control-Allow-Headers 輸入欄位中，輸入以逗號分隔之標頭清單的靜態字串，用戶端必須在資源的實際請求中提交這些標頭。使用主控台提供的 'Content-Type, X-Amz-Date, Authorization, X-Api-Key, X-Amz-Security-Token' 標頭清單，或指定您自己的標頭。
  - b. 使用主控台提供的 '\*' 值，做為 Access-Control-Allow-Origin 標頭值，以允許所有來源中的存取請求，或指定允許其存取資源的來源。
  - c. 選擇 Enable CORS and replace existing CORS headers (啟用 CORS 並取代現有的 CORS 標頭)。

The screenshot shows the 'Enable CORS' configuration for the '/hello' resource. Under the 'Actions' dropdown, 'Enable CORS' is selected. The configuration includes:

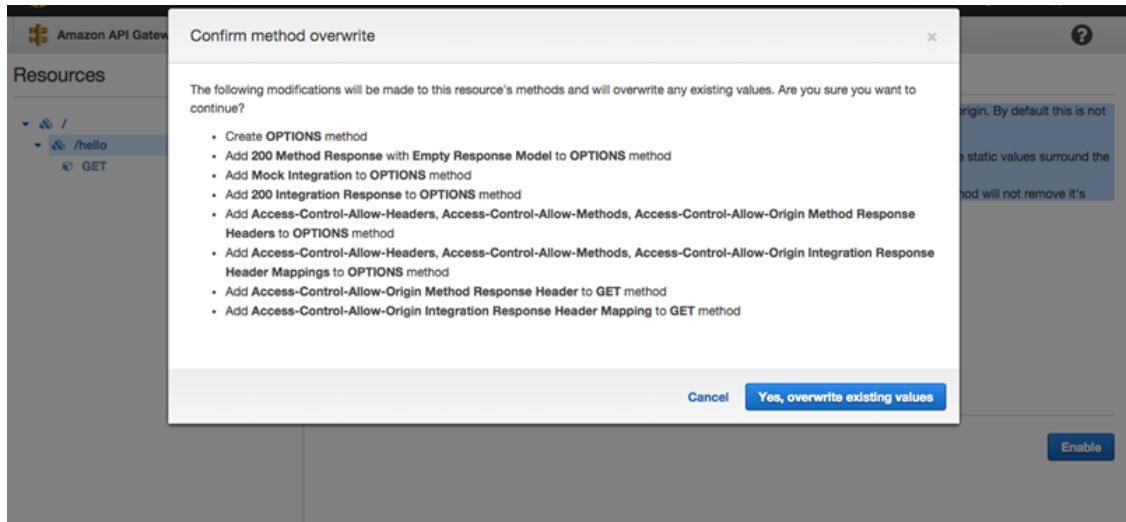
- Methods\***: GET, OPTIONS
- Access-Control-Allow-Methods**: GET,OPTIONS
- Access-Control-Allow-Headers**: 'Content-Type,X-Amz-Date,Authorization'
- Access-Control-Allow-Origin\***: '\*' (with a warning icon)

A blue button at the bottom right says 'Enable CORS and replace existing CORS headers'.

### Important

將上述說明應用在代理整合中的 ANY 方法時，不會設定任何適用的 CORS 標頭。反之，您的後端必須傳回適當的 CORS 標頭，例如 Access-Control-Allow-Origin。

6. 在 Confirm method changes (確認方法變更) 中，選擇 Yes, overwrite existing values (是，覆寫現有的值) 以確認新的 CORS 設定。



在 GET 方法上啟用 CORS 之後，如果資源中尚未有 OPTIONS 方法，則系統會將此方法新增至資源。OPTIONS 方法的 200 回應會自動設定為傳回三個 Access-Control-Allow-\* 標頭，來完成預檢交握。此外，根據預設也會設定實際 (GET) 方法，以其 200 回應中傳回 Access-Control-Allow-Origin 標頭。對於其他類型的回應，如果您不想要傳回 Cross-origin access 錯誤，則需要使用 '\*' 或特定來源手動進行設定，以傳回 Access-Control-Allow-Origin' 標頭。

在您的資源上啟用 CORS 支援之後，您必須部署或重新部署 API，新的設定才能生效。如需詳細資訊，請參閱the section called “從主控台部署 REST API” (p. 459)。

## 使用 API Gateway 匯入 API 在資源上啟用 CORS

如果您使用 [API Gateway 匯入 API \(p. 319\)](#)，則可以透過 OpenAPI 檔案來設定 CORS 支援。您必須先在傳回必要標頭的資源中定義 OPTIONS 方法。

### Note

Web 瀏覽器必須在每個接受 CORS 請求的 API 方法中設定 Access-Control-Allow-Headers 與 Access-Control-Allow-Origin 標頭。此外，某些瀏覽器會先向相同資源中的 OPTIONS 方法提出 HTTP 請求，然後預期收到相同的標頭。

下列範例為模擬整合建立 OPTIONS 方法。

```
/users
  options:
    summary: CORS support
    description: |
      Enable CORS by returning correct headers
    consumes:
      - application/json
    produces:
      - application/json
    tags:
      - CORS
  x-amazon-apigateway-integration:
    type: mock
    requestTemplates:
      application/json: |
        {
          "statusCode" : 200
        }
    responses:
      "default":
        statusCode: "200"
        responseParameters:
          method.response.header.Access-Control-Allow-Headers : "'Content-Type,X-Amz-Date,Authorization,X-Api-Key'"
          method.response.header.Access-Control-Allow-Methods : "'*'"
          method.response.header.Access-Control-Allow-Origin : "'*'"
        responseTemplates:
          application/json: |
            {}
    responses:
      200:
        description: Default response for CORS method
        headers:
          Access-Control-Allow-Headers:
            type: "string"
          Access-Control-Allow-Methods:
            type: "string"
          Access-Control-Allow-Origin:
            type: "string"
```

一旦您為資源設定 OPTIONS 方法，您就可以將必要的標頭新增至相同資源中需要接受 CORS 請求的其他方法。

1. 對回應類型宣告 Access-Control-Allow-Origin 與 Headers。

```
responses:
  200:
    description: Default response for CORS method
    headers:
      Access-Control-Allow-Headers:
```

```
        type: "string"
Access-Control-Allow-Methods:
    type: "string"
Access-Control-Allow-Origin:
    type: "string"
```

2. 在 `x-amazon-apigateway-integration` 標籤中，將這些標頭的對應設定為您的靜態值：

```
responses:
"default":
statusCode: "200"
responseParameters:
method.response.header.Access-Control-Allow-Headers : "'Content-Type,X-Amz-Date,Authorization,X-Api-Key'"
method.response.header.Access-Control-Allow-Methods : "'*!*'"
method.response.header.Access-Control-Allow-Origin : "'*!*'"
```

## 後端使用用戶端 SSL �凭證進行驗證

您可以使用 API Gateway 產生 SSL �凭證，並在後端中使用其公有金鑰來確認對後端系統的 HTTP 請求來自 API Gateway。這可讓您的 HTTP 後端控制和僅接受源自 Amazon API Gateway 的請求，即使可公開存取後端也是一樣。

### Note

有些後端伺服器可能不支援 SSL 用戶端驗證，因為 API Gateway 可以傳回 SSL �凭證錯誤。如需不相容的後端伺服器清單，請參閱 [the section called “重要說明” \(p. 630\)](#)。

API Gateway 所產生的 SSL �凭證是自我簽署的，而且只有在 API Gateway 主控台中或透過 API 才會顯示憑證的公有金鑰。

### 主題

- [使用 API Gateway 主控台產生用戶端憑證 \(p. 377\)](#)
- [設定 API 來使用 SSL �凭證 \(p. 377\)](#)
- [測試叫用以驗證用戶端憑證組態 \(p. 378\)](#)
- [設定後端 HTTPS 伺服器以驗證用戶端憑證 \(p. 378\)](#)
- [輪換到期的用戶端憑證 \(p. 378\)](#)
- [API Gateway 為 HTTP 與 HTTP 代理整合支援的憑證授權機構 \(p. 379\)](#)

## 使用 API Gateway 主控台產生用戶端憑證

1. 在主要導覽窗格中，選擇 Client Certificates (用戶端憑證)。
2. 從 Client Certificates (用戶端憑證) 窗格中，選擇 Generate Client Certificate (產生用戶端憑證)。
3. (選擇性) 針對 Edit (編輯)，選擇以新增所產生憑證的描述性標題，然後選擇 Save (儲存) 來儲存說明。API Gateway 會產生新的憑證，並傳回新憑證 GUID 以及 PEM 編碼的公有金鑰。

您現在已經準備好設定 API 來使用憑證。

## 設定 API 來使用 SSL �凭證

這些說明假設您已完成「[使用 API Gateway 主控台產生用戶端憑證 \(p. 377\)](#)」。

1. 在 API Gateway 主控台中，建立或開啟您要使用用戶端憑證的 API。請確定 API 已部署至階段。
2. 在所選取的 API 下選擇 Stages (階段)，然後選擇階段。

3. 在 Stage Editor (階段編輯器) 面板中，於 Client Certificate (用戶端憑證) 區段下選取憑證。
4. 若要儲存設定，請選擇 Save Changes (儲存變更)。

如果先前已在 API Gateway 主控台中部署該 API，您將需要重新部署 API，變更才能生效。

選取並儲存 API 的憑證之後，API Gateway 會將憑證用於 API 中 HTTP 整合的所有呼叫。

## 測試叫用以驗證用戶端憑證組態

1. 選擇 API 方法。在 Client (用戶端) 中，選擇 Test (測試)。
2. 從 Client Certificate (用戶端憑證) 中，選擇 Test (測試) 來呼叫方法請求。

API Gateway 會呈現所選擇的 SSL �凭證，讓 HTTP 後端驗證 API。

## 設定後端 HTTPS 伺服器以驗證用戶端憑證

這些說明是假設您已完成 [使用 API Gateway 主控台產生用戶端憑證 \(p. 377\)](#) 並下載了用戶端憑證的複本。您可以下載用戶端憑證，方式是呼叫 API Gateway REST API 的 `clientcertificate:by-id` 或 AWS CLI 的 `get-client-certificate`。

在設定後端 HTTPS 伺服器以驗證 API Gateway 的用戶端 SSL �凭證之前，您必須取得 PEM 編碼的私有金鑰，以及由受信任之憑證授權單位提供的伺服器端憑證。

如果伺服器網域名稱為 `myserver.mydomain.com`，則伺服器憑證的 CNAME 值必須為 `myserver.mydomain.com` 或 `*.mydomain.com`。

支援的憑證授權單位包括 [Let's Encrypt](#) 或 [the section called “為 HTTP 與 HTTP 代理整合支援的憑證授權機構” \(p. 379\)](#) 其中之一。

例如，假設用戶端憑證檔案為 `apig-cert.pem`，而伺服器私有金鑰和憑證檔案分別為 `server-key.pem` 與 `server-cert.pem`，則對於後端的 Node.js 伺服器，您可以將伺服器設定如下：

```
var fs = require('fs');
var https = require('https');
var options = {
  key: fs.readFileSync('server-key.pem'),
  cert: fs.readFileSync('server-cert.pem'),
  ca: fs.readFileSync('apig-cert.pem'),
  requestCert: true,
  rejectUnauthorized: true
};
https.createServer(options, function (req, res) {
  res.writeHead(200);
  res.end("hello world\n");
}).listen(443);
```

針對節點-express 應用程式，您可以使用 `client-certificate-auth` 模組，透過 PEM 編碼憑證來驗證用戶端請求。

針對其他 HTTPS 伺服器，請參閱伺服器的文件。

## 輪換到期的用戶端憑證

由 API Gateway 產生的用戶端憑證有效時間為 365 天。您必須在 API 階段的用戶端憑證過期之前輪換憑證，以避免 API 停機。您可以查看憑證的過期日期，方式是呼叫 API Gateway REST API 的 `clientCertificate:by-id` 或 `get-client-certificate` 的 AWS CLI 命令，並檢查傳回的 `expirationDate` 屬性。

要輪換用戶端憑證，請依照以下步驟：

- 產生新的用戶端憑證，方式是呼叫 API Gateway REST API 的 [clientcertificate:generate](#) 或 [generate-client-certificate](#) 的 AWS CLI 命令。在本教學中，我們將假設新的用戶端憑證 ID 為 `ndiqef`。
- 更新後端伺服器以納入新的用戶端憑證。請先不要移除現有的用戶端憑證。  
有些伺服器可能需要重新啟動以完成更新。請參閱伺服器文件，確認更新期間是否需要重新啟動伺服器。
- 更新 API 階段以使用新的用戶端憑證，方式是以新的用戶端憑證 ID (`ndiqef`) 呼叫 API Gateway REST API 的 [stage:update](#)：

```
PATCH /restapis/{restapi-id}/stages/stage1 HTTP/1.1
Content-Type: application/json
Host: apigateway.us-east-1.amazonaws.com
X-Amz-Date: 20170603T200400Z
Authorization: AWS4-HMAC-SHA256 Credential=...

{
  "patchOperations" : [
    {
      "op" : "replace",
      "path" : "/clientCertificateId",
      "value" : "ndiqef"
    }
  ]
}
```

- 或者呼叫 [update-stage](#) 的 CLI 命令。
- 更新後端伺服器以移除舊的用戶端憑證。
  - 從 API Gateway 刪除舊憑證，方式是呼叫 API Gateway REST API 的 [clientcertificate:delete](#)，指定舊憑證的 `clientCertificateId` (`a1b2c3`)：

```
DELETE /clientcertificates/a1b2c3
```

或者呼叫 [delete-client-certificate](#) 的 CLI 命令：

```
aws apigateway delete-client-certificate --client-certificate-id a1b2c3
```

若要在主控台中針對先前部署的 API 輪換用戶端憑證：

- 在主要導覽窗格中，選擇 Client Certificates (用戶端憑證)。
- 從 Client Certificates (用戶端憑證) 窗格中，選擇 Generate Client Certificate (產生用戶端憑證)。
- 開啟您要對其使用用戶端憑證的 API。
- 在所選取的 API 下選擇 Stages (階段)，然後選擇階段。
- 在 Stage Editor (階段編輯器) 面板中，於 Client Certificate (用戶端憑證) 區段下選取新憑證。
- 若要儲存設定，請選擇 Save Changes (儲存變更)。

您將需要重新部署 API，變更才能生效。

## API Gateway 為 HTTP 與 HTTP 代理整合支援的憑證授權機構

下列清單顯示 API Gateway 為 HTTP 與 HTTP 代理整合支援的憑證授權機構。

```
Alias name: mozillacert81.pem
MD5: D5:E9:81:40:C5:18:69:FC:46:2C:89:75:62:0F:AA:78
SHA256:
5C:58:46:8D:55:F5:8E:49:7E:74:39:82:D2:B5:00:10:B6:D1:65:37:4A:CF:83:A7:D4:A3:2D:B7:68:C4:40:8E
Alias name: mozillacert99.pem
MD5: 2B:70:20:56:86:82:A0:18:C8:07:53:12:28:70:21:72
SHA256:
97:8C:D9:66:F2:FA:A0:7B:A7:AA:95:00:D9:C0:2E:9D:77:F2:CD:AD:A6:AD:6B:A7:4A:F4:B9:1C:66:59:3C:50
Alias name: swisssignplatinumg2ca
MD5: C9:98:27:77:28:1E:3D:0E:15:3C:84:00:B8:85:03:E6
SHA256:
3B:22:2E:56:67:11:E9:92:30:0D:C0:B1:5A:B9:47:3D:AF:DE:F8:C8:4D:0C:EF:7D:33:17:B4:C1:82:1D:14:36
Alias name: mozillacert145.pem
MD5: 60:84:7C:5A:CE:DB:0C:D4:CB:A7:E9:FE:02:C6:A9:C0
SHA256:
D4:1D:82:9E:8C:16:59:82:2A:F9:3F:CE:62:BF:FC:DE:26:4F:C8:4E:8B:95:0C:5F:F2:75:D0:52:35:46:95:A3
Alias name: mozillacert37.pem
MD5: AB:57:A6:5B:7D:42:82:19:B5:D8:58:26:28:5E:FD:FF
SHA256:
E3:B6:A2:DB:2E:D7:CE:48:84:2F:7A:C5:32:41:C7:B7:1D:54:14:4B:FB:40:C1:1F:3F:1D:0B:42:F5:EE:A1:2D
Alias name: mozillacert4.pem
MD5: 4F:EB:F1:F0:70:C2:80:63:5D:58:9F:DA:12:3C:A9:C4
SHA256:
OB:5E:ED:4E:84:64:03:CF:55:E0:65:84:84:40:ED:2A:82:75:8B:F5:B9:AA:1F:25:3D:46:13:CF:A0:80:FF:3F
Alias name: mozillacert70.pem
MD5: 5E:80:9E:84:5A:0E:65:0B:17:02:F3:55:18:2A:3E:D7
SHA256:
06:3E:4A:FA:C4:91:DF:D3:32:F3:08:9B:85:42:E9:46:17:D8:93:D7:FE:94:4E:10:A7:93:7E:E2:9D:96:93:C0
Alias name: mozillacert88.pem
MD5: 73:9F:4C:4B:73:5B:79:E9:FA:BA:1C:EF:6E:CB:D5:C9
SHA256:
BC:10:4F:15:A4:8B:E7:09:DC:A5:42:A7:E1:D4:B9:DF:6F:05:45:27:E8:02:EA:A9:2D:59:54:44:25:8A:FE:71
Alias name: mozillacert134.pem
MD5: FC:11:B8:D8:08:93:30:00:6D:23:F9:7E:EB:52:1E:02
SHA256:
69:FA:C9:BD:55:FB:0A:C7:8D:53:BB:EE:5C:F1:D5:97:98:9F:D0:AA:AB:20:A2:51:51:BD:F1:73:3E:E7:D1:22
Alias name: mozillacert26.pem
MD5: DC:32:C3:A7:6D:25:57:C7:68:09:9D:EA:2D:A9:A2:D1
SHA256:
F1:C1:B5:0A:E5:A2:0D:D8:03:0E:C9:F6:BC:24:82:3D:D3:67:B5:25:57:59:B4:E7:1B:61:FC:E9:F7:37:5D:73
Alias name: buypassclass2ca
MD5: 46:A7:D2:FE:45:FB:64:5A:A8:59:90:9B:78:44:9B:29
SHA256:
9A:11:40:25:19:7C:5B:B9:5D:94:E6:3D:55:CD:43:79:08:47:B6:46:B2:3C:DF:11:AD:A4:A0:0E:FF:15:FB:48
Alias name: chunghwaepkirootca
MD5: 1B:2E:00:CA:26:06:90:3D:AD:FE:6F:15:68:D3:6B:B3
SHA256:
C0:A6:F4:DC:63:A2:4B:FD:CF:54:EF:2A:6A:08:2A:0A:72:DE:35:80:3E:2F:F5:FF:52:7A:E5:D8:72:06:DF:D5
Alias name: verisignclass2g2ca
MD5: 2D:BB:E5:25:D3:D1:65:82:3A:B7:0E:FA:E6:EB:E2:E1
SHA256:
3A:43:E2:20:FE:7F:3E:A9:65:3D:1E:21:74:2E:AC:2B:75:C2:OF:D8:98:03:05:BC:50:2C:AF:8C:2D:9B:41:A1
Alias name: mozillacert77.pem
MD5: CD:68:B6:A7:C7:C4:CE:75:E0:1D:4F:57:44:61:92:09
SHA256:
EB:04:CF:5E:B1:F3:9A:FA:76:2F:2B:B1:20:F2:96:CB:A5:20:C1:B9:7D:B1:58:95:65:B8:1C:B9:A1:7B:72:44
Alias name: mozillacert123.pem
MD5: C1:62:3E:23:C5:82:73:9C:03:59:4B:2B:E9:77:49:7F
SHA256:
07:91:CA:07:49:B2:07:82:AA:D3:C7:D7:BD:0C:DF:C9:48:58:35:84:3E:B2:D7:99:60:09:CE:43:AB:6C:69:27
Alias name: utndatacorpsgcc
MD5: B3:A5:3E:77:21:6D:AC:4A:C0:C9:FB:D5:41:3D:CA:06
SHA256:
85:FB:2F:91:DD:12:27:5A:01:45:B6:36:53:4F:84:02:4A:D6:8B:69:B8:EE:88:68:4F:F7:11:37:58:05:B3:48
Alias name: mozillacert15.pem
MD5: 88:2C:8C:52:B8:A2:3C:F3:F7:BB:03:EA:AE:AC:42:0B
```

```
SHA256:  
OF:99:3C:8A:EF:97:BA:AF:56:87:14:0E:D5:9A:D1:82:1B:B4:AF:AC:F0:AA:9A:58:B5:D5:7A:33:8A:3A:FB:CB  
Alias name: digicertglobalrootca  
MD5: 79:E4:A9:84:0D:7D:3A:96:D7:C0:4F:E2:43:4C:89:2E  
SHA256:  
43:48:A0:E9:44:4C:78:CB:26:5E:05:8D:5E:89:44:B4:D8:4F:96:62:BD:26:DB:25:7F:89:34:A4:43:C7:01:61  
Alias name: mozillacert66.pem  
MD5: 3D:41:29:CB:1E:AA:11:74:CD:5D:B0:62:AF:B0:43:5B  
SHA256:  
E6:09:07:84:65:A4:19:78:0C:B6:AC:4C:1C:0B:FB:46:53:D9:D9:CC:6E:B3:94:6E:B7:F3:D6:99:97:BA:D5:98  
Alias name: mozillacert112.pem  
MD5: 37:41:49:1B:18:56:9A:26:F5:AD:C2:66:FB:40:A5:4C  
SHA256:  
DD:69:36:FE:21:F8:F0:77:C1:23:A1:A5:21:C1:22:24:F7:22:55:B7:3E:03:A7:26:06:93:E8:A2:4B:0F:A3:89  
Alias name: utnuserfirstclientauthemailca  
MD5: D7:34:3D:EF:1D:27:09:28:E1:31:02:5B:13:2B:DD:F7  
SHA256:  
43:F2:57:41:2D:44:0D:62:74:76:97:4F:87:7D:A8:F1:FC:24:44:56:5A:36:7A:E6:0E:DD:C2:7A:41:25:31:AE  
Alias name: verisignc2g1.pem  
MD5: B3:9C:25:B1:C3:2E:32:53:80:15:30:9D:4D:02:77:3E  
SHA256:  
BD:46:9F:F4:5F:AA:E7:C5:4C:CB:D6:9D:3F:3B:00:22:55:D9:B0:6B:10:B1:D0:FA:38:8B:F9:6B:91:8B:2C:E9  
Alias name: mozillacert55.pem  
MD5: 74:9D:EA:60:24:C4:FD:22:53:3E:CC:3A:72:D9:29:4F  
SHA256:  
A4:31:0D:50:AF:18:A6:44:71:90:37:2A:86:AF:AF:8B:95:1F:FB:43:1D:83:7F:1E:56:88:B4:59:71:ED:15:57  
Alias name: mozillacert101.pem  
MD5: DF:F2:80:73:CC:F1:E6:61:73:FC:F5:42:E9:C5:7C:EE  
SHA256:  
62:F2:40:27:8C:56:4C:4D:D8:BF:7D:9D:4F:6F:36:6E:A8:94:D2:2F:5F:34:D9:89:A9:83:AC:EC:2F:FF:ED:50  
Alias name: mozillacert119.pem  
MD5: 94:14:77:7E:3E:5E:FD:8F:30:BD:41:B0:CF:E7:D0:30  
SHA256:  
CA:42:DD:41:74:5F:D0:B8:1E:B9:02:36:2C:F9:D8:BF:71:9D:A1:BD:1B:1E:FC:94:6F:5B:4C:99:F4:2C:1B:9E  
Alias name: verisignc3g1.pem  
MD5: EF:5A:F1:33:EF:F1:CD:BB:51:02:EE:12:14:4B:96:C4  
SHA256:  
A4:B6:B3:99:6F:C2:F3:06:B3:FD:86:81:BD:63:41:3D:8C:50:09:CC:4F:A3:29:C2:CC:F0:E2:FA:1B:14:03:05  
Alias name: mozillacert44.pem  
MD5: 72:E4:4A:87:E3:69:40:80:77:EA:BC:E3:F4:FF:F0:E1  
SHA256:  
96:0A:DF:00:63:E9:63:56:75:0C:29:65:DD:0A:08:67:DA:0B:9C:BD:6E:77:71:4A:EA:FB:23:49:AB:39:3D:A3  
Alias name: mozillacert108.pem  
MD5: 3E:45:52:15:09:51:92:E1:B7:5D:37:9F:B1:87:29:8A  
SHA256:  
EB:D4:10:40:E4:BB:3E:C7:42:C9:E3:81:D3:1E:F2:A4:1A:48:B6:68:5C:96:E7:CE:F3:C1:DF:6C:D4:33:1C:99  
Alias name: mozillacert95.pem  
MD5: 3D:3B:18:9E:2C:64:5A:E8:D5:88:CE:0E:F9:37:C2:EC  
SHA256:  
ED:F7:EB:BC:A2:7A:2A:38:4D:38:7B:7D:40:10:C6:66:E2:ED:B4:84:3E:4C:29:B4:AE:1D:5B:93:32:E6:B2:4D  
Alias name: keynectisrootca  
MD5: CC:4D:AE:FB:30:6B:D8:38:FE:50:EB:86:61:4B:D2:26  
SHA256:  
42:10:F1:99:49:9A:9A:C3:3C:8D:E0:2B:A6:DB:AA:14:40:8B:DD:8A:6E:32:46:89:C1:92:2D:06:97:15:A3:32  
Alias name: mozillacert141.pem  
MD5: A9:23:75:9B:BA:49:36:6E:31:C2:DB:F2:E7:66:BA:87  
SHA256:  
58:D0:17:27:9C:D4:DC:63:AB:DD:B1:96:A6:C9:90:6C:30:C4:E0:87:83:EA:E8:C1:60:99:54:D6:93:55:59:6B  
Alias name: equifaxsecureglobalebusinessca1  
MD5: 51:F0:2A:33:F1:F5:55:39:07:F2:16:7A:47:C7:5D:63  
SHA256:  
86:AB:5A:65:71:D3:32:9A:BC:D2:E4:E6:37:66:8B:A8:9C:73:1E:C2:93:B6:CB:A6:0F:71:63:40:A0:91:CE:AE  
Alias name: affirmtrustpremiumca  
MD5: C4:5D:0E:48:B6:AC:28:30:4E:0A:BC:F9:38:16:87:57  
SHA256:  
70:A7:3F:7F:37:6B:60:07:42:48:90:45:34:B1:14:82:D5:BF:0E:69:8E:CC:49:8D:F5:25:77:EB:F2:E9:3B:9A
```

```
Alias name: baltimorecodesigningca
MD5: 90:F5:28:49:56:D1:5D:2C:B0:53:D4:4B:EF:6F:90:22
SHA256:
A9:15:45:DB:D2:E1:9C:4C:CD:F9:09:AA:71:90:0D:18:C7:35:1C:89:B3:15:F0:F1:3D:05:C1:3A:8F:FB:46:87
Alias name: mozillacert33.pem
MD5: 22:2D:A6:01:EA:7C:0A:F7:F0:6C:56:43:3F:77:76:D3
SHA256:
A2:2D:BA:68:1E:97:37:6E:2D:39:7D:72:8A:AE:3A:9B:62:96:B9:FD:BA:60:BC:2E:11:F6:47:F2:C6:75:FB:37
Alias name: mozillacert0.pem
MD5: CA:3D:D3:68:F1:03:5C:D0:32:FA:B8:2B:59:E8:5A:DB
SHA256:
A5:31:25:18:8D:21:10:AA:96:4B:02:C7:B7:C6:DA:32:03:17:08:94:E5:FB:71:FF:FB:66:67:D5:E6:81:0A:36
Alias name: mozillacert84.pem
MD5: 49:63:AE:27:F4:D5:95:3D:D8:DB:24:86:B8:9C:07:53
SHA256:
79:3C:BF:45:59:B9:FD:E3:8A:B2:2D:F1:68:69:F6:98:81:AE:14:C4:B0:13:9A:C7:88:A7:8A:1A:FC:CA:02:FB
Alias name: mozillacert130.pem
MD5: 65:58:AB:15:AD:57:6C:1E:A8:A7:B5:69:AC:BF:FF:EB
SHA256:
F4:C1:49:55:1A:30:13:A3:5B:C7:BF:FE:17:A7:F3:44:9B:C1:AB:5B:5A:0A:E7:4B:06:C2:3B:90:00:4C:01:04
Alias name: mozillacert148.pem
MD5: 4C:56:41:E5:0D:BB:2B:E8:CA:A3:ED:18:08:AD:43:39
SHA256:
6E:A5:47:41:D0:04:66:7E:ED:1B:48:16:63:4A:A3:A7:9E:6E:4B:96:95:0F:82:79:DA:FC:8D:9B:D8:81:21:37
Alias name: mozillacert22.pem
MD5: 02:26:C3:01:5E:08:30:37:43:A9:D0:7D:CF:37:E6:BF
SHA256:
37:D5:10:06:C5:12:EA:AB:62:64:21:F1:EC:8C:92:01:3F:C5:F8:2A:E9:8E:E5:33:EB:46:19:B8:DE:B4:D0:6C
Alias name: verisignc1g1.pem
MD5: 97:60:E8:57:5F:D3:50:47:E5:43:0C:94:36:8A:B0:62
SHA256:
D1:7C:D8:EC:D5:86:B7:12:23:8A:48:2C:E4:6F:A5:29:39:70:74:2F:27:6D:8A:B6:A9:E4:6E:E0:28:8F:33:55
Alias name: mozillacert7.pem
MD5: 32:4A:4B:BB:C8:63:69:9B:BE:74:9A:C6:DD:1D:46:24
SHA256:
14:65:FA:20:53:97:B8:76:FA:A6:F0:A9:95:8E:55:90:E4:0F:CC:7F:AA:4F:B7:C2:C8:67:75:21:FB:5F:B6:58
Alias name: mozillacert73.pem
MD5: D6:39:81:C6:52:7E:96:69:FC:FC:CA:66:ED:05:F2:96
SHA256:
2C:E1:CB:OB:F9:D2:F9:E1:02:99:3F:BE:21:51:52:C3:B2:DD:0C:AB:DE:1C:68:E5:31:9B:83:91:54:DB:B7:F5
Alias name: mozillacert137.pem
MD5: D3:D9:BD:AE:9F:AC:67:24:B3:C8:1B:52:E1:B9:A9:BD
SHA256:
BD:81:CE:3B:4F:65:91:D1:1A:67:B5:FC:7A:47:FD:EF:25:52:1B:F9:AA:4E:18:B9:E3:DF:2E:34:A7:80:3B:E8
Alias name: swissignsilver2ca
MD5: E0:06:A1:C9:7D:CF:C9:FC:0D:CO:56:75:96:D8:62:13
SHA256:
BE:6C:4D:A2:BB:B9:BA:59:B6:F3:93:97:68:37:42:46:C3:CO:05:99:3F:A9:8F:02:0D:1D:ED:BE:D4:8A:81:D5
Alias name: mozillacert11.pem
MD5: 87:CE:0B:7B:2A:0E:49:00:E1:58:71:9B:37:A8:93:72
SHA256:
3E:90:99:B5:01:5E:8F:48:6C:00:BC:EA:9D:11:1E:E7:21:FA:BA:35:5A:89:BC:F1:DF:69:56:1E:3D:C6:32:5C
Alias name: mozillacert29.pem
MD5: D3:F3:A6:16:CO:FA:6B:1D:59:B1:2D:96:4D:0E:11:2E
SHA256:
15:F0:BA:00:A3:AC:7A:F3:AC:88:4C:07:2B:10:11:A0:77:BD:77:CO:97:F4:01:64:B2:F8:59:8A:BD:83:86:0C
Alias name: mozillacert62.pem
MD5: EF:5A:F1:33:EF:F1:CD:BB:51:02:EE:12:14:4B:96:C4
SHA256:
A4:B6:B3:99:6F:C2:F3:06:B3:FD:86:81:BD:63:41:3D:8C:50:09:CC:4F:A3:29:C2:CC:F0:E2:FA:1B:14:03:05
Alias name: mozillacert126.pem
MD5: 77:0D:19:B1:21:FD:00:42:9C:3E:0C:A5:DD:0B:02:8E
SHA1: 25:01:90:19:CF:FB:D9:99:1C:B7:68:25:74:8D:94:5F:30:93:95:42
SHA256:
AF:8B:67:62:A1:E5:28:22:81:61:A9:5D:5C:55:9E:E2:66:27:8F:75:D7:9E:83:01:89:A5:03:50:6A:BD:6B:4C
Alias name: securetrustca
```

```
MD5: DC:32:C3:A7:6D:25:57:C7:68:09:9D:EA:2D:A9:A2:D1
SHA256:
F1:C1:B5:0A:E5:A2:0D:D8:03:0E:C9:F6:BC:24:82:3D:D3:67:B5:25:57:59:B4:E7:1B:61:FC:E9:F7:37:5D:73
Alias name: soneraaclass1ca
MD5: 33:B7:84:F5:5F:27:D7:68:27:DE:14:DE:12:2A:ED:6F
SHA256:
CD:80:82:84:CF:74:6F:F2:FD:6E:B5:8A:A1:D5:9C:4A:D4:B3:CA:56:FD:C6:27:4A:89:26:A7:83:5F:32:31:3D
Alias name: mozillacert18.pem
MD5: F1:6A:22:18:C9:CD:DF:CE:82:1D:1D:B7:78:5C:A9:A5
SHA256:
44:04:E3:3B:5E:14:0D:CF:99:80:51:FD:FC:80:28:C7:C8:16:15:C5:EE:73:7B:11:1B:58:82:33:A9:B5:35:A0
Alias name: mozillacert51.pem
MD5: 18:98:C0:D6:E9:3A:FC:F9:B0:F5:0C:F7:4B:01:44:17
SHA256:
EA:A9:62:C4:FA:4A:6B:AF:EB:E4:15:19:6D:35:1C:CD:88:8D:4F:53:F3:FA:8A:E6:D7:C4:66:A9:4E:60:42:BB
Alias name: mozillacert69.pem
MD5: A6:B0:CD:85:80:DA:5C:50:34:A3:39:90:2F:55:67:73
SHA256:
25:30:CC:8E:98:32:15:02:BA:D9:6F:9B:1F:BA:1B:09:9E:2D:29:9E:0F:45:48:BB:91:4F:36:3B:C0:D4:53:1F
Alias name: mozillacert115.pem
MD5: 2B:9B:9E:E4:7B:6C:1F:00:72:1A:CC:C1:77:79:DF:6A
SHA256:
91:E2:F5:78:8D:58:10:EB:A7:BA:58:73:7D:E1:54:8A:8E:CA:CD:01:45:98:BC:0B:14:3E:04:1B:17:05:25:52
Alias name: verisignclass3g5ca
MD5: CB:17:E4:31:67:3E:E2:09:FE:45:57:93:F3:0A:FA:1C
SHA256:
9A:CF:AB:7E:43:C8:D8:80:D0:6B:26:2A:94:DE:EE:E4:B4:65:99:89:C3:D0:CA:F1:9B:AF:64:05:E4:1A:B7:DF
Alias name: utnuserfirsthardwareca
MD5: 4C:56:41:E5:0D:BB:2B:E8:CA:A3:ED:18:08:AD:43:39
SHA256:
6E:A5:47:41:D0:04:66:7E:ED:1B:48:16:63:4A:A3:A7:9E:6E:4B:96:95:0F:82:79:DA:FC:8D:9B:D8:81:21:37
Alias name: addtrustqualifiedca
MD5: 27:EC:39:47:CD:DA:5A:AF:E2:9A:01:65:21:A9:4C:BB
SHA256:
80:95:21:08:05:DB:4B:BC:35:5E:44:28:D8:FD:6E:C2:CD:E3:AB:5F:B9:7A:99:42:98:8E:B8:F4:DC:D0:60:16
Alias name: mozillacert40.pem
MD5: 56:5F:AA:80:61:12:17:F6:67:21:E6:2B:6D:61:56:8E
SHA256:
8D:A0:84:FC:F9:9C:E0:77:22:F8:9B:32:05:93:98:06:FA:5C:B8:11:E1:C8:13:F6:A1:08:C7:D3:36:B3:40:8E
Alias name: mozillacert58.pem
MD5: 01:5E:D8:6B:BD:6F:3D:8E:A1:31:F8:12:E0:98:73:6A
SHA256:
5E:DB:7A:C4:3B:82:A0:6A:87:61:E8:D7:BE:49:79:EB:F2:61:1F:7D:D7:9B:F9:1C:1C:6B:56:6A:21:9E:D7:66
Alias name: verisignclass3g3ca
MD5: CD:68:B6:A7:C7:C4:CE:75:E0:1D:4F:57:44:61:92:09
SHA256:
EB:04:CF:5E:B1:F3:9A:FA:76:2F:2B:B1:20:F2:96:CB:A5:20:C1:B9:7D:B1:58:95:65:B8:1C:B9:A1:7B:72:44
Alias name: mozillacert104.pem
MD5: 55:5D:63:00:97:BD:6A:97:F5:67:AB:4B:FB:6E:63:15
SHA256:
1C:01:C6:F4:DB:B2:FE:FC:22:55:8B:2B:CA:32:56:3F:49:84:4A:CF:C3:2B:7B:E4:B0:FF:59:9F:9E:8C:7A:F7
Alias name: mozillacert91.pem
MD5: 30:C9:E7:1E:6B:E6:14:EB:65:B2:16:69:20:31:67:4D
SHA256:
C1:B4:82:99:AB:A5:20:8F:E9:63:0A:CE:55:CA:68:A0:3E:DA:5A:51:9C:88:02:A0:D3:A6:73:BE:8F:8E:55:7D
Alias name: thawtepersonalfreemailca
MD5: 53:4B:1D:17:58:58:1A:30:A1:90:F8:6E:5C:F2:CF:65
SHA256:
5B:38:BD:12:9E:83:D5:A0:CA:D2:39:21:08:94:90:D5:0D:4A:AE:37:04:28:F8:DD:FF:FF:FA:4C:15:64:E1:84
Alias name: certplusclass3ppprimaryca
MD5: E1:4B:52:73:D7:1B:DB:93:30:E5:BD:E4:09:6E:BE:FB
SHA256:
CC:C8:94:89:37:1B:AD:11:1C:90:61:9B:EA:24:0A:2E:6D:AD:D9:9F:9F:6E:1D:4D:41:E5:8E:D6:DE:3D:02:85
Alias name: verisignc3g4.pem
MD5: 3A:52:E1:E7:FD:6F:3A:E3:6F:F3:6F:99:1B:F9:22:41
```

```
SHA256:  
69:DD:D7:EA:90:BB:57:C9:3E:13:5D:C8:5E:A6:FC:D5:48:0B:60:32:39:BD:C4:54:FC:75:8B:2A:26:CF:7F:79  
Alias name: swisssigngoldg2ca  
MD5: 24:77:D9:A8:91:D1:3B:FA:88:2D:C2:FF:F8:CD:33:93  
SHA256:  
62:DD:0B:E9:B9:F5:0A:16:3E:A0:F8:E7:5C:05:3B:1E:CA:57:EA:55:C8:68:8F:64:7C:68:81:F2:C8:35:7B:95  
Alias name: mozillacert47.pem  
MD5: ED:41:F5:8C:50:C5:2B:9C:73:E6:EE:6C:EB:C2:A8:26  
SHA256:  
E4:C7:34:30:D7:A5:B5:09:25:DF:43:37:0A:0D:21:6E:9A:79:B9:D6:DB:83:73:A0:C6:9E:B1:CC:31:C7:C5:2A  
Alias name: mozillacert80.pem  
MD5: 64:B0:09:55:CF:B1:D5:99:E2:BE:13:AB:A6:5D:EA:4D  
SHA256:  
BD:71:FD:F6:DA:97:E4:CF:62:D1:64:7A:DD:25:81:B0:7D:79:AD:F8:39:7E:B4:EC:BA:9C:5E:84:88:82:14:23  
Alias name: mozillacert98.pem  
MD5: 43:5E:88:D4:7D:1A:4A:7E:FD:84:2E:52:EB:01:D4:6F  
SHA256:  
3E:84:BA:43:42:90:85:16:E7:75:73:CO:99:2F:09:79:CA:08:4E:46:85:68:1F:F1:95:CC:BA:8A:22:9B:8A:76  
Alias name: mozillacert144.pem  
MD5: A3:EC:75:0F:2E:88:DF:FA:48:01:4E:0B:5C:48:6F:FB  
SHA256:  
79:08:B4:03:14:C1:38:10:0B:51:8D:07:35:80:7F:FB:FC:F8:51:8A:00:95:33:71:05:BA:38:6B:15:3D:D9:27  
Alias name: starfieldclass2ca  
MD5: 32:4A:4B:BB:C8:63:69:9B:BE:74:9A:C6:DD:1D:46:24  
SHA256:  
14:65:FA:20:53:97:B8:76:FA:A6:F0:A9:95:8E:55:90:E4:0F:CC:7F:AA:4F:B7:C2:C8:67:75:21:FB:5F:B6:58  
Alias name: mozillacert36.pem  
MD5: F0:96:B6:2F:C5:10:D5:67:8E:83:25:32:E8:5E:2E:E5  
SHA256:  
32:7A:3D:76:1A:BA:DE:A0:34:EB:99:84:06:27:5C:B1:A4:77:6E:FD:AE:2F:DF:6D:01:68:EA:1C:4F:55:67:D0  
Alias name: mozillacert3.pem  
MD5: 39:16:AA:B9:6A:41:E1:14:69:DF:9E:6C:3B:72:DC:B6  
SHA256:  
39:DF:7B:68:2B:7B:93:8F:84:71:54:81:CC:DE:8D:60:D8:F2:2E:C5:98:87:7D:0A:AA:C1:2B:59:18:2B:03:12  
Alias name: globalsignr2ca  
MD5: 94:14:77:7E:3E:5E:FD:8F:30:BD:41:B0:CF:E7:D0:30  
SHA256:  
CA:42:DD:41:74:5F:D0:B8:1E:B9:02:36:2C:F9:D8:BF:71:9D:A1:BD:1B:1E:FC:94:6F:5B:4C:99:F4:2C:1B:9E  
Alias name: mozillacert87.pem  
MD5: 6C:39:7D:A4:0E:55:59:B2:3F:D6:41:B1:12:50:DE:43  
SHA256:  
51:3B:2C:EC:B8:10:D4:CD:E5:DD:85:39:1A:DF:C6:C2:DD:60:D8:7B:B7:36:D2:B5:21:48:4A:A4:7A:0E:BE:F6  
Alias name: mozillacert133.pem  
MD5: D6:ED:3C:CA:E2:66:0F:AF:10:43:0D:77:9B:04:09:BF  
SHA256:  
7D:3B:46:5A:60:14:E5:26:C0:AF:FC:EE:21:27:D2:31:17:27:AD:81:1C:26:84:2D:00:6A:F3:73:06:CC:80:BD  
Alias name: mozillacert25.pem  
MD5: CB:17:E4:31:67:3E:E2:09:FE:45:57:93:F3:0A:FA:1C  
SHA256:  
9A:CF:AB:7E:43:C8:D8:80:D0:6B:26:2A:94:DE:EE:E4:B4:65:99:89:C3:D0:CA:F1:9B:AF:64:05:E4:1A:B7:DF  
Alias name: verisignclass1g2ca  
MD5: DB:23:3D:F9:69:FA:4B:B9:95:80:44:73:5E:7D:41:83  
SHA256:  
34:1D:E9:8B:13:92:AB:F7:F4:AB:90:A9:60:CF:25:D4:BD:6E:C6:5B:9A:51:CE:6E:D0:67:D0:0E:C7:CE:9B:7F  
Alias name: mozillacert76.pem  
MD5: 82:92:BA:5B:EF:CD:8A:6F:A6:3D:55:F9:84:F6:D6:B7  
SHA256:  
03:76:AB:1D:54:C5:F9:80:3C:E4:B2:E2:01:A0:EE:7E:EF:7B:57:B6:36:E8:A9:3C:9B:8D:48:60:C9:6F:5F:A7  
Alias name: mozillacert122.pem  
MD5: 1D:35:54:04:85:78:B0:3F:42:42:4D:BF:20:73:0A:3F  
SHA256:  
68:7F:A4:51:38:22:78:FF:F0:C8:B1:1F:8D:43:D5:76:67:1C:6E:B2:BC:EA:B4:13:FB:83:D9:65:D0:6D:2F:F2  
Alias name: mozillacert14.pem  
MD5: D4:74:DE:57:5C:39:B2:D3:9C:85:83:C5:C0:65:49:8A  
SHA256:  
74:31:E5:F4:C3:C1:CE:46:90:77:4F:0B:61:E0:54:40:88:3B:A9:A0:1E:D0:0B:A6:AB:D7:80:6E:D3:B1:18:CF
```

```
Alias name: equifaxsecureca
MD5: 67:CB:9D:C0:13:24:8A:82:9B:B2:17:1E:D1:1B:EC:D4
SHA256:
08:29:7A:40:47:DB:A2:36:80:C7:31:DB:6E:31:76:53:CA:78:48:E1:BE:BD:3A:0B:01:79:A7:07:F9:2C:F1:78
Alias name: mozillacert65.pem
MD5: A2:6F:53:B7:EE:40:DB:4A:68:E7:FA:18:D9:10:4B:72
SHA256:
BC:23:F9:8A:31:3C:B9:2D:E3:BB:FC:3A:5A:9F:44:61:AC:39:49:4C:4A:E1:5A:9E:9D:F1:31:E9:9B:73:01:9A
Alias name: mozillacert111.pem
MD5: F9:03:7E:CF:E6:9E:3C:73:7A:2A:90:07:69:FF:2B:96
SHA256:
59:76:90:07:F7:68:5D:0F:CD:50:87:2F:9F:95:D5:75:5A:5B:2B:45:7D:81:F3:69:2B:61:0A:98:67:2F:0E:1B
Alias name: certumtrustednetworkca
MD5: D5:E9:81:40:C5:18:69:FC:46:2C:89:75:62:0F:AA:78
SHA256:
5C:58:46:8D:55:F5:8E:49:7E:74:39:82:D2:B5:00:10:B6:D1:65:37:4A:CF:83:A7:D4:A3:2D:B7:68:C4:40:8E
Alias name: mozillacert129.pem
MD5: 92:65:58:8B:A2:1A:31:72:73:68:5C:B4:A5:7A:07:48
SHA256:
A0:45:9B:9F:63:B2:25:59:F5:FA:5D:4C:6D:B3:F9:F7:2F:F1:93:42:03:35:78:F0:73:BF:1D:1B:46:CB:B9:12
Alias name: mozillacert54.pem
MD5: B5:E8:34:36:C9:10:44:58:48:70:6D:2E:83:D4:B8:05
SHA256:
B4:78:B8:12:25:0D:F8:78:63:5C:2A:A7:EC:7D:15:5E:AA:62:5E:E8:29:16:E2:CD:29:43:61:88:6C:D1:FB:D4
Alias name: mozillacert100.pem
MD5: CD:E0:25:69:8D:47:AC:9C:89:35:90:F7:FD:51:3D:2F
SHA256:
49:E7:A4:42:AC:F0:EA:62:87:05:00:54:B5:25:64:B6:50:E4:F4:9E:42:E3:48:D6:AA:38:E0:39:E9:57:B1:C1
Alias name: mozillacert118.pem
MD5: 8F:5D:77:06:27:C4:98:3C:5B:93:78:E7:D7:7D:9B:CC
SHA256:
5F:OB:62:EA:B5:E3:53:EA:65:21:65:16:58:FB:B6:53:59:F4:43:28:0A:4A:FB:D1:04:D7:7D:10:F9:F0:4C:07
Alias name: gd-class2-root.pem
MD5: 91:DE:06:25:AB:DA:FD:32:17:0C:BB:25:17:2A:84:67
SHA256:
C3:84:6B:F2:4B:9E:93:CA:64:27:4C:0E:C6:7C:1E:CC:5E:02:4F:FC:AC:D2:D7:40:19:35:0E:81:FE:54:6A:E4
Alias name: mozillacert151.pem
MD5: 86:38:6D:5E:49:63:6C:85:5C:DB:6D:DC:94:B7:D0:F7
SHA256:
7F:12:CD:5F:7E:5E:29:0E:C7:D8:51:79:D5:B7:2C:20:A5:BE:75:08:FF:DB:5B:F8:1A:B9:68:4A:7F:C9:F6:67
Alias name: thawteprimaryrootcag3
MD5: FB:1B:5D:43:8A:94:CD:44:C6:76:F2:43:4B:47:E7:31
SHA256:
4B:03:F4:58:07:AD:70:F2:1B:FC:2C:AE:71:C9:FD:E4:60:4C:06:4C:F5:FF:B6:86:BA:E5:DB:AA:D7:FD:D3:4C
Alias name: quovaldisrootca
MD5: 27:DE:36:FE:72:B7:00:03:00:9D:F4:F0:1E:6C:04:24
SHA256:
A4:5E:DE:3B:BB:F0:9C:8A:E1:5C:72:EF:CO:72:68:D6:93:A2:1C:99:6F:D5:1E:67:CA:07:94:60:FD:6D:88:73
Alias name: thawteprimaryrootcag2
MD5: 74:9D:EA:60:24:C4:FD:22:53:3E:CC:3A:72:D9:29:4F
SHA256:
A4:31:0D:50:AF:18:A6:44:71:90:37:2A:86:AF:AF:8B:95:1F:FB:43:1D:83:7F:1E:56:88:B4:59:71:ED:15:57
Alias name: deprecateditsecca
MD5: A5:96:0C:F6:B5:AB:27:E5:01:C6:00:88:9E:60:33:E5
SHA256:
9A:59:DA:86:24:1A:FD:BA:A3:39:FA:9C:FD:21:6A:0B:06:69:4D:E3:7E:37:52:6B:BE:63:C8:BC:83:74:2E:CB
Alias name: entrustrootcag2
MD5: 4B:E2:C9:91:96:65:0C:F4:0E:5A:93:92:A0:0A:FE:B2
SHA256:
43:DF:57:74:B0:3E:7F:EF:5F:E4:0D:93:1A:7B:ED:F1:BB:2E:6B:42:73:8C:4E:6D:38:41:10:3D:3A:A7:F3:39
Alias name: mozillacert43.pem
MD5: 40:01:25:06:8D:21:43:6A:0E:43:00:9C:E7:43:F3:D5
SHA256:
50:79:41:C7:44:60:A0:B4:70:86:22:0D:4E:99:32:57:2A:B5:D1:B5:BB:CB:89:80:AB:1C:B1:76:51:A8:44:D2
Alias name: mozillacert107.pem
MD5: BE:EC:11:93:9A:F5:69:21:BC:D7:C1:C0:67:89:CC:2A
```

```
SHA256:  
F9:6F:23:F4:C3:E7:9C:07:7A:46:98:8D:5A:F5:90:06:76:A0:F0:39:CB:64:5D:D1:75:49:B2:16:C8:24:40:CE  
Alias name: trustcenterclass4caii  
MD5: 9D:FB:F9:AC:ED:89:33:22:F4:28:48:83:25:23:5B:E0  
SHA256:  
32:66:96:7E:59:CD:68:00:8D:9D:D3:20:81:11:85:C7:04:20:5E:8D:95:FD:D8:4F:1C:7B:31:1E:67:04:FC:32  
Alias name: mozillacert94.pem  
MD5: 46:A7:D2:FE:45:FB:64:5A:A8:59:90:9B:78:44:9B:29  
SHA256:  
9A:11:40:25:19:7C:5B:B9:5D:94:E6:3D:55:CD:43:79:08:47:B6:46:B2:3C:DF:11:AD:A4:A0:0E:FF:15:FB:48  
Alias name: mozillacert140.pem  
MD5: 5E:39:7B:DD:F8:BA:EC:82:E9:AC:62:BA:0C:54:00:2B  
SHA256:  
85:A0:DD:7D:D7:20:AD:B7:FF:05:F8:3D:54:2B:20:9D:C7:FF:45:28:F7:D6:77:B1:83:89:FE:A5:E5:C4:9E:86  
Alias name: ttelesecglobalrootclass3ca  
MD5: CA:FB:40:A8:4E:39:92:8A:1D:FE:8E:2F:C4:27:EA:EF  
SHA256:  
FD:73:DA:D3:1C:64:4F:F1:B4:3B:EF:0C:CD:DA:96:71:0B:9C:D9:87:5E:CA:7E:31:70:7A:F3:E9:6D:52:2B:BD  
Alias name: amzninternalcorcpa  
MD5: 7B:0E:9D:67:A9:3A:88:DD:BA:81:8D:A9:3C:74:AA:BB  
SHA256:  
01:29:04:6C:60:EF:5C:51:60:D3:9F:A2:3A:1D:0C:52:0A:AF:DA:4F:17:87:95:AA:66:82:01:9F:76:C9:11:DC  
Alias name: starfieldservicesrootg2ca  
MD5: 17:35:74:AF:7B:61:1C:EB:F4:F9:3C:E2:EE:40:F9:A2  
SHA256:  
56:8D:69:05:A2:C8:87:08:A4:B3:02:51:90:ED:CF:ED:B1:97:4A:60:6A:13:C6:E5:29:0F:CB:2A:E6:3E:DA:B5  
Alias name: mozillacert32.pem  
MD5: 0C:7F:DD:6A:F4:2A:B9:C8:9B:BD:20:7E:A9:DB:5C:37  
SHA256:  
B9:BE:A7:86:0A:96:2E:A3:61:1D:AB:97:AB:6D:A3:E2:1C:10:68:B9:7D:55:57:5E:D0:E1:12:79:C1:1C:89:32  
Alias name: mozillacert83.pem  
MD5: 2C:8C:17:5E:B1:54:AB:93:17:B5:36:5A:DB:D1:C6:F2  
SHA256:  
8C:4E:DF:D0:43:48:F3:22:96:9E:7E:29:A4:CD:4D:CA:00:46:55:06:1C:16:E1:B0:76:42:2E:F3:42:AD:63:0E  
Alias name: verisignroot.pem  
MD5: 8E:AD:B5:01:AA:4D:81:E4:8C:1D:D1:E1:14:00:95:19  
SHA256:  
23:99:56:11:27:A5:71:25:DE:8C:EF:EA:61:0D:DF:2F:A0:78:B5:C8:06:7F:4E:82:82:90:BF:B8:60:E8:4B:3C  
Alias name: mozillacert147.pem  
MD5: B3:A5:3E:77:21:6D:AC:4A:C0:C9:FB:D5:41:3D:CA:06  
SHA256:  
85:FB:2F:91:DD:12:27:5A:01:45:B6:36:53:4F:84:02:4A:D6:8B:69:B8:EE:88:68:4F:F7:11:37:58:05:B3:48  
Alias name: camerfirmachambersca  
MD5: 5E:80:9E:84:5A:0E:65:0B:17:02:F3:55:18:2A:3E:D7  
SHA256:  
06:3E:4A:FA:C4:91:DF:D3:32:F3:08:9B:85:42:E9:46:17:D8:93:D7:FE:94:4E:10:A7:93:7E:E2:9D:96:93:C0  
Alias name: mozillacert21.pem  
MD5: E0:06:A1:C9:7D:CF:C9:FC:0D:C0:56:75:96:D8:62:13  
SHA256:  
BE:6C:4D:A2:BB:B9:BA:59:B6:F3:93:97:68:37:42:46:C3:C0:05:99:3F:A9:8F:02:0D:1D:ED:BE:D4:8A:81:D5  
Alias name: mozillacert39.pem  
MD5: CE:78:33:5C:59:78:01:6E:18:EA:B9:36:A0:B9:2E:23  
SHA256:  
E6:B8:F8:76:64:85:F8:07:AE:7F:8D:AC:16:70:46:1F:07:C0:A1:3E:EF:3A:1F:F7:17:53:8D:7A:BA:D3:91:B4  
Alias name: mozillacert6.pem  
MD5: 91:DE:06:25:AB:DA:FD:32:17:0C:BB:25:17:2A:84:67  
SHA256:  
C3:84:6B:F2:4B:9E:93:CA:64:27:4C:0E:C6:7C:1E:CC:5E:02:4F:FC:AC:D2:D7:40:19:35:0E:81:FE:54:6A:E4  
Alias name: verisignuniversalrootca  
MD5: 8E:AD:B5:01:AA:4D:81:E4:8C:1D:D1:E1:14:00:95:19  
SHA256:  
23:99:56:11:27:A5:71:25:DE:8C:EF:EA:61:0D:DF:2F:A0:78:B5:C8:06:7F:4E:82:82:90:BF:B8:60:E8:4B:3C  
Alias name: mozillacert72.pem  
MD5: 80:3A:BC:22:C1:E6:FB:8D:9B:3B:27:4A:32:1B:9A:01  
SHA256:  
45:14:0B:32:47:EB:9C:C8:C5:B4:F0:D7:B5:30:91:F7:32:92:08:9E:6E:5A:63:E2:74:9D:D3:AC:A9:19:8E:DA
```

```
Alias name: geotrustuniversalca
MD5: 92:65:58:8B:A2:1A:31:72:73:68:5C:B4:A5:7A:07:48
SHA256:
A0:45:9B:9F:63:B2:25:59:F5:FA:5D:4C:6D:B3:F9:F7:2F:F1:93:42:03:35:78:F0:73:BF:1D:1B:46:CB:B9:12
Alias name: mozillacert136.pem
MD5: 49:79:04:B0:EB:87:19:AC:47:BO:BC:11:51:9B:74:D0
SHA256:
D7:A7:A0:FB:5D:7E:27:31:D7:71:E9:48:4E:BC:DE:F7:1D:5F:0C:3E:0A:29:48:78:2B:C8:3E:E0:EA:69:9E:F4
Alias name: mozillacert10.pem
MD5: F8:38:7C:77:88:DF:2C:16:68:2E:C2:E2:52:4B:B8:F9
SHA256:
21:DB:20:12:36:60:BB:2E:D4:18:20:5D:A1:1E:E7:A8:5A:65:E2:BC:6E:55:B5:AF:7E:78:99:C8:A2:66:D9:2E
Alias name: mozillacert28.pem
MD5: 5C:48:DC:F7:42:72:EC:56:94:6D:1C:CC:71:35:80:75
SHA256:
0C:2C:D6:3D:F7:80:6F:A3:99:ED:E8:09:11:6B:57:5B:F8:79:89:F0:65:18:F9:80:8C:86:05:03:17:8B:AF:66
Alias name: affirmtrustnetworkingca
MD5: 42:65:CA:BE:01:9A:9A:4C:A9:8C:41:49:CD:C0:D5:7F
SHA256:
0A:81:EC:5A:92:97:77:F1:45:90:4A:F3:8D:5D:50:9F:66:B5:E2:C5:8F:CD:B5:31:05:8B:0E:17:F3:F0:B4:1B
Alias name: mozillacert61.pem
MD5: 42:81:A0:E2:1C:E3:55:10:DE:55:89:42:65:96:22:E6
SHA256:
03:95:0F:B4:9A:53:1F:3E:19:91:94:23:98:DF:A9:E0:EA:32:D7:BA:1C:DD:9B:C8:5D:B5:7E:D9:40:OB:43:4A
Alias name: mozillacert79.pem
MD5: C4:5D:0E:48:B6:AC:28:30:4E:0A:BC:F9:38:16:87:57
SHA256:
70:A7:3F:7F:37:6B:60:07:42:48:90:45:34:B1:14:82:D5:BF:0E:69:8E:CC:49:8D:F5:25:77:EB:F2:E9:3B:9A
Alias name: affirmtrustcommercialca
MD5: 82:92:BA:5B:EF:CD:8A:6F:A6:3D:55:F9:84:F6:D6:B7
SHA256:
03:76:AB:1D:54:C5:F9:80:3C:E4:B2:E2:01:A0:EE:7E:EF:7B:57:B6:36:E8:A9:3C:9B:8D:48:60:C9:6F:5F:A7
Alias name: mozillacert125.pem
MD5: D6:A5:C3:ED:5D:DD:3E:00:C1:3D:87:92:1F:1D:3F:E4
SHA256:
73:C1:76:43:4F:1B:C6:D5:AD:F4:5B:0E:76:E7:27:28:7C:8D:E5:76:16:C1:E6:E6:14:1A:2B:2C:BC:7D:8E:4C
Alias name: mozillacert17.pem
MD5: 21:D8:4C:82:2B:99:09:33:A2:EB:14:24:8D:8E:5F:E8
SHA256:
76:7C:95:5A:76:41:2C:89:AF:68:8E:90:A1:C7:0F:55:6C:FD:6B:60:25:DB:EA:10:41:6D:7E:B6:83:1F:8C:40
Alias name: mozillacert50.pem
MD5: 2C:20:26:9D:CB:1A:4A:00:85:B5:B7:5A:AE:C2:01:37
SHA256:
35:AE:5B:DD:D8:F7:AE:63:5C:FF:BA:56:82:A8:F0:0B:95:F4:84:62:C7:10:8E:E9:A0:E5:29:2B:07:4A:AF:B2
Alias name: mozillacert68.pem
MD5: 73:3A:74:7A:EC:BB:A3:96:A6:C2:E4:E2:C8:9B:C0:C3
SHA256:
04:04:80:28:BF:1F:28:64:D4:8F:9A:D4:D8:32:94:36:6A:82:88:56:55:3F:3B:14:30:3F:90:14:7F:5D:40:EF
Alias name: starfieldrootg2ca
MD5: D6:39:81:C6:52:7E:96:69:FC:FC:CA:66:ED:05:F2:96
SHA256:
2C:E1:CB:0B:F9:D2:F9:E1:02:99:3F:BE:21:51:52:C3:B2:DD:0C:AB:DE:1C:68:E5:31:9B:83:91:54:DB:B7:F5
Alias name: mozillacert114.pem
MD5: B8:A1:03:63:B0:BD:21:71:70:8A:6F:13:3A:BB:79:49
SHA256:
B0:BF:D5:2B:B0:D7:D9:BD:92:BF:5D:4D:C1:3D:A2:55:C0:2C:54:2F:37:83:65:EA:89:39:11:F5:5E:55:F2:3C
Alias name: buypassclass3ca
MD5: 3D:3B:18:9E:2C:64:5A:E8:D5:88:CE:0E:F9:37:C2:EC
SHA256:
ED:F7:EB:BC:A2:7A:2A:38:4D:38:7B:7D:40:10:C6:66:E2:ED:B4:84:3E:4C:29:B4:AE:1D:5B:93:32:E6:B2:4D
Alias name: mozillacert57.pem
MD5: A8:0D:6F:39:78:B9:43:6D:77:42:6D:98:5A:CC:23:CA
SHA256:
F9:E6:7D:33:6C:51:00:2A:C0:54:C6:32:02:2D:66:DD:A2:E7:E3:FF:F1:0A:D0:61:ED:31:D8:BB:B4:10:CF:B2
Alias name: verisignc2g3.pem
MD5: F8:BE:C4:63:22:C9:A8:46:74:8B:B8:1D:1E:4A:2B:F6
```

```
SHA256:  
92:A9:D9:83:3F:E1:94:4D:B3:66:E8:BF:AE:7A:95:B6:48:0C:2D:6C:6C:2A:1B:E6:5D:42:36:B6:08:FC:A1:BB  
Alias name: verisignclass2g3ca  
MD5: F8:BE:C4:63:22:C9:A8:46:74:8B:B8:1D:1E:4A:2B:F6  
SHA256:  
92:A9:D9:83:3F:E1:94:4D:B3:66:E8:BF:AE:7A:95:B6:48:0C:2D:6C:6C:2A:1B:E6:5D:42:36:B6:08:FC:A1:BB  
Alias name: mozillacert103.pem  
MD5: E6:24:E9:12:01:AE:0C:DE:8E:85:C4:CE:A3:12:DD:EC  
SHA256:  
3C:FC:3C:14:D1:F6:84:FF:17:E3:8C:43:CA:44:0C:00:B9:67:EC:93:3E:8B:FE:06:4C:A1:D7:2C:90:F2:AD:B0  
Alias name: mozillacert90.pem  
MD5: 69:C1:0D:4F:07:A3:1B:C3:FE:56:3D:04:BC:11:F6:A6  
SHA256:  
55:92:60:84:EC:96:3A:64:B9:6E:2A:BE:01:CE:OB:A8:6A:64:FB:FE:BC:C7:AA:B5:AF:C1:55:B3:7F:D7:60:66  
Alias name: verisignc3g3.pem  
MD5: CD:68:B6:A7:C7:C4:CE:75:E0:1D:4F:57:44:61:92:09  
SHA256:  
EB:04:CF:5E:B1:F3:9A:FA:76:2F:2B:B1:20:F2:96:CB:A5:20:C1:B9:7D:B1:58:95:65:B8:1C:B9:A1:7B:72:44  
Alias name: mozillacert46.pem  
MD5: AA:8E:5D:D9:F8:DB:0A:58:B7:8D:26:87:6C:82:35:55  
SHA256:  
EC:C3:E9:C3:40:75:03:BE:E0:91:AA:95:2F:41:34:8F:F8:8B:AA:86:3B:22:64:BE:FA:C8:07:90:15:74:E9:39  
Alias name: godaddyclass2ca  
MD5: 91:DE:06:25:AB:DA:FD:32:17:0C:BB:25:17:2A:84:67  
SHA256:  
C3:84:6B:F2:4B:9E:93:CA:64:27:4C:0E:C6:7C:1E:CC:5E:02:4F:FC:AC:D2:D7:40:19:35:0E:81:FE:54:6A:E4  
Alias name: verisignc4g3.pem  
MD5: DB:C8:F2:27:2E:B1:EA:6A:29:23:5D:FE:56:3E:33:DF  
SHA256:  
E3:89:36:0D:0F:DB:AE:B3:D2:50:58:4B:47:30:31:4E:22:2F:39:C1:56:A0:20:14:4E:8D:96:05:61:79:15:06  
Alias name: mozillacert97.pem  
MD5: A2:33:9B:4C:74:78:73:D4:6C:E7:C1:F3:8D:CB:5C:E9  
SHA256:  
83:CE:3C:12:29:68:8A:59:3D:48:5F:81:97:3C:0F:91:95:43:1E:DA:37:CC:5E:36:43:0E:79:C7:A8:88:63:8B  
Alias name: mozillacert143.pem  
MD5: F1:BC:63:6A:54:E0:B5:27:F5:CD:E7:1A:E3:4D:6E:4A  
SHA256:  
E7:5E:72:ED:9F:56:0E:EC:6E:B4:80:00:73:A4:3F:C3:AD:19:19:5A:39:22:82:01:78:95:97:4A:99:02:6B:6C  
Alias name: mozillacert35.pem  
MD5: 3F:45:96:39:E2:50:87:F7:BB:FE:98:0C:3C:20:98:E6  
SHA256:  
92:BF:51:19:AB:EC:CA:D0:B1:33:2D:C4:E1:D0:5F:BA:75:B5:67:90:44:EE:0C:A2:6E:93:1F:74:4F:2F:33:CF  
Alias name: mozillacert2.pem  
MD5: 3A:52:E1:E7:FD:6F:3A:E3:6F:F3:6F:99:1B:F9:22:41  
SHA256:  
69:DD:D7:EA:90:BB:57:C9:3E:13:5D:C8:5E:A6:FC:D5:48:0B:60:32:39:BD:C4:54:FC:75:8B:2A:26:CF:7F:79  
Alias name: utnuserfirstobjectca  
MD5: A7:F2:E4:16:06:41:11:50:30:6B:9C:E3:B4:9C:B0:C9  
SHA256:  
6F:FF:78:E4:00:A7:0C:11:01:1C:D8:59:77:C4:59:FB:5A:F9:6A:3D:F0:54:08:20:D0:F4:B8:60:78:75:E5:8F  
Alias name: mozillacert86.pem  
MD5: 10:FC:63:5D:F6:26:3E:0D:F3:25:BE:5F:79:CD:67:67  
SHA256:  
E7:68:56:34:EF:AC:F6:9A:CE:93:9A:6B:25:5B:7B:4F:AB:EF:42:93:5B:50:A2:65:AC:B5:CB:60:27:E4:4E:70  
Alias name: mozillacert132.pem  
MD5: 14:F1:08:AD:9D:FA:64:E2:89:E7:1C:CF:A8:AD:7D:5E  
SHA256:  
77:40:73:12:C6:3A:15:3D:5B:C0:0B:4E:51:75:9C:DF:DA:C2:37:DC:2A:33:B6:79:46:E9:8E:9B:FA:68:0A:E3  
Alias name: addtrustclass1ca  
MD5: 1E:42:95:02:33:92:6B:B9:5F:C0:7F:DA:D6:B2:4B:FC  
SHA256:  
8C:72:09:27:9A:C0:4E:27:5E:16:D0:7F:D3:B7:75:E8:01:54:B5:96:80:46:E3:1F:52:DD:25:76:63:24:E9:A7  
Alias name: mozillacert24.pem  
MD5: 7C:A5:0F:F8:5B:9A:7D:6D:30:AE:54:5A:E3:42:A2:8A  
SHA256:  
66:8C:83:94:7D:A6:3B:72:4B:EC:E1:74:3C:31:A0:E6:AE:D0:DB:8E:C5:B3:1B:E3:77:BB:78:4F:91:B6:71:6F
```

```
Alias name: verisignc1g3.pem
MD5: B1:47:BC:18:57:D1:18:A0:78:2D:EC:71:E8:2A:95:73
SHA256:
CB:B5:AF:18:5E:94:2A:24:02:F9:EA:CB:C0:ED:5B:B8:76:EE:A3:C1:22:36:23:D0:04:47:E4:F3:BA:55:4B:65
Alias name: mozillacert9.pem
MD5: 37:85:44:53:32:45:1F:20:F0:F3:95:E1:25:C4:43:4E
SHA256:
76:00:29:5E:EF:E8:5B:9E:1F:D6:24:DB:76:06:2A:AA:AE:59:81:8A:54:D2:77:4C:D4:C0:B2:C0:11:31:E1:B3
Alias name: amzninternalrootca
MD5: 08:09:73:AC:E0:78:41:7C:0A:26:33:51:E8:CF:E6:60
SHA256:
OE:DE:63:C1:DC:7A:E8:11:F1:AB:BC:05:4F:59:EE:49:9D:62:9A:2F:DE:9C:A7:16:32:A2:64:29:3E:8B:66:AA
Alias name: mozillacert75.pem
MD5: 67:CB:9D:C0:13:24:8A:82:9B:B2:17:1E:D1:1B:EC:D4
SHA256:
08:29:7A:40:47:DB:A2:36:80:C7:31:DB:6E:31:76:53:CA:78:48:E1:BE:BD:3A:0B:01:79:A7:07:F9:2C:F1:78
Alias name: entrustevca
MD5: D6:A5:C3:ED:5D:DD:3E:00:C1:3D:87:92:1F:1D:3F:E4
SHA256:
73:C1:76:43:4F:1B:C6:D5:AD:F4:5B:0E:76:E7:27:28:7C:8D:E5:76:16:C1:E6:E6:14:1A:2B:2C:BC:7D:8E:4C
Alias name: secomsrootca2
MD5: 6C:39:7D:A4:0E:55:59:B2:3F:D6:41:B1:12:50:DE:43
SHA256:
51:3B:2C:EC:B8:10:D4:CD:E5:DD:85:39:1A:DF:C6:C2:DD:60:D8:7B:B7:36:D2:B5:21:48:4A:A4:7A:0E:BE:F6
Alias name: camerfirmachambersignca
MD5: 9E:80:FF:78:01:0C:2E:C1:36:BD:FE:96:90:6E:08:F3
SHA256:
13:63:35:43:93:34:A7:69:80:16:A0:D3:24:DE:72:28:4E:07:9D:7B:52:20:BB:8F:BD:74:78:16:EE:BE:BA:CA
Alias name: secomsrootca1
MD5: F1:BC:63:6A:54:E0:B5:27:F5:CD:E7:1A:E3:4D:6E:4A
SHA256:
E7:5E:72:ED:9F:56:0E:EC:6E:B4:80:00:73:A4:3F:C3:AD:19:19:5A:39:22:82:01:78:95:97:4A:99:02:6B:6C
Alias name: mozillacert121.pem
MD5: 1E:42:95:02:33:92:6B:B9:5F:CO:7F:DA:D6:B2:4B:FC
SHA256:
8C:72:09:27:9A:C0:4E:27:5E:16:D0:7F:D3:B7:75:E8:01:54:B5:96:80:46:E3:1F:52:DD:25:76:63:24:E9:A7
Alias name: mozillacert139.pem
MD5: 27:DE:36:FE:72:B7:00:03:00:9D:F4:F0:1E:6C:04:24
SHA256:
A4:5E:DE:3B:BB:F0:9C:8A:E1:5C:72:EF:C0:72:68:D6:93:A2:1C:99:6F:D5:1E:67:CA:07:94:60:FD:6D:88:73
Alias name: mozillacert13.pem
MD5: C5:A1:B7:FF:73:DD:D6:D7:34:32:18:DF:FC:3C:AD:88
SHA256:
6C:61:DA:C3:A2:DE:F0:31:50:6B:E0:36:D2:A6:FE:40:19:94:FB:D1:3D:F9:C8:D4:66:59:92:74:C4:46:EC:98
Alias name: mozillacert64.pem
MD5: 06:9F:69:79:16:66:90:02:1B:8C:8C:A2:C3:07:6F:3A
SHA256:
AB:70:36:36:5C:71:54:AA:29:C2:C2:9F:5D:41:91:16:3B:16:2A:22:25:01:13:57:D5:6D:07:FF:A7:BC:1F:72
Alias name: mozillacert110.pem
MD5: D0:A0:5A:EE:05:B6:09:94:21:A1:7D:F1:B2:29:82:02
SHA256:
9A:6E:C0:12:E1:A7:DA:9D:BE:34:19:4D:47:8A:D7:C0:DB:18:22:FB:07:1D:F1:29:81:49:6E:D1:04:38:41:13
Alias name: mozillacert128.pem
MD5: 0E:40:A7:6C:DE:03:5D:8F:D1:0F:E4:D1:8D:F9:6C:A9
SHA256:
CA:2D:82:A0:86:77:07:2F:8A:B6:76:4F:F0:35:67:6C:FE:3E:5E:32:5E:01:21:72:DF:3F:92:09:6D:B7:9B:85
Alias name: entrust2048ca
MD5: EE:29:31:BC:32:7E:9A:E6:E8:B5:F7:51:B4:34:71:90
SHA256:
6D:C4:71:72:E0:1C:BC:B0:BF:62:58:0D:89:5F:E2:B8:AC:9A:D4:F8:73:80:1E:0C:10:B9:C8:37:D2:1E:B1:77
Alias name: mozillacert53.pem
MD5: 7E:23:4E:5B:A7:A5:B4:25:E9:00:07:74:11:62:AE:D6
SHA256:
2D:47:43:7D:E1:79:51:21:5A:12:F3:C5:8E:51:C7:29:A5:80:26:EF:1F:CC:0A:5F:B3:D9:DC:01:2F:60:0D:19
Alias name: mozillacert117.pem
MD5: AC:B6:94:A5:9C:17:E0:D7:91:52:9B:B1:97:06:A6:E4
```

```
SHA256:  
16:AF:57:A9:F6:76:B0:AB:12:60:95:AA:5E:BA:DE:F2:2A:B3:11:19:D6:44:AC:95:CD:4B:93:DB:F3:F2:6A:EB  
Alias name: mozillacert150.pem  
MD5: C5:E6:7B:BF:06:D0:4F:43:ED:C4:7A:65:8A:FB:6B:19  
SHA256:  
EF:3C:B4:17:FC:8E:BF:6F:97:87:6C:9E:4E:CE:39:DE:1E:A5:FE:64:91:41:D1:02:8B:7D:11:C0:B2:29:8C:ED  
Alias name: thawteserverca  
MD5: EE:FE:61:69:65:6E:F8:9C:C6:2A:F4:D7:2B:63:EF:A2  
SHA256:  
87:C6:78:BF:B8:B2:5F:38:F7:E9:7B:33:69:56:BB:CF:14:4B:BA:CA:A5:36:47:E6:1A:23:25:BC:10:55:31:6B  
Alias name: secomvalicertclass1ca  
MD5: 65:58:AB:15:AD:57:6C:1E:A8:A7:B5:69:AC:BF:FF:EB  
SHA256:  
F4:C1:49:55:1A:30:13:A3:5B:C7:BF:FE:17:A7:F3:44:9B:C1:AB:5B:5A:0A:E7:4B:06:C2:3B:90:00:4C:01:04  
Alias name: mozillacert42.pem  
MD5: 74:01:4A:91:B1:08:C4:58:CE:47:CD:F0:DD:11:53:08  
SHA256:  
B6:19:1A:50:D0:C3:97:7F:7D:A9:9B:CD:AA:C8:6A:22:7D:AE:B9:67:9E:C7:OB:A3:B0:C9:D9:22:71:C1:70:D3  
Alias name: verisignc2g6.pem  
MD5: 7D:0B:83:E5:FB:7C:AD:07:4F:20:A9:B5:DF:63:ED:79  
SHA256:  
CB:62:7D:18:B5:8A:D5:6D:DE:33:1A:30:45:6B:C6:5C:60:1A:4E:9B:18:DE:DC:EA:08:E7:DA:AA:07:81:5F:F0  
Alias name: godaddyrootg2ca  
MD5: 80:3A:BC:22:C1:E6:FB:8D:9B:3B:27:4A:32:1B:9A:01  
SHA256:  
45:14:0B:32:47:EB:9C:C8:C5:B4:F0:D7:B5:30:91:F7:32:92:08:9E:6E:5A:63:E2:74:9D:D3:AC:A9:19:8E:DA  
Alias name: gtecybertrustglobalca  
MD5: CA:3D:D3:68:F1:03:5C:D0:32:FA:B8:2B:59:E8:5A:DB  
SHA256:  
A5:31:25:18:8D:21:10:AA:96:4B:02:C7:B7:C6:DA:32:03:17:08:94:E5:FB:71:FF:FB:66:67:D5:E6:81:0A:36  
Alias name: mozillacert106.pem  
MD5: 7B:30:34:9F:DD:0A:4B:6B:35:CA:31:51:28:5D:AE:EC  
SHA256:  
D9:5F:EA:3C:A4:EE:DC:E7:4C:D7:6E:75:FC:6D:1F:F6:2C:44:1F:0F:A8:BC:77:F0:34:B1:9E:5D:B2:58:01:5D  
Alias name: equifaxsecurebusinessca1  
MD5: 14:C0:08:E5:A3:85:03:A3:BE:78:E9:67:4F:27:CA:EE  
SHA256:  
2E:3A:2B:B5:11:25:05:83:6C:A8:96:8B:E2:CB:37:27:CE:9B:56:84:5C:6E:E9:8E:91:85:10:4A:FB:9A:F5:96  
Alias name: mozillacert93.pem  
MD5: 78:4B:FB:9E:64:82:0A:D3:B8:4C:62:F3:64:F2:90:64  
SHA256:  
C7:BA:65:67:DE:93:A7:98:AE:1F:AA:79:1E:71:2D:37:8F:AE:1F:93:C4:39:7F:EA:44:1B:B7:CB:E6:FD:59:95  
Alias name: quovaldisrootca3  
MD5: 31:85:3C:62:94:97:63:B9:AA:FD:89:4E:AF:6F:E0:CF  
SHA256:  
18:F1:FC:7F:20:5D:F8:AD:DD:EB:7F:E0:07:DD:57:E3:AF:37:5A:9C:4D:8D:73:54:6B:F4:F1:FE:D1:E1:8D:35  
Alias name: quovaldisrootca2  
MD5: 5E:39:7B:DD:F8:BA:EC:82:E9:AC:62:BA:0C:54:00:2B  
SHA256:  
85:A0:DD:7D:D7:20:AD:B7:FF:05:F8:3D:54:2B:20:9D:C7:FF:45:28:F7:D6:77:B1:83:89:FE:A5:E5:C4:9E:86  
Alias name: soneraaclass2ca  
MD5: A3:EC:75:0F:2E:88:DF:FA:48:01:4E:0B:5C:48:6F:FB  
SHA256:  
79:08:B4:03:14:C1:38:10:0B:51:8D:07:35:80:7F:FB:FC:F8:51:8A:00:95:33:71:05:BA:38:6B:15:3D:D9:27  
Alias name: mozillacert31.pem  
MD5: 7C:62:FF:74:9D:31:53:5E:68:4A:D5:78:AA:1E:BF:23  
SHA256:  
17:93:92:7A:06:14:54:97:89:AD:CE:2F:8F:34:F7:F0:B6:6D:0F:3A:E3:A3:B8:4D:21:EC:15:DB:BA:4F:AD:C7  
Alias name: mozillacert49.pem  
MD5: DF:3C:73:59:81:E7:39:50:81:04:4C:34:A2:CB:B3:7B  
SHA256:  
B7:B1:2B:17:1F:82:1D:AA:99:0C:D0:FE:50:87:B1:28:44:8B:A8:E5:18:4F:84:C5:1E:02:B5:C8:FB:96:2B:24  
Alias name: mozillacert82.pem  
MD5: 7F:30:78:8C:03:E3:CA:C9:0A:E2:C9:EA:1E:AA:55:1A  
SHA256:  
FC:BF:E2:88:62:06:F7:2B:27:59:3C:8B:07:02:97:E1:2D:76:9E:D1:0E:D7:93:07:05:A8:09:8E:FF:C1:4D:17
```

```
Alias name: mozillacert146.pem
MD5: 91:F4:03:55:20:A1:F8:63:2C:62:DE:AC:FB:61:1C:8E
SHA256:
48:98:C6:88:8C:0C:FF:B0:D3:E3:1A:CA:8A:37:D4:E3:51:5F:F7:46:D0:26:35:D8:66:46:CF:A0:A3:18:5A:E7
Alias name: baltimorecybertrustca
MD5: AC:B6:94:A5:9C:17:E0:D7:91:52:9B:B1:97:06:A6:E4
SHA256:
16:AF:57:A9:F6:76:B0:AB:12:60:95:AA:5E:BA:DE:F2:2A:B3:11:19:D6:44:AC:95:CD:4B:93:DB:F3:F2:6A:EB
Alias name: mozillacert20.pem
MD5: 24:77:D9:A8:91:D1:3B:FA:88:2D:C2:FF:F8:CD:33:93
SHA256:
62:DD:0B:E9:B9:F5:0A:16:3E:A0:F8:E7:5C:05:3B:1E:CA:57:EA:55:C8:68:8F:64:7C:68:81:F2:C8:35:7B:95
Alias name: mozillacert38.pem
MD5: 93:2A:3E:F6:FD:23:69:0D:71:20:D4:2B:47:99:2B:A6
SHA256:
A6:C5:1E:0D:A5:CA:0A:93:09:D2:E4:C0:E4:0C:2A:F9:10:7A:AE:82:03:85:7F:E1:98:E3:E7:69:E3:43:08:5C
Alias name: mozillacert5.pem
MD5: A1:0B:44:B3:CA:10:D8:00:6E:9D:0F:D8:0F:92:0A:D1
SHA256:
CE:CD:DC:90:50:99:D8:DA:DF:C5:B1:D2:09:B7:37:CB:E2:C1:8C:FB:2C:10:CO:FF:0B:CF:0D:32:86:FC:1A:A2
Alias name: mozillacert71.pem
MD5: 9E:80:FF:78:01:0C:2E:C1:36:BD:FE:96:90:6E:08:F3
SHA256:
13:63:35:43:93:34:A7:69:80:16:A0:D3:24:DE:72:28:4E:07:9D:7B:52:20:BB:8F:BD:74:78:16:EE:BE:BA:CA
Alias name: verisignclass3g4ca
MD5: 3A:52:E1:E7:FD:6F:3A:E3:6F:F3:6F:99:1B:F9:22:41
SHA256:
69:DD:D7:EA:90:BB:57:C9:3E:13:5D:C8:5E:A6:FC:D5:48:0B:60:32:39:BD:C4:54:FC:75:8B:2A:26:CF:7F:79
Alias name: mozillacert89.pem
MD5: DB:C8:F2:27:2E:B1:EA:6A:29:23:5D:FE:56:3E:33:DF
SHA256:
E3:89:36:0D:0F:DB:AE:B3:D2:50:58:4B:47:30:31:4E:22:2F:39:C1:56:A0:20:14:4E:8D:96:05:61:79:15:06
Alias name: mozillacert135.pem
MD5: 2C:8F:9F:66:1D:18:90:B1:47:26:9D:8E:86:82:8C:A9
SHA256:
D8:E0:FE:BC:1D:B2:E3:8D:00:94:0F:37:D2:7D:41:34:4D:99:3E:73:4B:99:D5:65:6D:97:78:D4:D8:14:36:24
Alias name: camerfirmachamberscommerceca
MD5: B0:01:EE:14:D9:AF:29:18:94:76:8E:F1:69:33:2A:84
SHA256:
0C:25:8A:12:A5:67:4A:EF:25:F2:8B:A7:DC:FA:EC:EE:A3:48:E5:41:E6:F5:CC:4E:E6:3B:71:B3:61:60:6A:C3
Alias name: mozillacert27.pem
MD5: CF:F4:27:0D:D4:ED:DC:65:16:49:6D:3D:DA:BF:6E:DE
SHA256:
42:00:F5:04:3A:C8:59:0E:BB:52:7D:20:9E:D1:50:30:29:FB:CB:D4:1C:A1:B5:06:EC:27:F1:5A:DE:7D:AC:69
Alias name: verisignc1g6.pem
MD5: 2F:A8:B4:DA:F6:64:4B:1E:82:F9:46:3D:54:1A:7C:B0
SHA256:
9D:19:0B:2E:31:45:66:68:5B:E8:A8:89:E2:7A:A8:C7:D7:AE:1D:8A:AD:DB:A3:C1:EC:F9:D2:48:63:CD:34:B9
Alias name: verisignclass3g2ca
MD5: A2:33:9B:4C:74:78:73:D4:6C:E7:C1:F3:8D:CB:5C:E9
SHA256:
83:CE:3C:12:29:68:8A:59:3D:48:5F:81:97:3C:0F:91:95:43:1E:DA:37:CC:5E:36:43:0E:79:C7:A8:88:63:8B
Alias name: mozillacert60.pem
MD5: B7:52:74:E2:92:B4:80:93:F2:75:E4:CC:D7:F2:EA:26
SHA256:
BF:0F:EE:FB:9E:3A:58:1A:D5:F9:E9:DB:75:89:98:57:43:D2:61:08:5C:4D:31:4F:6F:5D:72:59:AA:42:16:12
Alias name: mozillacert78.pem
MD5: 42:65:CA:BE:01:9A:9A:4C:A9:8C:41:49:CD:C0:D5:7F
SHA256:
0A:81:EC:5A:92:97:77:F1:45:90:4A:F3:8D:5D:50:9F:66:B5:E2:C5:8F:CD:B5:31:05:8B:0E:17:F3:F0:B4:1B
Alias name: gd_bundle-g2.pem
MD5: 96:C2:50:31:BC:0D:C3:5C:FB:A7:23:73:1E:1B:41:40
SHA256:
97:3A:41:27:6F:FD:01:E0:27:A2:AA:D4:9E:34:C3:78:46:D3:E9:76:FF:6A:62:0B:67:12:E3:38:32:04:1A:A6
Alias name: certumca
MD5: 2C:8F:9F:66:1D:18:90:B1:47:26:9D:8E:86:82:8C:A9
```

```
SHA256:  
D8:E0:FE:BC:1D:B2:E3:8D:00:94:0F:37:D2:7D:41:34:4D:99:3E:73:4B:99:D5:65:6D:97:78:D4:D8:14:36:24  
Alias name: deutschetelekomrootca2  
MD5: 74:01:4A:91:B1:08:C4:58:CE:47:CD:F0:DD:11:53:08  
SHA256:  
B6:19:1A:50:D0:C3:97:7F:7D:A9:9B:CD:AA:C8:6A:22:7D:AE:B9:67:9E:C7:0B:A3:B0:C9:D9:22:71:C1:70:D3  
Alias name: mozillacert124.pem  
MD5: 27:EC:39:47:CD:DA:5A:AF:E2:9A:01:65:21:A9:4C:BB  
SHA256:  
80:95:21:08:05:DB:4B:BC:35:5E:44:28:D8:FD:6E:C2:CD:E3:AB:5F:B9:7A:99:42:98:8E:B8:F4:DC:D0:60:16  
Alias name: mozillacert16.pem  
MD5: 41:03:52:DC:0F:F7:50:1B:16:F0:02:8E:BA:6F:45:C5  
SHA256:  
06:87:26:03:31:A7:24:03:D9:09:F1:05:E6:9B:CF:0D:32:E1:BD:24:93:FF:C6:D9:20:6D:11:BC:D6:77:07:39  
Alias name: secomevrootca1  
MD5: 22:2D:A6:01:EA:7C:0A:F7:F0:6C:56:43:3F:77:76:D3  
SHA256:  
A2:2D:BA:68:1E:97:37:6E:2D:39:7D:72:8A:AE:3A:9B:62:96:B9:FD:BA:60:BC:2E:11:F6:47:F2:C6:75:FB:37  
Alias name: mozillacert67.pem  
MD5: C5:DF:B8:49:CA:05:13:55:EE:2D:BA:1A:C3:3E:B0:28  
SHA256:  
CB:B5:22:D7:B7:F1:27:AD:6A:01:13:86:5B:DF:1C:D4:10:2E:7D:07:59:AF:63:5A:7C:F4:72:0D:C9:63:C5:3B  
Alias name: globalsignr3ca  
MD5: C5:DF:B8:49:CA:05:13:55:EE:2D:BA:1A:C3:3E:B0:28  
SHA256:  
CB:B5:22:D7:B7:F1:27:AD:6A:01:13:86:5B:DF:1C:D4:10:2E:7D:07:59:AF:63:5A:7C:F4:72:0D:C9:63:C5:3B  
Alias name: mozillacert113.pem  
MD5: EE:29:31:BC:32:7E:9A:E6:E8:B5:F7:51:B4:34:71:90  
SHA256:  
6D:C4:71:72:E0:1C:BC:B0:BF:62:58:0D:89:5F:E2:B8:AC:9A:D4:F8:73:80:1E:0C:10:B9:C8:37:D2:1E:B1:77  
Alias name: gdroot-g2.pem  
MD5: 80:3A:BC:22:C1:E6:FB:8D:9B:3B:27:4A:32:1B:9A:01  
SHA256:  
45:14:0B:32:47:EB:9C:C8:C5:B4:F0:D7:B5:30:91:F7:32:92:08:9E:6E:5A:63:E2:74:9D:D3:AC:A9:19:8E:DA  
Alias name: aolrootca2  
MD5: D6:ED:3C:CA:E2:66:0F:AF:10:43:0D:77:9B:04:09:BF  
SHA256:  
7D:3B:46:5A:60:14:E5:26:C0:AF:FC:EE:21:27:D2:31:17:27:AD:81:1C:26:84:2D:00:6A:F3:73:06:CC:80:BD  
Alias name: trustcenteruniversalcai  
MD5: 45:E1:A5:72:C5:A9:36:64:40:9E:F5:E4:58:84:67:8C  
SHA256:  
EB:F3:C0:2A:87:89:B1:FB:7D:51:19:95:D6:63:B7:29:06:D9:13:CE:0D:5E:10:56:8A:8A:77:E2:58:61:67:E7  
Alias name: aolrootca1  
MD5: 14:F1:08:AD:9D:FA:64:E2:89:E7:1C:CF:A8:AD:7D:5E  
SHA256:  
77:40:73:12:C6:3A:15:3D:5B:C0:0B:4E:51:75:9C:DF:DA:C2:37:DC:2A:33:B6:79:46:E9:8E:9B:FA:68:0A:E3  
Alias name: verisignc2g2.pem  
MD5: 2D:BB:E5:25:D3:D1:65:82:3A:B7:0E:FA:E6:EB:E2:E1  
SHA256:  
3A:43:E2:20:FE:7F:3E:A9:65:3D:1E:21:74:2E:AC:2B:75:C2:0F:D8:98:03:05:BC:50:2C:AF:8C:2D:9B:41:A1  
Alias name: mozillacert56.pem  
MD5: FB:1B:5D:43:8A:94:CD:44:C6:76:F2:43:4B:47:E7:31  
SHA256:  
4B:03:F4:58:07:AD:70:F2:1B:FC:2C:AE:71:C9:FD:E4:60:4C:06:4C:F5:FF:B6:86:BA:E5:DB:AA:D7:FD:D3:4C  
Alias name: verisignclass1g3ca  
MD5: B1:47:BC:18:57:D1:18:A0:78:2D:EC:71:E8:2A:95:73  
SHA256:  
CB:B5:AF:18:5E:94:2A:24:02:F9:EA:CB:C0:ED:5B:B8:76:EE:A3:C1:22:36:23:D0:04:47:E4:F3:BA:55:4B:65  
Alias name: mozillacert102.pem  
MD5: AA:C6:43:2C:5E:2D:CD:C4:34:C0:50:4F:11:02:4F:B6  
SHA256:  
EE:C5:49:6B:98:8C:E9:86:25:B9:34:09:2E:EC:29:08:BE:D0:B0:F3:16:C2:D4:73:0C:84:EA:F1:F3:D3:48:81  
Alias name: addtrustexternalca  
MD5: 1D:35:54:04:85:78:B0:3F:42:42:4D:BF:20:73:0A:3F  
SHA256:  
68:7F:A4:51:38:22:78:FF:F0:C8:B1:1F:8D:43:D5:76:67:1C:6E:B2:BC:EA:B4:13:FB:83:D9:65:D0:6D:2F:F2
```

```
Alias name: verisignc3g2.pem
MD5: A2:33:9B:4C:74:78:73:D4:6C:E7:C1:F3:8D:CB:5C:E9
SHA256:
83:CE:3C:12:29:68:8A:59:3D:48:5F:81:97:3C:0F:91:95:43:1E:DA:37:CC:5E:36:43:0E:79:C7:A8:88:63:8B
Alias name: verisignclass3ca
MD5: EF:5A:F1:33:EF:F1:CD:BB:51:02:EE:12:14:4B:96:C4
SHA256:
A4:B6:B3:99:6F:C2:F3:06:B3:FD:86:81:BD:63:41:3D:8C:50:09:CC:4F:A3:29:C2:CC:F0:E2:FA:1B:14:03:05
Alias name: mozillacert45.pem
MD5: 1B:2E:00:CA:26:06:90:3D:AD:FE:6F:15:68:D3:6B:B3
SHA256:
C0:A6:F4:DC:63:A2:4B:FD:CF:54:EF:2A:6A:08:2A:0A:72:DE:35:80:3E:2F:F5:FF:52:7A:E5:D8:72:06:DF:D5
Alias name: verisignc4g2.pem
MD5: 26:6D:2C:19:98:B6:70:68:38:50:54:19:EC:90:34:60
SHA256:
44:64:0A:0A:0E:4D:00:0F:BD:57:4D:2B:8A:07:BD:B4:D1:DF:ED:3B:45:BA:AB:A7:6F:78:57:78:C7:01:19:61
Alias name: digicertassuredidrootca
MD5: 87:CE:0B:7B:2A:0E:49:00:E1:58:71:9B:37:A8:93:72
SHA256:
3E:90:99:B5:01:5E:8F:48:6C:00:BC:EA:9D:11:1E:E7:21:FA:BA:35:5A:89:BC:F1:DF:69:56:1E:3D:C6:32:5C
Alias name: verisignclass1ca
MD5: 86:AC:DE:2B:C5:6D:C3:D9:8C:28:88:D3:8D:16:13:1E
SHA256:
51:84:7C:8C:BD:2E:9A:72:C9:1E:29:2D:2A:E2:47:D7:DE:1E:3F:D2:70:54:7A:20:EF:7D:61:0F:38:B8:84:2C
Alias name: mozillacert109.pem
MD5: 26:01:FB:D8:27:A7:17:9A:45:54:38:1A:43:01:3B:03
SHA256:
E2:3D:4A:03:6D:7B:70:E9:F5:95:B1:42:20:79:D2:B9:1E:DF:BB:1F:B6:51:A0:63:3E:AA:8A:9D:C5:F8:07:03
Alias name: thawtepremiumserverca
MD5: A6:6B:60:90:23:9B:3F:2D:BB:98:6F:D6:A7:19:0D:46
SHA256:
3F:9F:27:D5:83:20:4B:9E:09:C8:A3:D2:06:6C:4B:57:D3:A2:47:9C:36:93:65:08:80:50:56:98:10:5D:BC:E9
Alias name: verisigntsaca
MD5: F2:89:95:6E:4D:05:F0:F1:A7:21:55:7D:46:11:BA:47
SHA256:
CB:6B:05:D9:E8:E5:7C:D8:82:B1:0B:4D:B7:0D:E4:BB:1D:E4:2B:A4:8A:7B:D0:31:8B:63:5B:F6:E7:78:1A:9D
Alias name: mozillacert96.pem
MD5: CA:FB:40:A8:4E:39:92:8A:1D:FE:8E:2F:C4:27:EA:EF
SHA256:
FD:73:DA:D3:1C:64:4F:F1:B4:3B:EF:0C:CD:DA:96:71:0B:9C:D9:87:5E:CA:7E:31:70:7A:F3:E9:6D:52:2B:BD
Alias name: mozillacert142.pem
MD5: 31:85:3C:62:94:97:63:B9:AA:FD:89:4E:AF:6F:E0:CF
SHA256:
18:F1:FC:7F:20:5D:F8:AD:DD:EB:7F:E0:07:DD:57:E3:AF:37:5A:9C:4D:8D:73:54:6B:F4:F1:FE:D1:E1:8D:35
Alias name: thawteprimaryrootca
MD5: 8C:CA:DC:0B:22:CE:F5:BE:72:AC:41:1A:11:A8:D8:12
SHA256:
8D:72:2F:81:A9:C1:13:C0:79:1D:F1:36:A2:96:6D:B2:6C:95:0A:97:1D:B4:6B:41:99:F4:EA:54:B7:8B:FB:9F
Alias name: mozillacert34.pem
MD5: BC:6C:51:33:A7:E9:D3:66:63:54:15:72:1B:21:92:93
SHA256:
41:C9:23:86:6A:B4:CA:D6:B7:AD:57:80:81:58:2E:02:07:97:A6:CB:DF:4F:FF:78:CE:83:96:B3:89:37:D7:F5
Alias name: mozillacert1.pem
MD5: C5:70:C4:A2:ED:53:78:0C:C8:10:53:81:64:CB:D0:1D
SHA256:
B4:41:0B:73:E2:E6:EA:CA:47:FB:C4:2F:8F:A4:01:8A:F4:38:1D:C5:4C:FA:A8:44:50:46:1E:ED:09:45:4D:E9
Alias name: xrampglobalca
MD5: A1:0B:44:B3:CA:10:D8:00:6E:9D:0F:D8:0F:92:0A:D1
SHA256:
CE:CD:DC:90:50:99:D8:DA:DF:C5:B1:D2:09:B7:37:CB:E2:C1:8C:FB:2C:10:CO:FF:0B:CF:0D:32:86:FC:1A:A2
Alias name: mozillacert85.pem
MD5: AA:08:8F:F6:F9:7B:B7:F2:B1:A7:1E:9B:EA:EA:BD:79
SHA256:
BF:D8:8F:E1:10:1C:41:AE:3E:80:1B:F8:BE:56:35:0E:E9:BA:D1:A6:B9:BD:51:5E:DC:5C:6D:5B:87:11:AC:44
Alias name: valicertclass2ca
MD5: A9:23:75:9B:BA:49:36:6E:31:C2:DB:F2:E7:66:BA:87
```

```
SHA256:  
58:D0:17:27:9C:D4:DC:63:AB:DD:B1:96:A6:C9:90:6C:30:C4:E0:87:83:EA:E8:C1:60:99:54:D6:93:55:59:6B  
Alias name: mozillacert131.pem  
MD5: 34:FC:B8:D0:36:DB:9E:14:B3:C2:F2:DB:8F:E4:94:C7  
SHA256:  
A0:23:4F:3B:C8:52:7C:A5:62:8E:EC:81:AD:5D:69:89:5D:A5:68:0D:C9:1D:1C:B8:47:7F:33:F8:78:B9:5B:0B  
Alias name: mozillacert149.pem  
MD5: B0:01:EE:14:D9:AF:29:18:94:76:8E:F1:69:33:2A:84  
SHA256:  
0C:25:8A:12:A5:67:4A:EF:25:F2:8B:A7:DC:FA:EC:EE:A3:48:E5:41:E6:F5:CC:4E:E6:3B:71:B3:61:60:6A:C3  
Alias name: geotrustprimaryca  
MD5: 02:26:C3:01:5E:08:30:37:43:A9:D0:7D:CF:37:E6:BF  
SHA256:  
37:D5:10:06:C5:12:EA:AB:62:64:21:F1:EC:8C:92:01:3F:C5:F8:2A:E9:8E:E5:33:EB:46:19:B8:DE:B4:D0:6C  
Alias name: mozillacert23.pem  
MD5: 8C:CA:DC:0B:22:CE:F5:BE:72:AC:41:1A:11:A8:D8:12  
SHA256:  
8D:72:2F:81:A9:C1:13:C0:79:1D:F1:36:A2:96:6D:B2:6C:95:0A:97:1D:B4:6B:41:99:F4:EA:54:B7:8B:FB:9F  
Alias name: verisignc1g2.pem  
MD5: DB:23:3D:F9:69:FA:4B:B9:95:80:44:73:5E:7D:41:83  
SHA256:  
34:1D:E9:8B:13:92:AB:F7:F4:AB:90:A9:60:CF:25:D4:BD:6E:C6:5B:9A:51:CE:6E:D0:67:D0:0E:C7:CE:9B:7F  
Alias name: mozillacert8.pem  
MD5: 22:4D:8F:8A:FC:F7:35:C2:BB:57:34:90:7B:8B:22:16  
SHA256:  
C7:66:A9:BE:F2:D4:07:1C:86:3A:31:AA:49:20:E8:13:B2:D1:98:60:8C:B7:B7:CF:E2:11:43:B8:36:DF:09:EA  
Alias name: mozillacert74.pem  
MD5: 17:35:74:AF:7B:61:1C:EB:F4:F9:3C:E2:EE:40:F9:A2  
SHA256:  
56:8D:69:05:A2:C8:87:08:A4:B3:02:51:90:ED:CF:ED:B1:97:4A:60:6A:13:C6:E5:29:0F:CB:2A:E6:3E:DA:B5  
Alias name: mozillacert120.pem  
MD5: 64:9C:EF:2E:44:FC:C6:8F:52:07:D0:51:73:8F:CB:3D  
SHA256:  
CF:56:FF:46:A4:A1:86:10:9D:D9:65:84:B5:EE:B5:8A:51:0C:42:75:B0:E5:F9:4F:40:BB:AE:86:5E:19:F6:73  
Alias name: geotrustglobalca  
MD5: F7:75:AB:29:FB:51:4E:B7:77:5E:FF:05:3C:99:8E:F5  
SHA256:  
FF:85:6A:2D:25:1D:CD:88:D3:66:56:F4:50:12:67:98:CF:AB:AA:DE:40:79:9C:72:2D:E4:D2:B5:DB:36:A7:3A  
Alias name: mozillacert138.pem  
MD5: 91:1B:3F:6E:CD:9E:AB:EE:07:FE:1F:71:D2:B3:61:27  
SHA256:  
3F:06:E5:56:81:D4:96:F5:BE:16:9E:B5:38:9F:9F:2B:8F:F6:1E:17:08:DF:68:81:72:48:49:CD:5D:27:CB:69  
Alias name: mozillacert12.pem  
MD5: 79:E4:A9:84:0D:7D:3A:96:D7:CO:4F:E2:43:4C:89:2E  
SHA256:  
43:48:A0:E9:44:4C:78:CB:26:5E:05:8D:5E:89:44:B4:D8:4F:96:62:BD:26:DB:25:7F:89:34:A4:43:C7:01:61  
Alias name: comodoaaaca  
MD5: 49:79:04:B0:EB:87:19:AC:47:B0:BC:11:51:9B:74:D0  
SHA256:  
D7:A7:A0:FB:5D:7E:27:31:D7:71:E9:48:4E:BC:DE:F7:1D:5F:0C:3E:0A:29:48:78:2B:C8:3E:E0:EA:69:9E:F4  
Alias name: mozillacert63.pem  
MD5: F8:49:F4:03:BC:44:2D:83:BE:48:69:7D:29:64:FC:B1  
SHA256:  
3C:5F:81:FE:A5:FA:B8:2C:64:BF:A2:EA:EC:AF:CD:E8:E0:77:FC:86:20:A7:CA:E5:37:16:3D:F3:6E:DB:F3:78  
Alias name: certplusclass2primaryca  
MD5: 88:2C:8C:52:B8:A2:3C:F3:F7:BB:03:EA:AE:AC:42:0B  
SHA256:  
0F:99:3C:8A:EF:97:BA:AF:56:87:14:0E:D5:9A:D1:82:1B:B4:AF:AC:F0:AA:9A:58:B5:D5:7A:33:8A:3A:FB:CB  
Alias name: mozillacert127.pem  
MD5: F7:75:AB:29:FB:51:4E:B7:77:5E:FF:05:3C:99:8E:F5  
SHA256:  
FF:85:6A:2D:25:1D:CD:88:D3:66:56:F4:50:12:67:98:CF:AB:AA:DE:40:79:9C:72:2D:E4:D2:B5:DB:36:A7:3A  
Alias name: ttelesecglobalrootclass2ca  
MD5: 2B:9B:9E:E4:7B:6C:1F:00:72:1A:CC:C1:77:79:DF:6A  
SHA256:  
91:E2:F5:78:8D:58:10:EB:A7:BA:58:73:7D:E1:54:8A:8E:CA:CD:01:45:98:BC:0B:14:3E:04:1B:17:05:25:52
```

```
Alias name: mozillacert19.pem
MD5: 37:A5:6E:D4:B1:25:84:97:B7:FD:56:15:7A:F9:A2:00
SHA256:
C4:70:CF:54:7E:23:02:B9:77:FB:29:DD:71:A8:9A:7B:6C:1F:60:77:7B:03:29:F5:60:17:F3:28:BF:4F:6B:E6
Alias name: digicerthighassuranceevrootca
MD5: D4:74:DE:57:5C:39:B2:D3:9C:85:83:C5:C0:65:49:8A
SHA256:
74:31:E5:F4:C3:C1:CE:46:90:77:4F:0B:61:E0:54:40:88:3B:A9:A0:1E:D0:0B:A6:AB:D7:80:6E:D3:B1:18:CF
Alias name: amzninternalinfoseccag3
MD5: E9:34:94:02:BA:BB:31:6B:22:E6:2B:A9:C4:F0:26:04
SHA256:
81:03:OB:C7:E2:54:DA:7B:F8:B7:45:DB:DD:41:15:89:B5:A3:81:86:FB:4B:29:77:1F:84:0A:18:D9:67:6D:68
Alias name: mozillacert52.pem
MD5: 21:BC:82:AB:49:C4:13:3B:4B:B2:2B:5C:6B:90:9C:19
SHA256:
E2:83:93:77:3D:A8:45:A6:79:F2:08:0C:C7:FB:44:A3:B7:A1:C3:79:2C:B7:EB:77:29:FD:CB:6A:8D:99:AE:A7
Alias name: mozillacert116.pem
MD5: AE:B9:C4:32:4B:AC:7F:5D:66:CC:77:94:BB:2A:77:56
SHA256:
F3:56:BE:A2:44:B7:A9:1E:B3:5D:53:CA:9A:D7:86:4A:CE:01:8E:2D:35:D5:F8:F9:6D:DF:68:A6:F4:1A:A4:74
Alias name: globalsignca
MD5: 3E:45:52:15:09:51:92:E1:B7:5D:37:9F:B1:87:29:8A
SHA256:
EB:D4:10:40:E4:BB:3E:C7:42:C9:E3:81:D3:1E:F2:A4:1A:48:B6:68:5C:96:E7:CE:F3:C1:DF:6C:D4:33:1C:99
Alias name: mozillacert41.pem
MD5: 45:E1:A5:72:C5:A9:36:64:40:9E:F4:58:84:67:8C
SHA256:
EB:F3:C0:2A:87:89:B1:FB:7D:51:19:95:D6:63:B7:29:06:D9:13:CE:0D:5E:10:56:8A:8A:77:E2:58:61:67:E7
Alias name: mozillacert59.pem
MD5: 8E:AD:B5:01:AA:4D:81:E4:8C:1D:D1:E1:14:00:95:19
SHA256:
23:99:56:11:27:A5:71:25:DE:8C:EF:EA:61:0D:DF:2F:A0:78:B5:C8:06:7F:4E:82:82:90:BF:B8:60:E8:4B:3C
Alias name: mozillacert105.pem
MD5: 5B:04:69:EC:A5:83:94:63:18:A7:86:D0:E4:F2:6E:19
SHA256:
F0:9B:12:2C:71:14:F4:A0:9B:D4:EA:4F:4A:99:D5:58:B4:6E:4C:25:CD:81:14:0D:29:C0:56:13:91:4C:38:41
Alias name: trustcenterclass2caii
MD5: CE:78:33:5C:59:78:01:6E:18:EA:B9:36:A0:B9:2E:23
SHA256:
E6:B8:F8:76:64:85:F8:07:AE:7F:8D:AC:16:70:46:1F:07:C0:A1:3E:EF:3A:1F:F7:17:53:8D:7A:BA:D3:91:B4
Alias name: mozillacert92.pem
MD5: C9:3B:0D:84:41:FC:A4:76:79:23:08:57:DE:10:19:16
SHA256:
E1:78:90:EE:09:A3:FB:F4:F4:8B:9C:41:4A:17:D6:37:B7:A5:06:47:E9:BC:75:23:22:72:7F:CC:17:42:A9:11
Alias name: verisignc3g5.pem
MD5: CB:17:E4:31:67:3E:E2:09:FE:45:57:93:F3:0A:FA:1C
SHA256:
9A:CF:AB:7E:43:C8:D8:80:D0:6B:26:2A:94:DE:EE:E4:B4:65:99:89:C3:D0:CA:F1:9B:AF:64:05:E4:1A:B7:DF
Alias name: geotrustprimarycag3
MD5: B5:E8:34:36:C9:10:44:58:48:70:6D:2E:83:D4:B8:05
SHA256:
B4:78:B8:12:25:0D:F8:78:63:5C:2A:A7:EC:7D:15:5E:AA:62:5E:E8:29:16:E2:CD:29:43:61:88:6C:D1:FB:D4
Alias name: geotrustprimarycag2
MD5: 01:5E:D8:6B:BD:6F:3D:8E:A1:31:F8:12:E0:98:73:6A
SHA256:
5E:DB:7A:C4:3B:82:A0:6A:87:61:E8:D7:BE:49:79:EB:F2:61:1F:7D:D7:9B:F9:1C:1C:6B:56:6A:21:9E:D7:66
Alias name: mozillacert30.pem
MD5: 15:AC:A5:C2:92:2D:79:BC:E8:7F:CB:67:ED:02:CF:36
SHA256:
A7:12:72:AE:AA:A3:CF:E8:72:7F:B3:9F:0F:B3:D1:E5:42:6E:90:60:B0:6E:E6:F1:3E:9A:3C:58:33:CD:43
Alias name: affirmtrustpremiumeccca
MD5: 64:B0:09:55:CF:B1:D5:99:E2:BE:13:AB:A6:5D:EA:4D
SHA256:
BD:71:FD:F6:DA:97:E4:CF:62:D1:64:7A:DD:25:81:B0:7D:79:AD:F8:39:7E:B4:EC:BA:9C:5E:84:88:82:14:23
Alias name: mozillacert48.pem
MD5: B8:08:9A:F0:03:CC:1B:0D:C8:6C:0B:76:A1:75:64:23
```

SHA256:  
0F:4E:9C:DD:26:4B:02:55:50:D1:70:80:63:40:21:4F:E9:44:34:C9:B0:2F:69:7E:C7:10:FC:5F:EA:FB:5E:38

## 使用 AWS WAF 來保護您的 Amazon API Gateway API 避免受到常見的網路攻擊

AWS WAF 是一種 Web 應用程式防火牆，透過讓您設定一組規則（稱為 web 存取控制清單或 web ACL），協助保護 Web 應用程式和 API 不受攻擊，該組規則可根據您定義的自訂的 web 安全規則與條件來允許、封鎖或計數 web 請求。如需詳細資訊，請參閱 [AWS WAF 運作方式](#)。

您可以使用 AWS WAF 來保護 API Gateway API 免受常見網路攻擊，像是 SQL injection 和跨網站指令碼 (XSS) 攻擊，這些攻擊可能影響 API 可用性、效能和安全性危害，或耗用過多資源。例如，您可以建立規則，允許或封鎖來自指定 IP 地址或 CIDR 區塊的請求，或源自特定國家或區域，其中包含惡意 SQL 程式碼或惡意指令碼的請求。您也可以建立規則，以符合在 HTTP 標頭、方法、查詢字串、URI 和請求本文（限於前 8 KB）的指定字串或常規表達式模式。此外，您可以建立規則以封鎖來自特定使用者代理程式、惡意機器人和內容抓取器的攻擊。例如，您可以使用以速率為基礎的規則，以指定每個用戶端 IP 在尾隨、持續更新的 5 分鐘期間，允許的 Web 請求數。

### Important

AWS WAF 是您對抗網路攻擊的第一條防線。在 API 上啟用 AWS WAF 時，會先評估 AWS WAF 規則，再評估其他存取控制功能，例如 [資源政策 \(p. 324\)](#)、[IAM 政策 \(p. 333\)](#)、[Lambda 授權方 \(p. 348\)](#)，以及 [Amazon Cognito 授權方 \(p. 363\)](#)。例如，如果 AWS WAF 封鎖來自資源政策所允許之 CIDR 區塊的存取，則會優先使用 AWS WAF，且不會評估資源政策。

若要啟用 API 的 AWS WAF，您將需要：

1. 使用 AWS WAF 主控台、AWS 開發套件或 CLI 來建立區域性 web ACL，其中包含 AWS WAF 管理規則和您自己的自訂規則所需的組合。如需詳細資訊，請參閱 [AWS WAF 入門](#) 和 [建立和設定 Web 存取控制清單 \(Web ACL\)](#)。

### Important

API Gateway 需要區域性 web ACL。

2. 將 AWS WAF 區域性 web ACL 與 API 階段相關聯。您可以使用 AWS WAF 主控台、AWS SDK 或 CLI 或是 API Gateway 主控台、AWS 開發套件或 CLI 來這麼做。

## 使用 API Gateway 主控台來建立 API Gateway API 階段與 AWS WAF 區域性 Web ACL 的關聯

若要使用 API Gateway 主控台，將 AWS WAF 區域性 web ACL 與現有 API Gateway API 階段建立關聯，請使用以下步驟：

1. 登入位於 <https://console.aws.amazon.com/apigateway> 的 API Gateway 主控台。
2. 在 API (API) 導覽窗格中，選擇 API，然後選擇 Stages (階段)。
3. 在 Stages (階段) 窗格中，選擇階段的名稱。
4. 在 Stage Editor (階段編輯器) 窗格中，選擇 Settings (設定) 分頁。
5. 將區域性 web ACL 與 API 階段相關聯：
  - 在 AWS WAF Web ACL 下拉式清單中，選擇您想與此階段建立關聯的區域性 web ACL。

### Note

如果您需要的 Web ACL 不存在，選擇 Create WebACL (建立 WebACL)，然後選擇 Go to AWS WAF (前往 AWS WAF)，即可在新的瀏覽器分頁開啟 WAF 主控台並建立區域性 web ACL。然後，返回 API Gateway 主控台以將 web ACL 與該階段建立關聯。

6. 選擇 Save Changes (儲存變更)。

## 使用 AWS CLI 來建立 API Gateway API 階段與 AWS WAF 區域性 Web ACL 的關聯

若要使用 AWS CLI，將 AWS WAF 區域性 web ACL 與現有 API Gateway API 階段建立關聯，請在以下範例中呼叫 `associate-web-acl` 命令：

```
aws waf-regional associate-web-acl \
--web-acl-id 'aabc123a-fb4f-4fc6-becb-2b00831cadcf' \
--resource-arn 'arn:aws:apigateway:{region}::/restapis/4wk1k4onj3/stages/prod'
```

## 使用 AWS WAF REST API 來建立 API 階段與 AWS WAF 區域性 Web ACL 的關聯

若要使用 AWS WAF REST API，將 AWS WAF 區域性 web ACL 與現有 API Gateway API 階段建立關聯，請在以下範例中呼叫 `AssociateWebACL` 命令：

```
import boto3

waf = boto3.client('wafregional')

waf.associate_web_acl(
    WebACLId='aabc123a-fb4f-4fc6-becb-2b00831cadcf',
    ResourceArn='arn:aws:apigateway:{region}::/restapis/4wk1k4onj3/stages/prod'
)
```

## 建立和使用 API 金鑰的用量計劃

在您建立、測試和部署 API 之後，您可以使用 API Gateway 用量計劃讓它們可做為產品優惠供給客戶使用。您可以設定用量計劃和 API 金鑰，讓客戶依雙方一致同意之符合其業務需求和預算限制的請求速率和配額存取選取的 API。如有需要，您可以為 API 設定預設方法層級調節限制或為個別 API 方法設定調節限制。

### 什麼是用量計劃和 API 金鑰？

用量計劃指定能存取一或多個部署的 API 階段和方法的人員，也規定這些人可以存取 API 的數量和速度。計劃會使用 API 金鑰來識別 API 用戶端，並計量對每個金鑰相關的 API 階段的存取。它也可讓您設定調節限制和配額限制，這些是針對個別用戶端 API 金鑰所強制執行。

API 金鑰是英數字串值，您會將其發佈到應用程式開發人員客戶，以授與您 API 的存取權。您可以同時使用 API 金鑰與用量計劃 (p. 397) 或 Lambda 授權方 (p. 348)，以控制對 API 的存取。API Gateway 可以代表您產生 API 金鑰，或從 CSV 檔案 (p. 411) 匯入它們。您可以在 API Gateway 中產生 API 金鑰，或從外部來源將它匯入 API Gateway。如需詳細資訊，請參閱the section called “[使用 API Gateway 主控台設定 API 金鑰](#)” (p. 400)。

API 金鑰具有名稱和數值。(術語「API 金鑰」與「API 金鑰值」經常可以交換使用。) 金鑰值是介於 30 與 128 個字元之間的英數字串，例如，`apikey1234abcdefghij0123456789`。

#### Important

API 金鑰值必須是唯一的。如果您嘗試建立兩個名稱不同但數值相同的 API 金鑰，API Gateway 會將它們視為相同的 API 金鑰。

API 金鑰可以關聯到多個用量計劃。用量計劃可以關聯到多個階段。不過，特定 API 金鑰只能與每個 API 階段的一個用量計劃相關聯。

調節限制是請求速率限制，會套用到您新增到用量計劃的每個 API 金鑰。您也可以為 API 設定預設方法層級調節限制或為個別 API 方法設定調節限制。

配額限制是最大的請求數量與在指定時間間隔內提交的指定 API 金鑰。您可以設定個別的 API 方法根據用量計劃組態來要求 API 金鑰授權。此外，您可以使用 `get-usage` CLI 命令和 `usage:get` REST API 方法，來判定您的 API 客戶已使用的配額數量。

**Note**

調節和配額限制適用於在用量計劃中跨所有 API 階段彙總之個別 API 金鑰的請求。

## API 金鑰和用量計劃的最佳實務

以下是使用 API 金鑰和用量計劃時建議遵循的最佳實務。

- 請不要倚賴 API 金鑰做為您 API 的唯一身分驗證和授權方法。然而，如果您在用量計劃中具有多個 API，則對該用量計劃中的某個 API 具有有效 API 金鑰的使用者可以存取該用量計劃中的所有 API。反之，使用 IAM 角色、[Lambda 授權方 \(p. 348\)](#)或 [Amazon Cognito 使用者集區 \(p. 363\)](#)。
- 如果您是使用[開發人員入口網站 \(p. 579\)](#)來發佈 API，請注意，特定用量計劃中的所有 API 都是可訂閱的，即使您尚未讓客戶可以看到它們也一樣。

## 設定用量計劃的步驟？

下列步驟概述身為 API 擁有者的您，如何為客戶建立和設定用量計劃。

### 設定用量計劃

- 建立一或多個 API、設定方法要求 API 金鑰，以及將 API 部署至階段。
- 產生或匯入 API 金鑰來發佈到將使用您 API 的應用程式開發人員 (您的客戶)。
- 建立具有所需調節和配額限制的用量計劃。
- 將 API 階段和 API 金鑰與用量計劃建立關聯。

API 發起人必須在請求 API 的 `x-api-key` 標頭中提供已指派的 API 金鑰。

**Note**

若要在用量計劃中包含 API 方法，您必須設定個別的 API 方法要求 API 金鑰 ([p. 400](#))。使用者身分驗證和授權請不要使用 API 金鑰。使用 IAM 角色、[Lambda 授權方 \(p. 348\)](#)或 [Amazon Cognito 使用者集區 \(p. 363\)](#)。

## 選擇 API 金鑰來源

當您將 API 與用量計劃建立關聯並在 API 方法上啟用 API 金鑰時，每個對 API 傳入的請求必須包含 [API 金鑰 \(p. 5\)](#)。API Gateway 會讀取金鑰，並將其與在用量計劃中的金鑰進行比對。如果相符，則 API Gateway 會根據計劃的請求限制和配額調節請求。否則，它會擲出 `InvalidKeyParameter` 例外狀況。因此，發起人收到 `403 Forbidden` 回應。

API Gateway API 可以接收從以下兩種來源的 API 金鑰：

### HEADER

您將 API 金鑰發佈至客戶，並要求他們傳遞 API 金鑰，做為每個傳入請求的 `X-API-Key` 標頭。

#### AUTHORIZER

您可以讓 Lambda 授權方傳回 API 金鑰，當做授權回應的一部分。如需授權回應的詳細資訊，請參閱「[the section called “Amazon API Gateway Lambda 授權方的輸出” \(p. 358\)](#)」。

使用 API Gateway 主控台選擇 API 的 API 金鑰來源：

1. 登入 API Gateway 主控台。
2. 選擇現有的 API 或建立新的 API。
3. 在主導覽窗格中，在選擇或新建立的 API 下選擇 Settings (設定)。
4. 在 Settings (設定) 窗格的 API Key Source (API 金鑰來源) 區段下，從下拉式清單中選擇 HEADER 或 AUTHORIZER。
5. 選擇 Save Changes (儲存變更)。

若要使用 AWS CLI 選擇 API 的 API 金鑰來源，請呼叫 `update-rest-api` 命令，如下所示：

```
aws apigateway update-rest-api --rest-api-id 1234123412 --patch-operations
op=replace,path=/apiKeySource,value=AUTHORIZER
```

若要讓用戶端提交 API 金鑰，請在上述 CLI 命令中將 value 設為 HEADER。

若要使用 API Gateway REST API 選擇 API 的 API 金鑰來源，請呼叫 `restapi:update`，如下所示：

```
PATCH /restapis/fugvwdxtri/ HTTP/1.1
Content-Type: application/json
Host: apigateway.us-east-1.amazonaws.com
X-Amz-Date: 20160603T205348Z
Authorization: AWS4-HMAC-SHA256 Credential={access_key_ID}/20160603/us-east-1/
apigateway/aws4_request, SignedHeaders=content-length;content-type;host;x-amz-date,
Signature={sig4_hash}

{
  "patchOperations" : [
    {
      "op" : "replace",
      "path" : "/apiKeySource",
      "value" : "HEADER"
    }
  ]
}
```

若要讓授權方傳回 API 金鑰，請在之前的 value 輸入中將 AUTHORIZER 設為 patchOperations。

根據您選擇的 API 金鑰來源類型，使用以下程序的其中之一，在方法呼叫中使用來源於標頭的 API 金鑰或授權方傳回的 API 金鑰：

若要使用標頭來源 API 金鑰：

1. 使用所需的 API 方法建立 API。然後將 API 部署到階段。
2. 建立新的用量計劃或選擇現有的用量計劃。將已部署的 API 階段新增至用量計劃。將 API 金鑰連接到用量計劃，或在計劃中選擇現有的 API 金鑰。請記下所選的 API 金鑰值。
3. 設定 API 方法要求 API 金鑰。
4. 將 API 重新部署到同一個階段。如果您將 API 部署到新的階段，請務必更新用量計劃以連接新的 API 階段。

用戶端現在可以在呼叫 API 方法的同時，提供以所選 API 金鑰為標頭值的 `x-api-key` 標頭。

若要使用授權方來源 API 金鑰：

1. 使用所需的 API 方法建立 API。然後將 API 部署到階段。
2. 建立新的用量計劃或選擇現有的用量計劃。將已部署的 API 階段新增至用量計劃。將 API 金鑰連接到用量計劃，或在計劃中選擇現有的 API 金鑰。請記下所選的 API 金鑰值。
3. 建立字符類型的自訂 Lambda 授權方。包含 `usageIdentifierKey:{api-key}`，作為授權回應的根層級屬性，其中 `{api-key}` 代表上一個步驟中提及的 API 金鑰值。
4. 設定 API 方法請求 API 金鑰，也在方法中啟用 Lambda 授權方。
5. 將 API 重新部署到同一個階段。如果您將 API 部署到新的階段，請務必更新用量計劃以連接新的 API 階段。

用戶端現在可以不用明確提供任何 API 金鑰，呼叫需要 API 金鑰的方法。自動使用授權方傳回的 API 金鑰。

## 使用 API Gateway 主控台設定 API 金鑰

若要設定 API 金鑰，請執行下列步驟：

- 設定 API 方法以要求 API 金鑰。
- 為區域中的 API 建立或匯入 API 金鑰。

在設定 API 金鑰之前，您必須已建立 API 並將它部署至階段。

如需如何使用 API Gateway 主控台建立和部署 API 的說明，請分別參閱[在 Amazon API Gateway 中建立 REST API \(p. 167\)](#)和[在 Amazon API Gateway 中部署 REST API \(p. 457\)](#)。

### 主題

- [方法中需要 API 金鑰 \(p. 400\)](#)
- [建立 API 金鑰 \(p. 401\)](#)
- [匯入 API 金鑰 \(p. 402\)](#)

## 方法中需要 API 金鑰

下列程序說明如何設定 API 方法要求 API 金鑰。

### 設定 API 方法要求 API 金鑰

1. Sign in to the AWS Management Console and open the API Gateway console at <https://console.aws.amazon.com/apigateway/>.
2. 在 API Gateway 主導覽窗格中，選擇 Resources (資源)。
3. 在 Resources (資源) 下，建立新的方法或選擇現有的方法。
4. 選擇 Method Request (方法請求)。
5. 在 Authorization Settings (授權設定) 區段下，針對 API Key Required (需要 API 金鑰) 選擇 `true`。
6. 選取核取記號圖示來儲存設定。

The screenshot shows the AWS API Gateway console. On the left, the 'Resources' sidebar lists methods: GET, OPTIONS, POST for the root path, and GET, OPTIONS, POST for the /pets/{petId} path. The 'Actions' dropdown is open. On the right, the 'Method Execution' configuration for the GET method on the /pets/{petId} path is shown. The 'Authorization Settings' section indicates 'Authorization: NONE'. The 'API Key Required' dropdown is set to 'true' and is highlighted with a red oval. Below it are sections for 'URL Query String Parameters', 'HTTP Request Headers', and 'Request Models'.

7. 部署或重新部署 API 以使要求生效。

如果 API Key Required (需要 API 金鑰) 選項設為 `false`，而且您不執行之前的各項步驟，則方法不會使用任何與 API 階段相關聯的 API 金鑰。

## 建立 API 金鑰

如果您已建立或匯入 API 金鑰供用量計劃使用，您可以略過此操作和下一個程序。

### 建立 API 金鑰

1. Sign in to the AWS Management Console and open the API Gateway console at <https://console.aws.amazon.com/apigateway/>.
2. 在 API Gateway 主導覽窗格中，選擇 API Keys (API 金鑰)。
3. 從 Actions (動作) 下拉式選單中，選擇 Create API key (建立 API 金鑰)。

The screenshot shows the 'API Keys' management page. The left sidebar has 'APIs' selected, showing 'PetStore' and 'Usage Plans'. 'API Keys' is selected and highlighted. The main area shows a list of keys: 'MyFirstKey'. A context menu is open over 'MyFirstKey' with options: 'Create API key' (highlighted), 'Import API keys', and 'Actions'. A message 'Select an API key' is displayed.

4. 在 Create API Key (建立 API 金鑰) 中，執行下列操作：
  - a. 在 Name (名稱) 輸入欄位中輸入 API 金鑰名稱 (例如 `MyFirstKey`)。
  - b. 選擇 Auto Generate (自動產生) 讓 API Gateway 產生金鑰值，或選擇 Custom (自訂) 手動輸入金鑰。
  - c. 選擇 Save (儲存)。

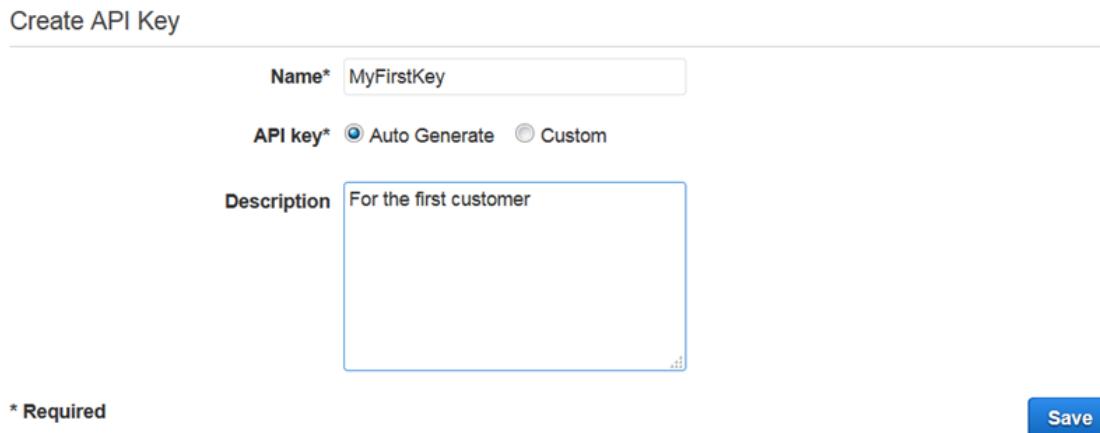
Create API Key

Name\* MyFirstKey

API key\*  Auto Generate  Custom

Description For the first customer

\* Required Save



5. 如有需要，請重複上述步驟建立多個 API 金鑰。

## 匯入 API 金鑰

下列程序說明如何匯入 API 金鑰供用量計劃使用。

### 匯入 API 金鑰

1. 在主導覽窗格中，選擇 API Keys (API 金鑰)。
2. 從 Actions (動作) 下拉式選單中，選擇 Import API keys (匯入 API 金鑰)。
3. 若要載入逗號分隔的金鑰檔案，請選擇 Select CSV File (選取 CSV 檔案)。您也可以手動輸入金鑰。如需檔案格式的資訊，請參閱「[the section called “API Gateway API 金鑰檔案格式” \(p. 411\)](#)」。

## Import API Keys

Use the field below to upload your existing API Keys as comma separated values (CSV). API Keys will be created in API Gateway and associated with a Usage Plan. Learn about the CSV format in the [API Gateway documentation](#).

[Select CSV File](#)

1	name,Key,Description(enabled,usageplanIds)
	2 ImportedKey,CWaiyZjNC212f9P7hcxG17Ae803jEdFu8pzfryqf,an imported key,true,abcdef

Fail on warnings    Ignore warnings   [Import](#)

4. 選擇 Fail on warnings (發出警告時失敗) 在發生錯誤時停止匯入，或選擇 Ignore warnings (忽略警告) 在發生錯誤時繼續匯入有效的金鑰項目。
5. 選擇 Import (匯入) 開始匯入選取的 API 金鑰。

既然您已設定 API 金鑰，就可以繼續[建立和使用用量計劃 \(p. 403\)](#)。

## 使用 API Gateway 主控台建立、設定和測試用量計劃

建立用量計劃之前，請確保您已設定所需的 API 金鑰。如需更多詳細資訊，請參閱 [使用 API Gateway 主控台設定 API 金鑰 \(p. 400\)](#)。

本節說明如何使用 API Gateway 主控台建立和使用用量計劃。

### 主題

- [遷移您的 API 到預設用量計劃 \(如需要\) \(p. 403\)](#)
- [建立用量計劃 \(p. 404\)](#)
- [測試用量計劃 \(p. 406\)](#)
- [維護計劃用量 \(p. 406\)](#)

### 遷移您的 API 到預設用量計劃 (如需要)

如果您在 2016 年 8 月 11 日推出用量計劃功能之後開始使用 API Gateway，就可以自動在所有支援的區域中啟用用量計劃。

如果您是在該日期之前開始使用 API Gateway，您可能需要遷移到預設用量計劃。在所選區域第一次使用用量計劃之前，系統會提示您 Enable Usage Plans (啟用用量計劃) 選項。當您啟用此選項時，您已為與現有 API 金鑰相關聯的每個唯一 API 階段，建立預設的用量計劃。在預設的用量計劃中，一開始並無設定調節和配額限制，而 API 金鑰和 API 階段之間的關聯會複製到用量計劃。API 的行為和以前一樣。不過，您必須使

用 `UsagePlan.apiStages` 屬性建立指定的 API 階段值 (`apiId` 和 `stage`) 與內含 API 金鑰的關聯性 (透過 `UsagePlanKey`)，而非使用 `ApiKey stageKeys` 屬性。

要查看您是否已遷移到預設用量計劃，請使用 `get-account` CLI 命令。在命令輸出之中，當用量計劃已啟用，`features` 清單就會包含 "UsagePlans" 項目。

您也可以使用 AWS CLI 將 API 邁移到預設用量計劃，如下所示：

#### 使用 AWS CLI 邁移到預設用量計劃

1. 呼叫 CLI 命令：`update-account`。
2. 針對 `cli-input-json` 參數，請使用下列 JSON：

```
[  
  {  
    "op": "add",  
    "path": "/features",  
    "value": "UsagePlans"  
  }  
]
```

## 建立用量計劃

下列程序說明如何建立用量計劃。

#### 建立用量計劃

1. 在 Amazon API Gateway 主導覽窗格中，選擇 Usage Plans (用量計劃)，然後選擇 Create (建立)。
2. 在 Create Usage Plan (建立用量計劃) 下，執行下列操作：
  - a. 針對 Name (名稱)，輸入您的計劃名稱 (例如 `Plan_A`)。
  - b. 針對 Description (描述)，輸入您的計劃描述。
  - c. 選取 Enable throttling (啟用調節)，並設定 Rate (費率) (例如 `100`) 和 Burst (爆量) (例如 `200`)。
  - d. 選擇 Enable quota (啟用配額)，並設定其所選時間間隔 (例如 Month (月)) 的限制 (例如 `5000`)。
  - e. 選擇 Next (下一步)。

**Create Usage Plan**

Usage Plans help you meter API usage. With Usage Plans, you can enforce a throttling and quota limit on each API key. Throttling limits define the maximum number of requests per second available to each key. Quota limits define the number of requests each API key is allowed to make over a period.

**Name\*** Plan\_A

**Description** Sample usage plan

**Throttling**

Enable throttling

Rate\* 100 requests per second

Burst\* 200 requests

**Quota**

Enable quota

5000 requests per Month

\* Required

Next

3. 若要在計畫中新增階段，請在 Associated API Stages (相關聯的 API 階段) 窗格中執行下列操作：
  - a. 選擇 Add API Stage (新增 API 階段)。
  - b. 從 API 下拉式清單中選擇 API (例如 **PetStore**)。
  - c. 從 Stage (階段) 下拉式清單中選擇階段 (例如 **Stage\_1**)。
  - d. 選擇核取記號圖示以儲存。

### Associated API Stages

Add API Stage		
API	Stage	Method Throttling
PetStore	prod	No Methods Configured
		Configure Method Throttling <span style="float: right;">X</span>

4. 要設定方法調節 (p. 469)，請執行下列動作：
  - a. 選擇 Configure Method Throttling (設定方法調節)。
  - b. 選擇 Add Resource/Method (新增資源/方法)。
  - c. 從 Resource (資源) 下拉式清單中，選擇資源。
  - d. 從 Method (方法) 下拉式清單中，選擇方法。

- e. 設定 Rate (requests per second) (費率 (根據每秒請求))，(例如 **100**) 和 Burst (爆量) (例如 **200**)。
  - f. 選擇核取記號圖示以儲存。
  - g. 選擇 Close (關閉)。
5. 若要在計劃中新增金鑰，請在 API Keys (API 金鑰) 標籤中執行下列操作：
- a. 若要使用現有的金鑰，請選擇 Add API Key to Usage Plan (將 API 金鑰新增至用量方案)。
  - b. 針對 Name (名稱)，輸入您要新增的金鑰名稱 (例如，**MyFirstKey**)。
  - c. 選擇核取記號圖示以儲存。
  - d. 視需要重複上述步驟，將其他的現有 API 金鑰新增至這個用量計劃。

## Usage Plan API Keys

Subscribe an API key to this usage plan. Choose "Add API Key" below to search through your existing API keys. Once a key is associated with a plan, API Gateway will meter all requests from the key and apply the plan's throttling and quota limits.

The screenshot shows the 'Usage Plan API Keys' section of the AWS API Gateway console. At the top, there are two buttons: 'Add API Key to Usage Plan' and 'Create API Key and add to Usage Plan'. Below these is a dropdown menu set to 'Results per page 100'. A table lists an API key named 'MyFirstKey (Hiorr...)'. To the right of the table are two small icons: a checkmark and an X. At the bottom of the table are navigation arrows ('<', '>') and page numbers ('Page 1'). At the very bottom are 'Back' and 'Done' buttons.

### Note

或者，若要新建 API 金鑰至將其新增至用量計劃，請選擇 Create API Key and add to Usage Plan (建立 API 金鑰並新增至用量計劃) 並遵循說明操作。

### Note

API 金鑰可以關聯到多個用量計劃。用量計劃可以關聯到多個階段。不過，特定 API 金鑰只能與每個 API 階段的一個用量計劃相關聯。

6. 若要完成用量計劃建立工作，請選擇 Done (完成)。
7. 如果您希望在用量計劃中新增更多的 API 階段，請選擇 Add API Stage (新增 API 階段) 重複前面的步驟。

## 測試用量計劃

若要測試用量計劃，您可以使用 AWS 開發套件、AWS CLI，或如 Postman 這類的 REST API 用戶端。如需使用 Postman 測試用量計劃的範例，請參閱 [測試用量計劃 \(p. 410\)](#)。

## 維護計劃用量

維護用量計劃涉及監控指定時段內已使用和剩餘的配額，以及如有需要，依指定的量擴充剩餘配額。下列程序說明如何監控和擴充配額。

## 監控已使用及剩餘的配額

1. 在 API Gateway 主導覽窗格中，選擇 Usage Plans (用量計劃)。
2. 從清單中選擇用量計劃的用量計劃。
3. 在指定的計劃中，選擇 API Keys (API 金鑰)。
4. 選擇 API 金鑰，然後在您監控的計劃中選擇 Usage (用量)，檢視 Subscriber's Traffic (訂閱者的流量)。
5. (選擇性) 依序選擇 Export (匯出)、From (開始) 日期和 To (結束) 日期、**JSON** 或 **CSV** 的匯出資料格式，然後選擇 Export (匯出)。

以下範例顯示匯出的檔案。

```
{  
  "thisPeriod": {  
    "px1KW6...qBazOJH": [  
      [  
        0,  
        5000  
      ],  
      [  
        0,  
        5000  
      ],  
      [  
        0,  
        10  
      ]  
    ],  
    "startDate": "2016-08-01",  
    "endDate": "2016-08-03"  
  }  
}
```

範例中的用量資料顯示某 API 用戶端自 2016 年 8 月 1 日到 2016 年 8 月 3 日的每日用量資料，依 API 金鑰 (px1KW6...qBazOJH) 識別。每個每日用量資料都會顯示已使用和剩餘的配額。在這個範例中，訂閱者並未用完任何分配的配額，而 API 擁有者或管理員已在第三天將剩餘的配額從 5000 降至 10。

## 擴充剩餘的配額

1. 重複前述程序的步驟 1–3。
2. 在用量計劃窗格中，從用量計劃視窗選擇 Extension (擴充)。
3. 針對 Remaining (剩餘的) 請求配額輸入數量。
4. 選擇 Save (儲存)。

## 使用 API Gateway REST API 設定 API 金鑰

若要設定 API 金鑰，請執行下列步驟：

- 設定 API 方法以要求 API 金鑰。
- 為區域中的 API 建立或匯入 API 金鑰。

在設定 API 金鑰之前，您必須已建立 API 並將它部署至階段。

如需建立和部署 API 的 REST API 呼叫，請參閱 [restapi:create](#) 和 [deployment:create](#)。

主題

- [方法中需要 API 金鑰 \(p. 408\)](#)
- [建立或匯入 API 金鑰 \(p. 408\)](#)

## 方法中需要 API 金鑰

方法中如需要 API 金鑰，請執行下列其中一項操作：

- 呼叫 `method:put` 來建立方法。在請求承載中將 `apiKeyRequired` 設為 `true`。
- 呼叫 `method:update`，設定為 `apiKeyRequired` 為 `true`。

## 建立或匯入 API 金鑰

若要建立或匯入 API 金鑰，請執行下列其中一項操作：

- 呼叫 `apikey:create`，建立 API 金鑰。
- 呼叫 `apikey:import` 從檔案匯入 API 金鑰。如需檔案格式，請參閱「[API Gateway API 金鑰檔案格式 \(p. 411\)](#)」。

建立 API 金鑰之後，您即可繼續 [使用 API Gateway CLI 和 REST API 建立、設定和測試用量計劃 \(p. 408\)](#)。

## 使用 API Gateway CLI 和 REST API 建立、設定和測試用量計劃

設定用量計劃之前，您必須已經完成下列操作：設定所選 API 的方法要求 API 金鑰、已將 API 部署或重新部署到階段，以及已建立或匯入一或多個 API 金鑰。如需詳細資訊，請參閱[使用 API Gateway REST API 設定 API 金鑰 \(p. 407\)](#)。

若要使用 API Gateway REST API 設定用量計劃，請使用下列指示，假設您已建立可新增至用量計劃的 API。

### 主題

- [遷移到預設用量計劃 \(p. 408\)](#)
- [建立用量計劃 \(p. 409\)](#)
- [使用 AWS CLI 來管理用量計劃 \(p. 409\)](#)
- [測試用量計劃 \(p. 410\)](#)

## 遷移到預設用量計劃

第一次建立用量計劃時，您可以下列內文呼叫 `account:update`，將與所選 API 金鑰相關聯的現有 API 階段遷移到用量計劃：

```
{  
  "patchOperations" : [ {  
    "op" : "add",  
    "path" : "/features",  
    "value" : "UsagePlans"  
  } ]  
}
```

如需遷移與 API 金鑰相關聯之 API 階段的詳細資訊，請參閱[在 API Gateway 主控台中遷移到預設用量計劃 \(p. 403\)](#)。

## 建立用量計劃

下列程序說明如何建立用量計劃。

### 使用 REST API 建立用量計劃

1. 呼叫 `usageplan:create` 來建立用量計劃。在承載中，指定計劃的名稱和說、相關聯的 API 階段、速率限制和配額。

記下產生的用量計劃識別符。下一個步驟需要此值。

2. 請執行下列其中一項：

- a. 呼叫 `usageplankey:create` 將 API 金鑰新增至用量計劃。在承載中指定 `keyId` 和 `keyType`。

若要將更多的 API 金鑰新增至用量計劃，請重複之前的呼叫，一次一個 API 金鑰。

- b. 呼叫 `apikey:import` 將一或多個 API 金鑰直接新增至指定的用量計劃。請求承載應該包含 API 金鑰值、相關聯的用量計劃識別符、指出用量計劃已啟用金鑰的布林值旗標，以及 API 金鑰名稱和說明 (如可能)。

以下 `apikey:import` 請求範例會將三個 API 金鑰 (識別為 `key`、`name` 和 `description`) 新增至一個用量計劃 (識別為 `usageplanIds`)：

```
POST /apikeys?mode=import&format=csv&failonwarnings=false HTTP/1.1
Host: apigateway.us-east-1.amazonaws.com
Content-Type: text/csv
Authorization: ...

key,name,description(enabled, usageplanIds)
abcdef1234ghijklmnop8901234567, importedKey_1, firstone, tRuE, n371pt
abcdef1234ghijklmnop0123456789, importedKey_2, secondone, TRUE, n371pt
abcdef1234ghijklmnop9012345678, importedKey_3, , true, n371pt
```

因此，會在 `UsagePlanKey` 中建立和新增三項 `UsagePlan` 資源。

您也可以用這種方式將 API 金鑰新增至多個用量計劃。若要這樣做，請將每個 `usageplanIds` 欄值變更成包含所選用量計劃識別符的逗號分隔字串，以一對引號 ("n371pt,m282qs" 或 'n371pt,m282qs') 括住。

#### Note

API 金鑰可以關聯到多個用量計劃。用量計劃可以關聯到多個階段。不過，特定 API 金鑰只能與每個 API 階段的一個用量計劃相關聯。

## 使用 AWS CLI 來管理用量計劃

以下程式碼範例說明如何透過呼叫 `update-usage-plan` 命令來在用量計劃中新增、移除或修改方法層級調節設定。

#### Note

請務必將 `us-east-1` 變更為您 API 的適當區域值。

若要為調節個別的資源和方法新增或取代速率限制：

```
aws apigateway --region us-east-1 update-usage-plan --usage-plan-id <planId> --patch-operations
op="replace",path="/apiStages/<apiId>:<stage>/throttle/<resourcePath>/<httpMethod>/rateLimit",value="0.1"
```

若要為調節個別的資源和方法新增或取代爆量限制：

```
aws apigateway --region us-east-1 update-usage-plan --usage-plan-id <planId>
--patch-operations op="replace",path="/apiStages/<apiId>:<stage>/
throttle/<resourcePath>/<httpMethod>/burstLimit",value="1"
```

若要為個別的資源和方法移除方法層級調節設定：

```
aws apigateway --region us-east-1 update-usage-plan --usage-plan-id <planId>
--patch-operations op="remove",path="/apiStages/<apiId>:<stage>/
throttle/<resourcePath>/<httpMethod>",value=""
```

要移除 API 的所有方法層級調節設定：

```
aws apigateway --region us-east-1 update-usage-plan --usage-plan-id <planId> --patch-
operations op="remove",path="/apiStages/<apiId>:<stage>/throttle ",value=""
```

此處為使用 Pet Store 範例 API 的範例：

```
aws apigateway --region us-east-1 update-usage-plan --usage-plan-id <planId> --patch-
operations
op="replace",path="/apiStages/<apiId>:<stage>/throttle",value='"/
pets/GET\":{\\"rateLimit\\":1.0,\\"burstLimit\\":1},\"//GET\":{\\"rateLimit\\":1.0,\\"burstLimit
\\":1}"}'
```

## 測試用量計劃

以在 [教學課程：匯入範例來建立 REST API \(p. 40\)](#) 中建立的 PetStore API 為例。假設 API 設定使用 Hiorr45VR...c4GJc 的 API 金鑰。下列步驟說明如何測試用量計劃。

### 測試您的用量計劃

- 在用量計劃中，對 API (例如 GET) 的 Pets 資源 (/pets) 提出 ?type=...&page=... 請求，查詢參數為 xbvxlpjch：

```
GET /testStage/pets?type=dog&page=1 HTTP/1.1
x-api-key: Hiorr45VR...c4GJc
Content-Type: application/x-www-form-urlencoded
Host: xbvxlpjch.execute-api.ap-southeast-1.amazonaws.com
X-Amz-Date: 20160803T001845Z
Authorization: AWS4-HMAC-SHA256 Credential={access_key_ID}/20160803/ap-southeast-1/
execute-api/aws4_request, SignedHeaders=content-type;host;x-amz-date;x-api-key,
Signature={sigv4_hash}
```

### Note

您必須向 API Gateway 的 execute-api 元件提交此請求，並在所需的 x-api-key 標頭中提供需要的 API 金鑰 (例如 Hiorr45VR...c4GJc)。

成功的回應會傳回 200 OK 狀態碼和包含後端請求結果的承載。如果您忘記設定 x-api-key 標頭或以錯誤的金鑰設定它，您會收到 403 Forbidden 回應。不過，如果您並未設定方法要求 API 金鑰，您可能會收得 200 OK 回應，無論您是否正確設定 x-api-key 標頭，而且會略過用量計劃的調節和配額限制。

有時，當發生 API Gateway 無法強制用量計劃調節限制或請求配額的內部錯誤時，API Gateway 會提供請求，但不會套用用量計劃指定的調節限制或配額。但是，它會在 CloudWatch 中記錄 Usage Plan check failed due to an internal error 錯誤訊息。您可以忽略這類偶發的錯誤。

## API Gateway API 金鑰檔案格式

API Gateway 可以從逗號分隔值 (CSV) 格式的外部檔案匯入 API 金鑰，然後建立已匯入金鑰與一或多個用量計劃的關聯性。匯入的檔案必須包含 Name 與 Key 欄位。欄標頭名稱不區分大小寫，各欄順序任意排列，如下範例所示：

```
Key, name  
apikey1234abcdefghij0123456789, MyFirstApiKey
```

Key 值必須介於 30 到 128 個字元之間。

API 金鑰檔案也可以有 Description、Enabled 或 UsagePlanIds 欄位，如下例所示：

```
Name, key, description, Enabled, usageplanIds  
MyFirstApiKey, apikey1234abcdefghij0123456789, An imported key, TRUE, c7y23b
```

當金鑰與多個用量計劃相關聯時，UsagePlanIds 值是用量計劃 ID 的逗號分隔字串，以一對雙引號或單引號括住，如下例所示：

```
Enabled, Name, key, UsageplanIds  
true, MyFirstApiKey, apikey1234abcdefghij0123456789, "c7y23b, glvrsr"
```

允許無法識別的欄位，但會被忽略。預設值為空白字串或 true 布林值。

相同的 API 金鑰可以匯入多次，以最新的版本覆寫前一個。兩個 API 金鑰如有相同的 key 值，則它們即完全相同。

## 記錄 API Gateway 中的 REST API

為了協助客戶了解及使用您的 API，您應該記錄 API。為了協助您記錄 API，API Gateway 可讓您新增及更新個別 API 實體的說明內容，這是 API 開發程序不可或缺的一部分。API Gateway 可存放來源內容，並讓您封存不同版本的文件。您可以建立文件版本與 API 階段的關聯、將特定階段的文件快照匯出到外部 OpenAPI 檔案，並在發佈文件時分發檔案。

若要記錄您的 API，您可以呼叫 [API Gateway REST API](#)、使用其中一個 [AWS 開發套件](#) 或 [適用於 API Gateway 的 AWS CLI](#)，或是使用 API Gateway 主控台。此外，您可以匯入或匯出外部 OpenAPI 檔案中定義的文件組件。在說明如何記錄 API 之前，我們將示範如何在 API Gateway 中表示 API 文件。

### 主題

- [API Gateway 中的 API 文件表示 \(p. 411\)](#)
- [使用 API Gateway 主控台記錄 API \(p. 419\)](#)
- [使用 API Gateway 主控台發佈 API 文件 \(p. 427\)](#)
- [使用 API Gateway REST API 記錄 API \(p. 427\)](#)
- [使用 API Gateway REST API 發佈 API 文件 \(p. 442\)](#)
- [匯入 API 文件 \(p. 448\)](#)
- [控制 API 文件的存取 \(p. 452\)](#)

## API Gateway 中的 API 文件表示

API Gateway API 文件是由與特定 API 實體相關聯的個別文件組件所組成，這些實體包括 API、資源、方法、請求、回應、訊息參數（即路徑、查詢、標頭），以及授權方與模型。

在 API Gateway 中，文件組件是以 [DocumentationPart](#) 資源表示。整個 API 文件是以 [DocumentationParts](#) 集合表示。

記錄 API 需要建立 [DocumentationPart](#) 執行個體、將其新增至 [DocumentationParts](#) 集合，並隨著 API 改進維護不同版本的文件組件。

#### 主題

- [文件組件 \(p. 412\)](#)
- [文件版本 \(p. 418\)](#)

## 文件組件

[DocumentationPart](#) 資源是存放適用於個別 API 實體之文件內容的 JSON 物件。其 `properties` 欄位包含對應鍵值對的文件內容。其 `location` 屬性識別相關聯的 API 實體。

內容對應的成形取決於身為 API 開發人員的您。金鑰/值對的值可以是字串、數值、布林值、物件或陣列。`location` 物件的成形取決於目標實體類型。

[DocumentationPart](#) 資源支援內容繼承：API 實體的文件內容適用於 API 實體的子系。如需子實體與內容繼承定義的詳細資訊，請參閱[從較一般規格的 API 實體繼承內容 \(p. 413\)](#)。

### 文件組件的位置

[DocumentationPart](#) 執行個體的 `location` 屬性可辨識相關聯內容適用的 API 實體。該 API 實體可以是 API Gateway REST API 資源，例如 [RestApi](#)、[Resource](#)、[Method](#)、[MethodResponse](#)、[Authorizer](#) 或 [Model](#)。該實體也可以是訊息參數，例如 URL 路徑參數、查詢字串參數、請求或回應標頭參數、請求或回應內文，或是回應狀態碼。

若要指定 API 實體，請將 `location` 物件的 `type` 屬性設定為 [API](#)、[AUTHORIZER](#)、[MODEL](#)、[RESOURCE](#)、[METHOD](#)、[PATH\\_PARAMETER](#)、[QUERY\\_PARAMETER](#)、[REQUEST\\_HEADER](#)、[RESPONSE\\_BODY](#) 其中之一。

根據 API 實體的 `type`，您可以指定其他 `location` 屬性，包括 [method](#)、[name](#)、[path](#) 與 [statusCode](#)。並非所有屬性對指定的 API 實體都有效。例如，`type`、`path`、`name` 與 `statusCode` 是 [RESPONSE](#) 實體的有效屬性；但只有 `type` 與 `path` 是 [RESOURCE](#) 實體的有效 `location` 屬性。在指定 API 實體之 `location` 的 [DocumentationPart](#) 中包含無效的欄位是錯誤的。

並非所有有效的 `location` 欄位都是必要的。例如，`type` 是對所有 API 實體有效且必要的 `location` 欄位。不過，`method`、`path` 與 `statusCode` 對 [RESPONSE](#) 實體有效，但不是必要的屬性。未明確指定時，有效的 `location` 欄位會假設其預設值。預設 `path` 值是 `/`，即 API 的根資源。`method` 或 `statusCode` 的預設值是 `*`，分別表示任何方法或狀態碼值。

### 文件組件的內容

`properties` 值已編碼為 JSON 字串。`properties` 值包含為符合您的文件需求所選擇的任何資訊。例如，以下是有效的內容對應：

```
{  
  "info": {  
    "description": "My first API with Amazon API Gateway."  
  },  
  "x-custom-info" : "My custom info, recognized by OpenAPI.",  
  "my-info" : "My custom info not recognized by OpenAPI."  
}
```

雖然 API Gateway 接受任何有效的 JSON 字串做為內容對應，但內容屬性會分為兩類來處理：[OpenAPI](#) 可識別的屬性與無法識別的屬性。在上述範例中，[OpenAPI](#) 將 `info`、`description` 與 `x-custom-info` 識別為標準 [OpenAPI](#) 物件、屬性或延伸。反之，`my-info` 不相容於 [OpenAPI](#) 規格。API Gateway 會將

與 OpenAPI 相容的內容屬性從相關聯的 DocumentationPart 執行個體傳播到 API 實體定義中。API Gateway 不會將不相容的內容屬性傳播到 API 實體定義中。

在另一個範例中，DocumentationPart 實體的目標 Resource 如下：

```
{  
    "location" : {  
        "type" : "RESOURCE",  
        "path": "/pets"  
    },  
    "properties" : {  
        "summary" : "The /pets resource represents a collection of pets in PetStore.",  
        "description": "... a child resource under the root...",  
    }  
}
```

在本例中，type 與 path 都是可識別 RESOURCE 類型目標的有效欄位。對於根資源 (/)，您可以省略 path 欄位。

```
{  
    "location" : {  
        "type" : "RESOURCE"  
    },  
    "properties" : {  
        "description" : "The root resource with the default path specification."  
    }  
}
```

這與下列 DocumentationPart 執行個體相同：

```
{  
    "location" : {  
        "type" : "RESOURCE",  
        "path": "/"  
    },  
    "properties" : {  
        "description" : "The root resource with an explicit path specification"  
    }  
}
```

## 從較一般規格的 API 實體繼承內容

選用之 location 欄位的預設值提供 API 實體的模式說明。使用 location 物件的預設值，您可以將 properties 對應中的一般說明新增至具有這種 location 模式類型的 DocumentationPart 執行個體。API Gateway 會從泛型 API 實體的 DocumentationPart 中擷取適用的 OpenAPI 文件屬性，再插入其 location 欄位符合一般 location 模式或符合確切值的特定 API 實體中，除非該特定實體已有相關聯的 DocumentationPart 執行個體。此行為也稱為來自較一般規格之 API 實體的內容繼承。

內容繼承不適用於特定 API 實體類型。如需詳細資訊，請參閱下列資料表。

當 API 實體符合多個 DocumentationPart 的位置模式時，實體會繼承具有最高優先順序與明確性之位置欄位的文件組件。優先順序為 path > statusCode。與 path 欄位比對時，API Gateway 會選擇具有最明確路徑值的實體。下表顯示一些範例。

案例	path	statusCode	name	備註	
1	/pets	*	id	與此位置	

案例	path	statusCode	name	備註	
				模式相關聯的文件會由符合位置模式的實體繼承。	
2	/pets	200	id	當案例 1 與案例 2 相符時，由於案例 2 比案例 1 更明確，因此與此位置模式相關聯的文件會由符合位置模式的實體繼承。	

案例	path	statusCode	name	備註
3	/pets/ petId	*	id	當案例 1、2 與 3 相符時，由於案例 3 的優先順序比案例 2 高且比案例 1 更明確，因此與此位置模式相關聯的文件會由符合位置模式的實體繼承。

以下是較泛型的 DocumentationPart 執行個體與較明確的執行個體的另一個對比範例。下列一般錯誤訊息 "Invalid request error" 會插入 400 錯誤回應的 OpenAPI 定義，除非遭到覆寫。

```
{
  "location" : {
    "type" : "RESPONSE",
    "statusCode": "400"
  },
  "properties" : {
    "description" : "Invalid request error."
  }
}
```

透過下列覆寫，400 資源上任何方法的 /pets 回應會改為具有 "Invalid petId specified" 的說明。

```
{
  "location" : {
    "type" : "RESPONSE",
    "path": "/pets",
    "statusCode": "400"
  },
}
```

```

    "properties" : {
        "description" : "Invalid petId specified."
    }
}

```

## DocumentationPart 的有效位置欄位

下表顯示與指定 API 實體類型相關聯之 DocumentationPart 資源的有效與必要欄位，以及適用的預設值。

API 實體	有效的位置欄位	必要的位置欄位	預設欄位值	是否可繼承內容
API	<pre>{     "location": {         "type": "API"     },     ... }</pre>	type	不適用	否
Resource	<pre>{     "location": {         "type": "RESOURCE",         "path": "resource_path"     },     ... }</pre>	type	path 的預設值為 /。	否
方法	<pre>{     "location": {         "type": "METHOD",         "path": "resource_path",         "method": "http_verb"     },     ... }</pre>	type	path 與 method 的預設值分別為 / 與 *。	是，符合 path 的字首，且符合任何值的 method。
查詢參數	<pre>{     "location": {         "type": "QUERY_PARAMETER",         "path": "resource_path",         "method": "HTTP_verb",         "name": "query_parameter_name"     },     ... }</pre>	type	path 與 method 的預設值分別為 / 與 *。	是，符合 path 的字首，且符合 method 的確切值。
請求內文	<pre>{     "location": {         "type": "REQUEST_BODY",         ... }</pre>	type	path 與 method 的預設值分別為 / 與 *。	是，符合 path 的字首，且符合 method 的確切值。

API 實體	有效的位置欄位	必要的位置欄位	預設欄位值	是否可繼承內容
	<pre>     "path":      "<b>resource_path</b>",      "method":      "<b>http_verb</b>"      },      ...  } </pre>			
請求標頭參數	<pre> {     "location": {         "type":          "REQUEST_HEADER",          "path":          "<b>resource_path</b>",          "method":          "<b>HTTP_verb</b>",          "name":          "<b>header_name</b>"      },      ... } </pre>	type, name	path 與 method 的預設值分別為 / 與 *。	是，符合 path 的字首，且符合 method 的確切值。
請求路徑參數	<pre> {     "location": {         "type":          "PATH_PARAMETER",          "path": "<b>resource</b>/  <b>path_parameter_name</b>",          "method":          "<b>HTTP_verb</b>",          "name":          "<b>path_parameter_name</b>"      },      ... } </pre>	type, name	path 與 method 的預設值分別為 / 與 *。	是，符合 path 的字首，且符合 method 的確切值。
回應	<pre> {     "location": {         "type": "RESPONSE",          "path":          "<b>resource_path</b>",          "method":          "<b>http_verb</b>",          "statusCode":          "<b>status_code</b>"      },      ... } </pre>	type	path、method 與 statusCode 的預設值分別為 /、* 與 *。	是，符合 path 的字首，且符合 method 與 statusCode 的確切值。

API 實體	有效的位置欄位	必要的位置欄位	預設欄位值	是否可繼承內容
回應標頭	<pre>{     "location": {         "type": "RESPONSE_HEADER",         "path": "resource_path",         "method": "http_verb",         "statusCode": "status_code",         "name": "header_name"     },     ... }</pre>	type, name	path、method 與 statusCode 的預設值分別為 /、* 與 *。	是，符合 path 的字首，且符合 method 與 statusCode 的確切值。
回應內文	<pre>{     "location": {         "type": "RESPONSE_BODY",         "path": "resource_path",         "method": "http_verb",         "statusCode": "status_code"     },     ... }</pre>	type	path、method 與 statusCode 的預設值分別為 /、* 與 *。	是，符合 path 的字首，且符合 method 與 statusCode 的確切值。
授權方	<pre>{     "location": {         "type": "AUTHORIZER",         "name": "authorizer_name"     },     ... }</pre>	type	不適用	否
模型	<pre>{     "location": {         "type": "MODEL",         "name": "model_name"     },     ... }</pre>	type	不適用	否

## 文件版本

文件版本是 API 之 [DocumentationParts](#) 集合的快照，並以版本識別符標記。發佈 API 的文件需要建立文件版本、將其與 API 階段建立關聯，並將特定階段版本的 API 文件匯出到外部 OpenAPI 檔案。在 API Gateway 中，文件快照是以 [DocumentationVersion](#) 資源表示。

當您更新 API 時，您會建立新版的 API。在 API Gateway 中，您會使用 [DocumentationVersions](#) 集合維護所有文件版本。

## 使用 API Gateway 主控台記錄 API

在本節中，我們會說明如何使用 API Gateway 主控台來建立及維護 API 的文件組件。

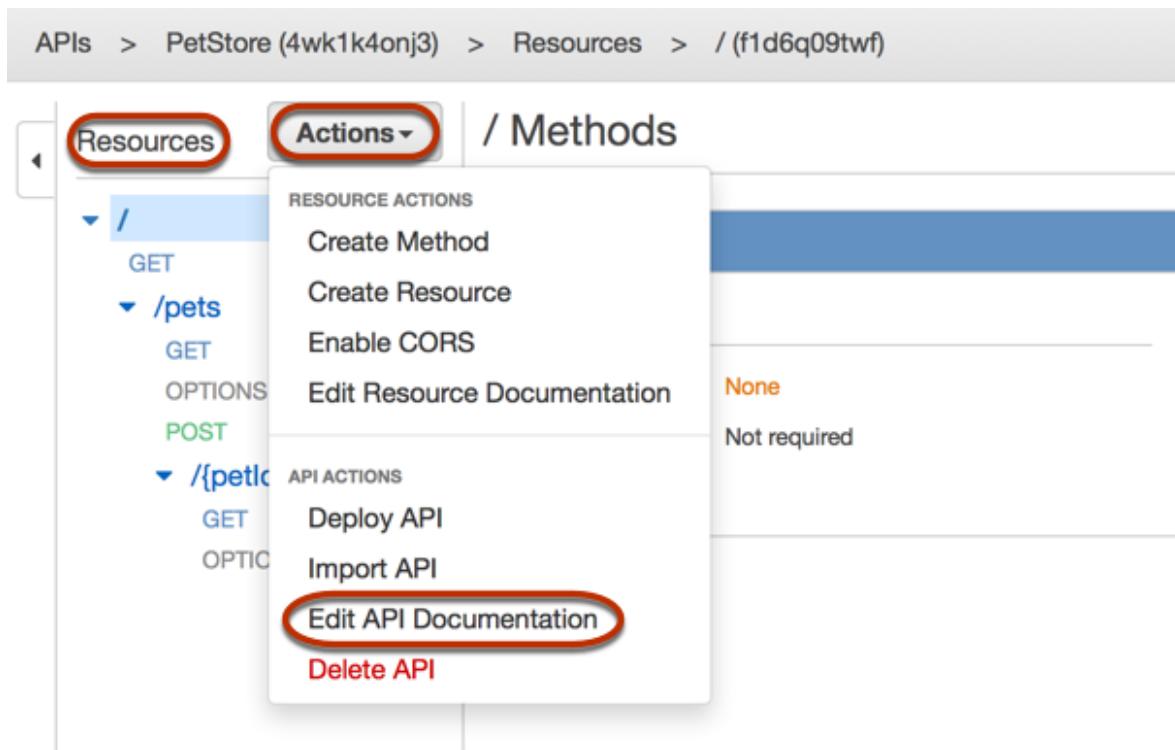
建立及編輯 API 文件的必要條件是您必須已建立 API。在本節中，我們以 [PetStore API](#) 為例。若要使用 API Gateway 主控台建立 API，請遵循教學課程：[匯入範例來建立 REST API \(p. 40\)](#)中的說明進行。

### 主題

- [記錄 API 實體 \(p. 419\)](#)
- [記錄 RESOURCE 實體 \(p. 422\)](#)
- [記錄 METHOD 實體 \(p. 422\)](#)
- [記錄 QUERY\\_PARAMETER 實體 \(p. 423\)](#)
- [記錄 PATH\\_PARAMETER 實體 \(p. 424\)](#)
- [記錄 REQUEST\\_HEADER 實體 \(p. 424\)](#)
- [記錄 REQUEST\\_BODY 實體 \(p. 425\)](#)
- [記錄 RESPONSE 實體 \(p. 425\)](#)
- [記錄 RESPONSE\\_HEADER 實體 \(p. 425\)](#)
- [記錄 RESPONSE\\_BODY 實體 \(p. 425\)](#)
- [記錄 MODEL 實體 \(p. 426\)](#)
- [記錄 AUTHORIZER 實體 \(p. 427\)](#)

## 記錄 API 實體

若要新增 API 實體的文件組件，請從 PetStore API 中選擇 Resources (資源)。選擇 Actions → Edit API Documentation (動作 → 編輯 API 文件) 選單項目。



如果未針對 API 建立文件組件，則會取得文件組件的 properties 對應編輯器。在文字編輯器中輸入下列 properties 對應，然後選擇 Save (儲存) 以建立文件組件。

```
{  
    "info": {  
        "description": "Your first API Gateway API.",  
        "contact": {  
            "name": "John Doe",  
            "email": "john.doe@api.com"  
        }  
    }  
}
```

#### Note

您不需要將 properties 對應編碼為 JSON 字串。API Gateway 主控台會為您字串化 JSON 物件。

## Documentation

Provide your API documentation in JSON format in the form below.

Type API

```
1  [
2    "info": {
3      "description" : "Your first API Gateway API.",
4      "contact": {
5        "name": "John Doe",
6        "email": "john.doe@api.com"
7      }
8    }
9  ]
```

**Close** **Save**

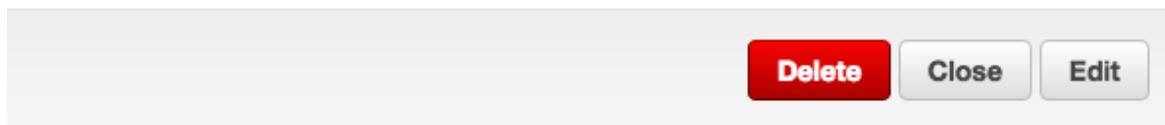
如果已建立文件組件，則會先取得 properties 對應檢視器，如下所示。

## Documentation

Specify your documentation part in JSON format in the form below. For more information, see the [Documentation Parts Documentation](#).

Type API

```
{  
  "info": {  
    "description": "Your first API Gateway API.",  
    "contact": {  
      "name": "John Doe",  
      "email": "john.doe@api.com"  
    }  
  }  
}
```



選擇 Edit (編輯) 會顯示如上所示的 `properties` 對應編輯器。

## 記錄 RESOURCE 實體

若要新增或編輯 API 根資源的文件組件，請選擇 Resource (資源) 樹狀目錄下的 /，然後選擇 Actions → Edit Resource Documentation (動作 → 編輯資源文件) 選單項目。

如果未針對此實體建立任何文件組件，則會取得 Documentation (文件) 視窗。在編輯器中，輸入有效的 `properties` 對應。然後選擇 Save (儲存) 與 Close (關閉)。

```
{  
  "description": "The PetStore's root resource."  
}
```

如果已針對 RESOURCE 實體定義文件組件，則會取得文件檢視器。選擇 Edit (編輯) 以開啟 Documentation (文件) 編輯器。修改現有的 `properties` 對應。選擇 Save (儲存)，然後選擇 Close (關閉)。

如果需要，請重複這些步驟來新增其他 RESOURCE 實體的文件組件。

## 記錄 METHOD 實體

若要新增或編輯 METHOD 實體的文件 (以根資源上的 GET 方法為例)，請選擇 / 資源下的 GET，然後選擇 Actions → Edit Method Documentation (動作 → 編輯方法文件) 選單項目。

若是新的文件組件，請在 Documentation (文件) 視窗的 Documentation (文件) 編輯器中，輸入下列 `properties` 對應。然後選擇 Save (儲存) 與 Close (關閉)。

```
{  
    "tags" : [ "pets" ],  
    "description" : "PetStore HTML web page containing API usage information"  
}
```

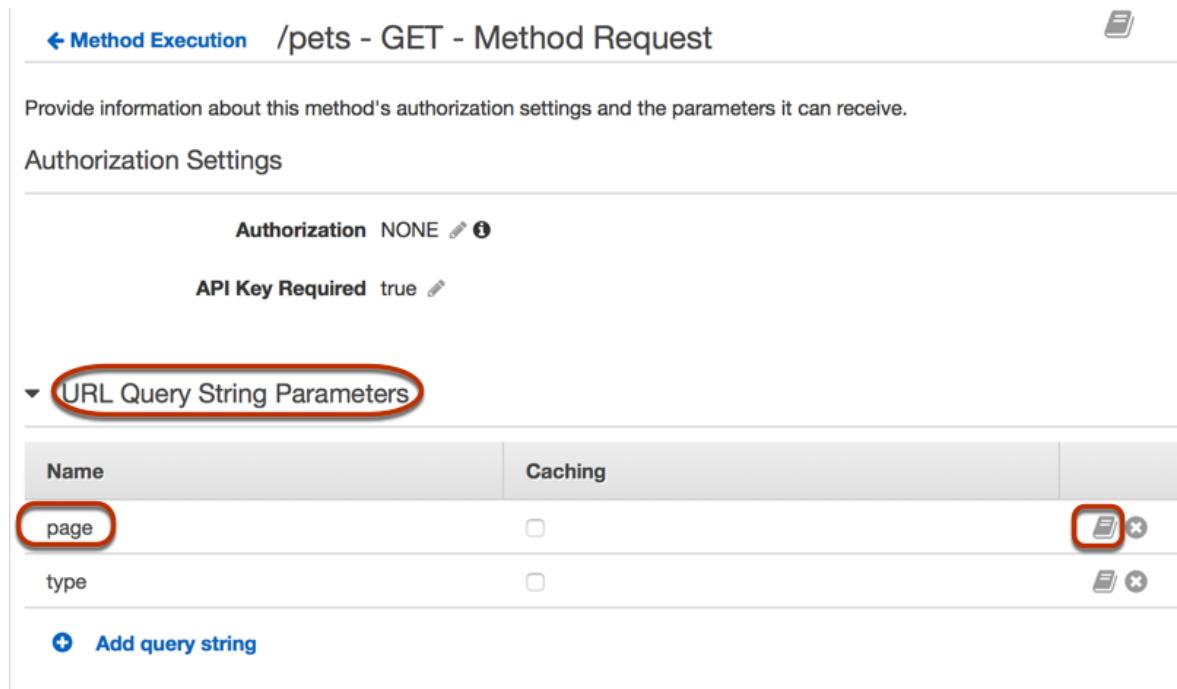
若是現有的文件，請從 Documentation (文件) 檢視器中選擇 Edit (編輯)。在 Documentation (文件) 編輯器中編輯文件內容，然後選擇 Save (儲存)。選擇 Close (關閉)。

從 Documentation (文件) 檢視器中，您也可以刪除文件組件。

如果需要，請重複這些步驟來新增其他方法的文件組件。

## 記錄 QUERY\_PARAMETER 實體

若要新增或編輯請求查詢參數的文件組件 (以 GET /pets?type=...&page=... 方法為例)，請從 Resources (資源) 樹狀目錄中，選擇 /pets 下的 GET。在 Method Execution (方法執行) 視窗中，選擇 Method Request (方法請求)。展開 URL Query String Parameters (URL 查詢字串參數) 區段。選擇 page 查詢參數，然後選擇書本圖示來開啟 Documentation (文件) 檢視器或編輯器。



← Method Execution /pets - GET - Method Request

Provide information about this method's authorization settings and the parameters it can receive.

Authorization Settings

Authorization NONE

API Key Required true

URL Query String Parameters

Name	Caching
page	<input type="checkbox"/>
type	<input type="checkbox"/>

+ Add query string

或者，您也可以從主要導覽窗格中，選擇 PetStore API 下的 Documentation (文件)。然後針對 Type (類型)，選擇 Query Parameter。在 PetStore 範例 API 中，這會顯示 page 與 type 查詢參數的文件組件。

Documentation

Create Documentation Part

Import Documentation

Publish Documentation

Add documentation to help developers understand how to interact with your API. Documentation parts can be shared across multiple resources and methods by specifying a wildcard value (\*) for method or status code, eg. documentation for a 200 response can be used in multiple locations. You can also import documentation by supplying a Swagger definition file, and publish documentation to a stage. For more information, reference the [documentation](#).

The screenshot shows the 'Documentation' page with a search bar and filters for Type (Query Parameter), Path (/pets), Method (All), and Name. Two entries are listed:

- Query Parameter /pets**:
  - Method: GET
  - Name: page (highlighted with a red circle)
  - Description: { "description": "Page number of results to return." }
- Query Parameter /pets**:
  - Method: GET
  - Name: type (highlighted with a red circle)
  - Description: { "description": "The type of pet to retrieve" }

對於已定義其他方法之查詢參數的 API，您可以針對 Path (路徑) 指定受影響資源的 path、從 Method (方法) 中選擇所需的 HTTP 方法，或是在 Name (名稱) 中輸入查詢參數名稱，來篩選您的選取範圍。

例如，選擇 page 查詢參數。選擇 Edit (編輯) 修改現有的文件。選擇 Save (儲存) 儲存變更。

若要新增 QUERY\_PARAMETER 實體的文件組件，請選擇 Create Documentation Part (建立文件組件)。針對 Type (類型)，選擇 Query Parameter。在 Path (路徑) 中，輸入資源路徑 (例如 /pets)。針對 Method (方法) 選擇 HTTP 動詞 (例如 GET)。在文字編輯器中，輸入 properties 說明。然後選擇 Save (儲存)。

如果需要，請重複這些步驟來新增其他請求查詢參數的文件組件。

## 記錄 PATH\_PARAMETER 實體

若要新增或編輯路徑參數的文件，請前往路徑參數所指定資源上之方法的 Method Request (方法請求)。展開 Request Paths (請求路徑) 區段。選擇路徑參數的書本圖示來開啟 Documentation (文件) 檢視器或編輯器。新增或修改文件組件的屬性。

或者，從主要導覽窗格中，選擇 PetStore API 下的 Documentation (文件)。針對 Type (類型)，選擇 Path Parameter。在清單中的路徑參數上，選擇 Edit (編輯)。修改內容，然後選擇 Save (儲存)。

若要新增未列出之路徑參數的文件，請選擇 Create Documentation Part (建立文件組件)。針對 Type (類型)，選擇 Path Parameter (路徑參數)。在 Path (路徑) 中設定資源路徑，從 Method (方法) 中選擇一個方法，然後在 Name (名稱) 中設定路徑參數名稱。新增文件的屬性，然後選擇 Save (儲存)。

如果需要，請重複這些步驟來新增或編輯其他路徑參數的文件組件。

## 記錄 REQUEST\_HEADER 實體

若要新增或編輯請求標頭的文件，請前往具有標頭參數之方法的 Method Request (方法請求)。展開 HTTP Request Headers (HTTP 請求標頭) 區段。選擇標頭的書本圖示來開啟 Documentation (文件) 檢視器或編輯器。新增或修改文件組件的屬性。

或者，從主要導覽窗格中，選擇 API 下的 Documentation (文件)。然後針對 Type (類型)，選擇 Request Header。在列出的請求標頭上選擇 Edit (編輯) 來變更文件。若要新增未列出之請求標頭的文件，請選擇 Create Documentation Part (建立文件組件)。針對 Type (類型)，選擇 Request Header (請求標頭)。在 Path (路徑) 中指定資源路徑。針對 Method (方法) 選擇一個方法。在 Name (名稱) 中，輸入標頭名稱。然後新增並儲存 properties 對應。

如果需要，請重複這些步驟來新增或編輯其他請求標頭的文件組件。

## 記錄 REQUEST\_BODY 實體

若要新增或編輯請求內文的文件，請前往方法的 Method Request (方法請求)。選擇 Request Body (請求內文) 的書本圖示，依序開啟 Documentation (文件) 檢視器與編輯器。新增或修改文件組件的屬性。

或者，從主要導覽窗格中，選擇 API 下的 Documentation (文件)。然後針對 Type (類型)，選擇 Request Body。在列出的請求內文上選擇 Edit (編輯) 來變更文件。若要新增未列出之請求內文的文件，請選擇 Create Documentation Part (建立文件組件)。針對 Type (類型)，選擇 Request Body。在 Path (路徑) 中設定資源路徑。針對 Method (方法) 選擇 HTTP 動詞。然後新增並儲存 properties 對應。

如果需要，請重複這些步驟來新增或編輯其他請求內文的文件組件。

## 記錄 RESPONSE 實體

若要新增或編輯回應的文件，請前往方法的 Method Response (方法回應)。選擇 Method Response (方法回應) 的書本圖示，依序開啟 Documentation (文件) 檢視器與編輯器。新增或修改文件組件的屬性。

Provide information about this method's response types, their headers and content types.

HTTP Status
200

**Add Response**

或者，從主要導覽窗格中，選擇 API 下的 Documentation (文件)。然後針對 Type (類型)，選擇 Response (status code)。在指定 HTTP 狀態碼的列出回應上選擇 Edit (編輯) 來變更文件。若要新增未列出之回應內文的文件，請選擇 Create Documentation Part (建立文件組件)。針對 Type (類型) 選擇 Response (status code) (回應 (狀態碼))。在 Path (路徑) 中設定資源路徑。針對 Method (方法) 選擇 HTTP 動詞。在 Status Code (狀態碼) 中，輸入 HTTP 狀態碼。然後新增並儲存文件組件屬性。

如果需要，請重複這些步驟來新增或編輯其他回應的文件組件。

## 記錄 RESPONSE\_HEADER 實體

若要新增或編輯回應標頭的文件，請前往方法的 Method Response (方法回應)。展開指定 HTTP 狀態的回應區段。在 Response Headers for **Status Code** (StatusCode 的回應標頭) 下，選擇回應標頭的書本圖示，依序開啟 Documentation (文件) 檢視器與編輯器。新增或修改文件組件的屬性。

或者，從主要導覽窗格中，選擇 API 下的 Documentation (文件)。然後針對 Type (類型)，選擇 Response Header。在列出的回應標頭上選擇 Edit (編輯) 來變更文件。若要新增未列出之回應標頭的文件，請選擇 Create Documentation Part (建立文件組件)。針對 Type (類型) 選擇 Response Header (回應標頭)。在 Path (路徑) 中設定資源路徑。針對 Method (方法) 選擇 HTTP 動詞。在 Status Code (狀態碼) 中，輸入 HTTP 狀態碼。在 Name (名稱) 中，輸回應標頭名稱。然後新增並儲存文件組件屬性。

如果需要，請重複這些步驟來新增或編輯其他回應標頭的文件組件。

## 記錄 RESPONSE\_BODY 實體

若要新增或編輯回應內文的文件，請前往方法的 Method Response (方法回應)。展開指定 HTTP 狀態的回應區段。選擇 Response Body for **Status Code** (StatusCode 的回應內文) 的書本圖示，依序開啟 Documentation (文件) 檢視器與編輯器。新增或修改文件組件的屬性。

或者，從主要導覽窗格中，選擇 API 下的 Documentation (文件)。然後針對 Type (類型)，選擇 Response Body。在列出的回應內文上選擇 Edit (編輯) 來變更文件。若要新增未列出之回應內文的文件，請選擇 Create Documentation Part (建立文件組件)。針對 Type (類型) 選擇 Response Body (回應內文)。在 Path (路徑) 中設定資源路徑。針對 Method (方法) 選擇 HTTP 動詞。在 Status Code (狀態碼) 中，輸入 HTTP 狀態碼。然後新增並儲存文件組件屬性。

如果需要，請重複這些步驟來新增或編輯其他回應內文的文件組件。

## 記錄 MODEL 實體

記錄 MODEL 實體需要建立及管理模型與每個模型之 DocumentationPart 的 properties 執行個體。例如，每個 API 隨附的 Error 模型預設具有下列結構描述定義：

```
{  
  "$schema": "http://json-schema.org/draft-04/schema#",  
  "title": "Error Schema",  
  "type": "object",  
  "properties": {  
    "message": { "type": "string" }  
  }  
}
```

且需要兩個 DocumentationPart 執行個體，一個用於 Model，另一個用於其 message 屬性：

```
{  
  "location": {  
    "type": "MODEL",  
    "name": "Error"  
  },  
  "properties": {  
    "title": "Error Schema",  
    "description": "A description of the Error model"  
  }  
}
```

和

```
{  
  "location": {  
    "type": "MODEL",  
    "name": "Error.message"  
  },  
  "properties": {  
    "description": "An error message."  
  }  
}
```

匯出 API 之後，DocumentationPart 的屬性會覆寫原始結構描述中的值。

若要新增或編輯模型的文件，請在主要導覽窗格中，前往 API 的 Models (模型)。選擇所列出之模型名稱的書本圖示，依序開啟 Documentation (文件) 檢視器與編輯器。新增或修改文件組件的屬性。

或者，從主要導覽窗格中，選擇 API 下的 Documentation (文件)。然後針對 Type (類型)，選擇 Model。在列出的模型上選擇 Edit (編輯) 來變更文件。若要新增未列出之模型的文件，請選擇 Create Documentation Part (建立文件組件)。針對 Type (類型) 選擇 Model (模型)。在 Name (名稱) 中，指定模型的名稱。然後新增並儲存文件組件屬性。

如果需要，請重複這些步驟來新增或編輯其他模型的文件組件。

## 記錄 AUTHORIZER 實體

若要新增或編輯授權方的文件，請在主要導覽窗格中，前往 API 的 Authorizers (授權方)。選擇所列出之授權方的書本圖示，依序開啟 Documentation (文件) 檢視器與編輯器。新增或修改文件組件的屬性。

或者，從主要導覽窗格中，選擇 API 下的 Documentation (文件)。然後針對 Type (類型)，選擇 Authorizer。在列出的授權方上選擇 Edit (編輯) 來變更文件。若要新增未列出之授權方的文件，請選擇 Create Documentation Part (建立文件組件)。針對 Type (類型) 選擇 Authorizer (授權方)。在 Name (名稱) 中，指定授權方的名稱。然後新增並儲存文件組件屬性。

如果需要，請重複這些步驟來新增或編輯其他授權方的文件組件。

若要新增授權方的文件組件，請選擇 Create Documentation Part (建立文件組件)。針對 Type (類型) 選擇 Authorizer (授權方)。針對授權方之 Name (名稱) 的有效 location 欄位，指定一個值。

新增並儲存 properties 對應編輯器中的文件內容。

如果需要，請重複這些步驟來新增另一個授權方的文件組件。

## 使用 API Gateway 主控台發佈 API 文件

下列程序說明如何發佈文件版本。

### 使用 API Gateway 主控台發佈文件版本

1. 從 API Gateway 主控台的主要導覽窗格中，選擇 API 的 Documentation (文件)。
2. 在 Documentation (文件) 窗格中，選擇 Publish Documentation (發佈文件)。
3. 設定發佈：
  - a. 針對 Stage (階段) 選擇可用的名稱。
  - b. 在 Version (版本) 中輸入版本識別符，例如 1.0.0。
  - c. (選擇性) 在 Description (說明) 中，提供發佈的相關說明。
4. 選擇 Publish (發佈)。

您現在可以將文件匯出到外部 OpenAPI 檔案，繼續下載已發佈的文件。

## 使用 API Gateway REST API 記錄 API

在本節中，我們會說明如何使用 API Gateway REST API 來建立及維護 API 的文件組件。

建立及編輯 API 的文件之前，請先建立 API。在本節中，我們以 PetStore API 為例。若要使用 API Gateway 主控台建立 API，請遵循教學課程：匯入範例來建立 REST API (p. 40) 中的說明進行。

### 主題

- [記錄 API 實體 \(p. 428\)](#)
- [記錄 RESOURCE 實體 \(p. 429\)](#)
- [記錄 METHOD 實體 \(p. 431\)](#)
- [記錄 QUERY\\_PARAMETER 實體 \(p. 433\)](#)
- [記錄 PATH\\_PARAMETER 實體 \(p. 434\)](#)
- [記錄 REQUEST\\_BODY 實體 \(p. 435\)](#)
- [記錄 REQUEST\\_HEADER 實體 \(p. 436\)](#)

- [記錄 RESPONSE 實體 \(p. 437\)](#)
- [記錄 RESPONSE\\_HEADER 實體 \(p. 438\)](#)
- [記錄 AUTHORIZER 實體 \(p. 439\)](#)
- [記錄 MODEL 實體 \(p. 440\)](#)
- [更新文件組件 \(p. 441\)](#)
- [列出文件組件 \(p. 442\)](#)

## 記錄 API 實體

若要新增 API 的文件，請新增 API 實體的 DocumentationPart 資源：

```
POST /restapis/restapi_id/documentation/parts HTTP/1.1
Host: apigateway.region.amazonaws.com
Content-Type: application/json
X-Amz-Date: YYYYMMDDTtttttz
Authorization: AWS4-HMAC-SHA256 Credential=access_key_id/YYYYMMDD/region/
apigateway/aws4_request, SignedHeaders=content-length;content-type;host;x-amz-date,
Signature=sigv4_secret

{
    "location" : {
        "type" : "API"
    },
    "properties": "{\n\t\"info\": {\n\t\t\"description\" : \"Your first API with Amazon API\nGateway.\n\t}\n}"
}
```

如果成功，此操作會傳回 201 Created 回應，包含新建立的 DocumentationPart 執行個體的承載。例如：

```
{
    ...
    "id": "s2e5xf",
    "location": {
        "path": null,
        "method": null,
        "name": null,
        "statusCode": null,
        "type": "API"
    },
    "properties": "{\n\t\"info\": {\n\t\t\"description\" : \"Your first API with Amazon API\nGateway.\n\t}\n}"
}
```

如果已新增文件組件，則會傳回 409 Conflict 回應，其中包含錯誤訊息 Documentation part already exists for the specified location: type 'API'."。在此情況下，您必須呼叫 documentationpart:update 操作。

```
PATCH /restapis/4wk1k4onj3/documentation/parts/part_id HTTP/1.1
Host: apigateway.region.amazonaws.com
Content-Type: application/json
X-Amz-Date: YYYYMMDDTtttttz
Authorization: AWS4-HMAC-SHA256 Credential=access_key_id/YYYYMMDD/region/
apigateway/aws4_request, SignedHeaders=content-length;content-type;host;x-amz-date,
Signature=sigv4_secret

{
    "patchOperations" : [ {
```

```
"op" : "replace",
"path" : "/properties",
"value" : "{\n\t\"info\": {\n\t\t\"description\" : \"Your first API with Amazon API\nGateway.\n\"}\n}"
}
```

成功的回應會傳回 200 OK 狀態碼與包含更新的 DocumentationPart 執行個體的承載。

## 記錄 RESOURCE 實體

若要新增 API 根資源的文件，請新增對應至 Resource 資源的 DocumentationPart 資源：

```
POST /restapis/restapi_id/documentation/parts HTTP/1.1
Host: apigateway.region.amazonaws.com
Content-Type: application/json
X-Amz-Date: YYYYMMDDTTtttttZ
Authorization: AWS4-HMAC-SHA256 Credential=access_key_id/YYYYMMDD/region,
apigateway/aws4_request, SignedHeaders=content-length;content-type;host;x-amz-date,
Signature=sigv4_secret

{
  "location" : {
    "type" : "RESOURCE",
  },
  "properties" : "{\n\t\"description\" : \"The PetStore root resource.\n\"}\n"
}
```

如果成功，此操作會傳回 201 Created 回應，包含新建立的 DocumentationPart 執行個體的承載。例如：

```
{
  "_links": {
    "curies": {
      "href": "http://docs.aws.amazon.com/apigateway/latest/developerguide/restapi-
documentationpart-{rel}.html",
      "name": "documentationpart",
      "templated": true
    },
    "self": {
      "href": "/restapis/4wk1k4onj3/documentation/parts/p76vqo"
    },
    "documentationpart:delete": {
      "href": "/restapis/4wk1k4onj3/documentation/parts/p76vqo"
    },
    "documentationpart:update": {
      "href": "/restapis/4wk1k4onj3/documentation/parts/p76vqo"
    }
  },
  "id": "p76vqo",
  "location": {
    "path": "/",
    "method": null,
    "name": null,
    "statusCode": null,
    "type": "RESOURCE"
  },
  "properties": "{\n\t\"description\" : \"The PetStore root resource.\n\"}\n"
}
```

未指定資源路徑時，資源會假設是根資源。您可以將 "path": "/" 新增至 properties 來明確指定。

若要建立 API 子資源的文件，請新增對應至 Resource 資源的 DocumentationPart 資源：

```
POST /restapis/restapi_id/documentation/parts HTTP/1.1
Host: apigateway.region.amazonaws.com
Content-Type: application/json
X-Amz-Date: YYYYMMDDTttttttZ
Authorization: AWS4-HMAC-SHA256 Credential=access_key_id/YYYYMMDD/region/
apigateway/aws4_request, SignedHeaders=content-length;content-type;host;x-amz-date,
Signature=sigv4_secret

{
    "location" : {
        "type" : "RESOURCE",
        "path" : "/pets"
    },
    "properties": "{\n\t\"description\" : \"A child resource under the root of PetStore.\n\"}\n"
}
```

如果成功，此操作會傳回 201 Created 回應，包含新建立的 DocumentationPart 執行個體的承載。例如：

```
{
    "_links": {
        "curies": {
            "href": "http://docs.aws.amazon.com/apigateway/latest/developerguide/restapi-
documentationpart-{rel}.html",
            "name": "documentationpart",
            "templated": true
        },
        "self": {
            "href": "/restapis/4wk1k4onj3/documentation/parts/qcht86"
        },
        "documentationpart:delete": {
            "href": "/restapis/4wk1k4onj3/documentation/parts/qcht86"
        },
        "documentationpart:update": {
            "href": "/restapis/4wk1k4onj3/documentation/parts/qcht86"
        }
    },
    "id": "qcht86",
    "location": {
        "path": "/pets",
        "method": null,
        "name": null,
        "statusCode": null,
        "type": "RESOURCE"
    },
    "properties": "{\n\t\"description\" : \"A child resource under the root of PetStore.\n\"}\n"
}
```

若要新增路徑參數所指定之子資源的文件，請新增映射至 Resource 資源的 DocumentationPart 資源：

```
POST /restapis/restapi_id/documentation/parts HTTP/1.1
Host: apigateway.region.amazonaws.com
Content-Type: application/json
X-Amz-Date: YYYYMMDDTttttttZ
Authorization: AWS4-HMAC-SHA256 Credential=access_key_id/YYYYMMDD/region/
apigateway/aws4_request, SignedHeaders=content-length;content-type;host;x-amz-date,
Signature=sigv4_secret

{
```

```
"location" : {
    "type" : "RESOURCE",
    "path" : "/pets/{petId}"
},
"properties": "{\n\t\"description\" : \"A child resource specified by the petId path\nparameter.\n\"}\n}
```

如果成功，此操作會傳回 201 Created 回應，包含新建立的 DocumentationPart 執行個體的承載。例如：

```
{
  "_links": {
    "curies": {
      "href": "http://docs.aws.amazon.com/apigateway/latest/developerguide/restapi-
documentationpart-{rel}.html",
      "name": "documentationpart",
      "templated": true
    },
    "self": {
      "href": "/restapis/4wk1k4onj3/documentation/parts/k6fpwb"
    },
    "documentationpart:delete": {
      "href": "/restapis/4wk1k4onj3/documentation/parts/k6fpwb"
    },
    "documentationpart:update": {
      "href": "/restapis/4wk1k4onj3/documentation/parts/k6fpwb"
    }
  },
  "id": "k6fpwb",
  "location": {
    "path": "/pets/{petId}",
    "method": null,
    "name": null,
    "statusCode": null,
    "type": "RESOURCE"
  },
  "properties": "{\n\t\"description\" : \"A child resource specified by the petId path\nparameter.\n\"}\n}
```

#### Note

RESOURCE 實體的 DocumentationPart 執行個體不可由其任何子資源繼承。

## 記錄 METHOD 實體

若要新增 API 方法的文件，請新增對應至 Method 資源的 DocumentationPart 資源：

```
POST /restapis/restapi_id/documentation/parts HTTP/1.1
Host: apigateway.region.amazonaws.com
Content-Type: application/json
X-Amz-Date: YYYYMMDDTTtttttZ
Authorization: AWS4-HMAC-SHA256 Credential=access_key_id/YYYYMMDD/region/
apigateway/aws4_request, SignedHeaders=content-length;content-type;host;x-amz-date,
Signature=sigv4_secret

{
  "location" : {
    "type" : "METHOD",
    "path" : "/pets",
    "method" : "GET"
  }
}
```

```
    },
    "properties": "{\n\t\"summary\" : \"List all pets.\n}\n}
```

如果成功，此操作會傳回 201 Created 回應，包含新建立的 DocumentationPart 執行個體的承載。例如：

```
{
  "_links": {
    "curies": {
      "href": "http://docs.aws.amazon.com/apigateway/latest/developerguide/restapi-documentationpart-{rel}.html",
      "name": "documentationpart",
      "templated": true
    },
    "self": {
      "href": "/restapis/4wk1k4onj3/documentation/parts/o64jbj"
    },
    "documentationpart:delete": {
      "href": "/restapis/4wk1k4onj3/documentation/parts/o64jbj"
    },
    "documentationpart:update": {
      "href": "/restapis/4wk1k4onj3/documentation/parts/o64jbj"
    }
  },
  "id": "o64jbj",
  "location": {
    "path": "/pets",
    "method": "GET",
    "name": null,
    "statusCode": null,
    "type": "METHOD"
  },
  "properties": "{\n\t\"summary\" : \"List all pets.\n}\n}
```

如果成功，此操作會傳回 201 Created 回應，包含新建立的 DocumentationPart 執行個體的承載。例如：

```
{
  "_links": {
    "curies": {
      "href": "http://docs.aws.amazon.com/apigateway/latest/developerguide/restapi-documentationpart-{rel}.html",
      "name": "documentationpart",
      "templated": true
    },
    "self": {
      "href": "/restapis/4wk1k4onj3/documentation/parts/o64jbj"
    },
    "documentationpart:delete": {
      "href": "/restapis/4wk1k4onj3/documentation/parts/o64jbj"
    },
    "documentationpart:update": {
      "href": "/restapis/4wk1k4onj3/documentation/parts/o64jbj"
    }
  },
  "id": "o64jbj",
  "location": {
    "path": "/pets",
    "method": "GET",
    "name": null,
    "statusCode": null,
```

```
        "type": "METHOD"
    },
    "properties": "{\n\t\"summary\" : \"List all pets.\n}"
}
```

如果上述請求中未指定 `location.method` 欄位，則會假設萬用字元 ANY 所表示的 \* 方法。

若要更新 METHOD 實體的文件內容，請呼叫 `documentationpart:update` 操作，提供新的 properties 映射：

```
PATCH /restapis/4wk1k4onj3/documentation/parts/part_id HTTP/1.1
Host: apigateway.region.amazonaws.com
Content-Type: application/json
X-Amz-Date: YYYYMMDDTTtttttZ
Authorization: AWS4-HMAC-SHA256 Credential=access_key_id/YYYYMMDD/region,
apigateway/aws4_request, SignedHeaders=content-length;content-type;host;x-amz-date,
Signature=sigv4_secret

{
    "patchOperations" : [ {
        "op" : "replace",
        "path" : "/properties",
        "value" : "{\n\t\"tags\" : [ \"pets\" ],\n\t\"summary\" : \"List all pets.\n\""
    } ]
}
```

成功的回應會傳回 200 OK 狀態碼與包含更新的 DocumentationPart 執行個體的承载。例如：

```
{
    "_links": {
        "curies": {
            "href": "http://docs.aws.amazon.com/apigateway/latest/developerguide/restapi-documentationpart-{rel}.html",
            "name": "documentationpart",
            "templated": true
        },
        "self": {
            "href": "/restapis/4wk1k4onj3/documentation/parts/o64bjbj"
        },
        "documentationpart:delete": {
            "href": "/restapis/4wk1k4onj3/documentation/parts/o64bjbj"
        },
        "documentationpart:update": {
            "href": "/restapis/4wk1k4onj3/documentation/parts/o64bjbj"
        }
    },
    "id": "o64bjbj",
    "location": {
        "path": "/pets",
        "method": "GET",
        "name": null,
        "statusCode": null,
        "type": "METHOD"
    },
    "properties": "{\n\t\"tags\" : [ \"pets\" ],\n\t\"summary\" : \"List all pets.\n\""
}
```

## 記錄 QUERY\_PARAMETER 實體

若要新增請求查詢參數的文件，請新增映射至 `QUERY_PARAMETER` 類型的 `DocumentationPart` 資源，其有效欄位為 `path` 與 `name`。

```
POST /restapis/restapi_id/documentation/parts HTTP/1.1
Host: apigateway.region.amazonaws.com
Content-Type: application/json
X-Amz-Date: YYYYMMDDTtttttz
Authorization: AWS4-HMAC-SHA256 Credential=access_key_id/YYYYMMDD/region/
apigateway/aws4_request, SignedHeaders=content-length;content-type;host;x-amz-date,
Signature=sigv4_secret

{
    "location" : {
        "type" : "QUERY_PARAMETER",
        "path" : "/pets",
        "method" : "GET",
        "name" : "page"
    },
    "properties": "{\n\t\"description\" : \"Page number of results to return.\"\\n\"}"
}
```

如果成功，此操作會傳回 201 Created 回應，包含新建立的 DocumentationPart 執行個體的承載。例如：

```
{
    "_links": {
        "curies": {
            "href": "http://docs.aws.amazon.com/apigateway/latest/developerguide/restapi-
documentationpart-{rel}.html",
            "name": "documentationpart",
            "templated": true
        },
        "self": {
            "href": "/restapis/4wk1k4onj3/documentation/parts/h9ht5w"
        },
        "documentationpart:delete": {
            "href": "/restapis/4wk1k4onj3/documentation/parts/h9ht5w"
        },
        "documentationpart:update": {
            "href": "/restapis/4wk1k4onj3/documentation/parts/h9ht5w"
        }
    },
    "id": "h9ht5w",
    "location": {
        "path": "/pets",
        "method": "GET",
        "name": "page",
        "statusCode": null,
        "type": "QUERY_PARAMETER"
    },
    "properties": "{\n\t\"description\" : \"Page number of results to return.\"\\n\"}"
}
```

properties 實體之文件組件的 QUERY\_PARAMETER 對應可由其 QUERY\_PARAMETER 子實體之一繼承。例如，如果您在 treats 後面新增 /pets/{petId} 資源、在 GET 上啟用 /pets/{petId}/treats 方法，並公開 page 查詢參數，則其子查詢參數會從 DocumentationPart 方法的相同名稱查詢參數繼承 properties 的 GET /pets 對應，除非您將 DocumentationPart 資源明確新增至 page 方法的 GET /pets/{petId}/treats 查詢參數。

## 記錄 PATH\_PARAMETER 實體

若要新增路徑參數的文件，請新增 PATH\_PARAMETER 實體的 DocumentationPart 資源。

```
POST /restapis/restapi_id/documentation/parts HTTP/1.1
```

```
Host: apigateway.region.amazonaws.com
Content-Type: application/json
X-Amz-Date: YYYYMMDDTtttttZ
Authorization: AWS4-HMAC-SHA256 Credential=access_key_id/YYYYMMDD/region/
apigateway/aws4_request, SignedHeaders=content-length;content-type;host;x-amz-date,
Signature=sigv4_secret

{
    "location" : {
        "type" : "PATH_PARAMETER",
        "path" : "/pets/{petId}",
        "method" : "*",
        "name" : "petId"
    },
    "properties": "{\n\t"description\" : \"The id of the pet to retrieve.\n}"
}
```

如果成功，此操作會傳回 201 Created 回應，包含新建立的 DocumentationPart 執行個體的承載。例如：

```
{
    "_links": {
        "curies": {
            "href": "http://docs.aws.amazon.com/apigateway/latest/developerguide/restapi-
documentationpart-{rel}.html",
            "name": "documentationpart",
            "templated": true
        },
        "self": {
            "href": "/restapis/4wk1k4onj3/documentation/parts/ckpgog"
        },
        "documentationpart:delete": {
            "href": "/restapis/4wk1k4onj3/documentation/parts/ckpgog"
        },
        "documentationpart:update": {
            "href": "/restapis/4wk1k4onj3/documentation/parts/ckpgog"
        }
    },
    "id": "ckpgog",
    "location": {
        "path": "/pets/{petId}",
        "method": "*",
        "name": "petId",
        "statusCode": null,
        "type": "PATH_PARAMETER"
    },
    "properties": "{\n\t"description\" : \"The id of the pet to retrieve.\n}"
}
```

## 記錄 REQUEST\_BODY 實體

若要新增請求內文的文件，請新增請求內文的 DocumentationPart 資源。

```
POST /restapis/restapi_id/documentation/parts HTTP/1.1
Host: apigateway.region.amazonaws.com
Content-Type: application/json
X-Amz-Date: YYYYMMDDTtttttZ
Authorization: AWS4-HMAC-SHA256 Credential=access_key_id/YYYYMMDD/region/
apigateway/aws4_request, SignedHeaders=content-length;content-type;host;x-amz-date,
Signature=sigv4_secret

{
```

```
"location" : {
    "type" : "REQUEST_BODY",
    "path" : "/pets",
    "method" : "POST"
},
"properties": "{\n\t\"description\" : \"A Pet object to be added to PetStore.\n\"}\n}
```

如果成功，此操作會傳回 201 Created 回應，包含新建立的 DocumentationPart 執行個體的承載。例如：

```
{
  "_links": {
    "curies": {
      "href": "http://docs.aws.amazon.com/apigateway/latest/developerguide/restapi-documentationpart-{rel}.html",
      "name": "documentationpart",
      "templated": true
    },
    "self": {
      "href": "/restapis/4wk1k4onj3/documentation/parts/kgmfr1"
    },
    "documentationpart:delete": {
      "href": "/restapis/4wk1k4onj3/documentation/parts/kgmfr1"
    },
    "documentationpart:update": {
      "href": "/restapis/4wk1k4onj3/documentation/parts/kgmfr1"
    }
  },
  "id": "kgmfr1",
  "location": {
    "path": "/pets",
    "method": "POST",
    "name": null,
    "statusCode": null,
    "type": "REQUEST_BODY"
  },
  "properties": "{\n\t\"description\" : \"A Pet object to be added to PetStore.\n\"}\n}
```

## 記錄 REQUEST\_HEADER 實體

若要新增請求標頭的文件，請新增請求標頭的 DocumentationPart 資源。

```
POST /restapis/restapi_id/documentation/parts HTTP/1.1
Host: apigateway.region.amazonaws.com
Content-Type: application/json
X-Amz-Date: YYYYMMDDTtttttZ
Authorization: AWS4-HMAC-SHA256 Credential=access_key_id/YYYYMMDD/region/apigateway/aws4_request, SignedHeaders=content-length;content-type;host;x-amz-date, Signature=sigv4_secret

{
  "location" : {
    "type" : "REQUEST_HEADER",
    "path" : "/pets",
    "method" : "GET",
    "name" : "x-my-token"
  },
  "properties": "{\n\t\"description\" : \"A custom token used to authorization the method invocation.\n\"}\n}
```

如果成功，此操作會傳回 201 Created 回應，包含新建立的 DocumentationPart 執行個體的承載。例如：

```
{  
  "_links": {  
    "curies": {  
      "href": "http://docs.aws.amazon.com/apigateway/latest/developerguide/restapi-documentationpart-{rel}.html",  
      "name": "documentationpart",  
      "templated": true  
    },  
    "self": {  
      "href": "/restapis/4wk1k4onj3/documentation/parts/h0m3uf"  
    },  
    "documentationpart:delete": {  
      "href": "/restapis/4wk1k4onj3/documentation/parts/h0m3uf"  
    },  
    "documentationpart:update": {  
      "href": "/restapis/4wk1k4onj3/documentation/parts/h0m3uf"  
    }  
  },  
  "id": "h0m3uf",  
  "location": {  
    "path": "/pets",  
    "method": "GET",  
    "name": "x-my-token",  
    "statusCode": null,  
    "type": "REQUEST_HEADER"  
  },  
  "properties": "{\n\t\"description\" : \"A custom token used to authorization the method invocation.\"\\n\"}  
}
```

## 記錄 RESPONSE 實體

若要新增某個狀態碼回應的文件，請新增對應至 MethodResponse 資源的 DocumentationPart 資源。

```
POST /restapis/restapi_id/documentation/parts HTTP/1.1  
Host: apigateway.region.amazonaws.com  
Content-Type: application/json  
X-Amz-Date: YYYYMMDDTTtttttZ  
Authorization: AWS4-HMAC-SHA256 Credential=access_key_id/YYYYMMDD/region,  
apigateway/aws4_request, SignedHeaders=content-length;content-type;host;x-amz-date,  
Signature=sigv4_secret  
  
{  
  "location": {  
    "path": "/",  
    "method": "*",  
    "name": null,  
    "statusCode": "200",  
    "type": "RESPONSE"  
  },  
  "properties": "{\n    \"description\" : \"Successful operation.\"\\n\"}  
}
```

如果成功，此操作會傳回 201 Created 回應，包含新建立的 DocumentationPart 執行個體的承載。例如：

```
{  
  "_links": {
```

```
"self": {
    "href": "/restapis/4wk1k4onj3/documentation/parts/lattew"
},
"documentationpart:delete": {
    "href": "/restapis/4wk1k4onj3/documentation/parts/lattew"
},
"documentationpart:update": {
    "href": "/restapis/4wk1k4onj3/documentation/parts/lattew"
}
},
"id": "lattew",
"location": {
    "path": "/",
    "method": "*",
    "name": null,
    "statusCode": "200",
    "type": "RESPONSE"
},
"properties": "{\n    \"description\" : \"Successful operation.\n}"
}
```

## 記錄 RESPONSE\_HEADER 實體

若要新增回應標頭的文件，請新增回應標頭的 DocumentationPart 資源。

```
POST /restapis/restapi_id/documentation/parts HTTP/1.1
Host: apigateway.region.amazonaws.com
Content-Type: application/json
X-Amz-Date: YYYYMMDDTttttttZ
Authorization: AWS4-HMAC-SHA256 Credential=access_key_id/YYYYMMDD/region/
apigateway/aws4_request, SignedHeaders=content-length;content-type;host;x-amz-date,
Signature=sigv4_secret

"location": {
    "path": "/",
    "method": "GET",
    "name": "Content-Type",
    "statusCode": "200",
    "type": "RESPONSE_HEADER"
},
"properties": "{\n    \"description\" : \"Media type of request\"\n}"
}
```

如果成功，此操作會傳回 201 Created 回應，包含新建立的 DocumentationPart 執行個體的承載。例如：

```
{
    "_links": {
        "curies": {
            "href": "http://docs.aws.amazon.com/apigateway/latest/developerguide/restapi-
documentationpart-{rel}.html",
            "name": "documentationpart",
            "templated": true
        },
        "self": {
            "href": "/restapis/4wk1k4onj3/documentation/parts/fev7j7"
        }
    },
    "documentationpart:delete": {
        "href": "/restapis/4wk1k4onj3/documentation/parts/fev7j7"
    },
    "documentationpart:update": {
        "href": "/restapis/4wk1k4onj3/documentation/parts/fev7j7"
    }
}
```

```
        },
        "id": "fev7j7",
        "location": {
            "path": "/",
            "method": "GET",
            "name": "Content-Type",
            "statusCode": "200",
            "type": "RESPONSE_HEADER"
        },
        "properties": "{\n            \"description\" : \"Media type of request\"\n        }"
    }
```

此 Content-Type 回應標頭的文件是 API 之任何回應的預設 Content-Type 標頭文件。

## 記錄 AUTHORIZER 實體

若要新增 API 授權方的文件，請新增映射至指定授權方的 DocumentationPart 資源。

```
POST /restapis/restapi_id/documentation/parts HTTP/1.1
Host: apigateway.region.amazonaws.com
Content-Type: application/json
X-Amz-Date: YYYYMMDDTtttttZ
Authorization: AWS4-HMAC-SHA256 Credential=access_key_id/YYYYMMDD/region,
apigateway/aws4_request, SignedHeaders=content-length;content-type;host;x-amz-date,
Signature=sigv4_secret

{
    "location" : {
        "type" : "AUTHORIZER",
        "name" : "myAuthorizer"
    },
    "properties": "{\n        \"description\" : \"Authorizes invocations of configured methods.\n    }"
}
```

如果成功，此操作會傳回 201 Created 回應，包含新建立的 DocumentationPart 執行個體的承載。例如：

```
{
    "_links": {
        "curies": {
            "href": "http://docs.aws.amazon.com/apigateway/latest/developerguide/restapi-
documentationpart-{rel}.html",
            "name": "documentationpart",
            "templated": true
        },
        "self": {
            "href": "/restapis/4wk1k4onj3/documentation/parts/pw3qw3"
        }
    },
    "documentationpart:delete": {
        "href": "/restapis/4wk1k4onj3/documentation/parts/pw3qw3"
    },
    "documentationpart:update": {
        "href": "/restapis/4wk1k4onj3/documentation/parts/pw3qw3"
    }
},
"id": "pw3qw3",
"location": {
    "path": null,
    "method": null,
    "name": "myAuthorizer",
    "statusCode": null,
```

```
        "type": "AUTHORIZER"
},
"properties": "{\n\t\"description\": \"Authorizes invocations of configured methods.\n\"\\n\""
}
```

#### Note

AUTHORIZER 實體的 [DocumentationPart](#) 執行個體不可由其任何子資源繼承。

## 記錄 MODEL 實體

記錄 MODEL 實體需要建立及管理模型與每個模型之 [DocumentationPart](#) 的 properties 執行個體。例如，每個 API 隨附的 Error 模型預設具有下列結構描述定義：

```
{
  "$schema" : "http://json-schema.org/draft-04/schema#",
  "title" : "Error Schema",
  "type" : "object",
  "properties" : {
    "message" : { "type" : "string" }
  }
}
```

且需要兩個 [DocumentationPart](#) 執行個體，一個用於 Model，另一個用於其 message 屬性：

```
{
  "location": {
    "type": "MODEL",
    "name": "Error"
  },
  "properties": {
    "title": "Error Schema",
    "description": "A description of the Error model"
  }
}
```

和

```
{
  "location": {
    "type": "MODEL",
    "name": "Error.message"
  },
  "properties": {
    "description": "An error message."
  }
}
```

匯出 API 之後，[DocumentationPart](#) 的屬性會覆寫原始結構描述中的值。

若要新增 API 模型的文件，請新增映射至指定模型的 [DocumentationPart](#) 資源。

```
POST /restapis/restapi\_id/documentation/parts HTTP/1.1
Host: apigateway.region.amazonaws.com
Content-Type: application/json
X-Amz-Date: YYYYMMDDTtttttZ
```

```
Authorization: AWS4-HMAC-SHA256 Credential=access_key_id/YYYYMMDD/region/  
apigateway/aws4_request, SignedHeaders=content-length;content-type;host;x-amz-date,  
Signature=sigv4_secret

{
    "location" : {
        "type" : "MODEL",
        "name" : "Pet"
    },
    "properties": "{\n\t\"description\" : \"Data structure of a Pet object.\n\"}"
}
```

如果成功，此操作會傳回 201 Created 回應，包含新建立的 DocumentationPart 執行個體的承載。例如：

```
{
    "_links": {
        "curries": {
            "href": "http://docs.aws.amazon.com/apigateway/latest/developerguide/restapi-documentationpart-{rel}.html",
            "name": "documentationpart",
            "templated": true
        },
        "self": {
            "href": "/restapis/4wk1k4onj3/documentation/parts/lkn4uq"
        },
        "documentationpart:delete": {
            "href": "/restapis/4wk1k4onj3/documentation/parts/lkn4uq"
        },
        "documentationpart:update": {
            "href": "/restapis/4wk1k4onj3/documentation/parts/lkn4uq"
        }
    },
    "id": "lkn4uq",
    "location": {
        "path": null,
        "method": null,
        "name": "Pet",
        "statusCode": null,
        "type": "MODEL"
    },
    "properties": "{\n\t\"description\" : \"Data structure of a Pet object.\n\"}"
}
```

請重複相同的步驟，來建立任何模型屬性的 DocumentationPart 執行個體。

#### Note

MODEL 實體的 DocumentationPart 執行個體不可由其任何子資源繼承。

## 更新文件組件

若要更新任何 API 實體類型的文件組件，請提交指定組件識別符之 DocumentationPart 執行個體的 PATCH 請求，將現有的 properties 映射取代為新的映射。

```
PATCH /restapis/4wk1k4onj3/documentation/parts/part_id HTTP/1.1
Host: apigateway.region.amazonaws.com
Content-Type: application/json
X-Amz-Date: YYYYMMDDTtttttz
Authorization: AWS4-HMAC-SHA256 Credential=access_key_id/YYYYMMDD/region/  
apigateway/aws4_request, SignedHeaders=content-length;content-type;host;x-amz-date,  
Signature=sigv4_secret
```

```
{  
    "patchOperations" : [ {  
        "op" : "replace",  
        "path" : "RESOURCE_PATH",  
        "value" : "NEW_properties_VALUE_AS_JSON_STRING"  
    } ]  
}
```

成功的回應會傳回 200 OK 狀態碼與包含更新的 DocumentationPart 執行個體的承載。

您可以透過單一 PATCH 請求來更新多個文件組件。

## 列出文件組件

若要列出任何 API 實體類型的文件組件，請在 DocumentationParts 集合提交 GET 請求。

```
GET /restapis/restapi_id/documentation/parts HTTP/1.1  
Host: apigateway.region.amazonaws.com  
Content-Type: application/json  
X-Amz-Date: YYYYMMDDTTtttttZ  
Authorization: AWS4-HMAC-SHA256 Credential=access_key_id/YYYYMMDD/region/  
apigateway/aws4_request, SignedHeaders=content-length;content-type;host;x-amz-date,  
Signature=sigv4_secret
```

成功的回應會傳回 200 OK 狀態碼與包含可用的 DocumentationPart 執行個體的承載。

## 使用 API Gateway REST API 發佈 API 文件

若要發佈 API 的文件，請建立、更新或取得文件快照，然後建立文件快照與 API 階段的關聯。建立文件快照時，您也可以同時將其與 API 階段建立關聯。

### 主題

- [建立文件快照並與 API 階段建立關聯。 \(p. 442\)](#)
- [建立文件快照 \(p. 443\)](#)
- [更新文件快照 \(p. 443\)](#)
- [取得文件快照 \(p. 443\)](#)
- [建立文件快照與 API 階段的關聯 \(p. 444\)](#)
- [下載與階段相關聯的文件快照 \(p. 444\)](#)

## 建立文件快照並與 API 階段建立關聯。

若要建立 API 文件組件的快照，並同時將其與 API 階段建立關聯，請提交下列 POST 請求：

```
POST /restapis/restapi_id/documentation/versions HTTP/1.1  
Host: apigateway.region.amazonaws.com  
Content-Type: application/json  
X-Amz-Date: YYYYMMDDTTtttttZ  
Authorization: AWS4-HMAC-SHA256 Credential=access_key_id/YYYYMMDD/region/  
apigateway/aws4_request, SignedHeaders=content-length;content-type;host;x-amz-date,  
Signature=sigv4_secret  
  
{  
    "documentationVersion" : "1.0.0",
```

```
        "stageName": "prod",
        "description" : "My API Documentation v1.0.0"
    }
```

如果成功，此操作會傳回 200 OK 回應，包含新建立的 DocumentationVersion 執行個體做為承載。

或者，您可以建立文件快照，但不先將其與 API 階段建立關聯，之後再呼叫 `restapi:update` 來建立快照與指定 API 階段的關聯。您也可以更新或查詢現有的文件快照，然後更新其階段關聯。我們將在接下來的四節中示範這些步驟。

## 建立文件快照

若要建立 API 文件組件的快照，請建立新的 DocumentationVersion 資源，並將其新增至 API 的 DocumentationVersions 集合：

```
POST /restapis/restapi_id/documentation/versions HTTP/1.1
Host: apigateway.region.amazonaws.com
Content-Type: application/json
X-Amz-Date: YYYYMMDDTtttttZ
Authorization: AWS4-HMAC-SHA256 Credential=access_key_id/YYYYMMDD/region/
apigateway/aws4_request, SignedHeaders=content-length;content-type;host;x-amz-date,
Signature=sigv4_secret

{
    "documentationVersion" : "1.0.0",
    "description" : "My API Documentation v1.0.0"
}
```

如果成功，此操作會傳回 200 OK 回應，包含新建立的 DocumentationVersion 執行個體做為承載。

## 更新文件快照

您只能透過修改對應 DocumentationVersion 資源的 description 屬性，來更新文件快照。下列範例示範如何更新文件快照的說明，該快照是以其版本識別符 `version` 來識別，例如 1.0.0。

```
PATCH /restapis/restapi_id/documentation/versions/version HTTP/1.1
Host: apigateway.region.amazonaws.com
Content-Type: application/json
X-Amz-Date: YYYYMMDDTtttttZ
Authorization: AWS4-HMAC-SHA256 Credential=access_key_id/YYYYMMDD/region/
apigateway/aws4_request, SignedHeaders=content-length;content-type;host;x-amz-date,
Signature=sigv4_secret

{
    "patchOperations": [
        {
            "op": "replace",
            "path": "/description",
            "value": "My API for testing purposes."
        }
    ]
}
```

如果成功，此操作會傳回 200 OK 回應，包含更新的 DocumentationVersion 執行個體做為承載。

## 取得文件快照

若要取得文件快照，請對指定的 DocumentationVersion 資源提交 GET 請求。下列範例示範如何取得指定版本識別符 1.0.0 的文件快照。

```
GET /restapis/<restapi_id>/documentation/versions/1.0.0 HTTP/1.1
Host: apigateway.region.amazonaws.com
Content-Type: application/json
X-Amz-Date: YYYYMMDDTTtttttZ
Authorization: AWS4-HMAC-SHA256 Credential=access_key_id/YYYYMMDD/region/
apigateway/aws4_request, SignedHeaders=content-length;content-type;host;x-amz-date,
Signature=sigv4_secret
```

## 建立文件快照與 API 階段的關聯

若要發佈 API 文件，請建立文件快照與 API 階段的關聯。您必須已建立 API 階段，才能建立文件版本與該階段的關聯。

若要使用 [API Gateway REST API](#) 建立文件快照與 API 階段的關聯，請呼叫 `stage:update` 操作，在 `stage.documentationVersion` 屬性上設定所需的文件版本：

```
PATCH /restapis/RESTAPI_ID/stages/STAGE_NAME
Host: apigateway.region.amazonaws.com
Content-Type: application/json
X-Amz-Date: YYYYMMDDTTtttttZ
Authorization: AWS4-HMAC-SHA256 Credential=access_key_id/YYYYMMDD/region/
apigateway/aws4_request, SignedHeaders=content-length;content-type;host;x-amz-date,
Signature=sigv4_secret

{
  "patchOperations": [
    {
      "op": "replace",
      "path": "/documentationVersion",
      "value": "VERSION_IDENTIFIER"
    }
  ]
}
```

## 下載與階段相關聯的文件快照

將文件組件版本與階段建立關聯之後，您可以使用 API Gateway 主控台、API Gateway REST API、其中一個開發套件或適用於 API Gateway 的 AWS CLI，將文件組件與 API 實體定義一起匯出到外部檔案。其程序與匯出 API 相同。匯出的檔案格式可以是 JSON 或 YAML。

使用 API Gateway REST API，您可以明確設定 `extension=documentation,integrations,authorizers` 查詢參數，在 API 匯出中包含 API 組件、API 整合與授權方。匯出 API 時，預設會包含文件組件，但會排除整合與授權方。API 匯出的預設輸出適用於文件分發。

若要使用 API Gateway REST API 將 API 文件匯出到外部 JSON OpenAPI 檔案，請提交下列 GET 請求：

```
GET /restapis/<restapi_id>/stages/<stage_name>/exports/swagger?extensions=documentation
HTTP/1.1
Accept: application/json
Host: apigateway.region.amazonaws.com
Content-Type: application/json
X-Amz-Date: YYYYMMDDTTtttttZ
Authorization: AWS4-HMAC-SHA256 Credential=access_key_id/YYYYMMDD/region/
apigateway/aws4_request, SignedHeaders=content-length;content-type;host;x-amz-date,
Signature=sigv4_secret
```

在本例中，`x-amazon-apigateway-documentation` 物件包含文件組件，而 API 實體定義包含 OpenAPI 支援的文件屬性。此輸出不包含整合或 Lambda 授權方（先前稱為自訂授權方）的詳細資訊。若要包含這兩種

詳細資訊，請設定 `extensions=integrations,authorizers,documentation`。若要包含整合的詳細資訊，但不包含授權方的詳細資訊，請設定 `extensions=integrations,documentation`。

您必須在請求中設定 `Accept:application/json` 標頭，以將結果輸出到 JSON 檔案。若要產生 YAML 輸出，請將請求標頭變更為 `Accept:application/yaml`。

舉例來說，我們將檢視在根資源 (GET) 上公開一個簡單 / 方法的 API。此 API 在 OpenAPI 定義檔中已定義四個 API 實體，分別屬於 API、MODEL、METHOD 與 RESPONSE 類型。每個 API、METHOD 與 RESPONSE 實體都已新增一個文件組件。呼叫上述文件匯出命令，我們會取得下列輸出，其中 `x-amazon-apigateway-documentation` 物件內所列的文件組件是標準 OpenAPI 檔案的延伸。

### OpenAPI 3.0

```
{
  "openapi": "3.0.0",
  "info": {
    "description": "API info description",
    "version": "2016-11-22T22:39:14Z",
    "title": "doc",
    "x-bar": "API info x-bar"
  },
  "paths": {
    "/": {
      "get": {
        "description": "Method description.",
        "responses": {
          "200": {
            "description": "200 response",
            "content": {
              "application/json": {
                "schema": {
                  "$ref": "#/components/schemas/Empty"
                }
              }
            }
          }
        }
      },
      "x-example": "x- Method example"
    },
    "x-bar": "resource x-bar"
  }
},
"x-amazon-apigateway-documentation": {
  "version": "1.0.0",
  "createdDate": "2016-11-22T22:41:40Z",
  "documentationParts": [
    {
      "location": {
        "type": "API"
      },
      "properties": {
        "description": "API description",
        "foo": "API foo",
        "x-bar": "API x-bar",
        "info": {
          "description": "API info description",
          "version": "API info version",
          "foo": "API info foo",
          "x-bar": "API info x-bar"
        }
      }
    },
    {
      "location": {
        "type": "API"
      }
    }
  ]
}
```

```
        "type": "METHOD",
        "method": "GET"
    },
    "properties": {
        "description": "Method description.",
        "x-example": "x- Method example",
        "foo": "Method foo",
        "info": {
            "version": "method info version",
            "description": "method info description",
            "foo": "method info foo"
        }
    }
},
{
    "location": {
        "type": "RESOURCE"
    },
    "properties": {
        "description": "resource description",
        "foo": "resource foo",
        "x-bar": "resource x-bar",
        "info": {
            "description": "resource info description",
            "version": "resource info version",
            "foo": "resource info foo",
            "x-bar": "resource info x-bar"
        }
    }
}
],
},
"x-bar": "API x-bar",
"servers": [
    {
        "url": "https://rznaap68yi.execute-api.ap-southeast-1.amazonaws.com/"
    }
],
"components": {
    "schemas": {
        "Empty": {
            "type": "object",
            "title": "Empty Schema"
        }
    }
}
```

## OpenAPI 2.0

```
{
    "swagger" : "2.0",
    "info" : {
        "description" : "API info description",
        "version" : "2016-11-22T22:39:14Z",
        "title" : "doc",
        "x-bar" : "API info x-bar"
    },
    "host" : "rznaap68yi.execute-api.ap-southeast-1.amazonaws.com",
```

```
"basePath" : "/test",
"schemes" : [ "https" ],
"paths" : {
  "/" : {
    "get" : {
      "description" : "Method description.",
      "produces" : [ "application/json" ],
      "responses" : {
        "200" : {
          "description" : "200 response",
          "schema" : {
            "$ref" : "#/definitions/Empty"
          }
        }
      },
      "x-example" : "x- Method example"
    },
    "x-bar" : "resource x-bar"
  }
},
"definitions" : {
  "Empty" : {
    "type" : "object",
    "title" : "Empty Schema"
  }
},
"x-amazon-apigateway-documentation" : {
  "version" : "1.0.0",
  "createdDate" : "2016-11-22T22:41:40Z",
  "documentationParts" : [ {
    "location" : {
      "type" : "API"
    },
    "properties" : {
      "description" : "API description",
      "foo" : "API foo",
      "x-bar" : "API x-bar",
      "info" : {
        "description" : "API info description",
        "version" : "API info version",
        "foo" : "API info foo",
        "x-bar" : "API info x-bar"
      }
    }
  }, {
    "location" : {
      "type" : "METHOD",
      "method" : "GET"
    },
    "properties" : {
      "description" : "Method description.",
      "x-example" : "x- Method example",
      "foo" : "Method foo",
      "info" : {
        "version" : "method info version",
        "description" : "method info description",
        "foo" : "method info foo"
      }
    }
  }, {
    "location" : {
      "type" : "RESOURCE"
    },
    "properties" : {
      "description" : "resource description",
      "foo" : "resource foo",
      "info" : {
        "version" : "resource info version",
        "description" : "resource info description",
        "foo" : "resource info foo"
      }
    }
  }
}
```

```
"x-bar" : "resource x-bar",
"info" : {
    "description" : "resource info description",
    "version" : "resource info version",
    "foo" : "resource info foo",
    "x-bar" : "resource info x-bar"
}
}
}
},
"x-bar" : "API x-bar"
}
```

針對文件組件的 `properties` 對應中定義的 OpenAPI 相容屬性，API Gateway 會將該屬性插入相關聯的 API 實體定義中。`x-something` 的屬性是標準 OpenAPI 延伸。此延伸會傳播到 API 實體定義中。例如，請參閱 `x-example` 方法的 GET 屬性。`foo` 之類的屬性不屬於 OpenAPI 規格，而且不會插入其相關聯的 API 實體定義中。

如果文件轉譯工具（例如 [OpenAPI UI](#)）可剖析 API 實體定義來擷取文件屬性，則 `DocumentationPart` 執行個體之任何非 OpenAPI 相容的 `properties` 屬性不適用於該工具。不過，如果文件轉譯工具可剖析 `x-amazon-apigateway-documentation` 物件來取得內容，或是如果該工具可呼叫 `restapi:documentation-parts` 與 `documentationpart:id` 從 API Gateway 擷取文件組件，則該工具可以顯示所有文件屬性。

若要將 API 實體定義中含有整合詳細資訊的文件匯出到 JSON OpenAPI 檔案，請提交下列 GET 請求：

```
GET /restapis/restapi_id/stages/stage_name/exports/swagger?
extensions=integrations,documentation HTTP/1.1
Accept: application/json
Host: apigateway.region.amazonaws.com
Content-Type: application/json
X-Amz-Date: YYYYMMDDTttttttZ
Authorization: AWS4-HMAC-SHA256 Credential=access_key_id/YYYYMMDD/region/
apigateway/aws4_request, SignedHeaders=content-length;content-type;host;x-amz-date,
Signature=sigv4_secret
```

若要將 API 實體定義中含有整合與授權方詳細資訊的文件匯出到 YAML OpenAPI 檔案，請提交下列 GET 請求：

```
GET /restapis/restapi_id/stages/stage_name/exports/swagger?
extensions=integrations,authorizers,documentation HTTP/1.1
Accept: application/yaml
Host: apigateway.region.amazonaws.com
Content-Type: application/json
X-Amz-Date: YYYYMMDDTttttttZ
Authorization: AWS4-HMAC-SHA256 Credential=access_key_id/YYYYMMDD/region/
apigateway/aws4_request, SignedHeaders=content-length;content-type;host;x-amz-date,
Signature=sigv4_secret
```

若要使用 API Gateway 主控台匯出並下載已發佈的 API 文件，請遵循[使用 API Gateway 主控台匯出 REST API \(p. 606\)](#)中的說明進行。

## 匯入 API 文件

如同匯入 API 實體定義，您可以將文件組件從外部 OpenAPI 檔案匯入 API Gateway 中的 API。您可以在有效 OpenAPI 定義檔的 `x-amazon-apigateway-documentation` 物件 (p. 538) 延伸中，指定要匯入的文件組件。匯入文件不會改變現有的 API 實體定義。

您可以在 API Gateway 中選擇將新指定的文件組件合併到現有的文件組件，或覆寫現有的文件組件。在 MERGE 模式下，OpenAPI 檔案中定義的新文件組件會新增至 API 的 `DocumentationParts` 集

合。如果匯入的 DocumentationPart 已存在，匯入的屬性會取代現有的屬性 (如果兩者不同)。其他現有的文件屬性則不受影響。在 OVERWRITE 模式下，會根據已匯入的 OpenAPI 定義檔來取代整個 DocumentationParts 集合。

## 使用 API Gateway REST API 匯入文件組件

若要使用 API Gateway REST API 匯入 API 文件，請呼叫 [documentationpart:import](#) 操作。下列範例示範如何透過單一 GET / 方法覆寫 API 的現有文件組件，並在成功時傳回 200 OK 回應。

OpenAPI 3.0

```
PUT /restapis/<restapi_id>/documentation/parts&mode=overwrite&failonwarnings=true
Host: apigateway.region.amazonaws.com
Content-Type: application/json
X-Amz-Date: YYYYMMDDTttttttZ
Authorization: AWS4-HMAC-SHA256 Credential=access_key_id/YYYYMMDD/region/
apigateway/aws4_request, SignedHeaders=content-length;content-type;host;x-amz-date,
Signature=sigv4_secret

{
  "openapi": "3.0.0",
  "info": {
    "description": "description",
    "version": "1",
    "title": "doc"
  },
  "paths": {
    "/": {
      "get": {
        "description": "Method description.",
        "responses": {
          "200": {
            "description": "200 response",
            "content": {
              "application/json": {
                "schema": {
                  "$ref": "#/components/schemas/Empty"
                }
              }
            }
          }
        }
      }
    }
  },
  "x-amazon-apigateway-documentation": {
    "version": "1.0.3",
    "documentationParts": [
      {
        "location": {
          "type": "API"
        },
        "properties": {
          "description": "API description",
          "info": {
            "description": "API info description 4",
            "version": "API info version 3"
          }
        }
      },
      {
        "location": {
          "type": "METHOD",
          "method": "GET"
        }
      }
    ]
  }
}
```

```
        },
        "properties": {
            "description": "Method description."
        }
    },
    {
        "location": {
            "type": "MODEL",
            "name": "Empty"
        },
        "properties": {
            "title": "Empty Schema"
        }
    },
    {
        "location": {
            "type": "RESPONSE",
            "method": "GET",
            "statusCode": "200"
        },
        "properties": {
            "description": "200 response"
        }
    }
]
},
"servers": [
{
    "url": "/"
}
],
"components": {
    "schemas": {
        "Empty": {
            "type": "object",
            "title": "Empty Schema"
        }
    }
}
}
```

## OpenAPI 2.0

```
PUT /restapis/<restapi_id>/documentation/parts&mode=overwrite&failonwarnings=true
Host: apigateway.region.amazonaws.com
Content-Type: application/json
X-Amz-Date: YYYYMMDDTtttttt
Authorization: AWS4-HMAC-SHA256 Credential=access_key_id/YYYYMMDD/region/
apigateway/aws4_request, SignedHeaders=content-length;content-type;host;x-amz-date,
Signature=sigv4_secret

{
    "swagger": "2.0",
    "info": {
        "description": "description",
        "version": "1",
        "title": "doc"
    },
    "host": "",
    "basePath": "/",
    "schemes": [
        "https"
    ],
    "paths": {
        "/": {

```

```
"get": {
    "description": "Method description.",
    "produces": [
        "application/json"
    ],
    "responses": {
        "200": {
            "description": "200 response",
            "schema": {
                "$ref": "#/definitions/Empty"
            }
        }
    }
},
"definitions": {
    "Empty": {
        "type": "object",
        "title": "Empty Schema"
    }
},
"x-amazon-apigateway-documentation": {
    "version": "1.0.3",
    "documentationParts": [
        {
            "location": {
                "type": "API"
            },
            "properties": {
                "description": "API description",
                "info": {
                    "description": "API info description 4",
                    "version": "API info version 3"
                }
            }
        },
        {
            "location": {
                "type": "METHOD",
                "method": "GET"
            },
            "properties": {
                "description": "Method description."
            }
        },
        {
            "location": {
                "type": "MODEL",
                "name": "Empty"
            },
            "properties": {
                "title": "Empty Schema"
            }
        },
        {
            "location": {
                "type": "RESPONSE",
                "method": "GET",
                "statusCode": "200"
            },
            "properties": {
                "description": "200 response"
            }
        }
    ]
}
```

```
}
```

成功時，此請求會傳回 200 OK 回應，其中包含承載中已匯入的 DocumentationPartId。

```
{
  "ids": [
    "kg3mth",
    "796rtf",
    "zhek4p",
    "5ukm9s"
  ]
}
```

此外，您也可以呼叫 `restapi:import` 或 `restapi:put`，在 `x-amazon-apigateway-documentation` 物件中提供文件組件，做為 API 定義之 OpenAPI 檔案輸入的一部分。若要從 API 匯入中排除文件組件，請在請求查詢參數中設定 `ignore=documentation`。

## 使用 API Gateway 主控台匯入文件組件

下列說明示範如何匯入文件組件。

使用主控台從外部檔案匯入 API 的文件組件

1. 從主控台的主要導覽窗格中，選擇 API 的 Documentation (文件)。
2. 在 Documentation (文件) 窗格中，選擇 Import Documentation (匯入文件)。
3. 選擇 Select OpenAPI File (選取 OpenAPI 檔案) 從磁碟機載入檔案，或將檔案內容複製並貼到檔案檢視中。如需範例，請參閱「[使用 API Gateway REST API 匯入文件組件 \(p. 449\)](#)」中的範例請求承載。
4. (選擇性) 選擇 Fail on warnings (發出警告時失敗) 或 Ignore warnings (忽略警告)，然後從 Import mode (匯入模式) 中選擇 Merge 或 Overwrite。
5. 選擇 Import (匯入)。

## 控制 API 文件的存取

如果您有專屬文件團隊負責撰寫及編輯 API 文件，您可以為開發人員（針對 API 開發）與撰寫人員或編輯人員（針對內容開發）設定不同的存取許可。這特別適用於有第三方廠商為您建立文件時。

若要授予您的文件團隊建立、更新及發佈 API 文件的存取權，您可以使用下列 IAM 政策將 IAM 角色指派給文件團隊，其中 `account_id` 是您文件團隊的 AWS 帳戶 ID。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "StmtDocPartsAddEditViewDelete",
      "Effect": "Allow",
      "Action": [
        "apigateway:GET",
        "apigateway:PUT",
        "apigateway:POST",
        "apigateway:PATCH",
        "apigateway:DELETE"
      ],
      "Resource": [
        "arn:aws:apigateway:<region>:restapis/<id>/docs/part/<partId>"
      ]
    }
  ]
}
```

```
    "Resource": [
        "arn:aws:apigateway::account_id:restapis/*documentation/*"
    ]
}
```

如需設定 API Gateway 資源存取許可的資訊，請參閱 [控制誰可以使用 IAM 政策建立及管理 API Gateway API \(p. 335\)](#)。

## 在 Amazon API Gateway 中更新和維護 REST API

維護檢視、更新和刪除現有 API 設定的 API 數量。您可以使用 API Gateway 主控台、AWS CLI、開發套件或 API Gateway REST API 來維護 API。更新 API 需要修改 API 的特定資源屬性或組態設定。資源更新必須重新部署 API，組態更新則不必。

下表詳列了可以更新的 API 資源。

### API 資源更新需要重新部署 API

Resource	備註
ApiKey	關於適用的屬性和支援的操作，請參閱 <a href="#">apikey:update</a> 。更新需要重新部署 API。
授權方	關於適用的屬性和支援的操作，請參閱 <a href="#">authorizer:update</a> 。更新需要重新部署 API。
DocumentationPart	關於適用的屬性和支援的操作，請參閱 <a href="#">documentationpart:update</a> 。更新需要重新部署 API。
DocumentationVersion	關於適用的屬性和支援的操作，請參閱 <a href="#">documentationversion:update</a> 。更新需要重新部署 API。
GatewayResponse	關於適用的屬性和支援的操作，請參閱 <a href="#">gatewayresponse:update</a> 。更新需要重新部署 API。
整合	關於適用的屬性和支援的操作，請參閱 <a href="#">integration:update</a> 。更新需要重新部署 API。
IntegrationResponse	關於適用的屬性和支援的操作，請參閱 <a href="#">integrationresponse:update</a> 。更新需要重新部署 API。
方法	關於適用的屬性和支援的操作，請參閱 <a href="#">method:update</a> 。更新需要重新部署 API。
MethodResponse	關於適用的屬性和支援的操作，請參閱 <a href="#">methodresponse:update</a> 。更新需要重新部署 API。
模型	關於適用的屬性和支援的操作，請參閱 <a href="#">model:update</a> 。更新需要重新部署 API。
RequestValidator	關於適用的屬性和支援的操作，請參閱 <a href="#">requestvalidator:update</a> 。更新需要重新部署 API。
Resource	關於適用的屬性和支援的操作，請參閱 <a href="#">resource:update</a> 。更新需要重新部署 API。
API	關於適用的屬性和支援的操作，請參閱 <a href="#">restapi:update</a> 。更新需要重新部署 API。
VpcLink	關於適用的屬性和支援的操作，請參閱 <a href="#">vpclink:update</a> 。更新需要重新部署 API。

下表詳列了可以更新的 API 組態。

## API 組態更新無須重新部署 API

組態	備註
帳戶	關於適用的屬性和支援的操作，請參閱 <a href="#">account:update</a> 。更新無須重新部署 API。
部署	關於適用的屬性和支援的操作，請參閱 <a href="#">deployment:update</a> 。
DomainName	關於適用的屬性和支援的操作，請參閱 <a href="#">domainname:update</a> 。更新無須重新部署 API。
BasePathMapping	關於適用的屬性和支援的操作，請參閱 <a href="#">basepathmapping:update</a> 。更新無須重新部署 API。
階段	關於適用的屬性和支援的操作，請參閱 <a href="#">stage:update</a> 。更新無須重新部署 API。
用途	關於適用的屬性和支援的操作，請參閱 <a href="#">usage:update</a> 。更新無須重新部署 API。
UsagePlan	關於適用的屬性和支援的操作，請參閱 <a href="#">usageplan:update</a> 。更新無須重新部署 API。

### 主題

- 在 API Gateway 中變更公有或私有 API 端點類型 (p. 454)
- 使用 API Gateway 主控台維護 API (p. 455)

## 在 API Gateway 中變更公有或私有 API 端點類型

變更 API 端點類型需要您更新 API 的組態。您可以使用 API Gateway 主控台、AWS CLI 或適用於 API Gateway 的 AWS 開發套件來變更現有的 API 類型。完成更新操作最多約需 60 秒。在這段期間，將可以使用您的 API，但無法再次變更其端點類型，直到目前變更完成為止。

支援下列端點類型的變更：

- 從邊緣最佳化改為區域或私有
- 從區域改為邊緣最佳化或私有
- 從私有改為區域

您不能將私有 API 改為邊緣最佳化 API。

請注意，如果您要將公有 API 從邊緣最佳化改為區域（反之亦然），邊緣最佳化 API 的行為可能與區域 API 的行為不同。例如，邊緣最佳化 API 會移除 Content-MD5 標頭。任何傳送到後端的 MD5 雜湊值都可以請求字串參數或內文屬性表示。不過，區域性 API 仍會傳送此標頭，雖然它可能會將此標頭名稱重新對應到一些其他名稱。了解差異有助於您決定如何將邊緣最佳化 API 更新為區域 API，或將區域 API 更新為邊緣最佳化 API。

### 主題

- 使用 API Gateway 主控台變更 API 端點類型 (p. 454)
- 使用 AWS CLI 變更 API 端點類型 (p. 455)

## 使用 API Gateway 主控台變更 API 端點類型

若要為您的 API 變更 API 端點類型，請執行下列任一組步驟：

將端點從區域或邊緣最佳化轉換為公有（反之亦然）

1. 登入 API Gateway 主控台，然後在主導覽窗格中選擇 API。

2. 選擇 + Create API (+ 建立 API) 下的 API 設定 (齒輪圖示)。
3. 在 Endpoint Configuration (端點組態) 下，將 Endpoint Type (端點類型) 選項從 Edge Optimized 變更為 Regional，或從 Regional 變更為 Edge Optimized。
4. 選擇 Save (儲存) 開始更新。

#### 將私有端點轉換為區域端點

1. 登入 API Gateway 主控台，然後在主導覽窗格中選擇 API。
2. 選擇 + Create API (+ 建立 API) 下的 API 設定 (齒輪圖示)。
3. 編輯您的 API 資源政策，以移除任何提及的 VPC 或 VPC 端點；如此一來，從您的 VPC 內、外部進行的 API 呼叫將可成功。
4. 變更 Endpoint Type (端點類型) 為 Regional。
5. 選擇 Save (儲存) 開始更新。
6. 從您的 API 移除資源政策。
7. 重新部署您的 API，變更才會生效。

## 使用 AWS CLI 變更 API 端點類型

若要使用 AWS CLI 更新 `{api-id}` 的邊緣最佳化 API，請呼叫 `update-rest-api`，如下所示：

```
aws apigateway update-rest-api \
--rest-api-id {api-id}
--patch-operations op=replace,path=/endpointConfiguration/types/EDGE,value=REGIONAL
```

成功回應會有 200 OK 狀態碼與類似下列的承載：

```
{
  "createdDate": "2017-10-16T04:09:31Z",
  "description": "Your first API with Amazon API Gateway. This is a sample API that integrates via HTTP with our demo Pet Store endpoints",
  "endpointConfiguration": {
    "types": "REGIONAL"
  },
  "id": "0gsnjtjck8",
  "name": "PetStore imported as edge-optimized"
}
```

反之，將區域 API 更新成邊緣最佳化 API，如下所示：

```
aws apigateway update-rest-api \
--rest-api-id {api-id}
--patch-operations op=replace,path=/endpointConfiguration/types/REGIONAL,value=EDGE
```

由於 `put-rest-api` 是用於更新 API 定義，所以不適用於更新 API 端點類型。

## 使用 API Gateway 主控台維護 API

### 主題

- 在 API Gateway 中檢視 API 清單 (p. 456)
- 在 API Gateway 中刪除 API (p. 456)
- 在 API Gateway 中刪除資源 (p. 456)
- 在 API Gateway 中檢視方法清單 (p. 456)

- 在 API Gateway 中刪除方法 (p. 457)

## 在 API Gateway 中檢視 API 清單

使用 API Gateway 主控台來檢視 API 清單。

### 使用 API Gateway 主控台來檢視 API 清單

您必須擁有 API Gateway 中可用的 API。請遵循[在 Amazon API Gateway 中建立 REST API \(p. 167\)](#)中的說明進行。

1. 登入位於 <https://console.aws.amazon.com/apigateway> 的 API Gateway 主控台。
2. 顯示 API 清單。

## 在 API Gateway 中刪除 API

您可以使用 API Gateway 主控台來刪除 API。

Warning

刪除 API 表示您無法再呼叫它。這個操作無法復原。

### 使用 API Gateway 主控台刪除 API

您必須至少已部署一次 API。請遵循[在 Amazon API Gateway 中部署 REST API \(p. 457\)](#)中的說明進行。

1. 登入位於 <https://console.aws.amazon.com/apigateway> 的 API Gateway 主控台。
2. 在含有您要刪除之 API 名稱的方塊中，選擇 Resources (資源)。
3. 選擇 Delete API (刪除 API)。
4. 當提示您刪除 API 時，請選擇 Ok (確定)。

## 在 API Gateway 中刪除資源

使用 API Gateway 主控台刪除資源。

Warning

當您刪除資源時，也會刪除其子資源和方法。刪除資源可能會導致某部分的對應 API 無法使用。刪除資源無法復原。

### 使用 API Gateway 主控台刪除資源

1. 登入位於 <https://console.aws.amazon.com/apigateway> 的 API Gateway 主控台。
2. 在包含您要刪除之資源的 API 名稱的方塊中，選擇 Resources (資源)。
3. 在 Resources (資源) 窗格中，選擇資源，然後選擇 Delete Resource (刪除資源)。
4. 出現提示時，選擇 Delete (刪除)。

## 在 API Gateway 中檢視方法清單

使用 API Gateway 主控台以檢視資源的方法清單。

### 使用 API Gateway 主控台檢視方法清單

您必須在 API Gateway 中有可用的方法。請遵循[教學：建置具有 HTTP 非代理整合的 API \(p. 52\)](#)中的說明進行。

1. 登入位於 <https://console.aws.amazon.com/apigateway> 的 API Gateway 主控台。
2. 在包含 API 名稱的方塊中，選擇 Resources (資源)。
3. 在 Resources (資源) 窗格中，顯示方法清單。

Tip

您可能需要選擇一或多個資源旁邊的箭頭，以顯示所有可用的方法。

## 在 API Gateway 中刪除方法

使用 API Gateway 主控台刪除方法。

Warning

刪除方法可能會導致某部分的對應 API 變成無法使用。刪除方法無法復原。

### 使用 API Gateway 主控台刪除方法

1. 登入位於 <https://console.aws.amazon.com/apigateway> 的 API Gateway 主控台。
2. 在包含方法 API 之名稱的方塊中，選擇 Resources (資源)。
3. 在 Resources (資源) 窗格中，選擇方法資源旁邊的箭頭。
4. 選擇方法，然後選擇 Delete Method (刪除方法)。
5. 出現提示時，選擇 Delete (刪除)。

## 在 Amazon API Gateway 中部署 REST API

API 在建立之後必須進行部署，這樣您的使用者才能呼叫該 API。

若要部署 API，請建立 API 部署，並建立它與階段的關聯。每個階段都是 API 的快照，而且可以供用戶端應用程式呼叫。

Important

每次您更新 API 時 (包含修改路由、方法、整合、授權方，以及階段設定以外的任何其他項目)，都必須將 API 重新部署至現有階段或新的階段。

在您的 API 演進時，您可以繼續將它部署至不同的階段以作為 API 的不同版本。您也可以將 API 更新部署為 [Canary Release 部署 \(p. 607\)](#)，並讓您的 API 用戶端透過生產發行在相同的階段上存取生產版本，以及透過 Canary Release 存取已更新的版本。

若要呼叫已部署的 API，用戶端會對 API URL 提交請求。此 URL 取決於 API 的通訊協定 (HTTP (S) 或 (WSS))、主機名稱、階段名稱和 (適用於 REST API) 資源路徑。主機名稱和階段名稱可決定 API 的基本 URL。

使用 API 的預設網域名稱，處於指定階段 (`{stageName}`) 的 (例如) REST API 之基本 URL 格式如下：

```
https://{restapi-id}.execute-api.{region}.amazonaws.com/{stageName}
```

為了讓使用者更容易使用 API 的預設基本 URL，您可以建立自訂網域名稱 (例如，`api.example.com`) 以取代 API 的預設主機名稱。若要在自訂網域名稱下方支援多個 API，您必須將 API 階段對應至基本路徑。

使用自訂網域名稱 `{api.example.com}` 以及自訂網域名稱下方映射至基本路徑 (`{basePath}`) 的 API 階段，REST API 的基本 URL 會變成下列項目：

```
https://{api.example.com}/{basePath}
```

針對每個階段，您可以調整預設帳戶層級請求調節限制以及啟用 API 快取，來最佳化 API 效能。您也可以啟用將 API 呼叫記錄至 CloudTrail 或 CloudWatch，以及選取後端的用戶端憑證來驗證 API 請求。此外，您也可以覆寫個別方法的階段層級設定，以及定義階段變數以在執行時間將階段特定環境內容傳遞至 API 整合。

階段啟用 API 的強大版本控制。例如，您可以將 API 部署至 test 階段和 prod 階段，並使用 test 階段作為測試組建，以及使用 prod 階段作為穩定組建。更新通過測試之後，您就可以將 test 階段提升為 prod 階段。提升的做法是將 API 重新部署至 prod 階段，或將階段變數 (p. 459) 值從階段名稱 test 更新為階段名稱 prod。

在本節中，我們討論如何使用 [API Gateway 主控台](#) 或呼叫 [API Gateway REST API](#) 來部署 API。例如，若要使用其他工具來執行相同操作，請參閱 [AWS CLI](#) 或 [AWS 開發套件](#) 的文件。

#### 主題

- 在 API Gateway 中部署 REST API (p. 458)
- 在 API Gateway 中設定階段 (p. 460)
- 在 API Gateway 中針對 REST API 產生開發套件 (p. 481)

## 在 API Gateway 中部署 REST API

在 API Gateway 中，REST API 部署由 [Deployment](#) 資源代表。它就像是 [RestApi](#) 資源代表之 API 的可執行檔。為了讓用戶端能呼叫您的 API，您必須建立部署，並建立與其相關聯的階段。階段由 [Stage](#) 資源代表，表示 API 的快照，包括方法、整合、模型、映射範本、Lambda 授權方 (先前稱作自訂授權方) 等。當您更新 API 時，可以建立新階段與現有部署的關聯，藉此重新部署 API。我們會在「[the section called “設定階段” \(p. 460\)](#)」中討論階段的建立。

#### 主題

- 使用 AWS CLI 建立部署 (p. 458)
- 從 API Gateway 主控台部署 REST API (p. 459)

## 使用 AWS CLI 建立部署

建立部署相當於初始化 [Deployment](#) 資源。您可以使用 API Gateway 主控台、AWS CLI、AWS 開發套件或 API Gateway REST API 建立部署。

若要使用 CLI 建立部署，請使用 `create-deployment` 命令：

```
aws apigateway create-deployment --rest-api-id <rest-api-id> --region <region>
```

在您將此部署與階段建立關聯前，都無法呼叫 API。若現已有階段，您可使用新建立的部署 ID (<deployment-id>) 更新階段的 [deploymentId](#) 屬性，來完成此操作。

```
aws apigateway update-stage --region <region> \
--rest-api-id <rest-api-id> \
--stage-name <stage-name> \
--patch-operations op='replace',path='/deploymentId',value='<deployment-id>'
```

當您第一次部署 API 時，可以同時建立階段和部署：

```
aws apigateway create-deployment --region <region> \
--rest-api-id <rest-api-id> \
--stage-name <stage-name>
```

這就是當您第一次部署 API 或將 API 重新部署到新階段時，API Gateway 主控台後端完成的操作。

## 從 API Gateway 主控台部署 REST API

第一次部署 REST API 之前，您必須先進行建立。如需詳細資訊，請參閱「[在 Amazon API Gateway 中建立 REST API \(p. 167\)](#)」。

### 主題

- [將 REST API 部署至階段 \(p. 459\)](#)
- [更新 REST API 部署的階段組態 \(p. 459\)](#)
- [設定 REST API 部署的階段變數 \(p. 459\)](#)
- [建立階段與 REST API 不同部署的關聯 \(p. 459\)](#)

### 將 REST API 部署至階段

API Gateway 主控台可讓您透過建立部署並將它與全新或現有階段建立關聯來部署 API。

#### Note

若要將 API Gateway 中的階段與不同的部署建立關聯，請參閱[建立階段與 REST API 不同部署的關聯 \(p. 459\)](#)。

1. 登入位於 <https://console.aws.amazon.com/apigateway> 的 API Gateway 主控台。
2. 在 API 導覽窗格中，選擇您要部署的 API。
3. 在 Resources (資源) 導覽窗格中，選擇 Actions (動作)。
4. 從 Actions (動作) 下拉式選單中，選擇 Deploy API (部署 API)。
5. 在 Deploy API (部署 API) 對話方塊中，從 Deployment stage (部署階段) 下拉式清單中選擇項目。
6. 如果您選擇 [New Stage] ([新增階段])，請在 Stage name (階段名稱) 中輸入名稱，並選擇性地在 Stage description (階段說明) 和 Deployment description (部署說明) 中提供階段和部署的描述。如果您選擇現有階段，則建議您在 Deployment description (部署描述) 中提供新部署的說明。
7. 選擇 Deploy (部署)，以使用預設階段設定將 API 部署至指定的階段。

### 更新 REST API 部署的階段組態

部署 API 之後，您可以修改階段設定來啟用或停用 API 快取、記錄或請求調節。您也可以選擇後端的用戶端憑證來驗證 API Gateway，並設定階段變數以在執行時間將部署內容傳遞至 API 整合。如需詳細資訊，請參閱[更新階段設定 \(p. 461\)](#)。

#### Important

修改階段設定之後，您必須重新部署 API，以讓變更生效。

#### Note

如果已更新的設定（例如啟用記錄）需要新的 IAM 角色，則您可以新增所需的 IAM 角色，而不需要重新部署 API。不過，可能需要幾分鐘的時間，新的 IAM 角色才能生效。發生這種情況之前，不會記錄您 API 呼叫的追蹤，即使您已啟用記錄選項也是一樣。

### 設定 REST API 部署的階段變數

針對部署，您可以設定或修改階段變數，以在執行時間將部署特定資料傳遞至 API 整合。您可以在 Stage Editor (階段編輯器) 的 Stage Variables (階段變數) 標籤上執行這項操作。如需詳細資訊，請參閱「[設定 REST API 部署的階段變數 \(p. 475\)](#)」中的說明。

### 建立階段與 REST API 不同部署的關聯

因為部署代表 API 快照，而階段定義快照的路徑，所以您可以選擇不同的部署階段組合，以控制使用者如何呼叫不同版本的 API。例如，如果您想要將 API 狀態復原到先前的部署，或將 API 的「私有分支」合併至公有分支，則這十分有用。

下列程序示範如何在 API Gateway 主控台中使用 Stage Editor (階段編輯器) 來執行此操作。假設您必須部署 API 多次。

1. 如果尚未在 Stage Editor (階段編輯器) 中，請從 API 主要導覽窗格的 API Stages (階段) 選項中選擇您要更新部署的階段。
2. 在 Deployment History (部署歷史記錄) 標籤上，選擇您要階段使用之部署旁邊的選項按鈕。
3. 選擇 Change Deployment (變更部署)。

## 在 API Gateway 中設定階段

階段是部署的具名參考，而這是 API 快照。您使用階段來管理和最佳化特定部署。例如，您可以設定階段設定來啟用快取、自訂請求調節、設定記錄、定義階段變數，或連線 Canary 版本以進行測試。

### 主題

- [使用 API Gateway 主控台設定階段 \(p. 460\)](#)
- [啟用 API 快取以提升回應能力 \(p. 463\)](#)
- [調節 API 請求以獲得較佳輸送 \(p. 468\)](#)
- [設定 API Gateway 的用戶端 SSL 身份驗證 \(p. 470\)](#)
- [在 API Gateway 中設定 AWS WAF \(p. 470\)](#)
- [在 API Gateway 中設定 API 階段的標籤 \(p. 470\)](#)
- [在 API Gateway 中設定 CloudWatch API 記錄 \(p. 472\)](#)
- [在 API Gateway 中設定 X-Ray 追蹤 \(p. 475\)](#)
- [在 API Gateway 中設定 CloudTrail 記錄 \(p. 475\)](#)
- [設定 REST API 部署的階段變數 \(p. 475\)](#)

## 使用 API Gateway 主控台設定階段

### 主題

- [建立新的階段 \(p. 460\)](#)
- [更新階段設定 \(p. 461\)](#)
- [刪除 API 的階段 \(p. 463\)](#)

## 建立新的階段

初次部署之後，您可以新增更多階段並與現有的部署相關聯。您可以使用 API Gateway 主控台建立與使用新的階段，或在部署 API 時選擇現有的階段。一般而言，您可以先為 API 部署新增新階段，再重新部署 API。若要使用 API Gateway 主控台來執行此作業，請遵循下列說明進行。

1. 登入位於 <https://console.aws.amazon.com/apigateway> 的 API Gateway 主控台。
2. 從 API 導覽窗格，選擇 API 下的 Stages (階段)。
3. 從 Stages (階段) 導覽窗格，選擇 Create (建立)。
4. 在 Create Stage (建立階段) 下，針對 Stage name (階段名稱) 輸入階段名稱，例如 **prod**。

### Note

階段名稱僅可含有英數字元、連字號與底線。長度上限為 128 字元。

5. (選擇性) 針對 Stage description (階段描述) 輸入階段描述
6. 從 Deployment (部署) 下拉式清單中，選擇您要與此階段相關聯之現有 API 部署的日期與時間。

7. 選擇 Create (建立)。

## 更新階段設定

成功部署 API 後，會使用預設設定填入階段。您可以使用主控台或 API Gateway REST API 來變更階段設定，包括 API 快取和記錄日誌。我們會在下面示範如何使用 API Gateway 主控台的 Stage Editor (階段編輯器) 來執行此操作。

### 使用 API Gateway 主控台更新階段設定

這些步驟假設您已經將 API 部署到階段。

1. 登入位於 <https://console.aws.amazon.com/apigateway> 的 API Gateway 主控台。
2. 在 API (API) 窗格中，選擇 API，然後選擇 Stages (階段)。
3. 在 Stages (階段) 窗格中，選擇階段的名稱。
4. 在 Stage Editor (階段編輯器) 窗格中，選擇 Settings (設定) 標籤。
5. 若要為階段啟用 API 快取，請選擇 Cache Settings (快取設定) 區段下的 Enable API cache (啟用 API 快取) 選項。然後，針對 Cache capacity (快取容量)、Encrypt cache data (加密快取資料)、Cache time-to-live (TTL) (快取存活期 (TTL)) 以及每個金鑰快取失效的任何需求，選擇所需的選項和關聯值。

如需階段層級快取設定的詳細資訊，請參閱[啟用 API 快取 \(p. 463\)](#)。

#### Important

如果您啟用 API 階段的 API 快取，您的 AWS 帳戶可能需要支付 API 快取的費用。快取不符合 AWS 免費方案資格。

#### Tip

您也可以覆寫個別方法的已啟用階段層級快取設定。若要執行此作業，請展開 Stages (階段) 次要導覽窗格下的階段，然後選擇方法。然後，在階段編輯器中，選擇 Settings (設定) 的 Override for this method (覆寫此方法) 選項。在 Cache Settings (快取設定) 區域中，您可以設定或清除 Enable Method Cache (啟用方法快取) 或自訂任何其他所需的選項。如需方法層級快取設定的詳細資訊，請參閱[啟用 API 快取 \(p. 463\)](#)。

6. 若要為與此 API Gateway API 階段相關聯的所有方法啟用 Amazon CloudWatch Logs，請執行下列操作：
  - a. 在 CloudWatch Settings (&CW; 設定) 區段中，選取 Enable CloudWatch Logs (啟用 &CWL;) 選項。

#### Tip

若要啟用方法層級的 CloudWatch 設定，請展開 Stages (階段) 次要導覽窗格下的階段，選擇每個您感興趣的方法，然後回到階段編輯器，選擇 Settings (設定) 的 Override for this method (覆寫此方法)。在隨後的 CloudWatch Settings (&CW; 設定) 區域中，請務必選取 Log to CloudWatch Logs (記錄至 &CWL;) 及任何其他所需的選項，然後選擇 Save Changes (儲存變更)。

#### Important

您的帳戶需要支付存取方法層級 CloudWatch 指標的費用，但不需要支付 API 或階段層級指標的費用。

- b. 對於 Log level (記錄層級)，選擇 ERROR (錯誤)，只將錯誤層級項目寫入 CloudWatch Logs，或選擇 INFO (資訊) 包含所有 ERROR (錯誤) 事件以及額外的資訊事件。
- c. 若要記錄完整的 API 呼叫請求和回應資訊，請選取 Log full requests/responses data (記錄完整的請求/回應資料)。除非選取 Log full requests/responses data (記錄完整的請求/回應資料) 選項，否則不記錄敏感資料。

### Important

將日誌設定為 ERROR (錯誤) , 然後選擇 Log full requests/responses data (記錄完整請求/回應資料) , 這會導致詳細記錄每一個請求。這是預期行為。

- d. 若要讓 API Gateway 向 CloudWatch 報告 API calls、Latency、Integration latency、400 errors 和 500 errors 的 API 指標，請選取 Enable Detailed CloudWatch Metrics (啟用詳細的 CloudWatch 指標) 選項。如需 CloudWatch 的詳細資訊，請參閱 [Amazon CloudWatch User Guide](#)。
- e. 選擇 Save Changes (儲存變更)。新的部署完成後，新的設定即生效。

### Important

若要針對所有或僅部分方法啟用 CloudWatch Logs , 您也必須指定 IAM 角色的 ARN , 讓 API Gateway 代表您的 IAM 使用者將資訊寫入 CloudWatch Logs。若要執行此作業，請從 APIs 主導覽窗格選擇 Settings (設定)。然後在 CloudWatch log role ARN (&CW; 日誌角色 ARN) 文字欄位中輸入 IAM 角色的 ARN。對於常見的應用程式案例，IAM 角色可以連接 AmazonAPIGatewayPushToCloudWatchLogs 的受管政策，它包含下列存取政策陳述式：

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "logs:CreateLogGroup",  
                "logs:CreateLogStream",  
                "logs:DescribeLogGroups",  
                "logs:DescribeLogStreams",  
                "logs:PutLogEvents",  
                "logs:GetLogEvents",  
                "logs:FilterLogEvents"  
            ],  
            "Resource": "*"  
        }  
    ]  
}
```

The IAM 角色也必須包含下列信任關係陳述式：

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

如需 CloudWatch 的詳細資訊，請參閱 [Amazon CloudWatch User Guide](#)。

7. 若要設定與此 API 相關聯之所有方法的階段層級調節限制，請在 Default Method Throttling (預設方法調節) 區段中執行下列操作：
  - a. 選擇 Enable throttling (啟用調節)。

- b. 針對 Rate (比率) , 輸入 API Gateway 可以提供卻不必傳回 429 Too Many Requests 回應的每秒階段層級穩定狀態請求數量上限。此階段層級速率限制不得超過 [設定和執行 REST API 的 API Gateway 限制 \(p. 628\)](#) 所指定的 [帳戶層級 \(p. 469\)](#) 速率限制。
  - c. 針對 Burst (爆量) , 輸入 API Gateway 可以提供卻不必傳回 429 Too Many Requests 回應的階段層級並行請求數量上限。此階段層級爆量不得超過 [設定和執行 REST API 的 API Gateway 限制 \(p. 628\)](#) 所指定的 [帳戶層級 \(p. 469\)](#) 爆量限制。
8. 若要覆寫個別方法的階段層級調節，請展開 Stages (階段) 次要導覽窗格下的階段，選擇一個方法，然後選擇 Settings (設定) 的 Override for this method (覆寫此方法)。在 Method Throttling (預設方法調節) 區域中，選取適當的選項。
  9. 若要將 AWS WAF web ACL 與該階段建立關聯，請從 Web ACL 下拉式清單中選擇 web ACL。

**Note**

如果有需要，選擇 Create Web ACL (建立 Web ACL)，即可在新的瀏覽器分頁中開啟 AWS WAF 主控台，並返回 API Gateway 主控台來建立 web ACL 與階段的關聯。

10. 如有需要，選擇 Block API Request if WebACL cannot be evaluated (Fail- Close) (如果無法評估 WebACL，則阻擋 API 請求 (失敗 - 關閉))。
11. 若要為 API 階段啟用 [AWS X-Ray](#) 追蹤：
  - a. 在 Stage Editor (階段編輯器) 窗格中，選擇 Logs/Tracing (日誌/追蹤) 標籤。
  - b. 若要啟用 X-Ray 追蹤，請在 X-Ray Tracing (X-Ray 追蹤) 下方選擇 Enable X-Ray Tracing (啟用 X-Ray 追蹤)。
  - c. 若要在 X-Ray 主控台中設定取樣規則，請選擇 Set X-Ray Sampling Rules (設定 X-Ray 取樣規則)。
  - d. 如有需要，選擇 Set X-Ray Sampling Rules (設定 X-Ray 取樣規則) 並前往 X-Ray 主控台來[設定取樣規則](#)。

如需詳細資訊，請參閱[使用 AWS X-Ray 追蹤 API Gateway API 執行 \(p. 518\)](#)。

12. 選擇 Save Changes (儲存變更)。重新部署 API 到階段之後，新設定就會生效。

## 刪除 API 的階段

當您不再需要階段時，可以將其刪除，以避免支付未使用資源的費用。我們在下面說明如何使用 API Gateway 主控台來刪除階段。

**Warning**

刪除階段可能會導致 API 發起人變成無法使用部分或所有對應的 API。刪除階段無法復原，但您可以重新建立階段，並將其與相同的部署建立關聯。

### 使用 API Gateway 主控台刪除階段

1. 登入位於 <https://console.aws.amazon.com/apigateway> 的 API Gateway 主控台。
2. 在包含階段 API 之名稱的方塊中，選擇 Stages (階段)。
3. 在 Stages (階段) 窗格中，選擇您要刪除的階段，然後選擇 Delete Stage (刪除階段)。
4. 出現提示時，選擇 Delete (刪除)。

## 啟用 API 快取以提升回應能力

您可以在 Amazon API Gateway 中啟用 API 快取，來快取您端點的回應。透過快取，您可以減少對端點進行的呼叫數目，並改善 API 請求的延遲。當您啟用階段的快取時，API Gateway 會在指定的存留期間 (TTL) (以秒為單位) 從您的端點快取回應。API Gateway 接著會從快取查閱端點回應 (而不是對端點提出請求) 來回應請求。API 快取的預設 TTL 值為 300 秒。最大 TTL 值為 3600 秒。TTL=0 表示已停用快取。

可以快取的回應大小上限是 1048576 個位元組。快取資料加密可能會在快取回應時增加回應的大小。

This is a HIPAA Eligible Service. For more information about AWS, U.S. Health Insurance Portability and Accountability Act of 1996 (HIPAA), and using AWS services to process, store, and transmit protected health information (PHI), see [HIPAA Overview](#).

**Important**

當您啟用階段的快取時，預設只有 GET 方法會啟用快取。這有助於確保 API 的安全和可用性。您可以透過[覆寫方法設定 \(p. 465\)](#)，來啟用其他方法的快取。

**Important**

快取是按小時計費，不符合 AWS 免費方案的資格。

## 啟用 Amazon API Gateway 快取

在 API Gateway 中，您可以啟用指定階段的快取。

當您啟用快取，您必須選擇快取容量。一般而言，容量越大效能越佳，但成本也越高。

API Gateway 可透過建立專用快取執行個體來啟用快取。此程序最多需要 4 分鐘的時間。

API Gateway 可透過移除現有的快取執行個體，以及修改容量以建立新的執行個體，來變更快取容量。所有現有的快取資料都會遭到刪除。

在 API Gateway 主控台中，您可以在具名 Stage Editor (階段編輯器) 的 Settings (設定) 標籤中設定快取。

設定指定階段的 API 快取：

1. 前往 API Gateway 主控台。
2. 選擇 API。
3. 選擇 Stages (階段)。
4. 在 API 的 Stages (階段) 清單中，選擇階段。
5. 選擇 Settings (設定) 索引標籤。
6. 選擇 Enable API cache (啟用 API 快取)。
7. 等候快取建立完成。

**Note**

API Gateway 需要約 4 分鐘的時間來完成快取的建立或刪除。建立快取時，Cache status (快取狀態) 值會從 CREATE\_IN\_PROGRESS 變更為 AVAILABLE。快取刪除完成時，Cache status (快取狀態) 值會從 DELETE\_IN\_PROGRESS 變更為空字串。

當您在階段的 Cache Settings (快取設定) 中啟用快取時，只會啟用 GET 方法的快取。為了確保 API 的安全和可用性，建議您不要變更此設定。但是，您可以透過[覆寫方法設定 \(p. 465\)](#)，來啟用其他方法的快取。

如果您想要確認快取是否如預期般運作，您有兩個一般選項：

- 檢查您 API 與階段的 CloudWatch 指標 CacheHitCount 與 CacheMissCount。
- 將時間戳記放在回應中。

**Note**

您不應該使用 CloudFront 回應中的 x-Cache 標頭，來判斷您的 API 是否由 API Gateway 快取執行個體所提供。

## 覆寫方法快取的 API Gateway 階段層級快取

您可以透過啟用或停用特定方法的快取、增加或減少其 TTL 期間，或是開啟或關閉快取回應的加密，來覆寫階段層級的快取設定。

如果您預期正在快取的方法會在其回應中收到敏感性資料，請在 Cache Settings (快取設定) 中，選擇 Encrypt cache data (加密快取資料)。

使用主控台來設定個別方法的 API 快取：

1. 登入位於 <https://console.aws.amazon.com/apigateway> 的 API Gateway 主控台。
2. 前往 API Gateway 主控台。
3. 選擇 API。
4. 選擇 Stages (階段)。
5. 在 API 的 Stages (階段) 清單中，展開階段，然後在 API 中選擇方法。
6. 在 Settings (設定) 中，選擇 Override for this method (覆寫這個方法)。
7. 在 Cache Settings (快取設定) 區域中，您可以設定或清除 Enable Method Cache (啟用方法快取) 或自訂任何其他所需的選項。(只有啟用 [階段層級快取 \(p. 464\)](#) 時才會顯示本區段。)

## 使用方法或整合參數作為快取金鑰來編製快取回應的索引

當快取方法或整合具有參數時 (其格式可能是自訂標頭、URL 路徑或查詢字串)，您可以使用部分或所有參數來形成快取金鑰。API Gateway 可以根據所使用的參數值來快取方法的回應。

### Note

設定資源的快取時必須提供快取金鑰。

例如，假設您有一個請求，其格式如下：

```
GET /users?type=... HTTP/1.1
host: example.com
...
```

在此請求中，type 可接受 admin 或 regular 值。如果您包含 type 參數作為快取金鑰的一部分，則會分別快取 GET /users?type=admin 的回應與 GET /users?type=regular 的回應。

當方法或整合請求接受多個參數時，您可以選擇包含部分或所有參數來建立快取金鑰。例如，您可以針對下列請求 (依 TTL 期間所列的順序提出)，僅在快取金鑰中包含 type 參數：

```
GET /users?type=admin&department=A HTTP/1.1
host: example.com
...
```

此請求的回應會被快取並用來服務下列請求：

```
GET /users?type=admin&department=B HTTP/1.1
host: example.com
...
```

若要在 API Gateway 主控台中包含方法或整合請求參數做為快取金鑰的一部分，請在新增參數之後選取 Caching (快取)。

[Method Execution /streams - GET - Method Request](#)

Provide information about this method's authorization settings and the parameters it can receive.

Authorization Settings ●

Authorization NONE

API Key Required false

URL Query String Parameters ●

Name	Caching	
query	<input checked="" type="checkbox"/>	

Add query string

HTTP Request Headers

Request Models [Create a Model](#) ●

## 在 API Gateway 中排清 API 階段快取

啟用 API 快取時，您可以排清 API 階段的整個快取，以確保您 API 的用戶端從您的整合端點取得最新回應。

若要排清 API 階段快取，您可以在 API Gateway 主控台的階段編輯器中，選擇 Settings (設定) 標籤之 Cache Settings (快取設定) 區段下的 Flush entire cache (排清整個快取) 按鈕。快取排清操作需要幾分鐘，完成排清之後快取狀態立即為 AVAILABLE。

Note

快取排清之後，會從整合端點處理回應，直到再度建立快取。在此期間，傳送至整合端點的請求數目可能會增加。這可能會暫時增加您的 API 整體延遲。

## 使 API Gateway 快取項目失效

您的 API 用戶端可使現有的快取項目失效，並從個別請求的整合端點將它重新載入。用戶端必須傳送含有 Cache-Control: max-age=0 標頭的請求。只要授權用戶端執行這項作業，用戶端就可以直接從整合端點 (而不是快取) 接收回應。這會以擷取自整合端點的新回應來取代現有的快取項目。

若要授予許可給用戶端，請將下列格式的政策連接到使用者的 IAM 執行角色。

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": "execute-api:Invoke",  
            "Resource": "  
                "arn:aws:execute-api:  
                    <region>:  
                    <account>:  
                    <restapi>/<resource>/{proxy+}"  
            }  
    ]  
}
```

```
"Action": [
    "execute-api:InvalidateCache"
],
"Resource": [
    "arn:aws:execute-api:region:account-id:api-id/stage-name/GET/resource-path-specifier"
]
}
```

此政策可讓 API Gateway 執行服務使一或多項指定資源請求的快取失效。若要指定一組目標資源，請針對 account-id、api-id 與 Resource 之 ARN 值中的其他項目使用萬用字元 (\*)。如需如何設定 API Gateway 執行服務許可的詳細資訊，請參閱 [使用 IAM 許可控制 API 的存取 \(p. 333\)](#)

如果您未強制執行 InvalidateCache 原則 (或在主控台中選擇 Require authorization (需要授權) 核取方塊)，則任何用戶端可能使 API 快取失效。如果大多數或所有用戶端使 API 快取失效，可能會顯著延長您的 API 延遲。

當您具備政策、啟用快取且需要授權時，您可以在 API Gateway 主控台中，從 Handle unauthorized requests (處理未經授權的請求) 選擇一個選項，來控制未經授權之請求的處理方式。

## test Stage Editor

Delete Stage

● Invoke URL: [https://\[REDACTED\].execute-api.us-east-1.amazonaws.com/test](https://[REDACTED].execute-api.us-east-1.amazonaws.com/test)

**Settings** **Stage Variables** **SDK Generation** **Export** **Deployment History**

Configure the metering and caching settings for the **test** stage.

**Cache Settings**

Cache status AVAILABLE **Flush entire cache**

**Enable API cache**

Enabling API cache increases cost and is not covered by the free tier. [See pricing for more details](#)

Cache capacity 0.5GB

Encrypt cache data

Cache time-to-live (TTL) 300

**Per-key cache invalidation**

**Require authorization**

**Handle unauthorized requests**  Ignore cache control header; Add a warning in response header

CloudWatch Settings  [Ignore cache control header; Add a warning in response header](#)

Ignore cache control header

Fail the request with 403 status code

Enable CloudWatch Logs  [?](#)

這三個選項會導致下列行為：

- Fail the request with 403 status code (請求失敗並顯示 403 狀態碼)：傳回 403 未授權回應。  
若要使用 API 來設定此選項，請使用 FAIL\_WITH\_403。
- Ignore cache control header; Add a warning in response header (忽略快取控制標頭；在回應標頭中新增警告)：處理請求並在回應中新增警告標頭。  
若要使用 API 來設定此選項，請使用 SUCCEED\_WITH\_RESPONSE\_HEADER。  
• Ignore cache control header (忽略快取控制標頭)：處理請求但不會在回應中新增警告標頭。  
若要使用 API 來設定此選項，請使用 SUCCEED\_WITHOUT\_RESPONSE\_HEADER。

## 調節 API 請求以獲得較佳輸送

為避免您的 API 無法處理過多請求，Amazon API Gateway 會使用**字符儲存貯體演算法**將請求節流到您的 API，而此演算法會將字符計算為請求。具體而言，API Gateway 會設定穩定狀態速率限制以及您帳戶中所有 API 請求提交的爆增。在字符儲存貯體演算法中，爆增就是最大儲存貯體大小。

請求提交超過穩定狀態請求率和爆增限制時，API Gateway 會讓超出限制請求失敗，並將 429 Too Many Requests 錯誤回應傳回給用戶端。發現這類例外狀況時，用戶端可以透過速率限制形式來重新提交失敗的請求，同時遵守 API Gateway 調節限制。

身為 API 開發人員，您可以設定個別 API 階段或方法的限制，來改善您帳戶中所有 API 的整體效能。或者，您可以啟用[用量方案 \(p. 397\)](#)來限制所指定之請求速率和配額內的用戶端請求提交。這會限制整體請求提交，讓它們不會明顯地通過帳戶層級調節限制。

### 主題

- [如何在 API Gateway 中套用調節限制設定 \(p. 468\)](#)
- [帳戶層級調節 \(p. 469\)](#)
- [預設方法調節和覆寫預設方法調節 \(p. 469\)](#)
- [在用量計劃中設定 API 層級和階段層次調節 \(p. 470\)](#)
- [在用量計劃中設定方法層次調節 \(p. 470\)](#)

## 如何在 API Gateway 中套用調節限制設定

在為階段設定中 API 設定限制設定和(選擇性) [用量計劃 \(p. 397\)](#)前，了解 Amazon API Gateway 如何套用調節限制設定很有用。

Amazon API Gateway 提供兩種基本類型的調節相關的設定：

- 在所有用戶端套用伺服器端調節限制。這些限制設定的存在是為防止您的 API — 和您的帳戶 — 接收過多請求。
- 根據用戶端的調節限制會套用到用戶端，該用戶端會使用與做為用戶端身分之使用政策關聯 API 金鑰。

API Gateway 調節相關的設定會以如下順序進行套用：

1. 您在[用量計劃 \(p. 404\)](#)中為 API 階段設定的 [根據用戶端的根據方法調節限制 \(p. 470\)](#)
2. 您在用量計劃中設定之[根據用戶端調節限制 \(p. 470\)](#)
3. 您在[API 階段設定 \(p. 461\)](#)中設定的 [預設根據方法限制和個別根據方法限制 \(p. 469\)](#)
4. [帳戶層級調節 \(p. 469\)](#)

## 帳戶層級調節

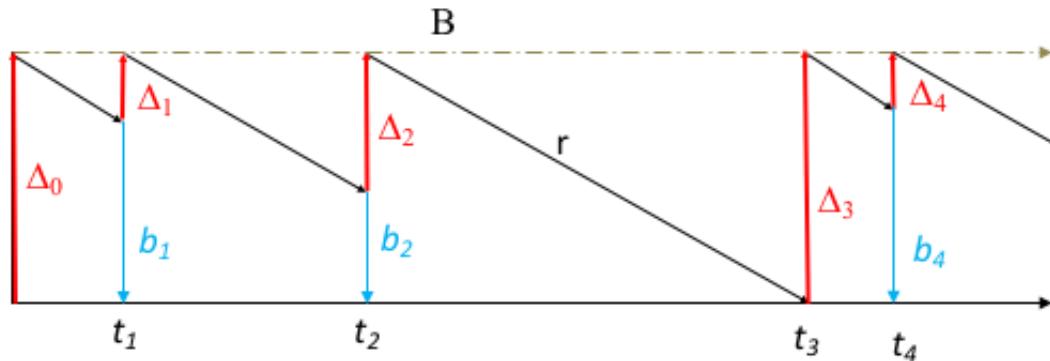
API Gateway 預設會將穩定狀態請求率限制為每秒 10,000 個請求 (rps)。它會將爆增 (即最大儲存貯體大小) 限制為 AWS 帳戶內所有 API 有 5,000 個請求。在 API Gateway 中，爆增限制會對應至 API Gateway 可在任何時間符合且未傳回 429 Too Many Requests 錯誤回應的最大並行請求提交數目。

為了協助了解這些調節限制，此處有一些關於爆增限制和預設帳戶層級速率的範例：

- 如果發起人平均在一秒期間內提交 10,000 個請求 (例如，每毫秒 10 個請求)，則 API Gateway 會處理所有請求，而不捨棄任何請求。
- 如果發起人在第一個毫秒傳送 10,000 個請求，則 API Gateway 會服務其中的 5,000 個請求，並在一秒期間內調節其他請求。
- 如果發起人在第一個毫秒提交 5,000 個請求，然後將另 5,000 個請求平均分佈到剩下的 999 毫秒 (例如，大約每毫秒 5 個請求)，則 API Gateway 會在一秒期間內處理全部 10,000 個請求，而不會傳回 429 Too Many Requests 錯誤回應。
- 如果發起人在第一個毫秒提交 5,000 個請求，並等到第 101 個毫秒來提交另 5,000 個請求，則 API Gateway 會在一秒期間內處理 6,000 個請求並節流剩下的請求。原因是依 10,000 rps 的速率，API Gateway 已在前 100 毫秒之後服務 1,000 個請求，因此會以相同的數量清空儲存貯體。接下來爆增 5,000 個請求，其中 1,000 個會填入儲存貯體，並放入佇列中以進行處理。另 4,000 個會超過儲存貯體容量，並予以捨棄。
- 如果發起人在第一個毫秒提交 5,000 個請求，並在第 101 毫秒提供 1000 個請求，然後將另 4,000 個請求平均分佈到剩下的 899 毫秒，則 API Gateway 會在一秒期間內處理全部 10,000 個請求，而不進行調節。

普遍來說，在任何指定的時間，儲存貯體包含  $b$  且最大儲存貯體容量是  $B$  時，可新增至儲存貯體的最大額外字符是  $\# = B - b$ 。這個最大額外字符數目對應至用戶端可提交但未收到任何 429 錯誤回應的最大額外並行請求數目。一般而言， $\#$  會因時間而不同。值的範圍是從儲存貯體已滿時 (即  $b=B$ ) 的零到儲存貯體為空時 (即  $B$ ) 的  $b=0$ 。範圍取決於請求處理率 (從儲存貯體移除字符的速率) 和速率限制率 (將字符新增至儲存貯體的速率)。

下列概要將  $\#$  一般行為 (最大額外並行請求) 顯示為時間函數。概要假設儲存貯體中的字符依合併速率  $r$  減少，並且是從空的儲存貯體開始。



帳戶層級速率限制可能會在請求時提高。若要請求提高帳戶層級調節限制，請與 [AWS Support 中心](#) 聯絡。如需詳細資訊，請參閱[API Gateway 限制 \(p. 627\)](#)。

## 預設方法調節和覆寫預設方法調節

您可以設定預設方法調節，來覆寫特定階段或 API 中個別方法的帳戶層級請求調節限制。預設方法調節限制受限於帳戶層級速率限制，即使您設定高於帳戶層級限制的預設方法調節限制也是一樣。

您可以在 API Gateway 主控台中設定預設方法調節限制，方法是使用 Stages (階段) 中的 Default Method Throttling (預設方法調節) 設定。如需使用主控台的說明，請參閱「[更新階段設定 \(p. 461\)](#)」。

您也可以呼叫 [API Gateway V1 和 V2 API 參考 \(p. 626\)](#) 來設定預設方法調節限制。

## 在用量計劃中設定 API 層級和階段層次調節

在 [用量計劃 \(p. 397\)](#) 中，您可以為在 Create Usage Plan (建立用量計劃) 下 API 或階段層級的所有方法設定預設根據方法調節限制，如[建立用量計劃 \(p. 404\)](#)中所示。

## 在用量計劃中設定方法層次調節

您可以在 Usage Plans (用量計劃) 中使用方法層級來設定額外調節限制，如[建立用量計劃 \(p. 404\)](#)中所示。在 API Gateway 主控台中，這些都是透過在 Configure Method Throttling (設定方法調節) 設定中指定 Resource=<resource> , Method=<method> 來設定。例如，對於 [PetStore 範例 \(p. 52\)](#)，您可能會指定 Resource=/pets , Method=GET。

## 設定API Gateway的用戶端 SSL 身份驗證

您可以使用 API Gateway 產生 SSL 憑證，並在後端中使用其公有金鑰來確認對後端系統的 HTTP 請求來自 API Gateway。您也可以使用 API Gateway 主控台來啟用 API 的用戶端 SSL 身份驗證。有关更多信息，请参阅 the section called “[設定 API 來使用 SSL 憑證” \(p. 377\)](#)。

## 在 API Gateway 中設定 AWS WAF

您可以使用 AWS WAF，來保護 Web 應用程式和 API 不受攻擊，方法為設定一組規則 (稱為 web 存取控制清單或 web ACL)，該組規則可根據您定義的可自訂 Web 安全規則與條件來允許、封鎖或計數 Web 請求。如需詳細資訊，請參閱the section called “[使用 AWS WAF 來保護您的 API 避免受到常見的網路攻擊” \(p. 396\)](#)。

## 在 API Gateway 中設定 API 階段的標籤

在 API Gateway 中，您可以將標籤新增至 API 階段、從階段中移除標籤或檢視標籤。若要執行此作業，您可以使用 API Gateway 主控台、AWS CLI/開發套件或 API Gateway REST API。

階段也可以繼承其父系 REST API 的標籤。如需詳細資訊，請參閱the section called “[標籤繼承” \(p. 622\)](#)。

如需有關標記 API Gateway 資源的詳細資訊，請參閱[標記 API Gateway 資源 \(p. 621\)](#)。

### 主題

- [使用 API Gateway 主控台設定 API 階段的標籤 \(p. 470\)](#)
- [使用 API Gateway REST API 設定 API 階段的標籤 \(p. 471\)](#)

## 使用 API Gateway 主控台設定 API 階段的標籤

下列程序說明如何設定 API 階段的標籤。

### 使用 API Gateway 主控台設定 API 階段的標籤

1. 登入 API Gateway 主控台。
2. 選擇現有的 API 或建立新的 API，其中包含資源、方法與對應的整合。
3. 選擇階段，或將 API 部署到新的階段。
4. 在 Stage Editor (階段編輯器) 中，選擇 Configure Tags (設定標籤) 按鈕。
5. 在 Tag Editor (標籤編輯器) 中，選擇 Add New Tag (新增標籤)。在 Key (索引鍵) 欄位中輸入標籤索引鍵 (例如 Department)，並在 Value (值) 欄位中輸入標籤值 (例如 Sales)。選擇核取記號圖示以儲存標籤。

6. 如果需要，請重複步驟 5 將更多標籤新增至 API 階段。每個階段的標籤數上限為 50。
7. 若要從階段中移除現有的標籤，請選擇標籤旁邊的垃圾桶圖示。
8. 選擇 Save Changes (儲存變更) 完成設定階段標籤。

如果先前已在 API Gateway 主控台中部署該 API，您將需要重新部署 API，變更才能生效。

## 使用 API Gateway REST API 設定 API 階段的標籤

您可以使用 API Gateway REST API 執行下列一項操作來設定 API 階段的標籤：

- 呼叫 `tags:tag` 來標記 API 階段。
- 呼叫 `tags:untag` 從 API 階段中刪除一或多個標籤。
- 呼叫 `stage:create` 以新增一或多個標籤到您建立的 API 階段。

您也可以呼叫 `tags:get` 來說明 API 階段中的標籤。

### 標記 API 階段

您可以在將 API (`m5zr3vnks7`) 部署到階段 (`test`) 之後，呼叫 `tags:tag` 來標記階段。必要的階段 Amazon Resource Name (ARN) (`arn:aws:apigateway:us-east-1::/restapis/m5zr3vnks7/stages/test`) 必須以 URL 編碼 (`arn%3Aaws%3Aapigateway%3Aus-east-1%3A%3A%2Frestapis%2Fm5zr3vnks7%2Fstages%2Ftest`)。

```
PUT /tags/arn%3Aaws%3Aapigateway%3Aus-east-1%3A%3A%2Frestapis%2Fm5zr3vnks7%2Fstages%2Ftest
{
  "tags" : {
    "Department" : "Sales"
  }
}
```

您也可以使用前一個請求，將現有的標籤更新為新的值。

您可以在呼叫 `stage:create` 建立階段時，將標籤新增至階段：

```
POST /restapis/<restapi_id>/stages
{
  "stageName" : "test",
  "deploymentId" : "adr134",
  "description" : "test deployment",
  "cacheClusterEnabled" : "true",
  "cacheClusterSize" : "500",
  "variables" : {
    "sv1" : "val1"
  },
  "documentationVersion" : "test",

  "tags" : {
    "Department" : "Sales",
    "Division" : "Retail"
  }
}
```

### 取消標記 API 階段

若要從階段中移除 `Department` 標籤，請呼叫 `tags:untag`：

```
DELETE /tags/arn%3Aaws%3Aapigateway%3Aus-east-1%3A%3A%2Frestapis%2Fm5zr3vnks7%2Fstages%2Ftest?tagKeys=Department
Host: apigateway.us-east-1.amazonaws.com
Authorization: ...
```

若要移除多個標籤，請在查詢表達式中使用標籤索引鍵的逗號分隔清單—例如？  
tagKeys=Department,Division,...。

### 說明 API 階段的標籤

若要說明指定階段的現有標籤，請呼叫 `tags:get`：

```
GET /tags/arn%3Aaws%3Aapigateway%3Aus-east-1%3A%3A%2Frestapis%2Fm5zr3vnks7%2Fstages%2Ftags
Host: apigateway.us-east-1.amazonaws.com
Authorization: ...
```

成功回應類似如下：

```
200 OK
{
  "_links": {
    "curies": {
      "href": "http://docs.aws.amazon.com/apigateway/latest/developerguide/restapi-tags-{rel}.html",
      "name": "tags",
      "templated": true
    },
    "tags:tag": {
      "href": "/tags/arn%3Aaws%3Aapigateway%3Aus-east-1%3A%3A%2Frestapis%2Fm5zr3vnks7%2Fstages%2Ftags"
    },
    "tags:untag": {
      "href": "/tags/arn%3Aaws%3Aapigateway%3Aus-east-1%3A%3A%2Frestapis%2Fm5zr3vnks7%2Fstages%2Ftags{?tagKeys}",
      "templated": true
    }
  },
  "tags": {
    "Department": "Sales"
  }
}
```

## 在 API Gateway 中設定 CloudWatch API 記錄

為了協助偵錯 API 請求執行或用戶端存取相關問題，您可以啟用 Amazon CloudWatch Logs 來記錄 API 呼叫。啟用之後，API Gateway 會將 API 呼叫記錄至 CloudWatch 中。如需詳細資訊，請參閱[the section called “使用 Amazon CloudWatch 監控 API 執行” \(p. 528\)](#)。

### API Gateway 的 CloudWatch 日誌格式

CloudWatch 有兩種類型的 API 記錄：執行記錄和存取記錄。在執行記錄中，API Gateway 會管理 CloudWatch Logs。此程序包含建立日誌群組和日誌串流，以及向日誌串流報告任何發起人的請求和回應。所記錄的資料包含錯誤或執行追蹤（例如請求或回應參數值或承載）、Lambda 授權方（先前稱作自訂授權方）所使用的資料、是否需要 API 金鑰、是否啟用用量方案，以此類推。

當您部署 API 時，API Gateway 會在日誌群組下方建立日誌群組和日誌串流。日誌群組的命名是遵循 API-Gateway-Execution-Logs\_{rest-api-id}/{stage\_name} 格式。在每個日誌群組內，日誌會進一步分割為日誌串流，而其在報告所記錄的資料時會依 Last Event Time（上次事件時間）排序。

在存取記錄中，您身為 API 開發人員且想要記錄誰已存取您的 API 以及發起人存取 API 的方式。您可以建立自己的日誌群組，或選擇 API Gateway 可管理的現有帳戶。您可以選取以所選擇格式表示的 [\\$context \(p. 274\)](#) 變數，以及選擇日誌群組做為目標，來指定存取詳細資訊。若要保留每個日誌的唯一性，則存取日誌格式必須包含 \$context.requestId。

**Note**

僅支援 \$context 變數，不支援 \$input 等。

選擇分析後端也會採用的日誌格式，例如[通用日誌格式](#) (CLF)、JSON、XML 或 CSV。您接著可以直接饋送其存取日誌，以計算和轉譯您的指標。若要定義日誌格式，請在階段的 [accessLogSettings/destinationArn](#) 屬性上設定日誌群組 ARN。您可以在 CloudWatch 主控台中取得日誌群組 ARN，但前提是選取 ARN 資料行進行顯示。若要定義存取日誌格式，請在階段的 [accessLogSetting/format](#) 屬性上設定選擇的格式。

一些常用存取日誌格式範例會顯示在 API Gateway 主控台中，並列出如下。

- CLF ([通用日誌格式](#)) :

```
$context.identity.sourceIp $context.identity.caller \
$context.identity.user [$context.requestTime] \
"$context.httpMethod $context.resourcePath $context.protocol" \
$context.status $context.responseLength $context.requestId
```

接續字元 (\) 顯示做為提醒，而且日誌格式不能有任何分行符號。

- JSON :

```
{ "requestId": "$context.requestId", \
  "ip": "$context.identity.sourceIp", \
  "caller": "$context.identity.caller", \
  "user": "$context.identity.user", \
  "requestTime": "$context.requestTime", \
  "httpMethod": "$context.httpMethod", \
  "resourcePath": "$context.resourcePath", \
  "status": "$context.status", \
  "protocol": "$context.protocol", \
  "responseLength": "$context.responseLength" \
}
```

接續字元 (\) 顯示做為提醒，而且日誌格式不能有任何分行符號。

- XML :

```
<request id="$context.requestId"> \
<ip>$context.identity.sourceIp</ip> \
<caller>$context.identity.caller</caller> \
<user>$context.identity.user</user> \
<requestTime>$context.requestTime</requestTime> \
<httpMethod>$context.httpMethod</httpMethod> \
<resourcePath>$context.resourcePath</resourcePath> \
<status>$context.status</status> \
<protocol>$context.protocol</protocol> \
<responseLength>$context.responseLength</responseLength> \
</request>
```

接續字元 (\) 顯示做為提醒，而且日誌格式不能有任何分行符號。

- CSV (逗號分隔值) :

```
$context.identity.sourceIp,$context.identity.caller, \
$context.identity.user,$context.requestTime,$context.httpMethod, \
```

```
$context.resourcePath,$context.protocol,$context.status,\$context.responseLength,$context.requestId
```

接續字元 (\) 顯示做為提醒，而且日誌格式不能有任何分行符號。

## CloudWatch 記錄的許可

若要啟用 CloudWatch Logs，您必須授予 API Gateway 讀取 CloudWatch 並將其寫入至您帳戶的適當許可。AmazonAPIGatewayPushToCloudWatchLogs 受管政策 (ARN 為 arn:aws:iam::aws:policy/service-role/AmazonAPIGatewayPushToCloudWatchLogs) 具有所有必要許可：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs>CreateLogGroup",
                "logs>CreateLogStream",
                "logs>DescribeLogGroups",
                "logs>DescribeLogStreams",
                "logs>PutLogEvents",
                "logs>GetLogEvents",
                "logs>FilterLogEvents"
            ],
            "Resource": "*"
        }
    ]
}
```

若要將這些許可授予您的帳戶，請建立 apigateway.amazonaws.com 做為其信任實體的 IAM 角色，並將先前的政策連接至 IAM 角色，然後在帳戶的 [cloudWatchRoleArn](#) 屬性上設定 IAM 角色 ARN。

若您在設定 IAM 角色 ARN 時發生錯誤，請檢查您的 AWS Security Token Service 帳戶設定，確認 AWS STS 在您使用的區域中已啟用。如需啟用 AWS STS 的詳細資訊，請參閱 IAM User Guide 中的 [在 AWS 區域管理 AWS STS](#)。

## 使用 API Gateway 主控台設定 API 記錄

若要設定 API 記錄，您必須已將 API 部署至階段。您也必須已設定您帳戶之適當的 CloudWatch Logs 角色 ([p. 474](#)) ARN。

1. 登入位於 <https://console.aws.amazon.com/apigateway> 的 API Gateway 主控台。
2. 從主導覽面板中選擇 Settings (設定)，並在 CloudWatch log role ARN (CloudWatch 日誌角色 ARN) 中輸入具有適當許可之 IAM 角色的 ARN。您需要執行這項操作一次。
3. 請執行下列其中一項：
  - a. 選擇現有 API，然後選擇階段。
  - b. 建立 API，並將它部署至階段。
4. 在 Stage Editor (階段編輯器) 中，選擇 Logs/Tracing (日誌/追蹤)。
5. 若要啟用執行記錄：
  - a. 在 CloudWatch Settings (CloudWatch 設定) 下選擇 Enable CloudWatch Logs (啟用 CloudWatch 日誌)。
  - b. 從下拉式功能表中，選擇 Error (錯誤) 或 Info (資訊)。
  - c. 如有需要，請選擇 Enable Detailed CloudWatch Metrics (啟用詳細的 CloudWatch 指標)。

如需 CloudWatch 指標的詳細資訊，請參閱 [the section called “使用 Amazon CloudWatch 監控 API 執行” \(p. 528\)](#)。

6. 若要啟用存取記錄：
  - a. 在 Custom Access Logging (自訂存取記錄) 下選擇 Enable Access Logging (啟用存取記錄)。
  - b. 在 CloudWatch Group (CloudWatch 群組) 中，輸入日誌群組的 ARN。ARN 的格式為 `arn:aws:logs:{region}:{account-id}:log-group:API-Gateway-Execution-Logs_{rest-api-id}/{stage-name}`。
  - c. 在 Log Format (日誌格式) 中，輸入日誌格式。您可以選擇 CLF、JSON、XML 或 CSV，以使用其中一個提供的範例做為指南。
7. 選擇 Save Changes (儲存變更)。

#### Note

您可以獨立啟用執行記錄和存取記錄。

API Gateway 現在已準備好將請求記錄至 API。當您更新階段設定、日誌或階段變數時，不需要重新部署 API。

## 在 API Gateway 中設定 X-Ray 追蹤

為了協助對 API 請求延遲的相關問題進行除錯，您可以啟用 AWS X-Ray 追蹤來追蹤 API 請求和下游服務。啟用之後，API Gateway 會在 X-Ray 中追蹤 API。如需詳細資訊，請參閱[“設定 AWS X-Ray” \(p. 519\)](#)。

## 在 API Gateway 中設定 CloudTrail 記錄

為 API 啟用 CloudTrail 記錄功能，您不需要執行任何作業。它會自動啟用。如需詳細資訊，請參閱[“透過 AWS CloudTrail 記錄向 Amazon API Gateway API 發出的呼叫” \(p. 527\)](#)。

## 設定 REST API 部署的階段變數

階段變數是名稱/值對，您可以定義為與 REST API 部署階段相關聯的組態屬性。它們的作用如同環境變數，可用於 API 設定和對應範本。

例如，您可以在階段組態中定義階段變數，然後將其值設為您 REST API 方法之 HTTP 整合的 URL 字串。之後，您可以從 API 設定使用相關聯的階段變數名稱參考 URL 字串。利用這種方式，您可以將階段變數值重設為對應的 URL，在每個階段的不同端點使用相同的 API 設定。您也可以存取對應範本中的階段變數，或將組態參數傳送到 AWS Lambda 或 HTTP 後端。

如需對應範本的詳細資訊，請參閱[API Gateway 映射範本和存取記錄變數參考 \(p. 274\)](#)。

## 使用案例

透過 API Gateway 中的部署階段，您可以管理每個 API 的多個發行階段，像是 alpha、beta 和生產。使用階段變數，您可以設定 API 部署階段與不同的後端端點互動。例如，您的 API 可將 GET 請求當做 HTTP 代理傳送到後端 Web 主機 (例如，`http://example.com`)。在這種情況下，後端 Web 主機是以階段變數設定，所以當開發人員呼叫您的生產端點時，API Gateway 會呼叫 `example.com`。當您呼叫您的 beta 端點時，API Gateway 會在 beta 階段使用階段變數中設定的值，並呼叫不同的 Web 主機 (例如，`beta.example.com`)。同樣地，階段變數可用來指定您 API 中每個階段的不同 AWS Lambda 函數名稱。

您也可以使用階段變數，透過您的對應範本將組態參數傳送到 Lambda 函數。例如，您可能希望在您 API 的多個階段中重複使用相同的 Lambda 函數，但此函數應該從不同的 Amazon DynamoDB 資料表讀取資

料，視呼叫的階段而定。在產生 Lambda 函數請求的對應範本中，您可以使用階段變數將資料表名稱傳送給 Lambda。

階段變數不會套用 API 規格的安全性定義部分。例如，您不能針對不同的階段使用不同的 Amazon Cognito 使用者集區。

## 範例

若要使用階段變數來自訂 HTTP 整合端點，您必須先設定指定名稱的階段變數，例如，`url`，然後為它指派值，例如 `example.com`。接著，從您的方法組態設定 HTTP 代理整合，而不是輸入端點的 URL，您可以指示 API Gateway 使用階段變數值 `http://${stageVariables.url}`。此值會指示 API Gateway 在執行時間替換您的階段變數 `${}`，視您 API 執行的階段而定。您可以類似的方式參考階段變數，指定 Lambda 函數名稱、AWS Service 代理路徑，或登入資料欄位中的 AWS 角色 ARN。

將 Lambda 函數名稱指定為階段變數值時，您必須在 Lambda 函數中手動設定許可。您可以使用 AWS Command Line Interface 來執行此操作。

```
aws lambda add-permission --function-name arn:aws:lambda:XXXXXX:your-lambda-function-name --source-arn arn:aws:execute-api:us-east-1:YOUR_ACCOUNT_ID:api_id/*/*HTTP_METHOD/resource --principal apigateway.amazonaws.com --statement-id apigateway-access --action lambda:InvokeFunction
```

以下範例會指派 API Gateway 許可，呼叫名為 `helloWorld` 的 Lambda 函數，它裝載在代表 API 方法的 AWS 帳戶的 US West (Oregon) 區域中。

```
arn arn:aws:execute-api:us-west-2:123123123123:bmmuvptwze/*/*GET/hello
```

以下是使用 AWS CLI 的相同命令。

```
aws lambda add-permission --function-name arn:aws:lambda:us-east-1:123123123123:function:helloWorld --source-arn arn:aws:execute-api:us-west-2:123123123123:bmmuvptwze/*/*GET/hello --principal apigateway.amazonaws.com --statement-id apigateway-access --action lambda:InvokeFunction
```

## 使用 Amazon API Gateway 主控台設定階段變數

在此教學中，您會了解如何使用 Amazon API Gateway 主控台，為範例 API 的兩個部署階段設定階段變數。開始之前，請務必先達成以下必要條件：

- 您必須擁有 API Gateway 中可用的 API。請遵循[在 Amazon API Gateway 中建立 REST API \(p. 167\)](#)中的說明進行。
- 您必須至少已部署一次 API。請遵循[在 Amazon API Gateway 中部署 REST API \(p. 457\)](#)中的說明進行。
- 您必須為已部署的 API 建立第一個階段。請遵循[建立新的階段 \(p. 460\)](#)中的說明進行。

### 使用 API Gateway 主控台宣告階段變數

1. 登入位於 <https://console.aws.amazon.com/apigateway> 的 API Gateway 主控台。
2. 建立 API，在 API 根資源建立 GET 方法，如果您尚未這樣做。將 HTTP Endpoint URL (端點 URL) 值設為 "`http://${stageVariables.url}`"，然後選擇 Save (儲存)。
3. 選擇 Deploy API (部署 API)。選擇 New Stage (新的階段)，然後針對 Stage name (階段名稱) 輸入 "beta"。選擇 Deploy (部署)。
4. 在 beta Stage Editor (beta 階段編輯器) 窗格中選擇 Stage Variables (階段變數) 標籤，然後選擇 Add Stage Variable (新增階段變數)。
5. 在 Name (名稱) 欄位中輸入 "url" 字串，在 Value (值) 欄位中輸入 "httpbin.org/get"。選擇核取記號圖示儲存此階段變數的設定。

6. 重複上述步驟，多新增兩個階段變數：version 和 function。分別將它們的值設為 "v-beta" 和 "HelloWorld"。

Note

將 Lambda 函數設定為階段變數值時，請使用函數的本機名稱，這可能會包括其別名或版本規格，如 HelloWorld、HelloWorld:1 或 HelloWorld:alpha。請勿使用函數的 ARN (例如，arn:aws:lambda:us-east-1:123456789012:function:HelloWorld)。API Gateway 主控台假設 Lambda 函數的階段變數值為不完整的函數名稱，會將指定的階段變數擴充為 ARN。

7. 從 Stages (階段) 導覽窗格，選擇 Create (建立)。在 Stage name (階段名稱) 中輸入 prod。從 Deployment (部署) 選取最新的部署，然後選擇 Create (建立)。
8. 至於 beta 階段，請將相同的三個階段變數 (url、version 和 function) 分別設成不同的值 ("petstore-demo-endpoint.execute-api.com/petstore/pets"、"v-prod" 和 "HelloEveryone")。

## 使用 Amazon API Gateway 階段變數

您可以使用 API Gateway 階段變數來存取不同 API 部署階段的 HTTP 和 Lambda 後端，將階段特定組態中繼資料傳送到 HTTP 後端當做查詢參數，傳送到 Lambda 函數當做輸入對應範本中產生的承載。

### 先決條件

您必須建立兩個階段，將 url 變數設定為兩個不同的 HTTP 端點：指派給兩個不同 Lambda 函數的 function 階段變數，以及包含階段特定中繼資料的 version 階段變數。請遵循「[使用 Amazon API Gateway 主控台設定階段變數 \(p. 476\)](#)」中的說明進行。

### 透過有階段變數的 API 存取 HTTP 端點

1. 在 Stages (階段) 導覽窗格中，選擇 beta。在 beta Stage Editor (beta 階段編輯器)中，選擇 Invoke URL (呼叫 URL) 連結。這會啟動 beta 階段在 API 根資源上的 GET 請求。

Note

Invoke URL (呼叫 URL) 連結在其 beta 階段會指向 API 的根資源。透過選擇連結導覽至 URL 會對根資源呼叫 beta 階段 GET 方法。如果方法是在子資源中定義，不是在根資源本身中定義，選擇 Invoke URL (呼叫 URL) 連結會回應 {"message": "Missing Authentication Token"} 錯誤回應。在這種情況下，您必須將特定子資源的名稱附加至 Invoke URL (呼叫 URL) 連結。

2. 您從 beta 階段 GET 請求取得的回應，如旁所示。您也可以使用瀏覽器導覽至 http://httpbin.org/get 驗證結果。此值已在 beta 階段指派給 url 變數。兩個回應完全相同。
3. 在 Stages (階段) 導覽窗格中，選擇 prod 階段。在 prod Stage Editor (prod 階段編輯器) 中，選擇 Invoke URL (呼叫 URL) 連結。這會啟動 prod 階段在 API 根資源上的 GET 請求。
4. 您從 prod 階段 GET 請求取得的回應，如旁所示。您可以使用瀏覽器導覽至 http://petstore-demo-endpoint-execute-api.com/petstore/pets 來驗證結果。此值已在 prod 階段指派給 url 變數。兩個回應完全相同。

### 在查詢參數表達式中透過階段變數將階段特定中繼資料傳送到 HTTP 後端

此程序說明如何使用查詢參數表達式中的階段變數值，將階段特定中繼資料傳送到 HTTP 後端。我們會使用在 version 中宣告的 [使用 Amazon API Gateway 主控台設定階段變數 \(p. 476\)](#) 階段變數。

1. 在 Resource (資源) 導覽窗格中選擇 GET 方法。若要將查詢字串參數新增到方法的 URL，請在 Method Execution (方法執行) 中選擇 Method Request (方法請求)。輸入參數名稱的 version。
2. 在 Method Execution (方法執行) 中，選擇 Integration Request (整合請求)。編輯 Endpoint URL (端點 URL) 值，將 ?version=\${stageVariables.version} 附加至之前定義的 URL 值；在本例中，它也會以 url 階段變數表示。選擇 Deploy API (部署 API) 來部署這些變更。

3. 在 Stages (階段) 導覽窗格中，選擇 beta 階段。從 beta Stage Editor (beta 階段編輯器) 驗證目前的階段是否處於最新的部署，然後選擇 Invoke URL (呼叫 URL) 連結。

Note

我們在此使用 beta 階段，是因為 HTTP 端點如 url 變數 "http://httpbin.org/get" 所指定，接受查詢參數表達式，並傳回它們做為其回應的 args 物件。

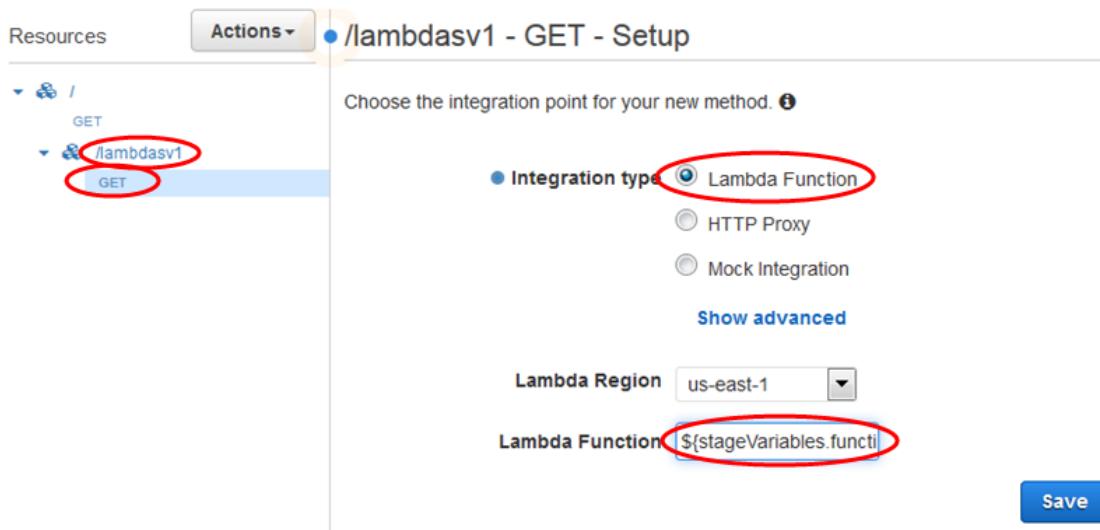
4. 回應如旁所示。請注意，指派給 v-beta 階段變數的 version，在後端傳送為 version 引數。

```
{  
    "args": {  
        "version": "v-beta"  
    },  
    "headers": {  
        "Accept": "application/json",  
        "Host": "httpbin.org",  
        "User-Agent": "AmazonAPIGateway_h4ah70cvmb"  
    },  
    "origin": "52.91.42.97",  
    "url": "http://httpbin.org/get?version=v-beta"  
}
```

### 透過 API 使用階段變數呼叫 Lambda 函數

此程序說明如何使用階段變數呼叫 Lambda 函數做為 API 的後端。我們會使用之前宣告的 function 階段變數。如需詳細資訊，請參閱 [使用 Amazon API Gateway 主控台設定階段變數 \(p. 476\)](#)。

1. 在 Resources (資源) 窗格中，在根目錄下建立 /lambdasv1 子資源，然後在子資源上建立 GET 方法。將 Integration type (整合類型) 設為 Lambda Function (Lambda 函數)，並在 Lambda Function (Lambda 函數) 中輸入 \${stageVariables.function}。選擇 Save (儲存)。

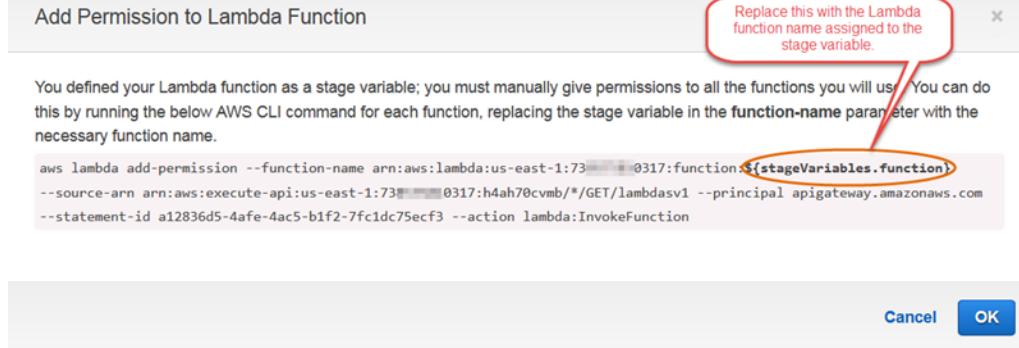


Tip

當提示您 Add Permission to Lambda Function (新增許可到 Lambda 函數) 時，請先記下 AWS CLI 命令，再選擇 OK (確定)。您必須對已指派或將指派給每個新建立 API 方法的 function 階段變數的每個 Lambda 函數執行命令。例如，如果 \${stageVariables.function} 值是 HelloWorld，而您尚未新增許可到此函數，您即必須執行以下 AWS CLI 命令：

```
aws lambda add-permission --function-name arn:aws:lambda:us-east-1:account-id:function:HelloWorld --source-arn arn:aws:execute-api:us-east-1:account-id:api-id/*/GET/lambdasv1 --principal apigateway.amazonaws.com --statement-id statement-id-guid --action lambda:InvokeFunction
```

如果不執行此作業，會在呼叫方法時造成 500 Internal Server Error 回應。請務必以指派給階段變數的 Lambda 函數名稱取代 \${stageVariables.function}。



2. 將 API 部署到可用的階段。
3. 在 Stages (階段) 導覽窗格中，選擇 beta 階段。確認您的最新部署位於 beta Stage Editor (beta 階段編輯器)。複製 Invoke URL (呼叫 URL) 連結，將其貼入瀏覽器的網址列，再將 /lambdasv1 附加至該 URL。這會透過 API 的 LambdaSv1 子資源的 GET 方法呼叫基礎 Lambda 函數。

#### Note

您的 HelloWorldLambda 函數實作以下程式碼。

```
exports.handler = function(event, context, callback) {
  if (event.version)
    callback(null, 'Hello, World! (' + event.version + ')');
  else
    callback(null, "Hello, world! (v-unknown)");
};
```

此實作會產生以下的回應。

```
"Hello, world! (v-unknown)"
```

### 透過階段變數將階段特定中繼資料傳送到 Lambda 函數

此程序說明如何使用階段變數將階段特定組態中繼資料傳送到 Lambda 函數。我們會使用之前宣告的 POST 階段變數，使用 version 方法和輸入對應範本產生承載。

1. 在 Resources (資源) 窗格中，選擇 /lambdasv1 子資源。在子資源建立 POST 方法，將 Integration type (整合類型) 設為 Lambda Function (Lambda 函數)，然後在 Lambda Function (Lambda 函數) 中輸入 \${stageVariables.function}。選擇 Save (儲存)。

#### Tip

這個步驟類似我們用來建立 GET 方法的步驟。如需詳細資訊，請參閱[透過 API 使用階段變數呼叫 Lambda 函數 \(p. 478\)](#)。

2. 在 /Method Execution (方法執行) 窗格中，選擇 Integration Request (整合請求)。在 Integration Request (整合請求) 窗格中，展開 Mapping Templates (對應範本)，然後選擇 Add mapping template (新增對應範本) 新增 application/json 內容類型的範本，如下所示。

The screenshot shows the AWS Lambda Function configuration interface. On the left, there's a tree view of resources: a root folder with a GET method, a sub-folder '/lambdasv1' containing a POST method (which is selected), and a GET method. On the right, under 'Actions', the 'Method Execution' tab is selected for the '/lambdasv1 - POST' integration request. The configuration includes:

- Integration type:** Lambda Function (selected)
- Lambda Region:** us-east-1
- Lambda Function:** \${stageVariables.function}
- Invoke with caller credentials:**
- Credentials cache:** Do not add caller credentials to cache key
- Body Mapping Templates:** A section for mapping templates based on Content-Type. For 'application/json', the mapping template is defined as:

```
1 #set($inputRoot = $input('$'))
2 {
3     "version": "$stageVariables.version"
4 }
```

#### Note

在對應範本中，參考的階段變數必須用引號括住 (如 "\$stageVariables.version" 或 "\${stageVariables.version}")，而在其他地方的參考則必須不用引號 (如 \${stageVariables.function})。

3. 將 API 部署到可用的階段。
4. 在 Stages (階段) 導覽窗格中，選擇 beta。在 beta Stage Editor (beta 階段編輯器) 中，驗證目前階段是否有最新的部署。複製 Invoke URL (呼叫 URL) 連結並貼入至 REST API 用戶端的 URL 輸入欄位，將 /lambdasv1 附加至該 URL，然後向基礎 Lambda 函數提交 POST 請求。

#### Note

您會收到以下回應。

```
"Hello, world! (v-beta)"
```

## Amazon API Gateway 階段變數參考

在下列情況下，您可以使用 API Gateway 階段變數。

## 參數對應表達式

階段變數可用在 API 方法請求或回應標頭參數的參數映射表達式中，沒有任何部分替換。在下列範例中，階段變數參考不使用 \$ 與包圍的 {...}。

- stageVariables.<variable\_name>

## 對應範本

階段變數可以用在對應範本的任何位置，如下例所示。

- { "name" : "\$stageVariables.<variable\_name>" }
- { "name" : "\${stageVariables.<variable\_name>}" }

## HTTP 整合 URI

階段變數可用作 HTTP 整合 URL 的一部分，如下例所示。

- 無協定的完整 URI，例如 `http://${stageVariables.<variable_name>}`
- 完整的網域：例如 `http://${stageVariables.<variable_name>}/resource/operation`
- 子網域：例如 `http://${stageVariables.<variable_name>}.example.com/resource/operation`
- 路徑，例如 `http://example.com/${stageVariables.<variable_name>}/bar`
- 查詢字串，例如 `http://example.com/foo?q=${stageVariables.<variable_name>}`

## AWS 整合 URI

階段變數可用作 AWS URI 動作或路徑元件的一部分，如下例所示。

- `arn:aws:apigateway:<region>:<service>:${stageVariables.<variable_name>}`

## AWS 整合 URI (Lambda 函數)

階段變數可用來取代 Lambda 函數名稱或版本/別名，如下例所示。

- `arn:aws:apigateway:<region>:lambda:path/2015-03-31/functions/arn:aws:lambda::<account_id>:function:${stageVariables.<function_variable_name>}/invocations`
- `arn:aws:apigateway:<region>:lambda:path/2015-03-31/functions/arn:aws:lambda::<account_id>:function:<function_name>:${stageVariables.<version_variable_name>}/invocations`

## AWS 整合登入資料

階段變數可用作 AWS 使用者/角色登入資料 ARN 的一部分，如下例所示。

- `arn:aws:iam::<account_id>:${stageVariables.<variable_name>}`

# 在 API Gateway 中針對 REST API 產生開發套件

若要以特定平台或特定語言的方式來呼叫您的 REST API，您必須產生特定平台或特定語言的 API 開發套件。目前，API Gateway 支援產生以 Java、JavaScript、適用於 Android 的 Java、適用於 iOS 的 Objective-C 或 Swift 以及 Ruby 撰寫的 API 開發套件。

本節說明如何產生 API Gateway API 的開發套件，並示範如何在 Java 應用程式、適用於 Android 的 Java 應用程式、適用於 iOS 的 Objective-C 與 Swift 應用程式以及 JavaScript 應用程式中，使用所產生的開發套件。

為了方便討論，我們使用此 API Gateway API (p. 487)，這會公開此[簡易計算機 \(p. 485\)](#) Lambda 函數。

繼續之前，請建立或匯入 API，並至少在 API Gateway 中部署一次。如需說明，請參閱「[在 Amazon API Gateway 中部署 REST API \(p. 457\)](#)」。

#### 主題

- [使用 API Gateway 主控台產生 API 開發套件 \(p. 482\)](#)
- [使用 AWS CLI 命令產生 API 開發套件 \(p. 484\)](#)
- [簡易計算機的 Lambda 函數 \(p. 485\)](#)
- [API Gateway 中的簡易計算機 API \(p. 487\)](#)
- [簡易計算機 API OpenAPI 定義 \(p. 492\)](#)

## 使用 API Gateway 主控台產生 API 開發套件

若要在 API Gateway 中產生特定平台或特定語言的 API 開發套件，您必須先建立、測試及部署階段中的 API。為了方便說明，我們在本節中使用[簡易計算機 \(p. 492\)](#) API 作為範例，來產生特定語言或特定平台的開發套件。如需如何建立、測試及部署此 API 的說明，請參閱[建立簡易計算機 API \(p. 487\)](#)。

#### 主題

- [產生 API 的 Java 開發套件 \(p. 482\)](#)
- [產生 API 的 Android 開發套件 \(p. 483\)](#)
- [產生 API 的 iOS 開發套件 \(p. 483\)](#)
- [產生 API 的 JavaScript 開發套件 \(p. 484\)](#)
- [產生 API 的 Ruby 開發套件 \(p. 484\)](#)

## 產生 API 的 Java 開發套件

### 在 API Gateway 中產生 API 的 Java 開發套件

1. 登入位於 <https://console.aws.amazon.com/apigateway> 的 API Gateway 主控台。
2. 在包含階段 API 之名稱的方塊中，選擇 Stages (階段)。
3. 在 Stages (階段) 窗格中，選擇階段的名稱。
4. 在 SDK Generation (開發套件產生) 標籤上，針對 Platform (平台)，選擇 Java，然後執行下列操作：
  - a. 針對 Service Name (服務名稱)，指定您開發套件的名稱。例如，SimpleCalcSdk。這會成為開發套件用戶端類別的名稱。此名稱會對應到開發套件專案資料夾之 pom.xml 檔案中 <name> 下的 <project> 標籤。請勿包含連字號。
  - b. 針對 Java Package Name (Java 套件名稱)，指定您開發套件的套件名稱。例如，examples.aws.apig.simpleCalc.sdk。此套件名稱會作為您開發套件程式庫的命名空間。請勿包含連字號。
  - c. 針對 Java Build System (Java 建置系統)，輸入 maven 或 gradle 以指定建置系統。
  - d. 針對 Java Group Id (Java 群組 ID)，輸入您的開發套件專案的群組識別符。例如，my-apig-api-examples。此識別符會對應到開發套件專案資料夾之 <groupId> 檔案中 <project> 下的 pom.xml 標籤。
  - e. 針對 Java Artifact Id (Java 成品 ID)，輸入您開發套件專案的成品識別符。例如，simple-calc-sdk。此識別符會對應到開發套件專案資料夾之 <artifactId> 檔案中 <project> 下的 pom.xml 標籤。

- f. 針對 Java Artifact Version (Java 成品版本) , 輸入版本識別符字符串。例如 , 1.0.0。此版本識別符會對應到開發套件專案資料夾之 <version> 檔案中 <project> 下的 pom.xml 標籤。
  - g. 針對 Source Code License Text (原始程式碼授權文字) , 輸入您原始程式碼的授權文字 (如果有)。
5. 選擇 Generate SDK (產生開發套件) , 然後遵循畫面上的說明下載 API Gateway 所產生的開發套件。
  6. 遵循「[使用 API Gateway 為 REST API 所產生的 Java 開發套件 \(p. 499\)](#)」中的說明使用所產生的開發套件。

每次更新 API , 都必須重新部署 API 並重新產生開發套件來包含更新。

## 產生 API 的 Android 開發套件

### 在 API Gateway 中產生 API 的 Android 開發套件

1. 登入位於 <https://console.aws.amazon.com/apigateway> 的 API Gateway 主控台。
2. 在包含階段 API 之名稱的方塊中 , 選擇 Stages (階段)。
3. 在 Stages (階段) 窗格中 , 選擇階段的名稱。
4. 在 SDK Generation (開發套件產生) 標籤上 , 針對 Platform (平台) , 選擇 Android 平台。
  - a. 針對 Group ID (群組 ID) , 輸入對應專案的唯一識別符。這會用於 pom.xml 檔案 (例如 com.mycompany)。
  - b. 針對 Invoker package (啟動程式套件) , 輸入所產生用戶端類別的命名空間 (例如 com.mycompany.clientsdk)。
  - c. 針對 Artifact ID (成品 ID) , 輸入已編譯 .jar 檔案的名稱 (不含版本)。這會用於 pom.xml 檔案 (例如 aws-apigateway-api-sdk)。
  - d. 針對 Artifact version (成品版本) , 輸入所產生用戶端的成品版本號碼。這會用於 pom.xml 檔案 , 且應該採用 major.minor.patch 模式 (例如 1.0.0)。
5. 選擇 Generate SDK (產生開發套件) , 然後遵循畫面上的說明下載 API Gateway 所產生的開發套件。
6. 遵循「[使用 API Gateway 為 REST API 所產生的 Android 開發套件 \(p. 502\)](#)」中的說明使用所產生的開發套件。

每次更新 API , 都必須重新部署 API 並重新產生開發套件來包含更新。

## 產生 API 的 iOS 開發套件

### 在 API Gateway 中產生 API 的 iOS 開發套件

1. 登入位於 <https://console.aws.amazon.com/apigateway> 的 API Gateway 主控台。
2. 在包含階段 API 之名稱的方塊中 , 選擇 Stages (階段)。
3. 在 Stages (階段) 窗格中 , 選擇階段的名稱。
4. 在 SDK Generation (開發套件產生) 標籤上 , 針對 Platform (平台) , 選擇 iOS (Objective-C) 或 iOS (Swift) 平台。
  - 在 Prefix (字首) 方塊中 , 輸入唯一的字首。

字首的作用如下：如果指派 SIMPLE\_CALC 作為具有 Input、Output 與 Result 模型之 SimpleCalc (p. 487) 的開發套件字首，所產生的開發套件會包含 SIMPLE\_CALCSimpleCalcClient 類別以封裝 API，包括方法請求/回應。此外，所產生的開發套件會包含 SIMPLE\_CALCInput、SIMPLE\_CALCOOutput 與 SIMPLE\_CALCResult 類別，分別代表輸入、輸出與結果，以表示請求輸入與回應輸出。如需詳細資訊，請參閱在 Objective-C 或 Swift 中使用 API Gateway 為 REST API 產生的 iOS 開發套件 (p. 508)。

5. 選擇 Generate SDK (產生開發套件) , 然後遵循畫面上的說明下載 API Gateway 所產生的開發套件。

6. 遵循「[在 Objective-C 或 Swift 中使用 API Gateway 為 REST API 產生的 iOS 開發套件 \(p. 508\)](#)」中的說明使用所產生的開發套件。

每次更新 API，都必須重新部署 API 並重新產生開發套件來包含更新。

## 產生 API 的 JavaScript 開發套件

### 在 API Gateway 中產生 API 的 JavaScript 開發套件

1. 登入位於 <https://console.aws.amazon.com/apigateway> 的 API Gateway 主控台。
2. 在包含階段 API 之名稱的方塊中，選擇 Stages (階段)。
3. 在 Stages (階段) 窗格中，選擇階段的名稱。
4. 在 SDK Generation (開發套件產生) 標籤上，針對 Platform (平台)，選擇 JavaScript。
5. 選擇 Generate SDK (產生開發套件)，然後遵循畫面上的說明下載 API Gateway 所產生的開發套件。
6. 遵循「[使用 API Gateway 為 REST API 所產生的 JavaScript 開發套件 \(p. 504\)](#)」中的說明使用所產生的開發套件。

每次更新 API，都必須重新部署 API 並重新產生開發套件來包含更新。

## 產生 API 的 Ruby 開發套件

### 在 API Gateway 中產生 API 的 Ruby 開發套件

1. 登入位於 <https://console.aws.amazon.com/apigateway> 的 API Gateway 主控台。
2. 在包含階段 API 之名稱的方塊中，選擇 Stages (階段)。
3. 在 Stages (階段) 窗格中，選擇階段的名稱。
4. 在 SDK Generation (開發套件產生) 標籤上，針對 Platform (平台)，選擇 Ruby。
  - a. 針對 Service Name (服務名稱)，指定您開發套件的名稱。例如，SimpleCalc。這會用來產生您 API 的 Ruby Gem 命名空間。此名稱必須全部都是字母 (a-zA-Z)，不含任何其他特殊字元或數字。
  - b. 針對 Ruby Gem Name (Ruby Gem 名稱)，指定 Ruby Gem 的名稱，以包含為您 API 產生的開發套件原始程式碼。預設是小寫的服務名稱加上 -sdk 字尾，例如 simplecalc-sdk。
  - c. 針對 Ruby Gem Version (Ruby Gem 版本)，指定所產生 Ruby Gem 的版本號碼。預設會設定為 1.0.0.
5. 選擇 Generate SDK (產生開發套件)，然後遵循畫面上的說明下載 API Gateway 所產生的開發套件。
6. 遵循「[使用 API Gateway 為 REST API 所產生的 Ruby 開發套件 \(p. 506\)](#)」中的說明使用所產生的開發套件。

每次更新 API，都必須重新部署 API 並重新產生開發套件來包含更新。

## 使用 AWS CLI 命令產生 API 開發套件

您可以使用 AWS CLI，透過呼叫 `get-sdk` 命令來產生及下載支援之平台的 API 開發套件。以下將示範其中一些支援的平台。

### 主題

- [使用 AWS CLI 產生及下載適用於 Android 的 Java 開發套件 \(p. 485\)](#)
- [使用 AWS CLI 產生及下載 JavaScript 開發套件 \(p. 485\)](#)

- 使用 AWS CLI 產生及下載 Ruby 開發套件 (p. 485)

## 使用 AWS CLI 產生及下載適用於 Android 的 Java 開發套件

若要在指定階段 (test) 產生及下載 API Gateway 所產生之 API (udpuvvzbkc) 的適用於 Android 的 Java 開發套件，請呼叫下列命令：

```
aws apigateway get-sdk \
--rest-api-id udpuvvzbkc \
--stage-name test \
--sdk-type android \
--parameters groupId='com.mycompany', \
    invokerPackage='com.mycompany.myApiSdk', \
    artifactId='myApiSdk', \
    artifactVersion='0.0.1' \
~/apps/myApi/myApi-android-sdk.zip
```

~/apps/myApi/myApi-android-sdk.zip 的最後一個輸入是名為 myApi-android-sdk.zip 之已下載開發套件檔案的路徑。

## 使用 AWS CLI 產生及下載 JavaScript 開發套件

若要在指定階段 (test) 產生及下載 API Gateway 所產生之 API (udpuvvzbkc) 的 JavaScript 開發套件，請呼叫下列命令：

```
aws apigateway get-sdk \
--rest-api-id udpuvvzbkc \
--stage-name test \
--sdk-type javascript \
~/apps/myApi/myApi-js-sdk.zip
```

~/apps/myApi/myApi-js-sdk.zip 的最後一個輸入是名為 myApi-js-sdk.zip 之已下載開發套件檔案的路徑。

## 使用 AWS CLI 產生及下載 Ruby 開發套件

若要在指定階段 (udpuvvzbkc) 產生及下載 API (test) 的 Ruby 開發套件，請呼叫下列命令：

```
aws apigateway get-sdk \
--rest-api-id udpuvvzbkc \
--stage-name test \
--sdk-type ruby \
--parameters service.name=myApiRubySdk,ruby.gem-name=myApi,ruby.gem- \
version=0.0.1 \
~/apps/myApi/myApi-ruby-sdk.zip
```

~/apps/myApi/myApi-ruby-sdk.zip 的最後一個輸入是名為 myApi-ruby-sdk.zip 之已下載開發套件檔案的路徑。

接下來，我們將示範如何使用所產生的開發套件來呼叫基礎 API。如需詳細資訊，請參閱 [透過產生的開發套件呼叫 REST API \(p. 499\)](#)。

## 簡易計算器的 Lambda 函數

我們將在圖中使用 Node.js Lambda 執行加、減、乘、除的二進位操作。

## 主題

- 簡易計算器之 Lambda 函數的輸入格式 (p. 486)
- 簡易計算器之 Lambda 函數的輸出格式 (p. 486)
- 簡易計算器的 Lambda 函數實作 (p. 486)
- 建立簡易計算器的 Lambda 函數 (p. 487)

## 簡易計算器之 Lambda 函數的輸入格式

此函數接受下列格式的輸入：

```
{ "a": "Number", "b": "Number", "op": "string"}
```

其中，op 可以是 (+, -, \*, /, add, sub, mul, div) 中的任一項。

## 簡易計算器之 Lambda 函數的輸出格式

若操作成功，其會傳回格式如下的結果：

```
{ "a": "Number", "b": "Number", "op": "string", "c": "Number"}
```

其中的 c 包含了計算結果。

## 簡易計算器的 Lambda 函數實作

以下是 Lambda 函數的實作：

```
console.log('Loading the calculator function');

exports.handler = function(event, context, callback) {
    console.log('Received event:', JSON.stringify(event, null, 2));
    if (event.a === undefined || event.b === undefined || event.op === undefined) {
        callback("400 Invalid Input");
    }

    var res = {};
    res.a = Number(event.a);
    res.b = Number(event.b);
    res.op = event.op;

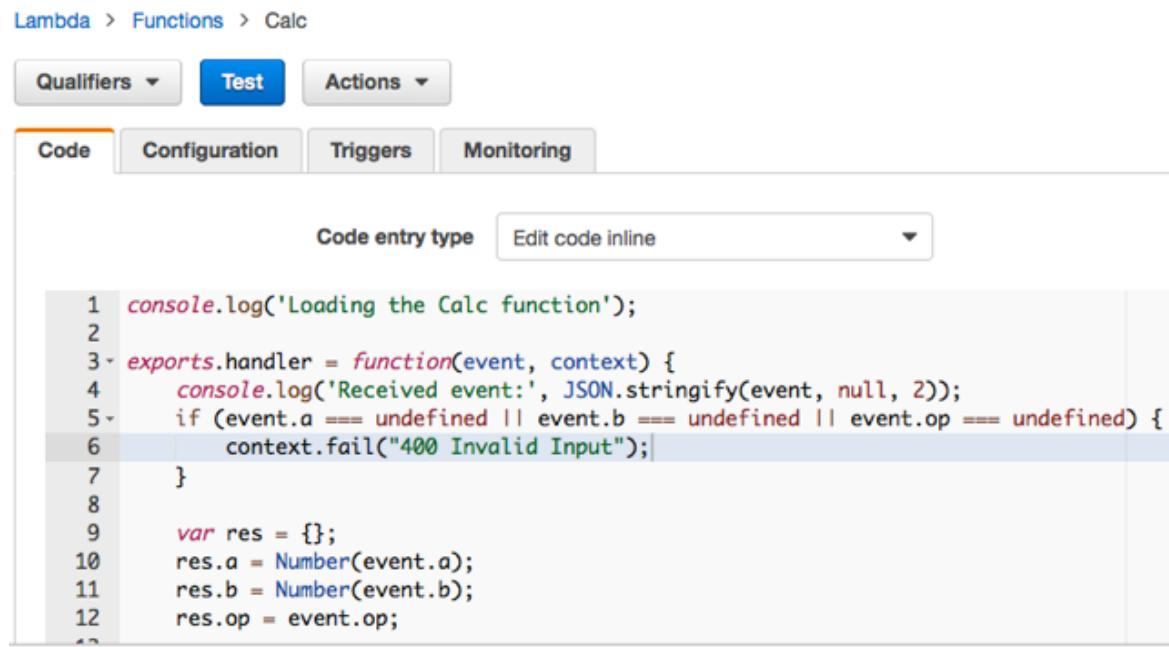
    if (isNaN(event.a) || isNaN(event.b)) {
        callback("400 Invalid Operand");
    }

    switch(event.op)
    {
        case "+":
        case "add":
            res.c = res.a + res.b;
            break;
        case "-":
        case "sub":
            res.c = res.a - res.b;
            break;
        case "*":
        case "mul":
            res.c = res.a * res.b;
            break;
        case "/":
        case "div":
            if (res.b === 0) {
                callback("400 Invalid Division");
            } else {
                res.c = res.a / res.b;
            }
            break;
    }
    callback(null, res);
}
```

```
        break;
    case "/":
    case "div":
        res.c = res.b === 0 ? NaN : Number(event.a) / Number(event.b);
        break;
    default:
        callback("400 Invalid Operator");
        break;
    }
    callback(null, res);
};
```

## 建立簡易計算器的 Lambda 函數

您可以使用 <https://console.aws.amazon.com/lambda/> 處的 AWS Lambda 主控台，將上列程式碼列表貼至線上程式碼編輯器來建立函數，如下所示：



## API Gateway 中的簡易計算器 API

我們的簡易計算器 API 公開三種方法 (GET、POST、PUT) 來呼叫 the section called “[簡易計算器的 Lambda 函數](#)” (p. 485)。此 API 的圖形呈現如下：

The screenshot shows the AWS Lambda console interface. On the left, there's a sidebar with links: APIs, SimpleCalc, Resources (which is selected and highlighted in orange), Stages, Authorizers, Models, Dashboard, Usage Plans, API Keys, Custom Domain Names, Client Certificates, and Settings. The main content area is titled 'Actions' and shows the 'Resources' tab. It displays a hierarchical structure of API endpoints. At the top level, there's a '/' endpoint with two methods: 'GET' and 'POST'. Below it, there's a '/{a}' endpoint with a single method 'ANY'. Underneath that is a '/{b}' endpoint also with a single 'ANY' method. Finally, there's a '/{op}' endpoint with a single 'GET' method.

這三種方法顯示提供後端 Lambda 函數輸入來執行相同操作的不同方式：

- GET /?a=...&b=...&op=... 方法會使用查詢參數來指定輸入。
- POST / 方法使用 {"a": "Number", "b": "Number", "op": "string"} 的 JSON 承載來指定輸入。
- GET /{a}/{b}/{op} 方法會使用路徑參數來指定輸入。

如果未定義，則 API Gateway 透過結合 HTTP 方法和路徑部分，來產生對應的開發套件方法名稱。根路徑部分 (/) 稱為 Api Root。例如，API 方法 GET /?a=...&b=...&op=... 的預設 Java 開發套件方法名稱是 getABOp、POST / 的預設開發套件方法名稱是 postApiRoot，而 GET /{a}/{b}/{op} 的預設開發套件方法名稱是 getABOp。個別開發套件可能會自訂慣例。請參閱開發套件特定方法名稱之所產生開發套件來源中的文件。

您可以且應該在每個 API 方法上指定 `operationName` 屬性，來覆寫預設開發套件方法名稱。您可以在使用 API Gateway REST API 建立 API 方法或更新 API 方法 時這麼做。在 API Swagger 定義中，您可以設定 `operationId` 以達到相同的結果。

顯示如何使用此 API 之 API Gateway 所產生的開發套件來呼叫這些方法之前，讓我們短暫回想一下如何設定它們。如需詳細說明，請參閱在 Amazon API Gateway 中建立 REST API (p. 167)。如果您是初次使用 API Gateway，請先參閱建置具有 Lambda 整合的 API Gateway API (p. 24)。

## 建立輸入和輸出的模型

為了在開發套件中指定強型別輸入，我們建立 API 的 Input 模型：

Provide a name, content type, and a schema for your model. Models use [JSON schema](#).

Model name\* Input

Content type\* application/json

Model description

Model schema\*

```
1 - {
2   "$schema": "http://json-schema.org/draft-04/schema#",
3   "type": "object",
4   "properties": {
5     "a": {"type": "number"},
6     "b": {"type": "number"},
7     "op": {"type": "string"}
8   },
9   "title": "Input"
10 }
```

\* Required      Cancel      Create model

同樣地，為了說明回應內文資料類型，我們在 API Gateway 建立下列模型：

```
{  
  "$schema": "http://json-schema.org/draft-04/schema#",  
  "type": "object",  
  "properties": {  
    "c": {"type": "number"}  
  },  
  "title": "Output"  
}
```

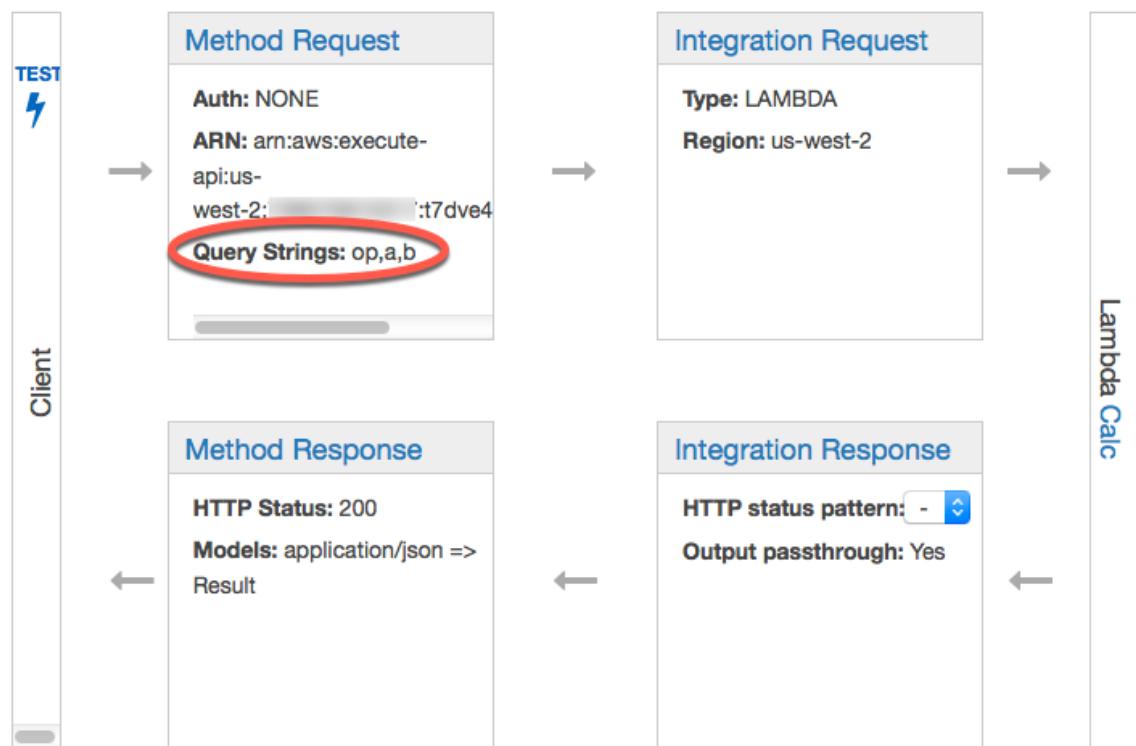
和

```
{  
  "$schema": "http://json-schema.org/draft-04/schema#",  
  "type": "object",  
  "properties": {  
    "input": {  
      "$ref": "https://apigateway.amazonaws.com/restapis/t7dve4zn36/models/Input"  
    },  
    "output": {  
      "$ref": "https://apigateway.amazonaws.com/restapis/t7dve4zn36/models/Output"  
    }  
  },  
  "title": "Result"  
}
```

## 設定 GET / 方法查詢參數

針對 GET /?a=..&b=..&op=.. 方法，在 Method Request (方法請求) 中宣告查詢參數：

## / - GET - Method Execution



### 設定承載做為後端輸入的資料模型

針對 POST / 方法，我們建立 Input 模型，並將它新增至方法請求來定義輸入資料的形狀。

[← Method Execution](#) / - POST - Method Request

Provide information about this method's authorization settings and the parameters it can receive.

Authorization Settings

Authorization NONE 

API Key Required false 

▶ URL Query String Parameters

▶ HTTP Request Headers

▼ Request Models [Create a Model](#)

Content type	Model name	
application/json	<b>Input</b> 	

 [Add model](#)

使用此模型，將 Input 物件執行個體化，您的 API 客戶就可以呼叫開發套件來指定輸入。沒有此模型，您的客戶就需要建立字典物件來代表 Lambda 函數的 JSON 輸入。

### 設定後端結果輸出的資料模型

針對所有這三種方法，我們建立 Result 模型，並將它新增至方法的 Method Response 來定義 Lambda 函數所傳回輸出的形狀。

[Method Execution](#) /{a}/{b}/{op} - GET - Method Response

Provide information about this method's response types, their headers and content types.

HTTP Status	
▼ 200 	

Response Headers for 200

Name	
No headers	
<a href="#">+ Add Header</a>	

Response Models for 200

Create a model	
Content type	Models
application/json	 
	<a href="#">+ Add Response Model</a>

[+ Add Response](#)

使用此模型，您的 API 客戶可以讀取 Result 物件的屬性來剖析成功的輸出。沒有此模型，客戶就需要建立字典物件來代表 JSON 輸出。

此外，您也可以遵循 Swagger API 定義 (p. 492)來建立和設定 API。

## 簡易計算器 API OpenAPI 定義

以下是簡易計算器 API 的 OpenAPI 定義。您可以將它匯入到您的帳戶。不過，匯入後，您需要重設 Lambda 函數 (p. 485)上的資源型許可。若要這樣做，請在 API Gateway 主控台中，從 Integration Request (整合請求) 重新選取您在帳戶中建立的 Lambda 函數。這會使 API Gateway 主控台重設必要的許可。或者，您可以對 Lambda 命令 `add-permission` 使用 AWS Command Line Interface。

### OpenAPI 2.0

```
{  
  "swagger": "2.0",  
  "info": {  
    "version": "2016-09-29T20:27:30Z",  
    "title": "SimpleCalc"  
  },  
  "host": "t6dve4zn25.execute-api.us-west-2.amazonaws.com",  
  "basePath": "/demo",  
  "schemes": [  
    "https"  
  ],  
  "paths": {  
    "/": {  
      "get": {  
        "consumes": [  
          "application/json"  
        ],  
        "produces": [  
          "application/json"  
        ]  
      }  
    }  
  }  
}
```

```
"parameters": [
  {
    "name": "op",
    "in": "query",
    "required": false,
    "type": "string"
  },
  {
    "name": "a",
    "in": "query",
    "required": false,
    "type": "string"
  },
  {
    "name": "b",
    "in": "query",
    "required": false,
    "type": "string"
  }
],
"responses": {
  "200": {
    "description": "200 response",
    "schema": {
      "$ref": "#/definitions/Result"
    }
  }
},
"x-amazon-apigateway-integration": {
  "requestTemplates": {
    "application/json": "#set($inputRoot = $input.path('$'))\n\n\"a\" : $input.params('a'),\n  \"b\" : $input.params('b'),\n  \"op\" : \"$input.params('op')\"\n"
  },
  "uri": "arn:aws:apigateway:us-west-2:lambda:path/2015-03-31/functions/arn:aws:lambda:us-west-2:123456789012:function:Calc/invocations",
  "passthroughBehavior": "when_no_templates",
  "httpMethod": "POST",
  "responses": {
    "default": {
      "statusCode": "200",
      "responseTemplates": {
        "application/json": "#set($inputRoot = $input.path('$'))\n\n\"input\" : {\n  \"a\" : $inputRoot.a,\n  \"b\" : $inputRoot.b,\n  \"op\" : \"$inputRoot.op\"\n},\n  \"output\" : {\n    \"c\" : $inputRoot.c\n  }\n}"
      }
    }
  },
  "type": "aws"
},
"post": {
  "consumes": [
    "application/json"
  ],
  "produces": [
    "application/json"
  ],
  "parameters": [
    {
      "in": "body",
      "name": "Input",
      "required": true,
      "schema": {
        "$ref": "#/definitions/Input"
      }
    }
  ]
}
```

```
        }
    ],
    "responses": {
        "200": {
            "description": "200 response",
            "schema": {
                "$ref": "#/definitions/Result"
            }
        }
    },
    "x-amazon-apigateway-integration": {
        "uri": "arn:aws:apigateway:us-west-2:lambda:path/2015-03-31/functions/arn:aws:lambda:us-west-2:123456789012:function:Calc/invocations",
        "passthroughBehavior": "when_no_match",
        "httpMethod": "POST",
        "responses": {
            "default": {
                "statusCode": "200",
                "responseTemplates": {
                    "application/json": "#set($inputRoot = $input.path('$'))\n{\n\"input\" : {\n    \"a\" : $inputRoot.a,\n    \"b\" : $inputRoot.b,\n    \"op\" : \"$inputRoot.op\"\n},\n    \"output\" : {\n        \"c\" : $inputRoot.c\n    }\n}\n"
                }
            }
        },
        "type": "aws"
    }
},
"/{a)": {
    "x-amazon-apigateway-any-method": {
        "consumes": [
            "application/json"
        ],
        "produces": [
            "application/json"
        ],
        "parameters": [
            {
                "name": "a",
                "in": "path",
                "required": true,
                "type": "string"
            }
        ],
        "responses": {
            "404": {
                "description": "404 response"
            }
        },
        "x-amazon-apigateway-integration": {
            "requestTemplates": {
                "application/json": "{\"statusCode\": 200}"
            },
            "passthroughBehavior": "when_no_match",
            "responses": {
                "default": {
                    "statusCode": "404",
                    "responseTemplates": {
                        "application/json": "{ \"Message\" : \"Can't $context.httpMethod $context.resourcePath\" }"
                    }
                }
            }
        },
        "type": "mock"
    }
}
```

```
        },
    },
    "/{a}/{b}": {
        "x-amazon-apigateway-any-method": {
            "consumes": [
                "application/json"
            ],
            "produces": [
                "application/json"
            ],
            "parameters": [
                {
                    "name": "a",
                    "in": "path",
                    "required": true,
                    "type": "string"
                },
                {
                    "name": "b",
                    "in": "path",
                    "required": true,
                    "type": "string"
                }
            ],
            "responses": {
                "404": {
                    "description": "404 response"
                }
            },
            "x-amazon-apigateway-integration": {
                "requestTemplates": {
                    "application/json": "{\"statusCode\": 200}"
                },
                "passThroughBehavior": "when_no_match",
                "responses": {
                    "default": {
                        "statusCode": "404",
                        "responseTemplates": {
                            "application/json": "{ \"Message\" : \"Can't $context.httpMethod\n$context.resourcePath\" }"
                        }
                    }
                },
                "type": "mock"
            }
        }
    },
    "/{a}/{b}/{op}": {
        "get": {
            "consumes": [
                "application/json"
            ],
            "produces": [
                "application/json"
            ],
            "parameters": [
                {
                    "name": "a",
                    "in": "path",
                    "required": true,
                    "type": "string"
                },
                {
                    "name": "b",
                    "in": "path",
                    "required": true,
                    "type": "string"
                }
            ]
        }
    }
}
```

```
        "type": "string"
    },
    {
        "name": "op",
        "in": "path",
        "required": true,
        "type": "string"
    }
],
"responses": {
    "200": {
        "description": "200 response",
        "schema": {
            "$ref": "#/definitions/Result"
        }
    }
},
"x-amazon-apigateway-integration": {
    "requestTemplates": {
        "application/json": "#set($inputRoot = $input.path('$'))\n\n\"a\" : $input.params('a'),\n  \"b\" : $input.params('b'),\n  \"op\" : \"$input.params('op')\"\n"
    },
    "uri": "arn:aws:apigateway:us-west-2:lambda:path/2015-03-31/functions/arn:aws:lambda:us-west-2:123456789012:function:Calc/invocations",
    "passthroughBehavior": "when_no_templates",
    "httpMethod": "POST",
    "responses": {
        "default": {
            "statusCode": "200",
            "responseTemplates": {
                "application/json": "#set($inputRoot = $input.path('$'))\n\n\"input\" : {\n    \"a\" : $inputRoot.a,\n    \"b\" : $inputRoot.b,\n    \"op\" : \"$inputRoot.op\"\n},\n    \"output\" : {\n        \"c\" : $inputRoot.c\n    }\n}"
            }
        },
        "type": "aws"
    }
}
},
"definitions": {
    "Input": {
        "type": "object",
        "properties": {
            "a": {
                "type": "number"
            },
            "b": {
                "type": "number"
            },
            "op": {
                "type": "string"
            }
        },
        "title": "Input"
    },
    "Output": {
        "type": "object",
        "properties": {
            "c": {
                "type": "number"
            }
        },
        "title": "Output"
    }
}
```

```
        },
        "Result": {
            "type": "object",
            "properties": {
                "input": {
                    "$ref": "#/definitions/Input"
                },
                "output": {
                    "$ref": "#/definitions/Output"
                }
            },
            "title": "Result"
        }
    }
}
```

## 在 Amazon API Gateway 中呼叫 REST API

呼叫已部署的 API 需要將請求提交至 API Gateway 元件服務的 URL 以執行 API (也稱為 `execute-api`)。

REST API 的基本 URL 格式如下所示：

```
https://{restapi_id}.execute-api.{region}.amazonaws.com/{stage_name}/
```

其中 *{restapi\_id}* 是 API 識別符，*{region}* 是已部署 API 的區域，而 *{stage\_name}* 是 API 部署的階段名稱。

### Important

在您可以叫用 API 前，您必須在 API Gateway 中進行部署。若要執行此操作，請遵循 [在 Amazon API Gateway 中部署 REST API \(p. 457\)](#) 中的說明。

### 主題

- [在 API Gateway 主控台中取得 API 的呼叫 URL \(p. 497\)](#)
- [使用 API Gateway 主控台測試 REST API 方法 \(p. 498\)](#)
- [使用 Postman 來呼叫 REST API \(p. 499\)](#)
- [透過產生的開發套件呼叫 REST API \(p. 499\)](#)
- [透過 AWS Amplify JavaScript 程式庫呼叫 REST API \(p. 517\)](#)
- [如何呼叫私有 API \(p. 517\)](#)

## 在 API Gateway 主控台中取得 API 的呼叫 URL

您可以在 API Gateway 主控台中的 API Stage Editor (階段編輯器) 中找到 REST API 的根 URL。它會列為頂端的 Invoke URL (呼叫 URL)。如果 API 的根資源公開 GET 方法而不需要使用者身分驗證，您可以按一下 Invoke URL (呼叫 URL) 連結來呼叫方法。您也可以合併 API 之已匯出 OpenAPI 定義檔的 host 與 basePath 欄位，來建構這個根 URL。

如果 API 允許匿名存取，您可以使用任何 Web 瀏覽器，將適當的引動過程 URL 複製並貼到瀏覽器的網址列，來呼叫任何 GET 方法呼叫。對於其他方法或任何需要身分驗證的呼叫，由於您必須指定承載或簽署請求，因此引動過程會更複雜。您可以使用其中一個 AWS 開發套件，在 HTML 頁面或用戶端應用程式後方的指令碼中處理這些作業。

若要進行測試，您可以使用 API Gateway 主控台來呼叫使用 API Gateway TestInvoke 功能的 API，這會略過呼叫 Invoke URL，並允許在部署 API 之前進行 API 測試。或者，您可以使用 [Postman](#) 應用程式來測試已成功部署的 API，而不需要撰寫指令碼或用戶端。

#### Note

引動過程 URL 中的查詢字串參數值不能包含 %%。

## 使用 API Gateway 主控台測試 REST API 方法

使用 API Gateway 主控台測試 REST API 方法。

#### 主題

- [先決條件 \(p. 498\)](#)
- [使用 API Gateway 主控台測試方法 \(p. 498\)](#)

## 先決條件

- 您必須指定所要測試之方法的設定。請遵循[在 API Gateway 中設定 REST API 方法 \(p. 188\)](#)中的說明進行。

## 使用 API Gateway 主控台測試方法

#### Important

使用 API Gateway 主控台測試方法可能會對資源產生無法復原的變更。使用 API Gateway 主控台測試方法與在 API Gateway 主控台以外呼叫方法相同。例如，如果您使用 API Gateway 主控台呼叫刪除 API 資源的方法，當方法呼叫成功時，則會刪除 API 的資源。

1. 登入位於 <https://console.aws.amazon.com/apigateway> 的 API Gateway 主控台。
2. 在包含方法 API 之名稱的方塊中，選擇 Resources (資源)。
3. 在 Resources (資源) 窗格中，選擇您要測試的方法。
4. 在 Method Execution (方法執行) 窗格的 Client (用戶端) 方塊中選擇 TEST。在任何顯示的方塊 (例如 Query Strings (查詢字串)、Headers (標頭) 與 Request Body (請求內文)) 中輸入值。

如需指定更多選項，請聯絡 API 擁有者。

5. 選擇 Test (測試)。下列資訊會隨即顯示：

- Request (請求) 是針對方法所呼叫的資源路徑。
- Status (狀態) 是回應的 HTTP 狀態碼。
- Latency (延遲) 是從發起人收到請求到傳回回應之間的時間。
- Response Body (回應內文) 是 HTTP 回應內文。
- Response Headers (回應標頭) 是 HTTP 回應標頭。

#### Tip

根據對應，HTTP 狀態碼、回應內文與回應標頭可能會與 Lambda 函數、HTTP 代理或 AWS 服務代理所傳送的內容不同。

- Logs (日誌) 是模擬的 Amazon CloudWatch Logs 項目，若在 API Gateway 主控台外部呼叫此方法，則會事先寫入。

#### Note

雖然 CloudWatch Logs 項目是模擬的，但方法呼叫的結果是真的。

除了使用 API Gateway 主控台，您還可以使用 AWS CLI 或適用於 API Gateway 的 AWS 開發套件來測試呼叫方法。若要使用 AWS CLI 來執行這項操作，請參閱 [test-invoke-method](#)。

## 使用 Postman 來呼叫 REST API

[Postman](#) 應用程式是一個可在 API Gateway 中測試 REST API 的便利工具。以下說明會逐步帶您了解使用 Postman 應用程式呼叫 API 的基本步驟。如需詳細資訊，請參閱 [Postman 說明](#)。

1. 啟動 Postman。
2. 在網址列中輸入請求的端點 URL，然後從網址列左側的下拉式清單中，選擇適當的 HTTP 方法。
3. 如有需要，請選擇 Authorization (授權) 標籤。針對授權 Type (類型)，選擇 AWS Signature (AWS 簽章)。在 AccessKey 輸入欄位中，輸入您的 AWS IAM 使用者存取金鑰 ID。在 SecretKey 中，輸入您的 IAM 使用者私密金鑰。指定符合引動過程 URL 中指定之區域的適當 AWS 區域。在 Service Name (服務名稱) 中輸入 **execute-api**。
4. 選擇 Headers (標頭) 標籤。(選擇性) 刪除任何現有的標頭。這會清除可能造成錯誤的任何過時設定。新增任何必要的自訂標頭。例如，如果已啟用 API 金鑰，您可以在這裡設定 **x-api-key:{api\_key}** 名稱/值對。
5. 選擇 Send (傳送) 以提交請求與接收回應。

如需使用 Postman 的範例，請參閱「[呼叫具有 API Gateway Lambda 授權方的 API \(p. 360\)](#)」。

## 透過產生的開發套件呼叫 REST API

本節示範如何在以 Java、適用於 Android 的 Java、JavaScript、Ruby、Objective-C 與 Swift 撰寫的用戶端應用程式中，透過產生的開發套件來呼叫 API。

### 主題

- [使用 API Gateway 為 REST API 所產生的 Java 開發套件 \(p. 499\)](#)
- [使用 API Gateway 為 REST API 所產生的 Android 開發套件 \(p. 502\)](#)
- [使用 API Gateway 為 REST API 所產生的 JavaScript 開發套件 \(p. 504\)](#)
- [使用 API Gateway 為 REST API 所產生的 Ruby 開發套件 \(p. 506\)](#)
- [在 Objective-C 或 Swift 中使用 API Gateway 為 REST API 產生的 iOS 開發套件 \(p. 508\)](#)

## 使用 API Gateway 為 REST API 所產生的 Java 開發套件

在本節中，我們以[簡易計算機 \(p. 492\)](#) API 為例，概述使用 API Gateway 為 REST API 所產生之 Java 開發套件的步驟。繼續之前，您必須完成[使用 API Gateway 主控台產生 API 開發套件 \(p. 482\)](#)中的步驟。

### 安裝及使用 API Gateway 所產生的 Java 開發套件

1. 將您稍早下載之 API Gateway 所產生的 .zip 檔案內容解壓縮。
2. 下載並安裝 [Apache Maven](#) (必須是 3.5 版或更新版本)。
3. 下載並安裝 [JDK](#) (必須是 1.8 版或更新版本)。
4. 設定 JAVA\_HOME 環境變數。
5. 前往 pom.xml 檔案所在之解壓縮的開發套件資料夾。此資料夾預設為 generated-code。執行 mvn install 命令，將已編譯的成品檔案安裝到您的本機 Maven 儲存庫。這會建立 target 資料夾，其中包含已編譯的開發套件程式庫。
6. 在空目錄中輸入下列命令建立用戶端專案 Stub，以使用安裝的開發套件程式庫呼叫 API。

```
mvn -B archetype:generate \
-DarchetypeGroupId=org.apache.maven.archetypes \
-DgroupId=examples.aws.apig.simpleCalc.sdk.app \
-DartifactId=SimpleCalc-sdkClient
```

### Note

上述命令中加入了分隔符號 \ 以利閱讀。整個命令應該放在一行且不使用分隔符號。

此命令會建立應用程式 Stub。應用程式 Stub 在專案根目錄 (上述命令中的 *SimpleCalc-sdkClient*) 下包含 pom.xml 檔案與 src 資料夾。一開始有兩個來源檔案：src/main/java/{package-path}/App.java 與 src/test/java/{package-path}/AppTest.java。在此範例中，{package-path} 是 examples/aws/apig/simpleCalc/sdk/app。此套件路徑衍生自 DarchetypeGroupId 值。您可以使用 App.java 檔案作為用戶端應用程式的範本，而且您可以視需要在相同的資料夾中新增其他檔案。您可以使用 AppTest.java 檔案作為應用程式的單元測試範本，而且您可以視需要將其他測試程式碼檔案新增至相同的測試資料夾。

7. 將所產生 pom.xml 檔案中的套件相依性更新為以下內容，並視需要替代成您專案的 groupId、artifactId、version 與 name 屬性：

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>examples.aws.apig.simpleCalc.sdk.app</groupId>
    <artifactId>SimpleCalc-sdkClient</artifactId>
    <packaging>jar</packaging>
    <version>1.0-SNAPSHOT</version>
    <name>SimpleCalc-sdkClient</name>
    <url>http://maven.apache.org</url>

    <dependencies>
        <dependency>
            <groupId>com.amazonaws</groupId>
            <artifactId>aws-java-sdk-core</artifactId>
            <version>1.11.94</version>
        </dependency>
        <dependency>
            <groupId>my-apig-api-examples</groupId>
            <artifactId>simple-calc-sdk</artifactId>
            <version>1.0.0</version>
        </dependency>

        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.12</version>
            <scope>test</scope>
        </dependency>

        <dependency>
            <groupId>commons-io</groupId>
            <artifactId>commons-io</artifactId>
            <version>2.5</version>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.5.1</version>
                <configuration>
                    <source>1.8</source>
                    <target>1.8</target>
                </configuration>
            </plugin>
        </plugins>
    </build>

```

```
</plugins>
</build>
</project>
```

#### Note

如果 aws-java-sdk-core 的相依成品有較新版本與以上指定的版本 (1.11.94) 不相容，則必須將 <version> 標籤更新為較新版本。

8. 接下來，我們將示範如何透過呼叫開發套件的 getABOp(GetABOpRequest req)、getApiRoot(GetApiRootRequest req) 與 postApiRoot(PostApiRootRequest req) 方法，使用開發套件來呼叫 API。這些方法分別對應到具有 {"a": x, "b": y, "op": "operator"} API 請求承載的 GET /{a}/{b}/{op}、GET /?a={x}&b={y}&op={operator} 與 POST / 方法。

請更新 App.java 檔案如下：

```
package examples.aws.apig.simpleCalc.sdk.app;

import java.io.IOException;

import com.amazonaws.opensdk.config.ConnectionConfiguration;
import com.amazonaws.opensdk.config.TimeoutConfiguration;

import examples.aws.apig.simpleCalc.sdk.*;
import examples.aws.apig.simpleCalc.sdk.model.*;
import examples.aws.apig.simpleCalc.sdk.SimpleCalcSdk.*;

public class App
{
    SimpleCalcSdk sdkClient;

    public App() {
        initSdk();
    }

    // The configuration settings are for illustration purposes and may not be a
    // recommended best practice.
    private void initSdk() {
        sdkClient = SimpleCalcSdk.builder()
            .connectionConfiguration(
                new ConnectionConfiguration()
                    .maxConnections(100)
                    .connectionMaxIdleMillis(1000))
            .timeoutConfiguration(
                new TimeoutConfiguration()
                    .httpRequestTimeout(3000)
                    .totalExecutionTimeout(10000)
                    .socketTimeout(2000))
            .build();
    }

    // Calling shutdown is not necessary unless you want to exert explicit control of
    // this resource.
    public void shutdown() {
        sdkClient.shutdown();
    }

    // GetABOpResult getABOp(GetABOpRequest getABOpRequest)
    public Output getResultWithPathParameters(String x, String y, String operator) {
        operator = operator.equals"+" ? "add" : operator;
        operator = operator.equals"/" ? "div" : operator;
```

```
GetABOpResult abopResult = sdkClient.getABOp(new
GetABOpRequest().a(x).b(y).op(operator));
    return abopResult.getResult().getOutput();
}

public Output getResultWithQueryParameters(String a, String b, String op) {
    GetApiRootResult rootResult = sdkClient.getApiRoot(new
GetApiRootRequest().a(a).b(b).op(op));
    return rootResult.getResult().getOutput();
}

public Output getResultByPostInputBody(Double x, Double y, String o) {
    PostApiRootResult postResult = sdkClient.postApiRoot(
        new PostApiRootRequest().input(new Input().a(x).b(y).op(o)));
    return postResult.getResult().getOutput();
}

public static void main( String[] args )
{
    System.out.println( "Simple calc" );
    // to begin
    App calc = new App();

    // call the SimpleCalc API
    Output res = calc.getResultWithPathParameters("1", "2", "-");
    System.out.printf("GET /1/2/-: %s\n", res.getc());

    // Use the type query parameter
    res = calc.getResultWithQueryParameters("1", "2", "+");
    System.out.printf("GET /?a=1&b=2&op=+: %s\n", res.getc());

    // Call POST with an Input body.
    res = calc.getResultByPostInputBody(1.0, 2.0, "*");
    System.out.printf("PUT /\n\n{\\"a\\":1, \\"b\\":2, \\"op\\":\\"*\\"}\n %s\n",
res.getc());

}
}
```

在上述範例中，用來執行個體化開發套件用戶端的組態設定僅供說明，不一定是建議的最佳實務。此外，呼叫 `sdkClient.shutdown()` 是選擇性的，特別是如果您需要精確控制何時釋放資源。

我們已示範使用 Java 開發套件呼叫 API 的基本模式。您可以延伸說明來呼叫其他 API 方法。

## 使用 API Gateway 為 REST API 所產生的 Android 開發套件

在本節中，我們將概述使用 API Gateway 為 REST API 所產生之 Android 開發套件的步驟。您必須已完成[使用 API Gateway 主控台產生 API 開發套件 \(p. 482\)](#)中的步驟，才能繼續往下進行。

### Note

所產生的開發套件與 Android 4.4 (含) 以前的版本不相容。如需詳細資訊，請參閱 the section called “[重要說明](#)” (p. 630)。

### 安裝及使用 API Gateway 所產生的 Android 開發套件

1. 將您稍早下載之 API Gateway 所產生的 .zip 檔案內容解壓縮。
2. 下載並安裝 [Apache Maven](#) (最好是 3.x 版)。
3. 下載並安裝 [JDK](#) (最好是 1.7 版或更新版本)。
4. 設定 JAVA\_HOME 環境變數。

5. 執行 mvn install 命令，將已編譯的成品檔案安裝到您的本機 Maven 儲存庫。這會建立 target 資料夾，其中包含已編譯的開發套件程式庫。
6. 將 target 資料夾中的開發套件檔案（其名稱衍生自您在產生開發套件時所指定的 Artifact Id (成品 ID) 與 Artifact Version (成品版本)，例如 simple-calcsdk-1.0.0.jar），連同 target/lib 資料夾中的所有其他程式庫，一起複製到您專案的 lib 資料夾中。

如果您使用 Android Studio，請在您的用戶端應用程式模組下建立一個 libs 資料夾，然後將必要的 .jar 檔案複製到此資料夾中。確認模組之 Gradle 檔案中的相依性區段包含以下內容。

```
compile fileTree(include: ['*.jar'], dir: 'libs')
compile fileTree(include: ['*.jar'], dir: 'app/libs')
```

確定未宣告重複的 .jar 檔案。

7. 使用 ApiClientFactory 類別初始化 API Gateway 所產生的開發套件。例如：

```
ApiClientFactory factory = new ApiClientFactory();

// Create an instance of your SDK. Here, 'SimpleCalcClient.java' is the compiled java
// class for the SDK generated by API Gateway.
final SimpleCalcClient client = factory.build(SimpleCalcClient.class);

// Invoke a method:
//   For the 'GET /?a=1&b=2&op=+' method exposed by the API, you can invoke it by
//   calling the following SDK method:

Result output = client.rootGet("1", "2", "+");

//   where the Result class of the SDK corresponds to the Result model of the API.
//

//   For the 'GET /{a}/{b}/{op}' method exposed by the API, you can call the following
//   SDK method to invoke the request,

Result output = client.aBOpGet(a, b, c);

//   where a, b, c can be "1", "2", "add", respectively.

//   For the following API method:
//     POST /
//     host: ...
//     Content-Type: application/json
//
//       { "a": 1, "b": 2, "op": "+" }
// you can call invoke it by calling the rootPost method of the SDK as follows:
Input body = new Input();
input.a=1;
input.b=2;
input.op="+";
Result output = client.rootPost(body);

//   where the Input class of the SDK corresponds to the Input model of the API.

// Parse the result:
//   If the 'Result' object is { "a": 1, "b": 2, "op": "add", "c":3 }, you retrieve
//   the result 'c' ) as

String result=output.c;
```

8. 若要使用 Amazon Cognito 登入資料提供者授權呼叫您的 API，請使用 ApiClientFactory 類別透過 API Gateway 所產生的開發套件來傳遞一組 AWS 登入資料，如下列範例所示。

```
// Use CognitoCachingCredentialsProvider to provide AWS credentials
// for the ApiClientFactory
AWSCredentialsProvider credentialsProvider = new CognitoCachingCredentialsProvider(
    context, // activity context
    "identityPoolId", // Cognito identity pool id
    Regions.US_EAST_1 // region of Cognito identity pool
);

ApiClientFactory factory = new ApiClientFactory()
    .credentialsProvider(credentialsProvider);
```

- 若要使用 API Gateway 所產生的開發套件來設定 API 金鑰，請使用類似如下的程式碼。

```
ApiClientFactory factory = new ApiClientFactory()
    .apiKey("YOUR_API_KEY");
```

## 使用 API Gateway 為 REST API 所產生的 JavaScript 開發套件

### Note

這些說明假設您已完成[使用 API Gateway 主控台產生 API 開發套件 \(p. 482\)](#)中的說明。

### Important

如果您的 API 只定義了 ANY 方法，則產生的軟體開發套件將不會包含 `apigClient.js` 檔案，而且您將需要自行定義 ANY 方法。

若要安裝，請啟動並呼叫 API Gateway 為 REST API 所產生的 JavaScript 開發套件。

- 將您稍早下載之 API Gateway 所產生的 `.zip` 檔案內容解壓縮。
- 針對 API Gateway 所產生之開發套件將會呼叫的所有方法，啟用跨來源資源分享 (CORS)。如需說明，請參閱[為 REST API 資源啟用 CORS \(p. 371\)](#)。
- 在您的網頁中，包含下列指令碼的參考。

```
<script type="text/javascript" src="lib/axios/dist/axios.standalone.js"></script>
<script type="text/javascript" src="lib/CryptoJS/rollups/hmac-sha256.js"></script>
<script type="text/javascript" src="lib/CryptoJS/rollups/sha256.js"></script>
<script type="text/javascript" src="lib/CryptoJS/components/hmac.js"></script>
<script type="text/javascript" src="lib/CryptoJS/components/enc-base64.js"></script>
<script type="text/javascript" src="lib/url-template/url-template.js"></script>
<script type="text/javascript" src="lib/apiGatewayCore/sigV4Client.js"></script>
<script type="text/javascript" src="lib/apiGatewayCore/apiGatewayClient.js"></script>
<script type="text/javascript" src="lib/apiGatewayCore/simpleHttpClient.js"></script>
<script type="text/javascript" src="lib/apiGatewayCore/utils.js"></script>
<script type="text/javascript" src="apigClient.js"></script>
```

- 在您的程式碼中，使用類似如下的程式碼來初始化 API Gateway 所產生的開發套件。

```
var apigClient = apigClientFactory.newClient();
```

若要使用 AWS 登入資料來初始化 API Gateway 所產生的開發套件，請使用類似如下的程式碼。如果您使用 AWS 登入資料，則會簽署 API 的所有請求。

```
var apigClient = apigClientFactory.newClient({
  accessKey: 'ACCESS_KEY',
  secretKey: 'SECRET_KEY',
});
```

若要搭配 API Gateway 所產生的開發套件使用 API 金鑰，請使用類似如下的程式碼將 API 金鑰當做參數傳遞給 Factory 物件。如果您使用 API 金鑰，則會將它指定為 x-api-key 標頭的一部分，而且 API 的所有請求都會經過簽署。這表示您必須為每個請求設定適當的 CORS Accept 標頭。

```
var apigClient = apigClientFactory.newClient({
  apiKey: 'API_KEY'
});
```

5. 使用類似如下的程式碼，在 API Gateway 中呼叫 API 方法。每個呼叫會傳回成功與失敗回呼的結果。

```
var params = {
  // This is where any modeled request parameters should be added.
  // The key is the parameter name, as it is defined in the API in API Gateway.
  param0: '',
  param1: ''
};

var body = {
  // This is where you define the body of the request,
};

var additionalParams = {
  // If there are any unmodeled query parameters or headers that must be
  // sent with the request, add them here.
  headers: {
    param0: '',
    param1: ''
  },
  queryParams: {
    param0: '',
    param1: ''
  }
};

apigClient.methodName(params, body, additionalParams)
  .then(function(result){
    // Add success callback code here.
  }).catch( function(result){
    // Add error callback code here.
  });

```

此處，**methodName** 是從方法請求的資源路徑與 HTTP 動詞建構而來。對於 SimpleCalc API，開發套件方法用於 API 方法

```
1. GET /?a=...&b=...&op=...
2. POST /
  {
    "a": ...,
    "b": ...,
    "op": ...
  }
3. GET /{a}/{b}/{op}
```

對應的開發套件方法如下所示：

```
1. rootGet(params);      // where params={"a": ..., "b": ..., "op": ...} is resolved to
                           the query parameters
```

```
2. rootPost(null, body); // where body={"a": ..., "b": ..., "op": ...}
3. aBOpGet(params);      // where params={"a": ..., "b": ..., "op": ...} is resolved to
                           the path parameters
```

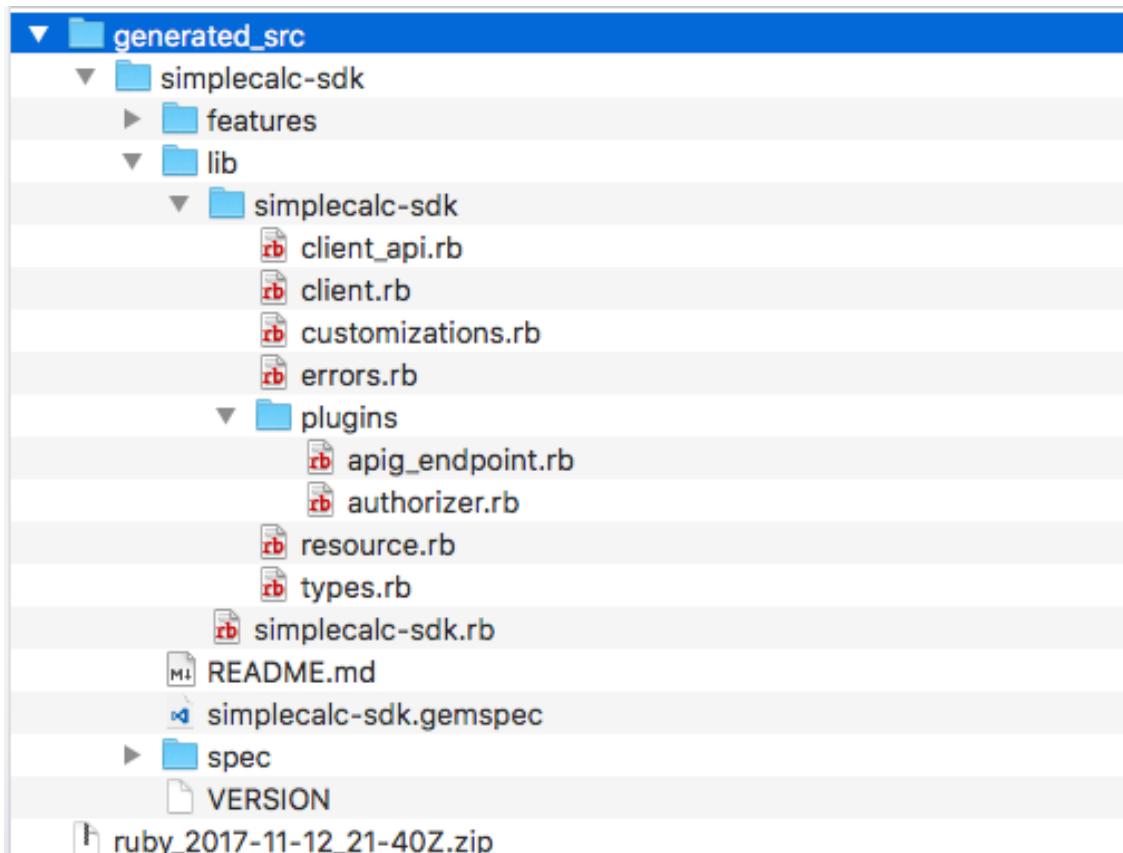
## 使用 API Gateway 為 REST API 所產生的 Ruby 開發套件

### Note

這些說明假設您已完成[使用 API Gateway 主控台產生 API 開發套件 \(p. 482\)](#)中的說明。

安裝、執行個體化和呼叫 API Gateway 為 REST API 產生的 Ruby 開發套件

1. 解壓縮下載的 Ruby 開發套件檔案。產生的開發套件來源如下所示。



2. 在終端機視窗中使用以下 shell 命令，從產生的開發套件來源建立 Ruby Gem：

```
# change to /simplecalc-sdk directory
cd simplecalc-sdk

# build the generated gem
gem build simplecalc-sdk.gemspec
```

在此之後，simplecalc-sdk-1.0.0.gem 即可使用。

3. 安裝 gem：

```
gem install simplecalc-sdk-1.0.0.gem
```

4. 建立用戶端應用程式。在應用程式中執行個體化和初始化 Ruby 開發套件用戶端：

```
require 'simplecalc-sdk'
client = SimpleCalc::Client.new(
  http_wire_trace: true,
  retry_limit: 5,
  http_read_timeout: 50
)
```

如果 API 具有已設定 AWS\_IAM 類型的授權，您可在初始化期間提供 accessKey 和 secretKey，將發起人的 AWS 登入資料包括在內：

```
require 'pet-sdk'
client = Pet::Client.new(
  http_wire_trace: true,
  retry_limit: 5,
  http_read_timeout: 50,
  access_key: 'ACCESS_KEY',
  secret_key: 'SECRET_KEY'
)
```

5. 在應用程式中透過開發套件呼叫 API。

Tip

如果您不熟悉開發套件方法呼叫慣例，您可以參閱產生的開發套件 client.rb 資料夾中的 lib 檔案。此資料夾包含每個受支援 API 方法呼叫的文件。

探索受支援的操作：

```
# to show supported operations:
puts client.operation_names
```

這會產生以下顯示畫面，分別對應到 GET /?a={.}&b={.}&op={.}、GET /{a}/{b}/{op} 和 POST / 的 API 方法，加上 {a:"...", b:"...", op:"..."} 格式的承載：

```
[:get_api_root, :get_ab_op, :post_api_root]
```

若要呼叫 GET /?a=1&b=2&op=+ API 方法，請呼叫以下 Ruby 開發套件方法：

```
resp = client.get_api_root({a:"1", b:"2", op:"+"})
```

若要呼叫 POST / API 方法加上 {a: "1", b: "2", "op": "+"} 承載，請呼叫以下 Ruby 開發套件方法：

```
resp = client.post_api_root(input: {a:"1", b:"2", op:"+"})
```

若要呼叫 GET /1/2/+ API 方法，請呼叫以下 Ruby 開發套件方法：

```
resp = client.get_ab_op({a:"1", b:"2", op:"+"})
```

成功的開發套件方法呼叫傳回以下回應：

```
resp : {
  result: {
    input: {
```

```
a: 1,  
b: 2,  
op: "+"  
},  
output: {  
    c: 3  
}  
}  
}
```

## 在 Objective-C 或 Swift 中使用 API Gateway 為 REST API 產生的 iOS 開發套件

在本教學中，我們會示範如何在 Objective-C 或 Swift 應用程式中使用 API Gateway 為 REST API 產生的 iOS 開發套件，以呼叫基礎 API。我們會以 [SimpleCalc API \(p. 487\)](#) 為例，說明下列主題：

- 如何將必要的 AWS Mobile 開發套件元件安裝到您的 Xcode 專案
- 如何在呼叫 API 的方法前，先建立 API 用戶端物件
- 如何透過 API 用戶端物件上對應的開發套件方法呼叫 API 方法
- 如何使用開發套件的對應模型類別準備方法輸入和剖析其結果

### 主題

- [使用產生的 iOS 開發套件 \(Objective-C\) 呼叫 API \(p. 508\)](#)
- [使用產生的 iOS 開發套件 \(Swift\) 呼叫 API \(p. 512\)](#)

### 使用產生的 iOS 開發套件 (Objective-C) 呼叫 API

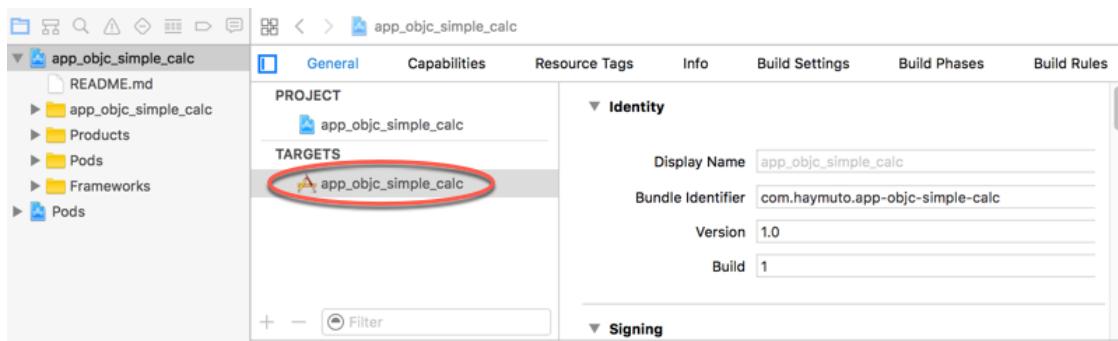
開始下列程序之前，您必須先在 Objective-C 中完成 [使用 API Gateway 主控台產生 API 開發套件 \(p. 482\)](#) 中的 iOS 步驟，並下載已產生之開發套件的 .zip 檔案。

#### 在 Objective-C 專案中安裝 AWS Mobile 開發套件和 API Gateway 產生的 iOS 開發套件

下列程序說明如何安裝開發套件。

#### 安裝和使用 API Gateway 在 Objective-C 中產生的 iOS 開發套件

1. 將您稍早下載之 API Gateway 所產生的 .zip 檔案內容解壓縮。使用 [SimpleCalc API \(p. 487\)](#)，您可能希望將解壓縮的開發套件資料夾重新命名成類似 `sdk_objc_simple_calc`。在這個開發套件資料夾中，有 `README.md` 檔案和 `Podfile` 檔案。`README.md` 檔案包含開發套件的安裝和使用說明。本教學提供這些說明的詳細資訊。安裝利用 [CocoaPods](#) 匯入所需的 API Gateway 程式庫和其他相依的 AWS Mobile 開發套件元件。您必須更新 `Podfile`，將開發套件匯入至您應用程式的 Xcode 專案。未封存的開發套件資料夾也包含 `generated-src` 資料夾，其中包含 API 之已產生開發套件的原始程式碼。
2. 啟動 Xcode 並建立新的 iOS Objective-C 專案。請記下專案的目標。您需要在 `Podfile` 中設定它。



3. 若要使用 CocoaPods 將 AWS Mobile SDK for iOS 匯入 Xcode 專案，請執行下列操作：

- a. 在終端機視窗中執行下列命令來安裝 CocoaPods：

```
sudo gem install cocoapods
pod setup
```

- b. 將 Podfile 檔案從解壓縮開發套件資料夾複製至包含 Xcode 專案檔的相同目錄。將下列區塊：

```
target '<YourXcodeTarget>' do
  pod 'AWSAPIGateway', '~> 2.4.7'
end
```

使用您專案的目標名稱：

```
target 'app_objc_simple_calc' do
  pod 'AWSAPIGateway', '~> 2.4.7'
end
```

如果您的 Xcode 專案已包含一個名為 Podfile 的檔案，請將下行程式碼新增到此檔案：

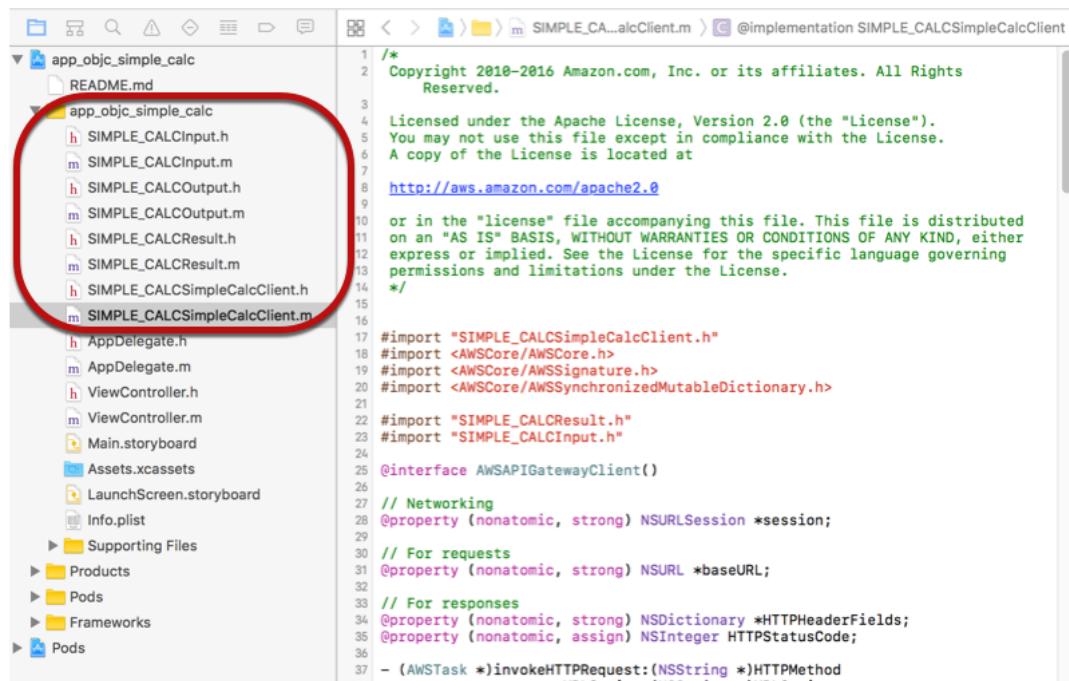
```
pod 'AWSAPIGateway', '~> 2.4.7'
```

- c. 開啟終端機視窗並執行下列命令：

```
pod install
```

這會安裝 API Gateway 元件和其他相依的 AWS Mobile 開發套件元件。

- d. 關閉 Xcode 專案，然後開啟 .xcworkspace 檔案重新啟動 Xcode。  
e. 從解壓縮的開發套件 .h 目錄將所有的 .m 和 generated-src 檔案新增到您的 Xcode 專案。



```
/*
Copyright 2010-2016 Amazon.com, Inc. or its affiliates. All Rights Reserved.

Licensed under the Apache License, Version 2.0 (the "License").
You may not use this file except in compliance with the License.
A copy of the License is located at

http://aws.amazon.com/apache2.0

or in the "license" file accompanying this file. This file is distributed
on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
express or implied. See the License for the specific language governing
permissions and limitations under the License.
*/

#import "SIMPLE_CALCSimpleCalcClient.h"
#import <AWSCore/AWSCore.h>
#import <AWSCore/AWSSignature.h>
#import <AWSCore/AWSynchronizedMutableDictionary.h>
...
@interface AWSGatewayClient()
...
// Networking
@property (nonatomic, strong) NSURLSession *session;
...
// For requests
@property (nonatomic, strong) NSURL *baseURL;
...
// For responses
@property (nonatomic, strong) NSDictionary *HTTPHeaderFields;
@property (nonatomic, assign) NSInteger HTTPStatusCode;
...
- (AWSTask *)invokeHTTPRequest:(NSString *)HTTPMethod
```

若要透過明確下載 AWS Mobile 開發套件或使用 [Carthage](#)，將 AWS Mobile SDK for iOS Objective-C 匯入您的專案，請遵循 README.md 檔案中的說明執行操作。請務必只使用其中一個選項來匯入 AWS Mobile 開發套件。

### 使用 Objective-C 專案中 API Gateway 產生的 iOS 開發套件呼叫 API 方法

當您使用此具有兩個方法輸入 (Input) 和輸出 (Result) 模型的 [SimpleCalc API \(p. 487\)](#) 的 SIMPLE\_CALC 字首產生開發套件後，在開發套件中，產生的 API 用戶端類別會成為 SIMPLE\_CALCSimpleCalcClient，而對應的資料類別分別是 SIMPLE\_CALCInput 和 SIMPLE\_CALCResult。API 請求和回應會對應至開發套件方法，如下所示：

- 下列的 API 請求：

```
GET /?a=...&b=...&op=...
```

會成為下列的開發套件方法：

```
(AWSTask *)rootGet:(NSString *)op a:(NSString *)a b:(NSString *)b
```

如果 AWSTask.result 模型已新增至方法回應，則 SIMPLE\_CALCResult 屬性為 Result 類型。否則，屬性為 NSDictionary 類型。

- 下列的此 API 請求：

```
POST /
{
    "a": "Number",
    "b": "Number",
    "op": "String"
}
```

會成為下列的開發套件方法：

```
(AWSTask *)rootPost:(SIMPLE_CALCInput *)body
```

- 下列的 API 請求：

```
GET /{a}/{b}/{op}
```

會成為下列的開發套件方法：

```
(AWSTask *)aBOpGet:(NSString *)a b:(NSString *)b op:(NSString *)op
```

下列程序說明如何在 Objective-C 應用程式原始碼中呼叫 API 方法；例如，在 `viewDidLoad` 檔案中作為 `ViewController.m` 委派的一部分。

#### 透過 API Gateway 所產生的 iOS 開發套件來呼叫 API

1. 匯入 API 用戶端類別標頭檔案，以能在應用程式中呼叫 API 用戶端類別：

```
#import "SIMPLE_CALCSimpleCalc.h"
```

`#import` 陳述式也針對兩個模型類別匯入 `SIMPLE_CALCInput.h` 和 `SIMPLE_CALCResult.h`。

2. 將 API 用戶端類別執行個體化：

```
SIMPLE_CALCSimpleCalcClient *apiInstance = [SIMPLE_CALCSimpleCalcClient defaultClient];
```

若要使用 Amazon Cognito 與 API，請先在預設的 `AWSServiceManager` 物件上設定 `defaultServiceConfiguration` 屬性，如下所示，再呼叫 `defaultClient` 方法建立 API 用戶端物件（如前例所示）：

```
AWSCognitoCredentialsProvider *creds = [[AWSCognitoCredentialsProvider alloc]
    initWithRegionType:AWSRegionUSEast1 identityPoolId:your_cognito_pool_id];
AWSConfiguration *configuration = [[AWSConfiguration alloc]
    initWithRegion:AWSRegionUSEast1 credentialsProvider:creds];
AWSServiceManager.defaultServiceManager.defaultServiceConfiguration = configuration;
```

3. 呼叫 `GET /?a=1&b=2&op=+` 方法來執行 1+2：

```
[[apiInstance rootGet: @"+" a:@"1" b:@"2"] continueWithBlock:^id _Nullable(AWSTask *
    _Nonnull task) {
    _textField1.text = [self handleApiResponse:task];
    return nil;
}];
```

協助程式函數 `handleApiResponse:task` 會在此將結果格式化為字串並顯示在文字欄位中 (`_textField1`)。

```
- (NSString *)handleApiResponse:(AWSTask *)task {
    if (task.error != nil) {
        return [NSString stringWithFormat: @"Error: %@", task.error.description];
    } else if (task.result != nil && [task.result isKindOfClass:[SIMPLE_CALCResult
        class]]) {
        return [NSString stringWithFormat:@"%@ %@ %@ = %@\n", task.result.input.a,
            task.result.input.op, task.result.input.b, task.result.output.c];
```

```
    }
    return nil;
}
```

產生的顯示畫面為  $1 + 2 = 3$ 。

4. 呼叫具有承載的 POST / 來執行 1-2：

```
SIMPLE_CALCInput *input = [[SIMPLE_CALCInput alloc] init];
    input.a = [NSNumber numberWithInt:1];
    input.b = [NSNumber numberWithInt:2];
    input.op = @"-";
    [[apiInstance rootPost:input] continueWithBlock:^id _Nullable(AWSTask * _Nonnull task) {
        _textField2.text = [self handleApiResponse:task];
        return nil;
    }];
}
```

產生的顯示畫面為  $1 - 2 = -1$ 。

5. 呼叫 GET /{a}/{b}/{op} 來執行 1/2：

```
[[apiInstance aBOpGet:@"1" b:@"2" op:@"div"] continueWithBlock:^id _Nullable(AWSTask * _Nonnull task) {
    _textField3.text = [self handleApiResponse:task];
    return nil;
}];

```

產生的顯示畫面為  $1 \text{ div } 2 = 0.5$ 。在此，div 用來代替 /，因為後端的簡單 Lambda 函數 (p. 485) 不處理 URL 編碼的路徑變數。

## 使用產生的 iOS 開發套件 (Swift) 呼叫 API

開始下列程序之前，您必須先在 Swift 中完成 [使用 API Gateway 主控台產生 API 開發套件 \(p. 482\)](#) 中的 iOS 步驟，並下載已產生之開發套件的 .zip 檔案。

### 主題

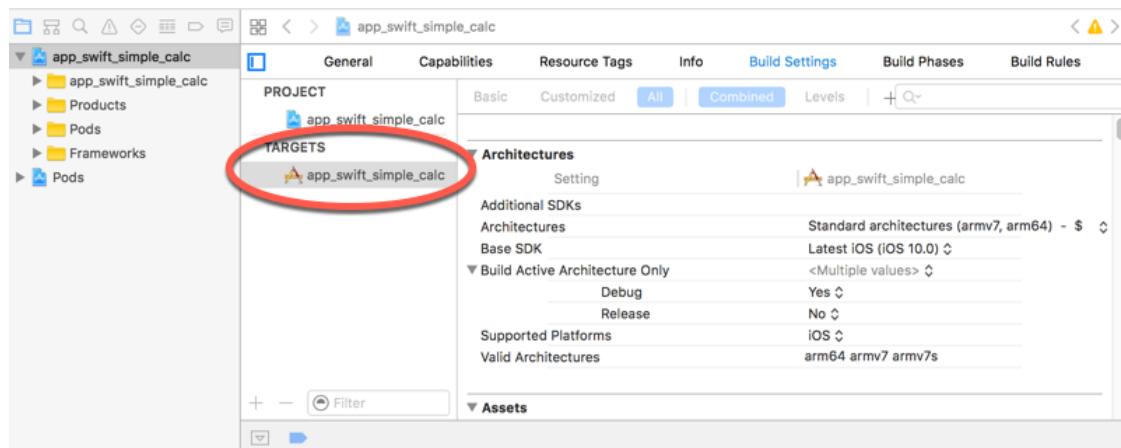
- [在 Swift 專案中安裝 AWS Mobile 開發套件和 API Gateway 產生的開發套件 \(p. 512\)](#)
- [在 Swift 專案中透過 API Gateway 所產生的 iOS 開發套件來呼叫 API 方法 \(p. 515\)](#)

### 在 Swift 專案中安裝 AWS Mobile 開發套件和 API Gateway 產生的開發套件

下列程序說明如何安裝開發套件。

#### 安裝和使用 API Gateway 在 Swift 中產生的 iOS 開發套件

1. 將您稍早下載之 API Gateway 所產生的 .zip 檔案內容解壓縮。使用 [SimpleCalc API \(p. 487\)](#)，您可能希望將解壓縮的開發套件資料夾重新命名成類似 `sdk_swift_simple_calc`。在這個開發套件資料夾中，有 `README.md` 檔案和 `Podfile` 檔案。`README.md` 檔案包含開發套件的安裝和使用說明。本教學提供這些說明的詳細資訊。安裝會利用 [CocoaPods](#) 來匯入所需的 AWS Mobile 開發套件元件。您必須更新 `Podfile`，以將開發套件匯入至您 Swift 應用程式的 Xcode 專案。未封存的開發套件資料夾也包含 `generated-src` 資料夾，其中包含 API 之已產生開發套件的原始程式碼。
2. 啟動 Xcode 並建立新的 iOS Swift 專案。請記下專案的目標。您需要在 `Podfile` 中設定它。



3. 若要使用 CocoaPods 將所需的 AWS Mobile 開發套件元件匯入至 Xcode 專案，請執行下列操作：
  - a. 如果未安裝，請在終端機視窗中執行下列命令來安裝 CocoaPods：

```
sudo gem install cocoapods
pod setup
```

- b. 將 Podfile 檔案從解壓縮開發套件資料夾複製至包含 Xcode 專案檔的相同目錄。將下列區塊：

```
target '<YourXcodeTarget>' do
  pod 'AWSAPIGateway', '~> 2.4.7'
end
```

取代為您專案的目標名稱，如下所示：

```
target 'app_swift_simple_calc' do
  pod 'AWSAPIGateway', '~> 2.4.7'
end
```

如果您的 Xcode 專案已包含具有正確目標的 Podfile，您可以只將下列程式碼行新增至 do ... end 迴圈：

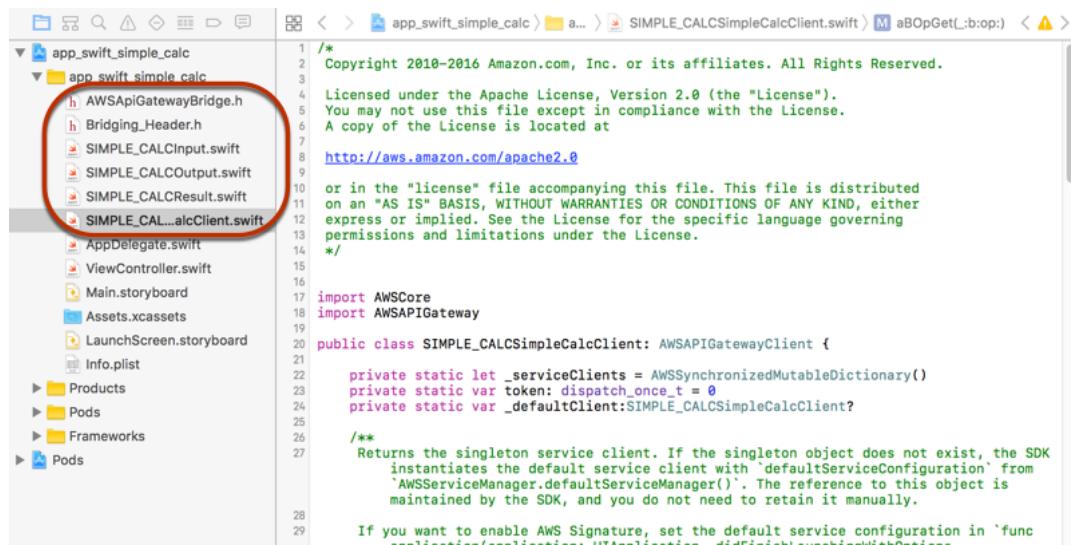
```
pod 'AWSAPIGateway', '~> 2.4.7'
```

- c. 開啟終端機視窗，並在應用程式目錄中執行下列命令：

```
pod install
```

這會將 API Gateway 元件和任何相依的 AWS Mobile 開發套件元件安裝至應用程式的專案中。

- d. 關閉 Xcode 專案，然後開啟 \*.xcworkspace 檔案重新啟動 Xcode。
  - e. 將所有開發套件的標頭檔案 (.h) 和 Swift 原始程式碼檔案 (.swift) 從擷取的 generated-src 目錄新增至 Xcode 專案。

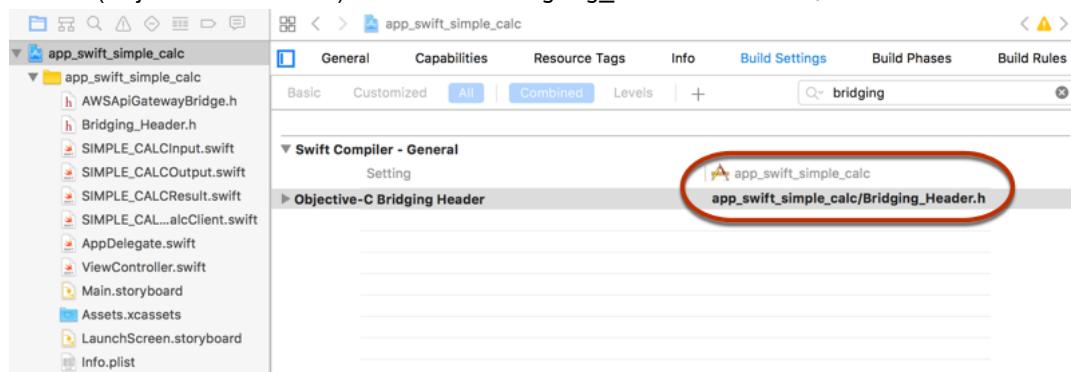


```

1 /*
2 Copyright 2010-2016 Amazon.com, Inc. or its affiliates. All Rights Reserved.
3
4 Licensed under the Apache License, Version 2.0 (the "License").
5 You may not use this file except in compliance with the License.
6 A copy of the License is located at
7
8 http://aws.amazon.com/apache2.0
9
10 or in the "license" file accompanying this file. This file is distributed
11 on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
12 express or implied. See the License for the specific language governing
13 permissions and limitations under the License.
14 */
15
16 import AWSCore
17 import AWSAPIGateway
18
19 public class SIMPLE_CALCSimpleCalcClient: AWSAPIGatewayClient {
20
21     private static let _serviceClients = AWSSynchronizedMutableDictionary()
22     private static var token: dispatch_once_t = 0
23     private static var _defaultClient:SIMPLE_CALCSimpleCalcClient?
24
25     /**
26      Returns the singleton service client. If the singleton object does not exist, the SDK
27      instantiates the default service client with 'defaultServiceConfiguration' from
28      'AWSServiceManager.defaultServiceManager()'. The reference to this object is
29      maintained by the SDK, and you do not need to retain it manually.
30
31      If you want to enable AWS Signature, set the default service configuration in 'func
32      application(application: UIApplication, didFinishLaunchingWithOptions:
33

```

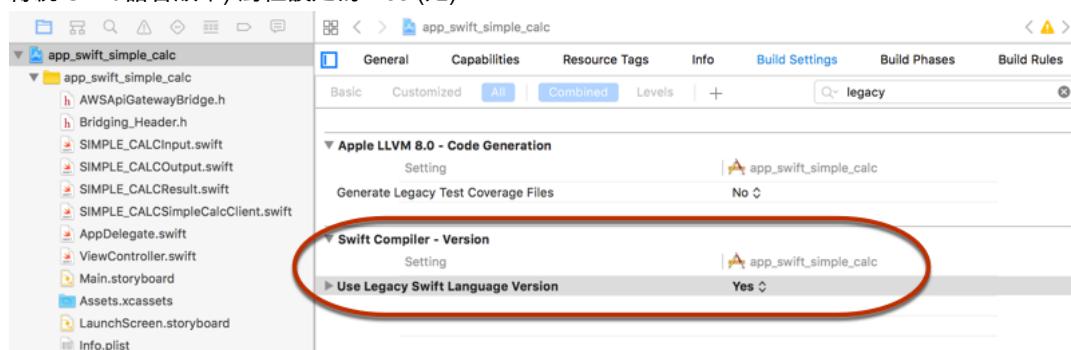
- f. 若要從 Swift 程式碼專案中啟用呼叫 AWS Mobile 開發套件的 Objective-C 程式庫，請在 Xcode 專案組態的 Swift Compiler - General (Swift 編譯器 - 一般) 設定下方，於 Objective-C Bridging Header (Objective-C 橋接標頭) 屬性上設定 Bridging\_Header.h 檔案路徑：



#### Tip

您可以在 Xcode 的搜尋方塊中輸入 **bridging**，以尋找 Objective-C Bridging Header (Objective-C 橋接標頭) 屬性。

- g. 建置 Xcode 專案，驗證已正確地設定它，再繼續進行。如果您的 Xcode 使用比 AWS Mobile 開發套件支援的 Swift 版本還要新的 Swift 版本，則會取得 Swift 編譯器錯誤。在這種情況下，於 Swift Compiler - Version (Swift 編譯器 - 版本) 設定下方，將 Use Legacy Swift Language Version (使用傳統 Swift 語言版本) 屬性設定為 Yes (是)：



若要明確下載 AWS Mobile 開發套件或使用 [Carthage](#) 以將 Swift 中適用於 iOS 的 AWS Mobile 開發套件匯入至專案，請遵循開發套件套件所隨附之 README.md 檔案中的說明。請務必只使用其中一個選項來匯入 AWS Mobile 開發套件。

## 在 Swift 專案中透過 API Gateway 所產生的 iOS 開發套件來呼叫 API 方法

當您使用兩個模型說明 API 請求和回應的輸入 (Input) 和輸出 (Result)，為這個 [SimpleCalc API \(p. 487\)](#) 產生字首為 SIMPLE\_CALC 的開發套件時，在開發套件中，產生的 API 用戶端類別會變成 SIMPLE\_CALCSimpleCalcClient，而對應的資料類別則分別是 SIMPLE\_CALCInput 和 SIMPLE\_CALCResult。API 請求和回應會對應至開發套件方法，如下所示：

- 下列的 API 請求：

```
GET /?a=...&b=...&op=...
```

會成為下列的開發套件方法：

```
public func rootGet(op: String?, a: String?, b: String?) -> AWSTask
```

如果 AWSTask.result 模型已新增至方法回應，則 SIMPLE\_CALCResult 屬性為 Result 類型。否則，它為 NSDictionary 類型。

- 下列的此 API 請求：

```
POST /  
  
{  
    "a": "Number",  
    "b": "Number",  
    "op": "String"  
}
```

會成為下列的開發套件方法：

```
public func rootPost(body: SIMPLE_CALCInput) -> AWSTask
```

- 下列的 API 請求：

```
GET /{a}/{b}/{op}
```

會成為下列的開發套件方法：

```
public func aBOpGet(a: String, b: String, op: String) -> AWSTask
```

下列程序說明如何在 Swift 應用程式原始碼中呼叫 API 方法；例如，在 viewDidLoad() 檔案中做為 ViewController.m 委派的一部分。

## 透過 API Gateway 所產生的 iOS 開發套件來呼叫 API

1. 將 API 用戶端類別執行個體化：

```
let client = SIMPLE_CALCSimpleCalcClient.defaultClient()
```

若要搭配使用 Amazon Cognito 與 API，請先設定預設 AWS 服務組態 (如下所示)，再取得 defaultClient 方法 (先前所示)：

```
let credentialsProvider =
    AWSCognitoCredentialsProvider(regionType: AWSRegionType.USEast1, identityPoolId:
        "my_pool_id")
let configuration = AWSServiceConfiguration(region: AWSRegionType.USEast1,
    credentialsProvider: credentialsProvider)
AWSServiceManager.defaultServiceManager().defaultServiceConfiguration = configuration
```

2. 呼叫 GET /?a=1&b=2&op=+ 方法來執行 1+2：

```
client.rootGet("/", a: "1", b:"2").continueWithBlock {(task: AWSTask) -> AnyObject? in
    self.showResult(task)
    return nil
}
```

其中，helper 函數 self.showResult(task) 會將結果或錯誤列印至主控台，例如：

```
func showResult(task: AWSTask) {
    if let error = task.error {
        print("Error: \(error)")
    } else if let result = task.result {
        if result is SIMPLE_CALCResult {
            let res = result as! SIMPLE_CALCResult
            print(String(format:@"%@ %@ %@ = %@", res.input!.a!, res.input!.op!,
res.input!.b!, res.output!.c!))
        } else if result is NSDictionary {
            let res = result as! NSDictionary
            print("NSDictionary: \(res)")
        }
    }
}
```

在生產應用程式中，您可以在文字欄位中顯示結果或錯誤。產生的顯示畫面為 1 + 2 = 3。

3. 呼叫具有承載的 POST / 來執行 1-2：

```
let body = SIMPLE_CALCInput()
body.a=1
body.b=2
body.op="-"
client.rootPost(body).continueWithBlock {(task: AWSTask) -> AnyObject? in
    self.showResult(task)
    return nil
}
```

產生的顯示畫面為 1 - 2 = -1。

4. 呼叫 GET /{a}/{b}/{op} 來執行 1/2：

```
client.aBOpGet("1", b:"2", op:"div").continueWithBlock {(task: AWSTask) -> AnyObject? in
    self.showResult(task)
    return nil
}
```

產生的顯示畫面為 1 div 2 = 0.5。在此，div 用來代替 /，因為後端的簡單 Lambda 函數 (p. 485)不處理 URL 編碼的路徑變數。

## 透過 AWS Amplify JavaScript 程式庫呼叫 REST API

AWS Amplify JavaScript 程式庫可用於向 API Gateway REST API 提出 API 請求。如需詳細資訊，請參閱 [AWS Amplify API 指南](#) 的說明。

### 如何呼叫私有 API

一旦您已部署[私有 API \(p. 183\)](#)，便可以透過私有 DNS (如果您已啟用私有 DNS 命名) 和公有 DNS 存取它。

若要為您的私有 API 取得 DNS 名稱，請執行下列動作：

1. 前往 <https://console.aws.amazon.com/vpc/> 以登入 Amazon VPC 主控台。
2. 在左側導覽窗格中，選擇 Endpoints (端點)，然後選擇您的 API Gateway 界面 VPC 端點。
3. 在 Details (詳細資訊) 窗格中，您將會在 DNS names (DNS 名稱) 欄位中看到 5 個值。前 3 個是您 API 的公有 DNS 名稱。其他 2 個是私有 DNS 名稱。

### 使用私有 DNS 名稱呼叫您的私有 API

如果您已啟用私有 DNS，可以使用私有 DNS 名稱存取您的私有 API，如下所示：

```
{restapi-id}.execute-api.{region}.amazonaws.com
```

呼叫 API 的基本 URL 格式如下：

```
https://{restapi-id}.execute-api.{region}.amazonaws.com/{stage}
```

例如，假設您在此範例中設定 GET /pets 和 GET /pets/{petId} 方法，並假設您的靜態 API ID 為 0qzs2sy7bh，且區域為 us-west-2，您可以在瀏覽器中輸入以下 URL 以測試您的 API：

```
https://0qzs2sy7bh.execute-api.us-west-2.amazonaws.com/test/pets
```

和

```
https://0qzs2sy7bh.execute-api.us-west-2.amazonaws.com/test/pets/1
```

或者，您可以使用下列 cURL 命令：

```
curl -X GET https://0qzs2sy7bh.execute-api.us-west-2.amazonaws.com/test/pets
```

和

```
curl -X GET https://0qzs2sy7bh.execute-api.us-west-2.amazonaws.com/test/pets/2
```

### 使用端點特定公有 DNS 主機名稱呼叫您的私有 API

您可以使用端點特定 DNS 主機名稱存取您的私有 API。這些公有 DNS 主機名稱包含 VPC 端點 ID 或您私有 API 的 API ID。

基本 URL 格式如下所示：

```
https://{public-dns-hostname}.execute-api.{region}.vpce.amazonaws.com/{stage}
```

例如，假設您在此範例中設定 GET /pets 和 GET /pets/{petId} 方法，並假設您 API 的 API ID 為 0qzs2sy7bh、其公有 DNS 主機名稱為 vpce-0c1308d7312217cd7-01234567，且您的區域為 us-west-2，您可以使用 Host cURL 命令標頭透過其 VPCE ID 測試您的 API，如下範例中所示：

```
curl -v https://vpce-0c1308d7312217cd7-01234567.execute-api.us-east-1.vpce.amazonaws.com/test/pets -H 'Host: 0qzs2sy7bh.execute-api.us-west-2.amazonaws.com'
```

或者，您可以使用以下格式的 x-apigw-api-id 標頭 cURL 命令，透過其 API ID 存取您的私有 API：

```
curl -v https://{vpce-id}.execute-api.{region}.vpce.amazonaws.com/test -H'x-apigw-api-id:{api-id}'
```

## 使用 AWS Direct Connect 存取您的私有 API

您也可以使用 AWS Direct Connect 在現場部署網路和 Amazon VPC 之間建立專屬私有連線，並使用公有 DNS 名稱經該連線來存取您的私有 API 端點。

您不能從現場部署網路使用私有 DNS 名稱存取您的私有 API。

# 追蹤、記錄並監控 API Gateway API

Amazon API Gateway 開發人員可以使用 [AWS X-Ray](#)、[AWS CloudTrail](#) 和 [Amazon CloudWatch](#) 等工具來追蹤、記錄並監控 API 執行和管理操作。

AWS X-Ray 是一種 AWS 服務，可讓您追蹤 Amazon API Gateway API 的延遲問題。X-Ray 會從 API Gateway 服務以及任何構成您 API 之上游或下游服務中收集中繼資料。X-Ray 使用此中繼資料來產生詳細的服務圖表，可描繪出延遲高峰以及其他會影響 API 應用程式效能的問題。

Amazon CloudWatch 會記錄 API 執行操作，這是 API 客戶或用戶端應用程式對 API Gateway execute-api 元件進行的呼叫。CloudWatch 指標包含快取、延遲和偵測到錯誤的相關統計資料。您可以使用 API Gateway 主控台中的 API 儀表板，或是使用 CloudWatch 主控台來查看 CloudWatch 日誌，為您的 API 實作或執行進行故障診斷。

AWS CloudTrail 會記錄 API Gateway API 管理操作，這是 API 開發人員或擁有者對 API Gateway apigateway 元件進行的 REST API 呼叫。您可以使用 CloudTrail 日誌來為 API 建立、部署與更新進行故障診斷。您也可以使用 Amazon CloudWatch 來監控 CloudTrail 日誌。

### 主題

- [使用 AWS X-Ray 追蹤 API Gateway API 執行 \(p. 518\)](#)
- [透過 AWS CloudTrail 記錄向 Amazon API Gateway API 發出的呼叫 \(p. 527\)](#)
- [使用 Amazon CloudWatch 監控 API 執行 \(p. 528\)](#)

## 使用 AWS X-Ray 追蹤 API Gateway API 執行

您可以使用 [AWS X-Ray](#)，在使用者請求通過 Amazon API Gateway API 進入基礎服務時追蹤和分析這些請求。API Gateway 支援對所有 API Gateway 端點類型進行 AWS X-Ray 追蹤：區域、最佳化邊緣和私有。您可以在提供 X-Ray 的所有區域中使用 AWS X-Ray 搭配 Amazon API Gateway。

X-Ray 提供整個請求的端對端檢視，因此您可以在 API 及其後端服務中分析延遲。您可以使用 X-Ray 服務對應來檢視整個請求的延遲，以及與 X-Ray 整合之下游服務的延遲。此外，您可以設定取樣規則，以根據您指

定的條件告知 X-Ray 要使用何種取樣率來記錄哪些請求。如果您從已被追蹤的服務呼叫 API Gateway API，即使 API 上未啟用 X-Ray 追蹤，API Gateway 也會傳遞追蹤。

您可以使用 API Gateway 管理主控台或使用 API Gateway API 或 CLI，為 API 階段啟用 X-Ray。

#### 主題

- [使用 API Gateway 設定 AWS X-Ray \(p. 519\)](#)
- [使用 AWS X-Ray 服務對應和追蹤檢視搭配 API Gateway \(p. 521\)](#)
- [針對 API Gateway API 設定 AWS X-Ray 取樣規則 \(p. 523\)](#)
- [了解適用於 Amazon API Gateway API 的 AWS X-Ray 追蹤 \(p. 524\)](#)

## 使用 API Gateway 設定 AWS X-Ray

本節提供如何使用 API Gateway 設定 [AWS X-Ray](#) 的詳細資訊。

#### 主題

- [X-Ray 會追蹤 API Gateway 的模式 \(p. 519\)](#)
- [X-Ray 追蹤的許可 \(p. 519\)](#)
- [在 API Gateway 主控台啟用 X-Ray 追蹤 \(p. 519\)](#)
- [使用 API Gateway CLI 啟用 AWS X-Ray 追蹤 \(p. 520\)](#)

### X-Ray 會追蹤 API Gateway 的模式

透過應用程式的請求路徑是使用追蹤 ID 進行追蹤。追蹤會收集由單一請求所產生的所有區段，一般為 HTTP GET 或 POST 請求。

針對 API Gateway API 提供兩種追蹤模式：

- **Passive (被動)：**如果您未在 API 階段上啟用 X-Ray 追蹤，這是預設設定。此方法表示僅有在上游服務已啟用 X-Ray 的情況下，系統才會追蹤 API Gateway API。
- **Active (主動)：**API Gateway API 階段具有此設定時，API Gateway 會根據 X-Ray 指定的取樣演算法來自動對 API 呼叫請求取樣。

在階段上啟用主動追蹤時，API Gateway 會在您的帳戶中建立服務連結角色 (若該角色不存在)。角色名為 `AWSServiceRoleForAPIGateway` 並會將 `APIGatewayServiceRolePolicy` 受管政策附加到該角色。如需服務連結角色的詳細資訊，請參閱[使用服務連結角色](#)。

#### Note

X-Ray 會套用取樣演算法以確保追蹤的效率，同時提供 API 收到之請求的代表範本。預設的取樣演算法為每秒 1 個請求，並對超過該限制的請求量取樣 5%。

您可以使用 API Gateway 管理主控台、API Gateway CLI 或 AWS 開發套件來為 API 變更追蹤。

### X-Ray 追蹤的許可

在階段上啟用 X-Ray 追蹤時，API Gateway 會在您的帳戶中建立服務連結角色 (若該角色不存在)。角色名為 `AWSServiceRoleForAPIGateway` 並會將 `APIGatewayServiceRolePolicy` 受管政策附加到該角色。如需服務連結角色的詳細資訊，請參閱[使用服務連結角色](#)。

### 在 API Gateway 主控台啟用 X-Ray 追蹤

您可以使用 Amazon API Gateway 主控台在 API 階段上啟用主動追蹤。

這些步驟假設您已經將 API 部署到階段。

1. 登入位於 <https://console.aws.amazon.com/apigateway> 的 API Gateway 主控台。
2. 在 API (API) 窗格中，選擇 API，然後選擇 Stages (階段)。
3. 在 Stages (階段) 窗格中，選擇階段的名稱。
4. 在 Stage Editor (階段編輯器) 窗格中，選擇 Logs/Tracing (日誌/追蹤) 標籤。
5. 若要啟用主動 X-Ray 追蹤，請在X-Ray Tracing (X-Ray 追蹤) 下方選擇 Enable X-Ray Tracing (啟用 X-Ray 追蹤)。
6. 如有需要，選擇 Set X-Ray Sampling Rules (設定 X-Ray 取樣規則) 並前往 X-Ray 主控台來設定取樣規則 (p. 523)。

為您的 API 階段啟用 X-Ray 之後，您可以使用 X-Ray 管理主控台來查看追蹤和服務對應。

## 使用 API Gateway CLI 啟用 AWS X-Ray 追蹤

若要在建立階段時使用 AWS CLI 來啟用 API 階段的主動 X-Ray 追蹤，請呼叫 `create-stage` 命令，如以下範例所示：

```
aws apigateway --endpoint-url {endpoint-url} create-stage \
  --rest-api-id {rest-api-id} \
  --stage-name {stage-name} \
  --deployment-id {deployment-id} \
  --region {region} \
  --tracing-enabled=true
```

以下是成功叫用的範例輸出：

```
{  
  "tracingEnabled": true,  
  "stageName": {stage-name},  
  "cacheClusterEnabled": false,  
  "cacheClusterStatus": "NOT_AVAILABLE",  
  "deploymentId": {deployment-id},  
  "lastUpdatedDate": 1533849811,  
  "createdDate": 1533849811,  
  "methodSettings": {}  
}
```

若要在建立階段時使用 AWS CLI 來停用 API 階段的主動 X-Ray 追蹤，請呼叫 `create-stage` 命令，如以下範例所示：

```
aws apigateway --endpoint-url {endpoint-url} create-stage \
  --rest-api-id {rest-api-id} \
  --stage-name {stage-name} \
  --deployment-id {deployment-id} \
  --region {region} \
  --tracing-enabled=false
```

以下是成功叫用的範例輸出：

```
{  
  "tracingEnabled": false,  
  "stageName": {stage-name},  
  "cacheClusterEnabled": false,  
  "cacheClusterStatus": "NOT_AVAILABLE",  
  "deploymentId": {deployment-id},  
  "lastUpdatedDate": 1533849811,  
  "createdDate": 1533849811,  
  "methodSettings": {}  
}
```

}

若要使用 AWS CLI 來啟用已部署 API 的主動 X-Ray 追蹤，請呼叫 [update-stage](#) 命令，如下所示：

```
aws apigateway update-stage \
  --rest-api-id {rest-api-id} \
  --stage-name {stage-name} \
  --patch-operations op=replace,path=/tracingEnabled,value=true
```

若要使用 AWS CLI 來停用已部署 API 的主動 X-Ray 追蹤，請呼叫 [update-stage](#) 命令，如以下範例所示：

```
aws apigateway update-stage \
  --endpoint-url {endpoint-url} \
  --rest-api-id {rest-api-id} \
  --stage-name {stage-name} \
  --region {region} \
  --patch-operations op=replace,path=/tracingEnabled,value=false
```

以下是成功叫用的範例輸出：

```
{
  "tracingEnabled": false,
  "stageName": {stage-name},
  "cacheClusterEnabled": false,
  "cacheClusterStatus": "NOT_AVAILABLE",
  "deploymentId": {deployment-id},
  "lastUpdatedDate": 1533850033,
  "createdDate": 1533849811,
  "methodSettings": {}
}
```

為 API 階段啟用 X-Ray 之後，請使用 X-Ray CLI 來擷取追蹤資訊。如需詳細資訊，請參閱[使用 AWS X-Ray API 搭配 AWS CLI](#)。

## 使用 AWS X-Ray 服務對應和追蹤檢視搭配 API Gateway

本節提供如何搭配 API Gateway 使用 [AWS X-Ray](#) 服務映射和追蹤檢視的詳細資訊。

如需服務映射和追蹤檢視的詳細資訊，以及如何解譯它們的詳細資訊，請參閱[AWS X-Ray 主控台](#)。

### 主題

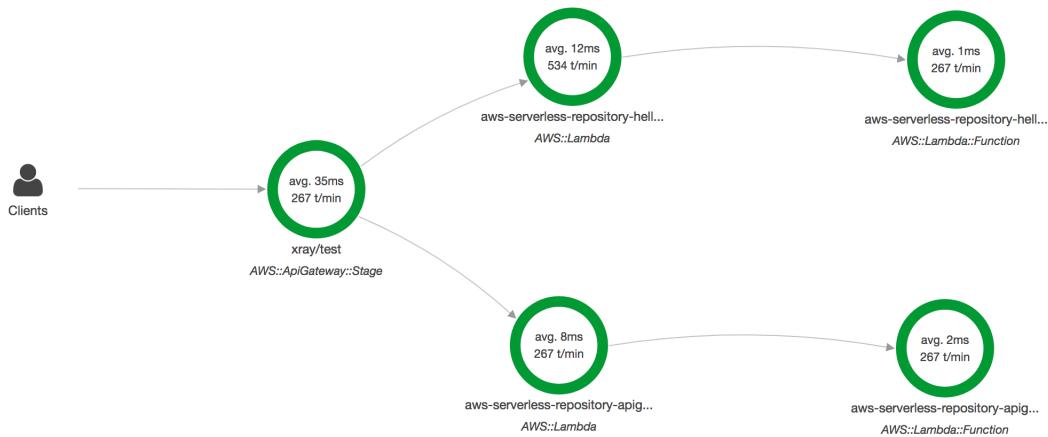
- [X-Ray 服務對應範例 \(p. 521\)](#)
- [X-Ray 追蹤檢視範例 \(p. 523\)](#)

## X-Ray 服務對應範例

AWS X-Ray 服務對應顯示您的 API 及其所有下游服務的相關資訊。在 API Gateway 中為 API 階段啟用 X-Ray 後，您會在服務對應中看到節點，其中包含 API Gateway 服務中所花費整體時間的相關資訊。您可以取得所選時間範圍內 API 回應時間之回應狀態和長條圖的詳細資訊。對於與 AWS 服務整合的 API，例如 AWS Lambda 和 Amazon DynamoDB，您會看到更多節點，提供這些服務的相關效能指標。每個 API 階段都會有一個服務對應。

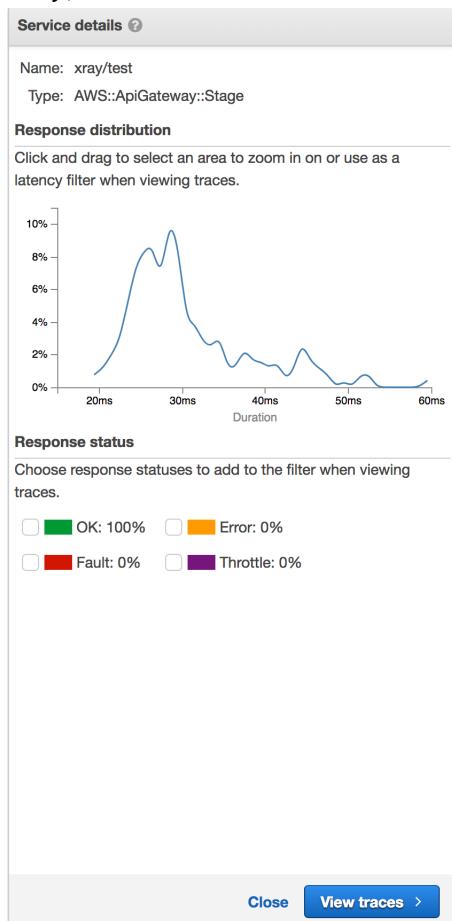
以下範例顯示名為 xray 之 API test 階段的服務對應。這個 API 已與 Lambda 授權方函數和 Lambda 後端函數進行 Lambda 整合。節點代表 API Gateway 服務、Lambda 服務以及兩個 Lambda 函數。

如需服務映射結構的詳細說明，請參閱[檢視服務映射](#)。



在服務對應上放大檢視即可查看 API 階段的追蹤檢視。此追蹤可顯示有關 API 的詳細資訊（以區段和子區段呈現）。例如，上面顯示之服務對應的追蹤包括 Lambda 服務和 Lambda 函數的區段。如需詳細資訊，請參閱[做為 AWS X-Ray 追蹤的 Lambda](#)。

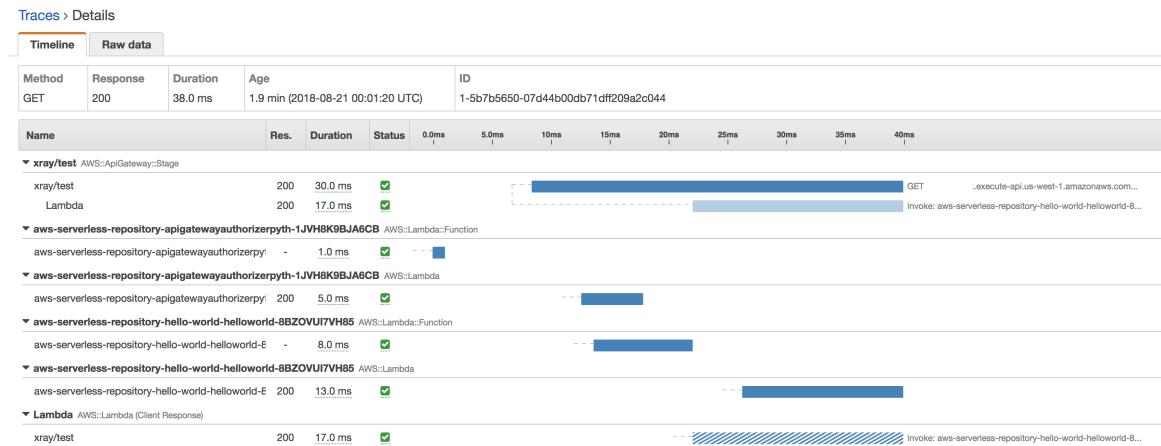
如果您在 X-Ray 服務對應上選擇節點或邊緣，X-Ray 主控台會顯示延遲分佈長條圖。您可以使用延遲長條圖來查看服務需要多久時間才能完成其請求。以下是先前服務對應中 API Gateway 階段的長條圖，名為 `xray/test`。如需延遲分佈長條圖的詳細說明，請參閱[在 AWS X-Ray 主控台中使用延遲長條圖](#)。



## X-Ray 追蹤檢視範例

下圖顯示為上述範例 API 產生的追蹤檢視，搭配 Lambda 後端函數和 Lambda 授權方函數。其中顯示成功的 API 方法請求，以及回應碼 200。

如需追蹤檢視的詳細說明，請參閱[檢視追蹤](#)。



## 針對 API Gateway API 設定 AWS X-Ray 取樣規則

您可以使用 AWS X-Ray 主控台或 SDK 來設定 Amazon API Gateway API 的取樣規則。取樣規則指定 X-Ray 應該為您的 API 記錄哪些請求。透過自訂取樣規則，您可以控制記錄的資料量，以及迅速修改取樣行為，而無需修改或重新部署程式碼。

指定您的 X-Ray 取樣規則之前，請先參閱《X-Ray 開發人員指南》中的下列主題：

- 在 AWS X-Ray 主控台中設定取樣規則
- 使用取樣規則搭配 X-Ray API

### 主題

- [API Gateway API 的 X-Ray 取樣規則選項值 \(p. 523\)](#)
- [X-Ray 取樣規則範例 \(p. 524\)](#)

## API Gateway API 的 X-Ray 取樣規則選項值

以下 X-Ray 取樣選項與 API Gateway 相關。字串值可以使用萬用字元來以符合單一字元 (?) 或零或多個字元 (\*)。如需詳細資訊，包括如何使用 Reservoir 和 Rate 設定的詳細說明，請參閱[在 AWS X-Ray 主控台中設定取樣規則](#)。

- Rule name (字串) — 規則的唯一名稱。
- Priority (介於 1 和 9999 之間的整數) — 取樣規則的優先順序。服務會以遞增的優先順序來評估規則，並使用符合的第一個規則來決定取樣決策。
- Reservoir (非負整數) — 在套用固定頻率前，每秒檢測的符合請求固定數量。服務不會直接使用蓄水池，而是集體套用至使用該規則的所有服務。
- Rate (比率) (從 0 到 100 之間的數字) — 在蓄水池用盡之後，要檢測的相符請求的百分比。
- Service name (服務名稱) (字串) — API 階段名稱，格式為 `{api-name}/{stage-name}`。例如，如果您準備將 [PetStore \(p. 40\)](#) 範例 API 部署到名為 test 的階段，則在您的取樣規則中應該指定的 Service name (服務名稱) 值是 `pets/test`。

- Service type (服務類型) (字串) — 對於 API Gateway API，可指定 **AWS::ApiGateway::Stage** 或 **AWS::ApiGateway::\***。
- Host (主機) (字串) — 來自 HTTP 託管標頭的主機名稱。將此值設定為 \* 可符合所有主機名稱。或者，您也可以指定符合全部或部分主機名稱，例如 `api.example.com` 或 `*.example.com`。
- Resource ARN (資源 ARN) (字串) — API 階段的 ARN，格式為 `arn:aws:execute-api:{region}:{account-id}:{api-id}/{stage-name}`，例如 `arn:aws:execute-api:us-east-1:123456789012:qsxrt/test`。

您可以從主控台或 API Gateway CLI 或 API 取得階段名稱。如需 ARN 格式的詳細資訊，請參閱[Amazon Web Services General Reference](#)。

- HTTP method (HTTP 方法) (字串) — 取樣的方法，例如 `GET`。
- URL path (URL 路徑) (字串) — API Gateway 不支援此選項。
- (選用) Attributes (屬性) (鍵和值) — 來自原始 HTTP 請求的標頭，例如 `Connection`、`Content-Length` 或 `Content-Type`。每個屬性值的長度最多為 32 個字元。

## X-Ray 取樣規則範例

### 取樣規則範例 #1

此規則會取樣 testxray API 在 test 階段的所有 GET 請求。

- Rule name (規則名稱) — `test-sampling`
- Priority (優先順序) — **17**
- Reservoir size (蓄水池大小) — **10**
- Fixed rate (固定頻率) — **10**
- Service name (服務名稱) — `testxray/test`
- Service type (服務類型) — **AWS::ApiGateway::Stage**
- HTTP method (HTTP 方法) — `GET`
- Resource ARN (資源 ARN) — \*
- Host (主機) — \*

### 取樣規則範例 #2

此規則會取樣 testxray API 在 prod 階段的所有請求。

- Rule name (規則名稱) — `prod-sampling`
- Priority (優先順序) — **478**
- Reservoir size (蓄水池大小) — **1**
- Fixed rate (固定頻率) — **60**
- Service name (服務名稱) — `testxray/prod`
- Service type (服務類型) — **AWS::ApiGateway::Stage**
- HTTP method (HTTP 方法) — \*
- Resource ARN (資源 ARN) — \*
- Host (主機) — \*
- Attributes (屬性) — { }

## 了解適用於 Amazon API Gateway API 的 AWS X-Ray 追蹤

本節討論適用於 Amazon API Gateway API 的 AWS X-Ray 追蹤區段、子區段和其他追蹤欄位。

在閱讀本節之前，請先參閱《X-Ray 開發人員指南》中的下列主題：

- [AWS X-Ray 主控台](#)
- [AWS X-Ray 區段文件](#)
- [X-Ray 概念](#)

#### 主題

- [為 API Gateway API 追蹤物件的範例 \(p. 525\)](#)
- [了解追蹤 \(p. 525\)](#)

## 為 API Gateway API 追蹤物件的範例

本節討論您可能會在 API Gateway API 的追蹤中看到的一些物件。

#### 註釋

註釋則顯示在區段和子區段中。它們在取樣規則中用做篩選追蹤的篩選運算式。如需詳細資訊，請參閱[在 AWS X-Ray 主控台中設定取樣規則](#)。

以下是 `annotations` 物件範例，其中 API 階段由 API ID 和 API 階段名稱識別：

```
"annotations": {  
    "aws:api_id": "a1b2c3d4e5",  
    "aws:api_stage": "dev"  
}
```

#### AWS 資源資料

`aws` 物件僅顯示在區段中。以下是符合預設取樣規則的 `aws` 物件範例。如需深入了解取樣規則，請參閱[在 AWS X-Ray 主控台中設定取樣規則](#)。

```
"aws": {  
    "xray": {  
        "sampling_rule_name": "Default"  
    },  
    "api_gateway": {  
        "account_id": "123412341234",  
        "rest_api_id": "a1b2c3d4e5",  
        "stage": "dev",  
        "request_id": "a1b2c3d4-a1b2-a1b2-a1b2c3d4e5f6"  
    }  
}
```

## 了解追蹤

以下是 API Gateway 階段的追蹤區段。如需組成追蹤區段之欄位的詳細說明，請參閱 AWS X-Ray Developer Guide 中的 [AWS X-Ray 區段文件](#)。

```
{  
    "Document": {  
        "id": "a1b2c3d4a1b2c3d4",  
        "name": "testxray/dev",  
        "start_time": 1533928226.229,  
        "end_time": 1533928226.614,  
        "metadata": {  
            "default": {
```

```
        "extended_request_id": "abcde12345abcde=",
        "request_id": "a1b2c3d4-a1b2-a1b2-a1b2-a1b2c3d4e5f6"
    }
},
"http": {
    "request": {
        "url": "https://example.com/dev?
username=demo&message=hellofromdemo/",
        "method": "GET",
        "client_ip": "192.0.2.0",
        "x_forwarded_for": true
    },
    "response": {
        "status": 200,
        "content_length": 0
    }
},
"aws": {
    "xray": {
        "sampling_rule_name": "Default"
    },
    "api_gateway": {
        "account_id": "123412341234",
        "rest_api_id": "a1b2c3d4e5",
        "stage": "dev",
        "request_id": "a1b2c3d4-a1b2-a1b2-a1b2-a1b2c3d4e5f6"
    }
},
"annotations": {
    "aws:api_id": "a1b2c3d4e5",
    "aws:api_stage": "dev"
},
"trace_id": "1-a1b2c3d4-a1b2c3d4a1b2c3d4a1b2c3d4",
"origin": "AWS::ApiGateway::Stage",
"resource_arn": "arn:aws:apigateway:us-east-1::/restapis/a1b2c3d4e5/stages/
dev",
"subsegments": [
    {
        "id": "abcdefghijklmno",
        "name": "Lambda",
        "start_time": 1533928226.233,
        "end_time": 1533928226.6130002,
        "http": {
            "request": {
                "url": "https://example.com/2015-03-31/functions/
arn:aws:lambda:us-east-1:123412341234:function:xray123/invocations",
                "method": "GET"
            },
            "response": {
                "status": 200,
                "content_length": 62
            }
        },
        "aws": {
            "function_name": "xray123",
            "region": "us-east-1",
            "operation": "Invoke",
            "resource_names": [
                "xray123"
            ]
        },
        "namespace": "aws"
    }
]
},
"Id": "a1b2c3d4a1b2c3d4"
```

}

## 透過 AWS CloudTrail 記錄向 Amazon API Gateway API 發出的呼叫

Amazon API Gateway 已與 AWS CloudTrail 服務整合，此服務可記錄使用者、角色或 AWS 服務在 API Gateway 中採取的動作。CloudTrail 會將 API Gateway 的所有 REST API 呼叫擷取為事件，包括來自 API Gateway 主控台的呼叫，以及對 API Gateway API 發出的程式碼呼叫。如果您建立追蹤記錄，就可以持續傳送 CloudTrail 事件至 Amazon S3 儲存貯體，包括 API Gateway 的事件。如果您不設定追蹤記錄，仍然可以透過 CloudTrail 主控台中的 Event history (事件歷史記錄) 檢視最新的事件。使用由 CloudTrail 收集的資訊，您就可以判斷送至 API Gateway 的請求、提出請求的 IP 地址、提出請求的人員、提出請求的時間，以及其他詳細資訊。

若要進一步了解 CloudTrail，請參閱 [AWS CloudTrail User Guide](#)。

### CloudTrail 中的 API Gateway 資訊

CloudTrail AWS 當您建立帳戶時，系統會在您的 帳戶中啟用。當 Amazon API Gateway 中發生活動時，該活動會與其他 AWS 服務事件一同記錄在 Event history (事件歷史記錄) 中的 CloudTrail 事件中。您可以檢視、搜尋和下載 AWS 帳戶的最新事件。如需詳細資訊，請參閱 [使用 CloudTrail 事件歷程記錄檢視事件](#)。

如需您 AWS 帳戶中正在進行事件的記錄（包含 API Gateway 的事件），請建立線索。追蹤記錄可讓 CloudTrail 將日誌檔案交付到 Amazon S3 儲存貯體。根據預設，當您 在主控台建立追蹤記錄時，追蹤記錄會套用到所有區域。該追蹤會記錄來自 AWS 分割區中所有區域的事件，並將日誌檔案交付到您指定的 Amazon S3 儲存貯體。此外，您可以設定其他 AWS 服務，以進一步分析和處理 CloudTrail 日誌中所收集的事件資料。如需詳細資訊，請參閱：

- [建立追蹤的概觀](#)
- [CloudTrail 支援的服務和整合](#)
- [設定 CloudTrail 的 Amazon SNS 通知](#)
- [接收多個區域的 CloudTrail 日誌檔案及接收多個帳戶的 CloudTrail 日誌檔案](#)

CloudTrail 會記錄所有 Amazon API Gateway 動作，並記錄在 [API Gateway V1 和 V2 API 參考 \(p. 626\)](#) 中。例如，在 API Gateway 中建立新 API、資源或方法的呼叫，會在 CloudTrail 日誌檔產生項目。

每一筆事件或記錄項目都會包含產生請求者的資訊。身分資訊可協助您判斷下列事項：

- 該請求是否使用根或 IAM 使用者登入資料提出。
- 提出該請求時，是否使用了特定角色或聯合身分使用者的暫時安全登入資料。
- 該請求是否由另一項 AWS 服務提出。

如需詳細資訊，請參閱 [CloudTrail 使用者身分元素](#)。

### 了解 API Gateway 日誌檔案項目

追蹤記錄是一種組態，能讓事件以日誌檔案的形式交付至您指定的 Amazon S3 儲存貯體。CloudTrail 日誌檔案包含一個或多個日誌項目。事件代表從任何來源的單一請求，並包含有關請求的動作、動作的日期和時間、請求參數等資訊。CloudTrail 日誌檔不是公有 API 呼叫的排序堆疊追蹤記錄，因此不會出現以任何特定順序顯示。

以下範例顯示的 CloudTrail 日誌項目示範了 API Gateway 取得資源的動作：

```
{  
    Records: [  
        {  
            eventVersion: "1.03",  
            userIdentity: {  
                type: "Root",  
                principalId: "AKIAI44QH8DHBEXAMPLE",  
                arn: "arn:aws:iam::123456789012:root",  
                accountId: "123456789012",  
                accessKeyId: "AKIAIOSFODNN7EXAMPLE",  
                sessionContext: {  
                    attributes: {  
                        mfaAuthenticated: "false",  
                        creationDate: "2015-06-16T23:37:58Z"  
                    }  
                }  
            },  
            eventTime: "2015-06-17T00:47:28Z",  
            eventSource: "apigateway.amazonaws.com",  
            eventName: "GetResource",  
            awsRegion: "us-east-1",  
            sourceIPAddress: "203.0.113.11",  
            userAgent: "example-user-agent-string",  
            requestParameters: {  
                restApiId: "3rbEXAMPLE",  
                resourceId: "5tfEXAMPLE",  
                template: false  
            },  
            responseElements: null,  
            requestID: "6d9c4bfc-148a-11e5-81b6-7577cEXAMPLE",  
            eventID: "4d293154-a15b-4c33-9e0a-ff5eeEXAMPLE",  
            readOnly: true,  
            eventType: "AwsApiCall",  
            recipientAccountId: "123456789012"  
        },  
        ... additional entries ...  
    ]  
}
```

## 使用 Amazon CloudWatch 監控 API 執行

您可以使用 CloudWatch 來監控 API 執行，該服務會收集並處理來自 API Gateway 的原始資料，進而將這些資料轉換為便於讀取且幾近即時的指標。這些統計資料會保存兩週的期間，以便您存取歷史資訊，並更清楚 web 應用程式或服務的執行方式。根據預設，API Gateway 指標資料會自動在一分鐘內傳送給 CloudWatch。如需詳細資訊，請參閱 Amazon CloudWatch User Guide 中的[什麼是 Amazon CloudWatch？](#)。

API Gateway 回報的指標可提供資訊，您可透過不同方式加以分析。下列清單說明一些常見的指標用法。這些是建議，以協助您開始，而不是的完整清單。

- 監控 IntegrationLatency 指標，以測量後端的回應能力。
- 監控 Latency 指標，以測量您 API 呼叫的整體回應能力。
- 監控 CacheHitCount 與 CacheMissCount 指標，以最佳化快取容量來達到所需的效能。

### 主題

- [Amazon API Gateway 維度與指標 \(p. 529\)](#)
- [以 API Gateway 中的 API 儀表板檢視 CloudWatch 指標 \(p. 530\)](#)
- [在 CloudWatch 主控台中檢視 API Gateway 指標 \(p. 531\)](#)

- 在 CloudWatch 主控台中檢視 API Gateway 日誌事件 (p. 531)
- AWS 中的監控工具 (p. 532)

## Amazon API Gateway 維度與指標

以下列出 API Gateway 傳送給 Amazon CloudWatch 的指標和維度。如需詳細資訊，請參閱[使用 Amazon CloudWatch 監控 API 執行 \(p. 528\)](#)。

### API Gateway 指標

Amazon API Gateway 每分鐘將指標資料傳送至 CloudWatch。

AWS/ApiGateway 命名空間包含下列指標。

指標	描述
4XXError	<p>在指定期間內擷取的用戶端錯誤數目。</p> <p>Sum 統計資訊代表此指標，即指定期間內的 4XXError 錯誤總數。Average 統計資訊代表 4XXError 錯誤率，即期間內的 4XXError 錯誤總數除以要求總數。分母對應至 Count 指標 (下方)。</p> <p>Unit: Count</p>
5XXError	<p>在指定期間內擷取的伺服器端錯誤數目。</p> <p>Sum 統計資訊代表此指標，即指定期間內的 5XXError 錯誤總數。Average 統計資訊代表 5XXError 錯誤率，即期間內的 5XXError 錯誤總數除以要求總數。分母對應至 Count 指標 (下方)。</p> <p>Unit: Count</p>
CacheHitCount	<p>在指定期間內從 API 快取提供的要求數目。</p> <p>Sum 統計資訊代表此指標，即指定期間內的快取命中總數。Average 統計資訊代表快取命中率，即期間內的快取命中總數除以要求總數。分母對應至 Count 指標 (下方)。</p> <p>Unit: Count</p>
CacheMissCount	<p>啟用 API 快取時，在指定期間內從後端提供的要求數目。</p> <p>Sum 統計資訊代表此指標，即指定期間內的快取未命中總數。Average 統計資訊代表快取未命中率，即期間內的快取未命中總數除以要求總數。分母對應至 Count 指標 (下方)。</p> <p>Unit: Count</p>
Count	<p>指定期間內的 API 要求總數。</p> <p>SampleCount 統計資訊代表此指標。</p> <p>Unit: Count</p>
IntegrationLatency	API Gateway 將要求轉送給後端時與收到來自後端的回應時之間的時間。

指標	描述
	Unit: Millisecond
Latency	API Gateway 收到來自用戶端的要求時與它將回應傳回給用戶端時之間的時間。延遲包含整合延遲與其他 API Gateway 額外負荷。 Unit: Millisecond

## 指標的維度

您可以使用下表中的維度來篩選 API Gateway 指標。

維度	描述
ApiName	篩選所指定 API 名稱之 REST API 的 API Gateway 指標。
ApiName, Method, Resource, Stage	篩選所指定 API 名稱、階段、資源與方法之 API 方法的 API Gateway 指標。  除非您已明確啟用詳細 CloudWatch 指標，否則 API Gateway 將不會傳送這類指標。您可以在主控台中執行這項操作，方法是在階段 Settings (設定) 索引標籤下選取 Enable CloudWatch Metrics (啟用 CloudWatch 指標)。或者，您可以呼叫 <a href="#">更新階段 AWS CLI 命令</a> ，將 metricsEnabled 屬性更新為 true。  啟用這些指標會產生您帳戶的額外費用。如需定價資訊，請參閱 <a href="#">Amazon CloudWatch 定價</a> 。
ApiName, Stage	篩選所指定 API 名稱和階段之 API 階段資源的 API Gateway 指標。

## 以 API Gateway 中的 API 儀表板檢視 CloudWatch 指標

您可以使用 API Gateway 主控台的 API 儀表板顯示在 API Gateway 中已部署 API 的 CloudWatch 指標。這些會顯示為 API 活動在一段時間內的摘要。

### 主題

- [先決條件 \(p. 530\)](#)
- [檢查儀表板中的 API 活動 \(p. 531\)](#)

## 先決條件

- 您必須擁有在 API Gateway 中建立的 API。請遵循[在 Amazon API Gateway 中建立 REST API \(p. 167\)](#)中的說明進行。
- 您必須至少已部署一次 API。請遵循[在 Amazon API Gateway 中部署 REST API \(p. 457\)](#)中的說明進行。
- 要取得個別方法的 CloudWatch 指標，您必須在特定階段為那些方法啟用 CloudWatch Logs，如[更新階段設定 \(p. 461\)](#)中所述。您的帳戶需要支付存取方法層級日誌的費用，但不需要支付存取 API 或階段層級日誌的費用。

## 檢查儀表板中的 API 活動

1. 登入位於 <https://console.aws.amazon.com/apigateway> 的 API Gateway 主控台。
2. 選擇 API 的名稱。
3. 在所選 API 之下，選擇 Dashboard (儀表板)。
4. 要顯示一段時間內的 API 活動摘要，請在 Stage (階段) 選擇所需的階段。
5. 使用 From (開始) 與 To (結束) 輸入日期範圍。
6. 重新整理 (如需要) 並檢視個別圖形中顯示的個別指標，而圖形的標題包括 API Calls (API 呼叫)、Integration Latency (整合延遲)、Latency (延遲)、4xx Error (4xx 錯誤) 和 5xx Error (5xx 錯誤)。CacheHitCount 和 CacheMissCount 圖形只會在 API 快取啟用時顯示。

Tip

要檢查方法層級的 CloudWatch 指標，請確保您已在方法層級啟用 CloudWatch Logs。如需設定方法層級記錄的詳細資訊，請參閱[使用 API Gateway 主控台更新階段設定 \(p. 461\)](#)。

## 在 CloudWatch 主控台中檢視 API Gateway 指標

指標會先依服務命名空間分組，再依各命名空間內不同的維度組合分類。

### 使用 CloudWatch 主控台檢視 API Gateway 指標

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. 如有必要請變更區域。請在導覽列中選擇您的 AWS 資源所在的區域。如需詳細資訊，請參閱[區域與端點](#)。
3. 在導覽窗格中，選擇指標。
4. 在 All Metrics (所有指標) 標籤中，選擇 API Gateway。
5. 若要依階段檢視指標，請選擇 By Stage (依階段) 面板。然後選取所需的 API 與指標名稱。
6. 若要依特定 API 檢視指標，請選擇 By Api Name (依 API 名稱) 面板。然後選取所需的 API 與指標名稱。

### 使用 AWS CLI; 檢視指標

1. 在命令提示字元中，使用下列命令來列出指標：

```
aws cloudwatch list-metrics --namespace "AWS/ApiGateway"
```

2. 若要每隔 5 分鐘檢視一次特定統計資料 (例如 Average)，請呼叫下列命令：

```
aws cloudwatch get-metric-statistics --namespace AWS/ApiGateway --metric-name Count
--start-time 2011-10-03T23:00:00Z --end-time 2017-10-05T23:00:00Z --period 300 --
statistics Average
```

## 在 CloudWatch 主控台中檢視 API Gateway 日誌事件

### 使用 CloudWatch 主控台檢視記錄的 API 請求與回應

1. 在導覽窗格中，選擇日誌。
2. 在 Log Groups (日誌群組) 資料表下，選擇 API-Gateway-Execution-Logs\_{rest-api-id}/{stage-name} 名稱的日誌群組。
3. 在 Log Streams (日誌串流) 資料表下，選擇一個日誌串流。您可以使用時間戳記協助尋找感興趣的日誌串流。

4. 選擇 Text (文字) 以檢視原始文字，或選擇 Row (資料列) 以逐列檢視事件。

#### Important

CloudWatch 可讓您刪除日誌群組或串流。不要手動刪除 API Gateway API 日誌群組或串流；讓 API Gateway 管理這些資源。手動刪除日誌群組或串流可能會導致 API 請求與回應不被記錄。如果發生這種情況，您可以刪除 API 的整個日誌群組，然後重新部署 API。這是因為 API Gateway 會針對部署 API 時的 API 階段，來建立日誌群組或日誌串流。

## AWS 中的監控工具

AWS 提供您可用來監控 API Gateway 的多種工具。您可以設定其中一些工具為您自動監控，但其他工具則需要手動介入。建議您盡可能自動化監控任務。

### AWS 中的自動化監控工具

您可以使用下列自動化監控工具來監看 API Gateway，並在發生錯誤時進行回報：

- Amazon CloudWatch Alarms (Amazon CloudWatch 警示) – 監看指定時段內的單一指標，並根據與多個時段內給定之閾值相對的指標值來執行一或多個動作。動作是傳送到 Amazon Simple Notification Service (Amazon SNS) 主題或 Amazon EC2 Auto Scaling 政策的通知。CloudWatch 警示不會只因為處於特定狀態而呼叫動作，狀態必須已變更，且在指定的期間數內維持此狀態。如需詳細資訊，請參閱 [使用 Amazon CloudWatch 監控 API 執行 \(p. 528\)](#)。
- Amazon CloudWatch Logs – 監控、存放及存取來自 AWS CloudTrail 或其他來源的日誌檔案。如需詳細資訊，請參閱 Amazon CloudWatch User Guide 中的 [監控日誌檔案](#)。
- Amazon CloudWatch Events – 比對事件並將其路由至一或多個目標函數或串流，以進行變更、擷取狀態資訊，以及採取修正動作。如需更多資訊，請參閱 Amazon CloudWatch User Guide 中的 [什麼是 Amazon CloudWatch Events？](#)。
- AWS CloudTrail 日誌監控 – 在帳戶之間共享日誌檔、將 CloudTrail 日誌檔傳送至 CloudWatch Logs，以對其進行即時監控、以 Java 撰寫日誌處理應用程式，以及驗證日誌檔在由 CloudTrail 交付之後沒有發生變更。如需詳細資訊，請參閱 AWS CloudTrail User Guide 中的 [使用 CloudTrail 日誌檔](#)。

### 手動監控工具

監控 API Gateway 的另一個重要部分包含手動監控 CloudWatch 警示未涵蓋的項目。API Gateway、CloudWatch 和其他 AWS 主控台儀表板提供您 AWS 環境狀態的快速瀏覽檢視。建議您也檢查 API 執行上的日誌檔。

- API Gateway 儀表板顯示指定 API 階段在指定時間內的下列統計資料：
  - API Calls (API 呼叫)
  - Cache Hit (快取命中)，只在啟用 API 快取時。
  - Cache Miss (快取遺漏)，只在啟用 API 快取時。
  - Latency (延遲)
  - Integration Latency (整合延遲)
  - 4XX Error (4XX 錯誤)
  - 5XX Error (5XX 錯誤)
- CloudWatch 首頁會顯示：
  - 目前警示與狀態
  - 警示與資源的圖表
  - 服務運作狀態

此外，您還可以使用 CloudWatch 執行下列動作：

- 建立 [自定儀表板](#)來監控您注重的服務

- 繪製指標資料圖表，以對問題進行故障診斷並探索趨勢
- 搜尋與瀏覽所有 AWS 資源指標
- 建立與編輯要通知發生問題的警示

## 建立 CloudWatch 警示來監控 API Gateway

您可以建立 CloudWatch 警示，其在警示變更狀態時傳送 Amazon SNS 訊息。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。此動作是傳送到 Amazon SNS 主題或 Auto Scaling 政策的通知。警示僅會針對持續狀態變更呼叫動作。CloudWatch 警示不會只叫用動作，因為它們處於特定狀態；狀態必須已變更，並且已維護指定數目的時段。

# OpenAPI 的 API Gateway 延伸

API Gateway 延伸支援 AWS 專屬授權和 API Gateway 專屬 API 整合。在本節中，我們會說明適用於 REST API 之 OpenAPI 規定的 API Gateway 延伸。

### Tip

若要了解如何在應用程式中使用 API Gateway 延伸，您可以使用 API Gateway 主控台建立 API，再將它匯出到 OpenAPI 定義檔。如需如何匯出 API 的詳細資訊，請參閱「[匯出 REST API \(p. 604\)](#)」。

### 主題

- [x-amazon-apigateway-any-method 物件 \(p. 533\)](#)
- [x-amazon-apigateway-api-key-source 屬性 \(p. 534\)](#)
- [x-amazon-apigateway-authorizer 物件 \(p. 535\)](#)
- [x-amazon-apigateway-authtype 屬性 \(p. 537\)](#)
- [x-amazon-apigateway-binary-media-types 屬性 \(p. 538\)](#)
- [x-amazon-apigateway-documentation 物件 \(p. 538\)](#)
- [x-amazon-apigateway-gateway-responses 物件 \(p. 539\)](#)
- [x-amazon-apigateway-gateway-responses.gatewayResponse 物件 \(p. 540\)](#)
- [x-amazon-apigateway-gateway-responses.responseParameters 物件 \(p. 540\)](#)
- [x-amazon-apigateway-gateway-responses.responseTemplates 物件 \(p. 541\)](#)
- [x-amazon-apigateway-integration 物件 \(p. 542\)](#)
- [x-amazon-apigateway-integration.requestTemplates 物件 \(p. 544\)](#)
- [x-amazon-apigateway-integration.requestParameters 物件 \(p. 545\)](#)
- [x-amazon-apigateway-integration.responses 物件 \(p. 545\)](#)
- [x-amazon-apigateway-integration.response 物件 \(p. 546\)](#)
- [x-amazon-apigateway-integration.responseTemplates 物件 \(p. 547\)](#)
- [x-amazon-apigateway-integration.responseParameters 物件 \(p. 548\)](#)
- [x-amazon-apigateway-request-validator 屬性 \(p. 548\)](#)
- [x-amazon-apigateway-requestValidators 物件 \(p. 549\)](#)
- [x-amazon-apigateway-requestValidators.requestValidator 物件 \(p. 550\)](#)

## x-amazon-apigateway-any-method 物件

在 [OpenAPI 路徑項目物件](#) 中指定 API Gateway 全部截獲 ANY 方法的 [OpenAPI 操作物件](#)。此物件會與其他操作物件並存，並會截獲未明確宣告的任何 HTTP 方法。

下表列有 API Gateway 延伸的屬性。如需其他 OpenAPI 操作屬性，請參閱 OpenAPI 規定。

#### 屬性

屬性名稱	類型	描述
x-amazon-apigateway-integration	x-amazon-apigateway-integration 物件 (p. 542)	指定方法與後端的整合。 這是 OpenAPI 操作物件的延伸屬性。整合類型可為 AWS、AWS_PROXY、HTTP、HTTP_PROXY 或 MOCK。

## x-amazon-apigateway-any-method 範例

下列範例會整合代理資源 {proxy+} 之 ANY 方法與 Lambda 函數 TestSimpleProxy。

```
"/{proxy+}": {
  "x-amazon-apigateway-any-method": {
    "produces": [
      "application/json"
    ],
    "parameters": [
      {
        "name": "proxy",
        "in": "path",
        "required": true,
        "type": "string"
      }
    ],
    "responses": {},
    "x-amazon-apigateway-integration": {
      "uri": "arn:aws:apigateway:us-east-1:lambda:path/2015-03-31/functions/arn:aws:lambda:us-east-1:123456789012:function:TestSimpleProxy/invocations",
      "passthroughBehavior": "when_no_match",
      "httpMethod": "POST",
      "type": "aws_proxy"
    }
  }
}
```

## x-amazon-apigateway-api-key-source 屬性

指定接收 API 金鑰的來源，以調節需要金鑰的 API 方法。這個 API 層級的屬性是 String 類型。

指定請求的 API 金鑰來源。有效值為：

- HEADER，從請求的 X-API-Key 標頭接收 API 金鑰。
- AUTHORIZER 適用於從 Lambda 授權方 (先前稱為自訂授權方) 的 UsageIdentifierKey 接收 API 金鑰。

## x-amazon-apigateway-api-key-source 範例

下列範例會將 X-API-Key 標頭設為 API 金鑰來源。

OpenAPI 2.0

```
{
  "swagger" : "2.0",
  "info" : {
```

```

        "title" : "Test1"
    },
    "schemes" : [ "https" ],
    "basePath" : "/import",
    "x-amazon-apigateway-api-key-source" : "HEADER",
    .
    .
    .
}
```

## x-amazon-apigateway-authorizer 物件

定義要套用的 Lambda 授權方 (以前稱為自訂授權方)，以授權在 API Gateway 中呼叫的方法。這個物件是 [OpenAPI 安全定義](#) 物件的延伸屬性。

### 屬性

屬性名稱	類型	描述
type	string	授權方的類型。這是必要屬性。 為發起人身分內嵌於授權字符串的 授權方指定 "token"。為發起人 身分包含在請求參數的授權方指 定 "request"。
authorizerUri	string	授權方 Lambda 函數的統一資源 識別符 (URI)。語法如下：  <code>"arn:aws:apigateway:us-east-1:lambda:path/2015-03-31/functions/arn:aws:lambda:us-east-1:<i>account-id</i>:function:<i>auth_function_name</i>/invocations"</code>
authorizerCredentials	string	呼叫授權方的必要登入 資料 (如果有)，格式為 IAM 執行角色的 ARN。例 如，"arn:aws:iam:: <i>account-id</i> : <i>IAM_role</i> "。
identitySource	string	做為身分來源之請求參數的對應 表達式逗號分隔清單。僅適用於 "request" 類型的授權方。
identityValidationExpression	string	驗證做為傳入身分之字符的一般 表達式。例如，"^x-[a-z]+"
authorizerResultTtlInSeconds	string	產生之 IAM 政策的快取秒數。

## x-amazon-apigateway-authorizer 範例

下列 OpenAPI 安全定義範例會指定類型為 "token" 且名為 test-authorizer 的 Lambda 授權方。

```

"securityDefinitions" : {
    "test-authorizer" : {
```

```

    "type" : "apiKey",                                // Required and the value must be "apiKey"
    for an API Gateway API.
    "name" : "Authorization",                         // The name of the header containing the
    authorization token.
    "in" : "header",                                 // Required and the value must be "header"
    for an API Gateway API.
    "x-amazon-apigateway-authtype" : "oauth2",        // Specifies the authorization mechanism
    for the client.
    "x-amazon-apigateway-authorizer" : {               // An API Gateway Lambda authorizer
    definition
        "type" : "token",                            // Required property and the value must
    "token"
        "authorizerUri" : "arn:aws:apigateway:us-east-1:lambda:path/2015-03-31/functions/
    arn:aws:lambda:us-east-1:account-id:function:function-name/invocations",
        "authorizerCredentials" : "arn:aws:iam::account-id:role",
        "identityValidationExpression" : "^x-[a-z]+",
        "authorizerResultTtlInSeconds" : 60
    }
}
}
}

```

下列 OpenAPI 操作物件程式碼片段會設定 GET /http 使用上述指定的 Lambda 授權方。

```

"/http" : {
    "get" : {
        "responses" : { },
        "security" : [ {
            "test-authorizer" : [ ]
        }],
        "x-amazon-apigateway-integration" : {
            "type" : "http",
            "responses" : {
                "default" : {
                    "statusCode" : "200"
                }
            },
            "httpMethod" : "GET",
            "uri" : "http://api.example.com"
        }
    }
}

```

下列 OpenAPI 安全定義範例會指定類型為 "request" 的 Lambda 授權方，其身分來源為單一標頭參數 (auth)。 securityDefinitions 名為 request\_authorizer\_single\_header。

```

"securityDefinitions": {
    "request_authorizer_single_header" : {
        "type" : "apiKey",
        "name" : "auth",                      // The name of a single header or query parameter as
        the identity source.
        "in" : "header",                     // The location of the single identity source request
        parameter. The valid value is "header" or "query"
        "x-amazon-apigateway-authtype" : "custom",
        "x-amazon-apigateway-authorizer" : {
            "type" : "request",
            "identitySource" : "method.request.header.auth",   // Request parameter mapping
            expression of the identity source. In this example, it is the 'auth' header.
            "authorizerCredentials" : "arn:aws:iam::123456789012:role/AWSepIntegTest-CS-
LambdaRole",
            "authorizerUri" : "arn:aws:apigateway:us-east-1:lambda:path/2015-03-31/functions/
    arn:aws:lambda:us-east-1:123456789012:function:APIGateway-Request-Authorizer:vtwo/
    invocations",
        }
    }
}

```

```
        "authorizerResultTtlInSeconds" : 300
    }
}
```

下列 OpenAPI 安全定義範例會指定類型為 "request" 的 Lambda 授權方，其身分來源為一個標頭 (HeaderAuth1) 和一個查詢字串參數 QueryString1。

```
"securityDefinitions": {
    "request_authorizer_header_query" : {
        "type" : "apiKey",
        "name" : "Unused",           // Must be "Unused" for multiple identity sources or
non header or query type of request parameters.
        "in" : "header",            // Must be "header" for multiple identity sources or
non header or query type of request parameters.
        "x-amazon-apigateway-authtype" : "custom",
        "x-amazon-apigateway-authorizer" : {
            "type" : "request",
            "identitySource" : "method.request.header.HeaderAuth1,
method.request.querystring.QueryString1",    // Request parameter mapping expressions of
the identity sources.
            "authorizerCredentials" : "arn:aws:iam::123456789012:role/AWSIntegTest-CS-
LambdaRole",
            "authorizerUri" : "arn:aws:apigateway:us-east-1:lambda:path/2015-03-31/functions/
arn:aws:lambda:us-east-1:123456789012:function:APIGateway-Request-Authorizer:vtwo/
invocations",
            "authorizerResultTtlInSeconds" : 300
        }
    }
}
```

下列 OpenAPI 安全定義範例會指定類型為 "request" 的 API Gateway Lambda 授權方，其身分來源為單一階段變數 (stage)。

```
"securityDefinitions": {
    "request_authorizer_single_stagevar" : {
        "type" : "apiKey",
        "name" : "Unused",           // Must be "Unused", for multiple identity sources or
non header or query type of request parameters.
        "in" : "header",            // Must be "header", for multiple identity sources or
non header or query type of request parameters.
        "x-amazon-apigateway-authtype" : "custom",
        "x-amazon-apigateway-authorizer" : {
            "type" : "request",
            "identitySource" : "stageVariables.stage",    // Request parameter mapping
expression of the identity source. In this example, it is the stage variable.
            "authorizerCredentials" : "arn:aws:iam::123456789012:role/AWSIntegTest-CS-
LambdaRole",
            "authorizerUri" : "arn:aws:apigateway:us-east-1:lambda:path/2015-03-31/functions/
arn:aws:lambda:us-east-1:123456789012:function:APIGateway-Request-Authorizer:vtwo/
invocations",
            "authorizerResultTtlInSeconds" : 300
        }
    }
}
```

## x-amazon-apigateway-authtype 屬性

指定描述 Lambda 授權方 (先前稱作自訂授權方) 的選擇性客戶定義資訊。其用於匯入和匯出 API Gateway API，而不影響功能。

這個屬性是 [OpenAPI 安全定義操作](#)物件的延伸屬性。

## x-amazon-apigateway-authtype 範例

下列範例會使用 OAuth 2 設定 Lambda 授權方的類型。

```
"cust-authorizer" : {
    "type" : "...", // required
    "name" : "...", // name of the identity source header
    "in" : "header", // must be header
    "x-amazon-apigateway-authtype" : "oauth2", // Specifies the authorization mechanism
for the client.
    "x-amazon-apigateway-authorizer" : {
        ...
    }
}
```

下列安全定義範例會使用 AWS Signature 第 4 版指定授權：

```
"sigv4" : {
    "type" : "apiKey",
    "name" : "Authorization",
    "in" : "header",
    "x-amazon-apigateway-authtype" : "awsSigv4"
}
```

## 另請參閱

[authorizer.authType](#)

## x-amazon-apigateway-binary-media-types 屬性

指定 API Gateway 要支援的二進位媒體類型清單，例如 application/octet-stream、image/jpeg 等。此延伸是一種 JSON 陣列。應該包含它做為 OpenAPI 文件的最上層廠商延伸。

## x-amazon-apigateway-binary-media-types 範例

下列範例示範 API 的編碼查詢順序。

```
"x-amazon-apigateway-binary-media-types: [ "application/octet", "image/jpeg" ]
```

## x-amazon-apigateway-documentation 物件

定義要匯入 API Gateway 的文件部分。這個物件是包含 DocumentationPart 執行個體陣列的 JSON 物件。

### 屬性

屬性名稱	類型	描述
documentationParts	Array	匯出或匯入的 DocumentationPart 執行個體陣列。
version	String	匯出文件部分快照的版本識別符。

## x-amazon-apigateway-documentation 範例

下列 OpenAPI 的 API Gateway 延伸範例會定義要從 API Gateway 中之 API 匯入或匯出的 DocumentationParts 執行個體。

```
{ ...  
  "x-amazon-apigateway-documentation": {  
    "version": "1.0.3",  
    "documentationParts": [  
      {  
        "location": {  
          "type": "API"  
        },  
        "properties": {  
          "description": "API description",  
          "info": {  
            "description": "API info description 4",  
            "version": "API info version 3"  
          }  
        }  
      },  
      {  
        ... // Another DocumentationPart instance  
      }  
    ]  
  }  
}
```

## x-amazon-apigateway-gateway-responses 物件

將 API 的閘道回應定義為鍵值對之字串對 [GatewayResponse](#) 的映射。

### 屬性

屬性名稱	類型	描述
<code>responseType</code>	<code>x-amazon-apigateway-gateway-responses.gatewayResponse</code> (p. 54)	指定之 <code>responseType</code> 的 <code>GatewayResponse</code> 。

## x-amazon-apigateway-gateway-responses 範例

下列 OpenAPI 的 API Gateway 延伸範例會定義包含兩個 [GatewayResponse](#) 執行個體的 [GatewayResponses](#) 映射，一個為 `DEFAULT_4XX` 類型，另一個為 `INVALID_API_KEY` 類型。

```
{  
  "x-amazon-apigateway-gateway-responses": {  
    "DEFAULT_4XX": {  
      "responseParameters": {  
        "gatewayresponse.header.Access-Control-Allow-Origin": "'domain.com'"  
      },  
      "responseTemplates": {  
        "application/json": "{\"message\": test 4xx b }"  
      }  
    },  
    "INVALID_API_KEY": {  
      "statusCode": "429",  
      "responseTemplates": {  
        "application/json": "{\"message\": test forbidden }"  
      }  
    }  
  }  
}
```

```
    }
}
}
```

## x-amazon-apigateway-gateway- responses.gatewayResponse 物件

定義指定回應類型的閘道回應，包括狀態碼、任何適用的回應參數，或回應範本。

### 屬性

屬性名稱	類型	描述
<code>responseParameters</code>	x-amazon-apigateway-gateway- responses.responseParameters (p. 5)	指定 <code>GatewayResponse</code> 參數，即標頭參數。該參數值可使用任何傳入請求參數 (p. 270) 值或靜態自訂值。
<code>responseTemplates</code>	x-amazon-apigateway-gateway- responses.responseTemplates (p. 5)	指定閘道回應的對應範本。該範本不由 VTL 引擎處理。
<code>statusCode</code>	string	閘道回應的 HTTP 狀態碼。

## x-amazon-apigateway-gateway-responses.gatewayResponse 範例

下列 OpenAPI 的 API Gateway 延伸範例會定義 `GatewayResponse`，以自訂 `INVALID_API_KEY` 回應傳回狀態碼 456、傳入請求的 `api-key` 標頭值和 "Bad api-key" 訊息。

```
"INVALID_API_KEY": {
  "statusCode": "456",
  "responseParameters": {
    "gatewayresponse.header.api-key": "method.request.header.api-key"
  },
  "responseTemplates": {
    "application/json": "{\"message\": \"Bad api-key\"}"
  }
}
```

## x-amazon-apigateway-gateway- responses.responseParameters 物件

定義鍵值對之字串對字串的對應，以從傳入請求參數或使用字串常值產生閘道回應參數。

### 屬性

屬性名稱	類型	描述
<code>gatewayresponse.param- position.param-name</code>	string	<code>param-position</code> 可以是 <code>header</code> 、 <code>path</code> 或 <code>querystring</code> 。如需更多詳細資訊

屬性名稱	類型	描述
		訊，請參閱 <a href="#">將方法請求資料對應到整合請求參數 (p. 270)</a> 。

## x-amazon-apigateway-gateway-responses.responseParameters 範例

下列 OpenAPI 延伸範例示範 [GatewayResponse](#) 回應參數映射表達式，以允許 \*.example.domain 網域上資源的 CORS 支援。

```
"responseParameters": {  
    "gatewayresponse.header.Access-Control-Allow-Origin": "*.*.example.domain",  
    "gatewayresponse.header.from-request-header" : method.request.header.Accept,  
    "gatewayresponse.header.from-request-path" : method.request.path.petId,  
    "gatewayresponse.header.from-request-query" : method.request.querystring.qname  
}
```

## x-amazon-apigateway-gateway- responses.responseTemplates 物件

為指定的閘道回應，將 [GatewayResponse](#) 映射範本定義為鍵值對之字串對字串的映射。每個鍵值對的鍵都是內容類型，例如 "application/json"，而值則是簡單變數替換的字串化對應範本。GatewayResponse 對應範本不由 [Velocity 範本語言 \(VTL\)](#) 引擎處理。

### 屬性

屬性名稱	類型	描述
<a href="#">content-type</a>	string	僅支援簡單變數替換的 GatewayResponse 內文對應範本，用以自訂閘道回應內文。

## x-amazon-apigateway-gateway-responses.responseTemplates 範例

下列 OpenAPI 延伸範例示範用以將 API Gateway 產生之錯誤回應自訂為應用程式專屬格式的 [GatewayResponse](#) 映射範本。

```
"responseTemplates": {  
    "application/json": "{ \"message\": $context.error.messageString, \"type\": $context.error.responseType, \"statusCode\": '488' }  
}
```

下列 OpenAPI 延伸範例示範用以使用靜態錯誤訊息覆寫 API Gateway 產生之錯誤回應的 [GatewayResponse](#) 映射範本。

```
"responseTemplates": {
```

```
    "application/json": "{ \"message\": 'API-specific errors' }" }
```

## x-amazon-apigateway-integration 物件

指定用於此方法的後端整合詳細資訊。此延伸是 [OpenAPI 操作](#)物件的延伸屬性。該結果是 API Gateway 整合物件。

### 屬性

屬性名稱	類型	描述
cacheKeyParameters	string 陣列	要快取其值的請求參數清單。
cacheNamespace	string	相關快取參數的 API 專屬標籤群組。
connectionId	string	私有整合的 <a href="#">VpcLink</a> ID。
connectionType	string	整合連線類型。有效值是私有整合的 "VPC_LINK" 或 "INTERNET"。
credentials	string	為 AWS IAM 角色登入資料指定適當 IAM 角色的 ARN。如未指定，登入資料會預設使用必須手動新增才能讓 API 存取資源的資源型許可。如需詳細資訊，請參閱 <a href="#">使用資源政策授予許可</a> 。請注意：使用 IAM 登入資料時，請確保為達最佳效能而部署此 API 的區域已啟用 <a href="#">AWS STS 區域端點</a> 。
contentHandling	string	請求承載編碼轉換類型。有效值為 1) CONVERT_TO_TEXT，將二進位承載轉換為 Base64 編碼的字串或將文字承載轉換為 utf-8 編碼的字串，或以原生方式傳遞文字承載，不予修改；和 2) CONVERT_TO_BINARY，將文字承載轉換為 Base64 編碼的 blob，或以原生方式傳遞二進位承載，不予修改。
httpMethod	string	用於整合請求中的 HTTP 方法。若要呼叫 Lambda 函數，該值必須為 POST。
passthroughBehavior	string	指定未對應內容類型的請求承載如何傳遞經整合請求，而無需修改。支援的值為 when_no_templates、when_no_match 和 never。如需詳細資訊，請參閱 <a href="#">Integration.passthroughBehavior</a> 。
requestParameters	x-amazon-apigateway-integration.requestParameters 物件 (p. 545)	指定從方法請求參數對應到整合請求參數。支援的請求參數為

屬性名稱	類型	描述
		querystring、path、header 和 body。
<b>requestTemplates</b>	x-amazon-apigateway-integration.requestTemplates 物件 (p. 544)	指定 MIME 類型的請求承載對應範本。
<b>responses</b>	x-amazon-apigateway-integration.responses 物件 (p. 545)	定義方法的回應，並指定從整合回應到方法回應所需的參數對應或承載對應。
<b>timeoutInMillis</b>	integer	整合逾時，介於 50 毫秒到 29,000 毫秒之間。
<b>type</b>	string	與指定後端整合的類型。有效值為 <ul style="list-style-type: none"> <li>• http 或 http_proxy：用於與 HTTP 後端整合；</li> <li>• aws_proxy：用於與 AWS Lambda 函數整合；</li> <li>• aws：用於與 AWS Lambda 函數或其他 AWS 服務（例如 Amazon DynamoDB、Amazon Simple Notification Service 或 Amazon Simple Queue Service）整合；</li> <li>• mock：用於與 API Gateway 整合，但不呼叫任何後端。</li> </ul> 如需整合類型的詳細資訊，請參閱 <a href="#">integration:type</a> 。
<b>uri</b>	string	後端的端點 URI。若是 aws 類型的整合，這是 ARN 值。若是 HTTP 整合，則這是 HTTP 端點的 URL，包括 https 或 http 結構描述。

## x-amazon-apigateway-integration 範例

下列範例會將 API 的 POST 方法與後端中的 Lambda 函數整合。基於示範目的，以下範例之 requestTemplates 和 responseTemplates 中示範的對應範本範例，會假設套用到下列 JSON 格式的承載：{ "name": "value\_1", "key": "value\_2", "redirect": { "url": "..." } }，以產生 { "stage": "value\_1", "user-id": "value\_2" } 的 JSON 輸出或 <stage>value\_1</stage> 的 XML 輸出。

```
"x-amazon-apigateway-integration" : {
    "type" : "aws",
    "uri" : "arn:aws:apigateway:us-east-1:lambda:path/2015-03-31/functions/
arn:aws:lambda:us-east-1:012345678901:function:HelloWorld/invocations",
    "httpMethod" : "POST",
    "credentials" : "arn:aws:iam::012345678901:role/apigateway-invoke-lambda-exec-role",
    "requestTemplates" : {
```

```
        "application/json" : "#set ($root=$input.path('$')) { \"stage\": \"$root.name\", \"user-id\": \"$root.key\" }",
        "application/xml" : "#set ($root=$input.path('$')) <stage>$root.name</
stage> "
    },
    "requestParameters" : {
        "integration.request.path.stage" : "method.request.querystring.version",
        "integration.request.querystring.provider" : "method.request.querystring.vendor"
    },
    "cacheNamespace" : "cache namespace",
    "cacheKeyParameters" : [],
    "responses" : {
        "2\\d{2}" : {
            "statusCode" : "200",
            "responseParameters" : {
                "method.response.header.requestId" : "integration.response.header.cid"
            },
            "responseTemplates" : {
                "application/json" : "#set ($root=$input.path('$')) { \"stage\": \"$root.name\", \"user-id\": \"$root.key\" }",
                "application/xml" : "#set ($root=$input.path('$')) <stage>$root.name</
stage> "
            }
        },
        "302" : {
            "statusCode" : "302",
            "responseParameters" : {
                "method.response.header.Location" :
"integration.response.body.redirect.url"
            }
        },
        "default" : {
            "statusCode" : "400",
            "responseParameters" : {
                "method.response.header.test-method-response-header" : "'static value'"
            }
        }
    }
}
```

請注意，對應範本中 JSON 字串的雙引號 ("") 必須是逸出字串 (\")。

## x-amazon-apigateway-integration.requestTemplates 物件

為指定 MIME 類型的請求承載指定對應範本。

屬性

屬性名稱	類型	描述
<b>MIME type</b>	string	MIME 類型範例為 <code>application/json</code> 。如需建立對應範本的相關資訊，請參閱「 <a href="#">對應範本 (p. 248)</a> 」。

## x-amazon-apigateway-integration.requestTemplates 範例

下列範例會為 application/json 和 application/xml MIME 類型的請求承載設定對應範本。

```
"requestTemplates" : {
    "application/json" : "#set ($root=$input.path('$')) { \"stage\": \"$root.name\",
    \"user-id\": \"$root.key\" }",
    "application/xml" : "#set ($root=$input.path('$')) <stage>$root.name</stage> "
}
```

## x-amazon-apigateway-integration.requestParameters 物件

指定從具名方法請求參數對應到整合請求參數。方法請求參數必須先定義才能參考。

### 屬性

屬性名稱	類型	描述
integration.request.< <i>param-type</i> >.< <i>param-name</i> >	string	該值必須是格式為 method.request.< <i>param-type</i> >.< <i>param-name</i> > 的預先定義方法請求參數，其中 < <i>param-type</i> > 可以是 querystring、path、header 或 body。至於 body 參數，< <i>param-name</i> > 是開頭沒有 \$. 的 JSON 路徑表達式。

## x-amazon-apigateway-integration.requestParameters 範例

下列請求參數對應範例會將方法請求的查詢 (version)、標頭 (x-user-id) 和路徑 (service) 參數分別轉譯為整合請求的查詢 (stage)、標頭 (x-userid) 和路徑參數 (op)。

### Note

如果您是透過 OpenAPI 或 AWS CloudFormation 建立資源，則應該以單引號括住靜態值。  
若要從主控台新增此值，請輸入 application/json，無需引號。

```
"requestParameters" : {
    "integration.request.querystring.stage" : "method.request.querystring.version",
    "integration.request.header.x-userid" : "method.request.header.x-user-id",
    "integration.request.path.op" : "method.request.path.service"
},
```

## x-amazon-apigateway-integration.responses 物件

定義方法的回應，並指定從整合回應到方法回應的參數對應或承載對應。

## 屬性

屬性名稱	類型	描述
#####	x-amazon-apigateway-integration.response 物件 (p. 546)	<p>用以將整合回應對應至方法回應的一般表達式選項。若是 HTTP 整合，此 regex 會套用至整合回應狀態碼。若是 Lambda 呼叫，則此 regex 會在 Lambda 函數執行拋出 <a href="#">例外狀況</a> 時，套用至 AWS Lambda 傳回之錯誤資訊物件的 <code>errorMessage</code> 欄位，做為失敗回應內文。</p> <p><b>Note</b></p> <p>##### 屬性名稱是指回應狀態碼或描述一組回應狀態碼的一般表達式。它不會映射到 API Gateway REST API 中 <a href="#">IntegrationResponse</a> 資源的任何識別符。</p>

## x-amazon-apigateway-integration.responses 範例

下列範例示範來自 2xx 和 302 回應的回應清單。在 2xx 回應方面，方法回應對應自 application/json 或 application/xml MIME 類型的整合回應承載。這個回應使用提供的對應範本。至於 302 回應，方法回應則會傳回 Location 標頭，該標頭的值來自整合回應承載的 `redirect.url` 屬性。

```

"responses" : {
    "2\\\d{2}" : {
        "statusCode" : "200",
        "responseTemplates" : {
            "application/json" : "#set ($root=$input.path('$')) { \\\"stage\\\": \\\"$root.name\\\", \\\"user-id\\\": \\\"$root.key\\\" }",
            "application/xml" : "#set ($root=$input.path('$')) <stage>$root.name</stage> "
        }
    },
    "302" : {
        "statusCode" : "302",
        "responseParameters" : {
            "method.response.header.Location" : "integration.response.body.redirect.url"
        }
    }
}

```

## x-amazon-apigateway-integration.response 物件

定義回應，並指定從整合回應到方法回應的參數對應或承載對應。

## 屬性

屬性名稱	類型	描述
statusCode	string	方法回應的 HTTP 狀態碼，例如 "200"。這必須對應到 OpenAPI 操作 responses 欄位中的相符回應。
responseTemplates	x-amazon-apigateway-integration.responseTemplates 物件 (p. 547)	指定回應承載的 MIME 類型專屬對應範本。
responseParameters	x-amazon-apigateway-integration.responseParameters 物件 (p. 548)	指定回應的參數對應。只有整合回應的 header 和 body 參數可以對應到方法的 header 參數。
contentHandling	string	回應承載編碼轉換類型。有效值為 1) CONVERT_TO_TEXT，將二進位承載轉換為 Base64 編碼的字串或將文字承載轉換為 utf-8 編碼的字串，或以原生方式傳遞文字承載，不予修改；和 2) CONVERT_TO_BINARY，將文字承載轉換為 Base64 編碼的 blob，或以原生方式傳遞二進位承載，不予修改。

## x-amazon-apigateway-integration.response 範例

下列範例會為從後端產生 302 或 application/json MIME 類型之承載的方法，定義 application/xml 回應。該回應使用提供的對應範本，並會傳回出自於方法之 Location 標頭中整合回應的重新導向 URL。

```
{  
    "statusCode" : "302",  
    "responseTemplates" : {  
        "application/json" : "#set ($root=$input.path('$')) { \\\"stage\\\": \\\"$root.name\\\",  
        \\\"user-id\\\": \\\"$root.key\\\" }",  
        "application/xml" : "#set ($root=$input.path('$')) <stage>$root.name</stage> "  
    },  
    "responseParameters" : {  
        "method.response.header.Location": "integration.response.body.redirect.url"  
    }  
}
```

## x-amazon-apigateway-integration.responseTemplates 物件

為指定 MIME 類型的回應承載指定對應範本。

## 屬性

屬性名稱	類型	描述
<i>MIME type</i>	string	指定對應範本，將整合回應內文轉換成指定 MIME 類型的方法回應內文。如需建立對應範本的相關資訊，請參閱「 <a href="#">對應範本 (p. 248)</a> 」。 <i>MIME ##</i> 範例為 application/json。

## x-amazon-apigateway-integration.responseTemplate 範例

下列範例會為 application/json 和 application/xml MIME 類型的請求承載設定對應範本。

```
"responseTemplates" : {
    "application/json" : "#set ($root=$input.path('$')) { \"stage\": \"$root.name\",
    \"user-id\": \"$root.key\" }",
    "application/xml" : "#set ($root=$input.path('$')) <stage>$root.name</stage> "
}
```

## x-amazon-apigateway-integration.responseParameters 物件

指定從整合方法回應參數對應到方法回應參數。只有整合回應參數的 header 和 body 類型可以對應到方法回應的 header 類型。

## 屬性

屬性名稱	類型	描述
method.response.header.< <i>param name</i> >	string	具名參數值只能從整合回應參數的 header 和 body 類型產生。

## x-amazon-apigateway-integration.responseParameters 範例

下列範例會將整合回應的 body 和 header 參數對應到方法回應的兩個 header 參數。

```
"responseParameters" : {
    "method.response.header.Location" : "integration.response.body.redirect.url",
    "method.response.header.x-user-id" : "integration.response.header.x-userid"
}
```

## x-amazon-apigateway-request-validator 屬性

指定請求驗證程式，方法是參考 *request\_validator\_name* 對應的 x-amazon-apigateway-request-validators 物件 (p. 549)，以對包含的 API 或方法進行請求驗證。此延伸的值是 JSON 字串。

此延伸可在 API 層級或方法層級指定。API 層級的驗證程式適用於所有方法，除非該驗證程式經方法層級的驗證程式所覆寫。

## x-amazon-apigateway-request-validator 範例

下列範例會在 API 層級套用 basic 請求驗證程式，同時會在 parameter-only 請求套用 POST / validation 請求驗證程式。

OpenAPI 2.0

```
{  
    "swagger": "2.0",  
    "x-amazon-apigateway-request-validation" : {  
        "basic" : {  
            "validateRequestBody" : true,  
            "validateRequestParameters" : true  
        },  
        "params-only" : {  
            "validateRequestBody" : false,  
            "validateRequestParameters" : true  
        }  
    },  
    "x-amazon-apigateway-request-validator" : "basic",  
    "paths" : {  
        "/validation" : {  
            "post" : {  
                "x-amazon-apigateway-request-validator" : "params-only",  
                ...  
            }  
        }  
    }  
}
```

## x-amazon-apigateway-request-validation 物件

定義為包含之 API 支援的請求驗證程式，做為驗證程式名稱和相關請求驗證規則之間的對應。此延伸適用於 API。

### 屬性

屬性名稱	類型	描述
<code>request_validator_name</code>	<code>x-amazon-apigateway-request-validation.requestValidator 物件</code> (p. 550)	<p>指定包含具名驗證程式的驗證規則。例如：</p> <pre>"basic" : {     "validateRequestBody" : true,     "validateRequestParameters" : true },</pre> <p>若要將此驗證程式套用到特定方法，請將驗證程式名稱 (basic) 做為 <code>x-amazon-apigateway-request-validator</code> 屬性 (p. 548) 屬性值的參考。</p>

## x-amazon-apigateway-request-validation 範例

下列範例示範 API 的一組請求驗證程式，做為驗證程式名稱和相關請求驗證規則之間的對應。

OpenAPI 2.0

```
{  
    "swagger": "2.0",  
    ...  
    "x-amazon-apigateway-request-validation": {  
        "basic": {  
            "validateRequestBody": true,  
            "validateRequestParameters": true  
        },  
        "params-only": {  
            "validateRequestBody": false,  
            "validateRequestParameters": true  
        }  
    },  
    ...  
}
```

## x-amazon-apigateway-request-validation.requestValidator 物件

將請求驗證程式的驗證規則指定為 [x-amazon-apigateway-request-validation](#) 物件 (p. 549) 對應定義的一部 分。

### 屬性

屬性名稱	類型	描述
validateRequestBody	Boolean	指定驗證請求內文 (true) 或不驗證 (false)。
validateRequestParameters	Boolean	指定驗證必要的請求參數 (true) 或不驗證 (false)。

## x-amazon-apigateway-request-validation.requestValidator 範例

下列範例示範僅限參數的請求驗證程式：

```
"params-only": {  
    "validateRequestBody": false,  
    "validateRequestParameters": true  
}
```

# 在 Amazon API Gateway 中建立、部署和呼叫 WebSocket API

如需對 API Gateway 中 WebSocket 支援的簡介，請參閱 [the section called “使用 API Gateway 來建立 WebSocket API” \(p. 3\)](#)。

## 主題

- [關於在 API Gateway 中的 WebSocket API \(p. 551\)](#)
- [在 API Gateway 中建立 WebSocket API \(p. 555\)](#)
- [在 API Gateway 中設定 WebSocket API 的路由 \(p. 556\)](#)
- [在 API Gateway 中設定 WebSocket API 整合 \(p. 557\)](#)
- [在 API Gateway 中設定 WebSocket API 的路由回應 \(p. 561\)](#)
- [在 API Gateway 中部署 WebSocket API \(p. 562\)](#)
- [呼叫 WebSocket API \(p. 564\)](#)
- [在 API Gateway 中控制 WebSocket API 的存取 \(p. 566\)](#)
- [使用 CloudWatch 監控 WebSocket API 執行 \(p. 568\)](#)
- [API Gateway 內的 WebSocket 選擇表達式 \(p. 570\)](#)
- [API Gateway WebSocket API 映射範本參考 \(p. 575\)](#)

## 關於在 API Gateway 中的 WebSocket API

您可以在 API Gateway 中建立 WebSocket API，做為 AWS 服務（例如，Lambda 或 DynamoDB）或 HTTP 端點的狀態前端。WebSocket API 會根據自用端應用程式接收的訊息內容來後端。

WebSocket API 與 REST API 不同（其會接收並回應請求），會支援用端應用程式和後端之間的雙向通訊。該後端可以將回呼訊息傳送到連線用端。

在 WebSocket API 中，是根據您設定的路由將傳入 JSON 訊息導向到後端整合。（會將非 JSON 訊息導向您設定的 `$default` 路由。）

路由包含的路由金鑰，是您可以在路由選擇表達式經評估後即預期的值。`routeSelectionExpression` 是在 API 層級定義的屬性。它指定的 JSON 屬性預計會出現在訊息承載。如需路由選擇表達式的詳細資訊，請參閱[the section called “” \(p. 570\)](#)。

例如，如果您的 JSON 訊息包含 `action` 屬性，以及您想要根據此屬性執行不同動作，路由選擇表達式可能會  `${request.body.action}`。您的路由表將指定要執行哪一個動作，方法是將 `action` 屬性值與您在資料表中定義的自訂路由鍵值進行比對。

有三種預先定義的路由可供使用：`$connect`、`$disconnect` 和 `$default`。此外，您可以建立自訂路由。

- API Gateway 會在用端和 WebSocket API 之間的持續連線進行起始化時呼叫 `$connect` 路由。
- 當用端或伺服器中斷與 API 的連線時，API Gateway 會呼叫 `$disconnect` 路由。
- 若發現相符的路由，針對該訊息評估路由選擇表達式後，API Gateway 即會呼叫自訂路由；該比對會判斷要呼叫哪些整合。
- 如果未發現相符路由，或無法針對該訊息來評估路由選擇表達式時，API Gateway 會呼叫 `$default` 路由。

如需 \$connect 和 \$disconnect 路由的相關資訊，請參閱[the section called “管理連線使用者和用戶端應用程式” \(p. 552\)](#)。

如需 \$default 路由和自訂路由的相關資訊，請參閱[the section called “呼叫後端整合” \(p. 552\)](#)。

後端服務可將資料傳送到連線的用戶端應用程式。如需詳細資訊，請參閱[the section called “將資料從後端服務傳送到連線的用戶端” \(p. 555\)](#)。

## 管理連線使用者和用戶端應用程式：\$connect 和 \$disconnect 路由

### 主題

- [\\$connect 路由 \(p. 552\)](#)
- [\\$disconnect 路由 \(p. 552\)](#)

## \$connect 路由

用戶端應用程式會透過傳送 WebSocket 升級請求來連線到 WebSocket API。如果請求成功，會在建立連線的同時執行 \$connect 路由。

由於 WebSocket 連線是狀態連接，您可以僅對 \$connect 路由設定授權。AuthN/AuthZ 將只會在連線時執行。

在與 \$connect 路由關聯的整合執行完成後，升級請求會處於等待中，且將不會建立實際的連線。如果 \$connect 請求失敗（例如，由於 AuthN/AuthZ 故障或整合故障），將不會進行連線。

### Note

如果在進行 \$connect 時授權失敗，就不會建立連線，用戶端將會收到 401 或 403 回應。

為 \$connect 設定整合是選用的。然而，此選項可是非常有用，因為後端會接收到連接的用戶端使用者 ID。您應在以下情況考慮設定 \$connect 整合：

- 您希望在用戶端連線和中斷連線時收到通知。
- 您想要調節連線或控制連線的人員。
- 您想要後端使用回呼 URL 將訊息傳回給用戶端。
- 您想將每個連線 ID 和其他資訊存放到資料庫（例如，Amazon DynamoDB）。

## \$disconnect 路由

\$disconnect 路由會在連線關閉後執行。

連線可以由伺服器或用戶端關閉。因為連線在執行時已關閉，\$disconnect 是最佳作業事件。API Gateway 會盡力將 \$disconnect 事件交付到整合，但無法保證交付。

後端可以透過使用 @connections API 來起使化中斷連線。如需詳細資訊，請參閱[the section called “在後端服務使用 @connections 命令” \(p. 565\)](#)。

## 呼叫您的後端整合：\$default 路由和自訂路由

### 主題

- [使用路由來處理訊息 \(p. 553\)](#)
- [\\$default 路由 \(p. 553\)](#)

- [自訂路由 \(p. 553\)](#)
- [使用 API Gateway WebSocket API 整合，以連接到商業邏輯 \(p. 554\)](#)
- [WebSocket API 和 REST API 之間的重要差異 \(p. 554\)](#)
- [處理二進位承載 \(p. 555\)](#)

## 使用路由來處理訊息

在 API Gateway WebSocket API 中，您可以將訊息從用戶端傳送到後端服務，反之亦然。後端與在 WebSocket 中的 HTTP 請求/回應模型不同，可以將訊息傳送給用戶端，而用戶端不需採取任何動作。

訊息可以是 JSON 或非 JSON。不過，只能夠根據訊息內容將 JSON 訊息路由到特定的整合。會透過 `$default` 路由將非 JSON 訊息傳遞到後端。

### Note

API Gateway 將支援高達 128 KB 的訊息承載，影格大小上限為 32 KB。如果訊息超過 32 KB，則必須分割成多個影格，每個為 32 KB 或以下。如果接收到更大的訊息 (或影格)，則該連線會關閉，並出現程式碼 1009。

目前不支援二進位承載。如果接收到二進位影格，則該連線會關閉，並出現程式碼 1003。不過，您可以將二進位承載轉換為文字。請參閱「[the section called “處理二進位承載” \(p. 555\)](#)」。

可透過 WebSocket API API Gateway，將 JSON 訊息路由以根據訊息內容執行特定的後端服務。當用戶端透過其 WebSocket 連線傳送訊息後，此會造成對 WebSocket API 發出路由請求。會透過 API Gateway 中之對應的路由金鑰來將請求對應至路由。您可以透過使用 AWS CLI 或使用 AWS 開發套件，在 API Gateway 主控台中設定 WebSocket API 的路由請求。

### Note

您可以在建立整合前後，在 AWS CLI 和 AWS 開發套件中建立路由。目前主控台不支援整合的重複使用，因此您必須先建立路由，然後建立該路由的整合。

您可以將 API Gateway 設定為對路由執行驗證，再繼續進行整合請求。如果驗證失敗，API Gateway 失敗請求，而不呼叫後端，傳送與以下用戶端類似的 "Bad request body" 閘道回應，以及在 CloudWatch Logs 中發佈驗證結果：

```
{"message" : "Bad request body", "connectionId": "{connectionId}", "messageId": "{messageId}"}
```

這可減少對後端不必要的呼叫，讓您專注在 API 的其他要求。

您也可以為 API 路由定義路由回應，以啟用雙向通訊。路由回應說明會在特定路由整合完成時，將哪些資料傳送到用戶端。例如，如果您希望用戶端將訊息傳送到後端，而不接收回應 (單向通訊)，您不需要定義路由的回應。不過，如果您不提供路由回應，API Gateway 不會將任何與您整合相關的資訊傳送到用戶端。

## \$default 路由

每個 API Gateway WebSocket API 都可以有一個 `$default` 路由。這是一個特殊路由值，使用方式如下：

- 您可以使用該值搭配定義的路由金鑰，來為不符合任何定義路由金鑰的內送訊息指定「備用」路由 (例如，一般偽裝整合，會傳回特定的錯誤訊息)。
- 您可以使用該值與任何定義路由金鑰，來指定將路由委派到後端元件的 Proxy 模型。
- 您可以使用該值來為非 JSON 承載指定路由。

## 自訂路由

如果您想要根據訊息內容來叫用特定的整合，則可透過建立自訂路由來這麼做。

自訂路由會使用您指定的路由金鑰和整合。當內送訊息包含 JSON 屬性，而且該屬性的判斷值符合路由金鑰值時，API Gateway 會叫用整合。(如需詳細資訊，請參閱「[the section called “關於 WebSocket API” \(p. 551\)](#)」)。

例如，假設您想要建立聊天空間應用程式。您可以透過建立路由選擇表達式為 `$request.body.action` 的 WebSocket API 來開始。然後，您可以接著定義兩個路由：`joinroom` 和 `sendmessage`。用戶端應用程式可以透過傳送如下訊息，來叫用 `joinroom` 路由：

```
{"action": "joinroom", "roomname": "developers"}
```

其可以透過傳送如下訊息，來叫用 `sendmessage` 路由：

```
{"action": "sendmessage", "message": "Hello everyone"}
```

## 使用 API Gateway WebSocket API 整合，以連接到商業邏輯

在為 API Gateway WebSocket API 設定路由後，您必須指定您想使用的整合。如同路由 (可能會擁有路由請求和路由回應)，整合可以擁有整合請求和整合回應。整合請求包含後端預期的資訊，以處理來自您用戶端的請求。整合回應包含後端傳回到 API Gateway 的資料，而且可能會用於建構訊息以傳送到用戶端 (如果路由回應已定義)。

如需設定整合的詳細資訊，請參閱[the section called “設定 WebSocket API 整合” \(p. 557\)](#)。

## WebSocket API 和 REST API 之間的重要差異

WebSocket API 整合與 REST API 整合類似，但差異如下：

- 目前，您必須先在 API Gateway 主控台中建立路由，然後建立整合做為該路由的目標。不過，您可以在 API 和 CLI 中，以任何順序單獨建立路由和整合。
- 您可以為多個路由使用單一整合。例如，如果您有一組動作彼此間密切關聯，您可能需要所有路由來移至單一 Lambda 函數。您不必多次定義整合的詳細資訊，您可以指定一次，並將其指派給每個相關的路由。

### Note

目前主控台不支援整合的重複使用，因此您必須先建立路由，然後建立該路由的整合。

您可以在 AWS CLI 和 AWS 開發套件中，透過將路由的目標設定為

`"integrations/{integration-id}"` 值，來重複使用整合，`{integration-id}` 是要與路由關聯之整合的唯一 ID。

- API Gateway 提供您可以在路由和整合使用的多種[選擇表達式 \(p. 570\)](#)。您不需要倚賴內容類型來選取輸入範本或輸出映射。如同路由選擇表達式，您可以定義由 API Gateway 評估的選擇表達式來適當的項目。如果未找到相符範本，所有這些都會回復為 `$default` 範本。
  - 在整合請求中，範本選擇表達式支援 `$request.body.<json_path_expression>` 和靜態值。
  - 在整合回應中，範本選擇表達式支援 `$request.body.<json_path_expression>`、`$integration.response.statuscode` 和 `$integration.response.header.<headerName>`。

在 HTTP 通訊協定 (其中請求和回應會同步傳送) 中；通訊基本上是單向。在 WebSocket 協定中，通訊是雙向。回應是非同步的，用戶端收到的順序與用戶端訊息的傳送順序不一定相同。此外，後端可以將訊息給傳送用戶端。

### Note

對於設定為使用 `AWS_PROXY` 或 `LAMBDA_PROXY` 整合的路由，通訊是單向，API Gateway 不會自動將後端回應傳遞至路由回應。例如，在 `LAMBDA_PROXY` 整合的情況下，就不會將 Lambda 函數傳回的內文傳回給用戶端。如果您希望用戶端接收整合回應，您必須定義路由回應，以進行雙向的通訊。

## 處理二進位承載

API Gateway WebSocket API 在內送訊息承載中目前不支援二進位影格。如果用戶端應用程式傳送的是二進位影格，API Gateway 會拒絕接收和中斷與用戶端的連線並出現程式碼 1003。

此行為有一個解決方法。如果用戶端傳送文字編碼的二進位資料 (例如，Base64) 做為文字框架，您可以將整合的 `contentHandlingStrategy` 屬性設定為 `CONVERT_TO_BINARY`，以從 Base64 編碼字串的承載轉換到二進位。

要為非 Proxy 整合二進位承載傳回路由回應，您可以將整合回應的 `contentHandlingStrategy` 屬性設定為 `CONVERT_TO_TEXT`，以從二進位到的承載轉換 Base64 編碼字串。

## 將資料從後端服務傳送到連線的用戶端

API Gateway WebSocket API 為您提供以下方式，將資料從後端服務傳送到連線用戶端：

- 整合可以傳送回應，其回應會由您已定義的路由來傳回用戶端。
- 您可以使用 @connections API 來傳送 POST 請求。如需詳細資訊，請參閱[the section called “在後端服務使用 @connections 命令” \(p. 565\)](#)。

## 在 API Gateway 中建立 WebSocket API

您可以使用 AWS CLI `create-api` 命令，或在 AWS 開發套件中使用 `CreateApi` 命令，在 API Gateway 主控台建立 WebSocket API。下列程序顯示如何建立新的 WebSocket API。

### Note

WebSocket API 僅支援 TLS 1.2。不支援舊版 TLS。

## 使用 AWS CLI 命令建立 WebSocket API

如下範例所示，使用 AWS CLI 建立 WebSocket API 需要呼叫 `create-api` 命令，這會建立含 `$request.body.action` 路由選擇表達式的 API：

```
aws apigatewayv2 --region us-east-1 create-api --name "myWebSocketApi3" --protocol-type WEBSOCKET --route-selection-expression '$request.body.action'
```

輸出範例：

```
{  
    "ApiKeySelectionExpression": "$request.header.x-api-key",  
    "Name": "myWebSocketApi3",  
    "CreatedDate": "2018-11-15T06:23:51Z",  
    "ProtocolType": "WEBSOCKET",  
    "RouteSelectionExpression": "'$request.body.action'",  
    "ApiId": "aabcccddee"  
}
```

## 使用 API Gateway 主控台建立 WebSocket API

您可以選擇 WebSocket 通訊協定並為 API 提供名稱，以在主控台中建立 WebSocket API。

### Important

一旦建立 API 後，您無法更改您選擇的通訊協定。您無法將 WebSocket API 轉換到 REST API，反之亦然。

## 使用 API Gateway 主控台建立 WebSocket API

1. 登入 API Gateway 主控台，然後選擇 Create API (建立 API)。
2. 在 Choose the protocol (選擇通訊協定) 下，選擇 WebSocket (WebSocket)。
3. 在 Create a new API (建立新 API) 下，選擇 New API (新增 API)。
4. 在 Settings (設定) 下，在 API name (API 名稱) 欄位中，輸入 API 的名稱，例如 **PetStore**。
5. 輸入 API 的 Route Selection Expression (路由選擇表達式)，例如，`$request.body.action`。  
如需路由選擇表達式的詳細資訊，請參閱[the section called “” \(p. 570\)](#)。
6. 如有需要，輸入 API 的 Description (說明)。
7. 選擇 Create API (建立 API)。

# 在 API Gateway 中設定 WebSocket API 的路由

## 主題

- [在 API Gateway 中設定 WebSocket API 的路由 \(p. 556\)](#)
- [指定路由請求設定 \(p. 557\)](#)

# 在 API Gateway 中設定 WebSocket API 的路由

首次新建的 WebSocket API 有三個預先定義的路由：`$connect`、`$disconnect` 和 `$default`，您可以使用主控台、API 或 AWS CLI 來加以建立。如有需要，您可以建立自訂路由。如需詳細資訊，請參閱 [the section called “關於 WebSocket API” \(p. 551\)](#)。

## Note

在 CLI 中，您可在建立整合之前或之後建立路由，也可以在多個路由重複使用相同整合。

## 主題

- [使用 API Gateway 主控台建立路由 \(p. 556\)](#)
- [使用 AWS CLI 建立路由 \(p. 556\)](#)

# 使用 API Gateway 主控台建立路由

## 使用 API Gateway 主控台建立路由

1. 登入 API Gateway 主控台、選擇 API，然後選擇 Routes (路由)。
2. 欲建立預先定義路由之一 (`$connect`、`$disconnect` 和 `$default`)，請選擇其名稱。
3. 如有需要，您可以建立自訂路由。方法是在 New Route Key (新的路由金鑰) 文字方塊輸入路由金鑰名稱，並選擇核取記號圖示。

## Note

建立自訂路由時，請勿在路由金鑰名稱字首使用 `$`。此字首保留供預先定義路由使用。

# 使用 AWS CLI 建立路由

欲使用 AWS CLI 來建立路由，請呼叫 `create-route`，如下列範例所示：

```
aws apigatewayv2 --region us-east-1 create-route --api-id aabbccdde --route-key $default
```

輸出範例：

```
{  
    "ApiKeyRequired": false,  
    "AuthorizationType": "NONE",  
    "RouteKey": "$default",  
    "RouteId": "1122334"  
}
```

## 指定路由請求設定

### 指定 \$connect 的路由請求設定

設定 API 的 \$connect 路由時，可使用下列選用設定，以啟用您 API 的授權。如需詳細資訊，請參閱[the section called “\\$connect 路由” \(p. 552\)](#)。

- Authorization (授權)：若無須授權，可指定為 NONE。否則，您可指定：
  - AWS\_IAM，以使用標準的 AWS IAM 政策來控制您 API 的存取。
  - CUSTOM，以指定您之前建立的 Lambda 授權方函數，藉此實作 API 的授權。授權方可位於您自己的 AWS 帳戶或其他 AWS 帳戶。如需 Lambda 授權方的詳細資訊，請參閱[使用 API Gateway Lambda 授權方 \(p. 348\)](#)。

#### Note

在 API Gateway 主控台中，只有在您設定授權方函數 (如[the section called “使用主控台設定 Lambda 授權方” \(p. 355\)](#) 所述) 後，才會看到 CUSTOM 設定。

#### Important

Authorization (授權) 設定會套用至整個 API，不只是 \$connect 路由。\$connect 路由會保護其他路由，因為每次連線都會呼叫該路由。

- API Key Required (需要 API 金鑰)：API 的 \$connect 路由可選擇要求 API 金鑰。您可以同時使用 API 金鑰與用量計劃，以控制並追蹤對 API 的存取。如需詳細資訊，請參閱[the section called “使用 API 金鑰的用量計劃” \(p. 397\)](#)。

## 使用 API Gateway 主控台設定 \$connect 路由請求

使用 API Gateway 主控台來設定 WebSocket API 的 \$connect 路由請求：

1. 登入 API Gateway 主控台、選擇 API，然後選擇 Routes (路由)。
2. 在 Routes (路由) 底下，選擇 \$connect。
3. 在路由概觀窗格中，選擇 Route Request (路由請求)。
4. 在 Access Settings (存取設定) 底下，設定路由設定，如下所示：
  - a. 欲編輯 Authorization (授權) 設定，請選擇鉛筆圖示。從下拉式功能表選擇所需設定，然後選擇核取記號圖示來儲存新的設定。
  - b. 欲編輯 API Key Required (需要 API 金鑰) 設定，請選擇鉛筆圖示。從下拉式功能表選擇 true 或 false，然後選擇核取記號圖示來儲存新的設定。

## 在 API Gateway 中設定 WebSocket API 整合

### 主題

- 在 API Gateway 中設定 WebSocket API 整合請求 (p. 558)
- 在 API Gateway 中設定 WebSocket API 整合回應 (p. 560)

## 在 API Gateway 中設定 WebSocket API 整合請求

設定整合請求包含下列事項：

- 選擇路由金鑰來整合後端。
- 指定欲呼叫的後端端點，例如 AWS 服務或 HTTP 端點。
- 視需要指定一個或多個請求範本，設定路由請求資料轉換為整合請求資料的方式。

## 使用 API Gateway 主控台設定 WebSocket API 整合請求

使用 API Gateway 主控台在 WebSocket API 內的路由新增整合請求

1. 登入 API Gateway 主控台、選擇 API，然後選擇 Routes (路由)。
2. 在 Routes (路由) 底下選擇路由。
3. 在 Route Overview (路由概觀) 窗格中，選擇 Integration Request (整合請求)。
4. 針對 Integration Type (整合類型)，選擇以下其中一項：
  - 若您的 API 將整合您在此帳戶或另一個帳戶中已建立的 AWS Lambda 函數，此時請選擇 Lambda Function (Lambda 函數)。  
若要在 AWS Lambda 中建立新的 Lambda 函數、設定 Lambda 函數的資源許可，或執行其他 Lambda 服務動作，請改選擇 AWS Service (AWS 服務)。
    - 若您的 API 將整合現有 HTTP 端點，請選擇 HTTP (HTTP)。如需詳細資訊，請參閱[在 API Gateway 中設定 HTTP 整合 \(p. 224\)](#)。
    - 若您希望直接從 API Gateway 產生 API 回應，不使用整合後端，請選擇 Mock (模擬)。如需詳細資訊，請參閱[在 API Gateway 中設定模擬整合 \(p. 234\)](#)。
    - 若您的 API 將整合 AWS 服務，請選擇 AWS Service (AWS 服務)。
    - 若您的 API 將使用 VpcLink 做為私有整合端點，請選擇 VPC Link (VPC 連結)。如需詳細資訊，請參閱[設定 API Gateway 私有整合 \(p. 229\)](#)。
5. 如果您選擇 Lambda Function (&LAM; 函數)，請執行下列操作：
  - a. 針對 Lambda Region (&LAM; 區域)，選擇對應到您 Lambda 函數建立所在區域的區域識別符。例如，如已在 US East (N. Virginia) 區域建立 Lambda 函數，請選擇 `us-east-1`。如需區域名稱和識別符清單，請參閱 Amazon Web Services 一般參考中的 [AWS Lambda](#)。
  - b. 若您希望使用 [Lambda 代理整合 \(p. 206\)](#)或[跨帳戶 Lambda 代理整合 \(p. 29\)](#)，請選擇 Use Lambda Proxy integration (使用 Lambda 代理整合) 核取方塊。
  - c. 針對 Lambda Function (Lambda 函數)，請以下列方法之一指定函數：
    - 若您的 Lambda 函數位於同一個帳戶，請開始輸入函數名稱，然後從下拉式清單中選擇函數。

### Note

函數名稱可選擇納入其別名或版本規格，如 `HelloWorld`、`HelloWorld:1` 或 `HelloWorld:alpha`。

- 若函數位於不同帳戶，請輸入該函數的 ARN。
- d. 若您希望 API Gateway 使用傳入請求內的登入資料來呼叫 Lambda 函數，請選擇 Invoke with caller credentials (使用發起人登入資料呼叫)。
- e. 針對 Execution role (執行角色)，請輸入 Lambda 呼叫角色的 ARN，讓 API Gateway 能夠呼叫您的 Lambda 函數。

- f. 若要使用預設 29 秒的逾時數值，請保持勾選 Use Default Timeout (使用預設逾時)。若要設定自訂逾時，請取消勾選核取方塊，然後輸入介於 50 和 29000 毫秒之間的逾時數值。
- g. 選擇 Save (儲存)。
6. 若您選擇 HTTP (HTTP)，請依照 [the section called “ 使用主控台設定整合請求” \(p. 203\)](#) 步驟 4 的指示。
7. 若您選擇 Mock (模擬)，請前往 Request Templates (請求範本) 步驟。
8. 若您選擇 AWS Service (AWS 服務)，請依照 [the section called “ 使用主控台設定整合請求” \(p. 203\)](#) 步驟 6 的指示。
9. 若您選擇 VPC Link (VPC 連結)，請執行下列操作：
  - a. 若您希望將請求代理到 VPCLink 端點，請選擇 Use Proxy integration (使用代理整合) 核取方塊。
  - b. 從 VPC Link (VPC 連結) 下拉式清單中，選擇 [Use Stage Variables]，然後在清單下方的文字方塊中輸入 \${stageVariables.vpcLinkId}。

API 部署到階段後，我們將定義 vpcLinkId 階段變數，並將其值設定為 VpcLink 的 ID。
  - c. 針對 HTTP method (HTTP 方法)，選擇最符合 HTTP 後端中方法的 HTTP 方法類型。
  - d. 針對 Endpoint URL (端點 URL)，請輸入您希望此整合使用之 HTTP 後端的 URL。
  - e. 若要使用預設 29 秒的逾時數值，請保持勾選 Use Default Timeout (使用預設逾時)。若要設定自訂逾時，請取消勾選核取方塊，然後輸入介於 50 和 29000 毫秒之間的逾時數值。
10. 在 Request Templates (請求範本) 底下執行下列動作：
  - 針對 Template Selection Expression (範本選擇表達式)，請選擇鉛筆圖示，然後將文字 template 取代為範本選擇表達式。API Gateway 將於訊息承載尋找此表達式。若找到將對其評估，而結果範本金鑰值將用於選取在訊息承載內套用至資料的資料映射範本。

如需範本選擇表達式的相關資訊，請參閱 [the section called “” \(p. 571\)](#)。

## 使用 AWS CLI 設定整合請求

您可使用 AWS CLI 設定 WebSocket API 內路由的整合請求，建立模擬整合，如下列範例所示：

1. 建立名為 integration-params.json 的檔案，其中內容如下：

```
{"PassthroughBehavior": "WHEN_NO_MATCH", "TimeoutInMillis": 29000, "ConnectionType": "INTERNET", "RequestTemplates": {"application/json": "{\"statusCode\":200}"}, "IntegrationType": "MOCK"}
```

2. 執行 `create-integration` 命令，如下範例所示：

```
aws apigatewayv2 --region us-east-1 create-integration --api-id aabbccdde --cli-input-json file://integration-params.json
```

此範例的範例輸出如下：

```
{  
    "PassthroughBehavior": "WHEN_NO_MATCH",  
    "TimeoutInMillis": 29000,  
    "ConnectionType": "INTERNET",  
    "IntegrationResponseSelectionExpression": "${response.statusCode}",  
    "RequestTemplates": {  
        "application/json": "{\"statusCode\":200}"  
    },  
    "IntegrationId": "0abcdef",  
    "IntegrationType": "MOCK"
```

}

或者，您可使用 AWS CLI 來設定代理整合的整合請求，如下列範例所示：

1. 在 Lambda 主控台中建立 Lambda 函數，並賦予基本的 Lambda 執行角色。
2. 執行 `create-integration` 命令，如下列範例所示：

```
aws apigatewayv2 create-integration --api-id aabbccdde --integration-type
AWS_PROXY --integration-method POST --integration-uri arn:aws:apigateway:us-
east-1:lambda:path/2015-03-31/functions/arn:aws:lambda:us-
east-1:123412341234:function:simpleproxy-echo-e2e/invocations
```

此範例的範例輸出如下：

```
{
    "PassthroughBehavior": "WHEN_NO_MATCH",
    "IntegrationMethod": "POST",
    "TimeoutInMillis": 29000,
    "ConnectionType": "INTERNET",
    "IntegrationUri": "arn:aws:apigateway:us-east-1:lambda:path/2015-03-31/functions/
arn:aws:lambda:us-east-1:123412341234:function:simpleproxy-echo-e2e/invocations",
    "IntegrationId": "abcdefg",
    "IntegrationType": "AWS_PROXY"
}
```

## 在 API Gateway 中設定 WebSocket API 整合回應

### 主題

- [整合回應概觀 \(p. 560\)](#)
- [使用 API Gateway 主控台設定整合回應 \(p. 561\)](#)
- [使用 AWS CLI 設定整合回應 \(p. 561\)](#)

## 整合回應概觀

API Gateway 的整合回應是從後端服務建模並操控回應的方式。REST API 與 WebSocket API 整合回應的設定方式有些許差異（如下述），但概念上的行為相同。

WebSocket 路由可設定為雙向或單向通訊。若路由具備路由回應，則此設定為雙向通訊。否則，此設定為單向通訊。

- 路由設定為雙向通訊時，整合回應可讓您設定回傳訊息承載的轉換，類似 REST API 的整合回應。
- 若路由設定為單向通訊，則不論整合回應的設定為何，WebSocket 管道在訊息處理後都不會回傳回應。

此文件後續部分假定您選擇將路由設定為雙向通訊。

整合可分為代理整合和非代理整合。

### Important

以代理整合而言，API Gateway 會自動將後端輸出以完整的承載傳遞至發起人。此時沒有整合回應。

以非代理整合而言，您必須設定至少一個整合回應：

- 在沒有明確選擇時，其中一個整合回應最好設定為全部截獲。將整合回應金鑰設定為 \$default 就是此預設案例的代表。
- 在其他情況下，整合回應金鑰會以常規表達式運作，應遵循 "/expression/" 的格式。

以非代理 HTTP 整合而言：

- API Gateway 將嘗試比對後端回應的 HTTP 狀態碼。整合回應金鑰此時會以常規表達式運作，若找不到針對項目，將選擇 \$default 做為整合回應。
- 範本選擇表達式運作方式完全相同，如上所述。例如：
  - /2\d\d/：接收並轉換成功的回應
  - /4\d\d/：接收並轉換不正確的請求錯誤
  - \$default：接收並轉換所有意外回應

如需範本選擇表達式目前限制的資訊，請參閱 [the section called “” \(p. 571\)](#)。

## 使用 API Gateway 主控台設定整合回應

使用 API Gateway 主控台設定 WebSocket API 的路由整合回應：

- 登入 API Gateway 主控台、選擇 API，然後選擇 Routes (路由)。
- 選擇路由。
- 選擇 Integration Response (整合回應)。
- 在 Integration Responses (整合回應) 底下，在 Response Selection Expression (回應選擇表達式) 文字方塊中輸入值。
- Response Key (回應金鑰) 底下，請選擇 Add Response (新增回應)。
  - 欲定義整合回應金鑰，請在 New Response Key (新的回應金鑰) 文字方塊內輸入金鑰名稱，然後選擇核取記號圖示。
  - 選擇 Template Selection Expression (範本選擇表達式) 旁的鉛筆圖示，然後輸入供 API Gateway 從傳出訊息中尋找的表達式。此表達式的評估結果應為映射至您回應範本之一的整合回應金鑰值。

如需範本選擇表達式的相關資訊，請參閱 [the section called “” \(p. 571\)](#)。

## 使用 AWS CLI 設定整合回應

欲使用 AWS CLI 來設定 WebSocket API 的整合回應，請呼叫 `create-integration-response` 命令。下列 CLI 命令為回應設定為 200 的範例：

```
aws apigatewayv2 create-integration-response \
--api-id vaz7da96z6 \
--integration-response-key 200
```

# 在 API Gateway 中設定 WebSocket API 的路由回應

WebSocket 路由可設定為雙向或單向通訊。若路由具備路由回應，則此設定為雙向通訊。否則，此設定為單向通訊。路由回應係用於啟用雙向通訊，其中您的 API 可將回應做為用戶端訊息的內容傳送回用戶端。

您可使用 API Gateway 主控台、AWS CLI 或 AWS 開發套件來設定路由回應與回應選擇表達式。如需路由回應的詳細資訊，請參閱 [the section called “呼叫後端整合” \(p. 552\)](#)。

如需路由回應選擇表達式的詳細資訊，請參閱 [the section called “” \(p. 572\)](#)。

#### 主題

- [使用 API Gateway 主控台設定路由回應 \(p. 562\)](#)
- [使用 AWS CLI 設定路由回應 \(p. 562\)](#)

## 使用 API Gateway 主控台設定路由回應

使用 API Gateway 主控台來設定 WebSocket API 的路由回應：

1. 登入 API Gateway 主控台，選擇 API。
2. 在 Routes (路由) 底下選擇路由。
3. 在路由概觀窗格中，選擇 Route Response (路由回應)。
4. 在 Response Modeling (回應模型) 底下，於 Response Selection Expression (回應選擇表達式) 方塊中輸入欲使用的回應選擇表達式，然後選擇核取記號圖示。
5. 在 Route Responses (路由回應) 底下，請選擇 Response Key (回應金鑰) 下方的 Add Response (新增回應)。

#### Note

目前 WebSocket API 的路由回應僅支援 \$default。

6. 輸入回應金鑰名稱並選擇核取記號圖示。

## 使用 AWS CLI 設定路由回應

欲使用 AWS CLI 來設定 WebSocket API 的路由回應，請呼叫 `create-route-response` 命令，如下列範例所示。

```
aws apigatewayv2 --region us-east-1 create-route-response --api-id aabbcccddee --route-id 1122334 --route-response-key $default
```

輸出範例：

```
{  
    "RouteResponseId": "abcdef",  
    "RouteResponseKey": "$default"  
}
```

## 在 API Gateway 中部署 WebSocket API

建立 WebSocket API 後，您必須將其部署以您的使用者叫用。

若要部署 API，請建立 API 部署 ([p. 5](#))，並建立它與階段 ([p. 6](#)) 的關聯。每個階段都是 API 的快照，而且可以供用戶端應用程式呼叫。

#### Important

每次您更新 API 時 (包含修改路由、方法、整合、授權方，以及階段設定以外的任何其他項目)，都必須將 API 重新部署至現有階段或新的階段。

在預設情況下，每個 API 只能有 10 個階段，所以建議重複使用。

若要呼叫已部署的 WebSocket API，用戶端會將訊息傳送到 API 的 URL。URL 的決定方式為 API 的主機名稱和階段名稱。

Note

API Gateway 將支援高達 128 KB 的承載，影格大小上限為 32 KB。如果訊息超過 32 KB，則必須分割成多個影格，每個為 32 KB 或以下。

使用 API 的預設網域名稱，處於指定階段 (*{stageName}*) 的 (例如) WebSocket API URL 格式如下：

```
wss://{api-id}.execute-api.{region}.amazonaws.com/{stageName}
```

為了讓使用者更容易使用 WebSocket API 的 URL，您可以建立自訂網域名稱 (例如，api.example.com) 以取代 API 的預設主機名稱。其組態程序與 REST API 相同。如需詳細資訊，請參閱[the section called “設定 API 自訂網域名稱” \(p. 590\)](#)。

階段啟用 API 的強大版本控制。例如，您可以將 API 部署至 test 階段和 prod 階段，並使用 test 階段作為測試組建，以及使用 prod 階段作為穩定組建。更新通過測試之後，您就可以將 test 階段提升為 prod 階段。您可以將 API 重新部署到 prod 階段來進行提升。

主題

- [使用 AWS CLI 建立 WebSocket API 部署 \(p. 563\)](#)
- [使用 API Gateway 主控台建立 WebSocket API 部署 \(p. 564\)](#)

## 使用 AWS CLI 建立 WebSocket API 部署

若要使用 AWS CLI 建立部署，如以下範例所示使用 `create-deployment` 命令：

```
aws apigatewayv2 --region us-east-1 create-deployment --api-id aabbccddeee
```

輸出範例：

```
{  
    "DeploymentId": "fedcba",  
    "DeploymentStatus": "DEPLOYED",  
    "CreatedDate": "2018-11-15T06:49:09Z"  
}
```

在您將此部署與階段建立關聯前，都無法呼叫已部署的 API。您可以建立新階段或重複使用您之前建立的階段。

若要建立新的階段，並將其與部署建立關聯，請使用如下範例所示的 `create-stage` 命令：

```
aws apigatewayv2 --region us-east-1 create-stage --api-id aabbccddeee --deployment-id fedcba  
--stage-name test
```

輸出範例：

```
{  
    "StageName": "test",  
    "CreatedDate": "2018-11-15T06:50:28Z",  
    "DeploymentId": "fedcba",  
    "DefaultRouteSettings": {  
        "MetricsEnabled": false,  
        "ThrottlingBurstLimit": 5000,
```

```
        "DataTraceEnabled": false,  
        "ThrottlingRateLimit": 10000.0  
    },  
    "LastUpdatedDate": "2018-11-15T06:50:28Z",  
    "StageVariables": {},  
    "RouteSettings": {}  
}
```

若要重複使用現有階段，您可使用 `update-stage` 命令搭配新建立的部署 ID (`{deployment-id}`) 來更新階段的 `deploymentId` 屬性。

```
aws apigatewayv2 update-stage --region {region} \  
  --api-id {api-id} \  
  --stage-name {stage-name} \  
  --deployment-id {deployment-id}
```

## 使用 API Gateway 主控台建立 WebSocket API 部署

若要使用 API Gateway 主控台為 WebSocket API 建立部署：

1. 登入 API Gateway 主控台，然後選擇 API。
2. 從 Actions (動作) 下拉式選單中，選擇 Deploy API (部署 API)。
3. 從下拉式清單中選擇所需的階段，或輸入新階段的名稱。

## 呼叫 WebSocket API

一旦您已部署 WebSocket API，用戶端應用程式可以連接到其中和傳送訊息，且後端可將訊息傳送到連線的用戶端應用程式：

- 您可以使用 `wscat` 來連接 WebSocket API 和將訊息傳送到其中，以模擬用戶端行為。請參閱「[the section called “使用 wscat 以連接到 WebSocket API 和將訊息傳送到其中” \(p. 564\)](#)」。
- 您可以從後端服務使用 `@connections` API，來將回呼訊息傳送到連線用戶端、取得連線資訊，或中斷用戶端。請參閱「[the section called “在後端服務使用 @connections 命令” \(p. 565\)](#)」。
- 用戶端應用程式可以使用自己的 WebSocket 程式庫來叫用 WebSocket API。

## 使用 wscat 以連接到 WebSocket API 和將訊息傳送到其中

`wscat` 公用程式是一種便捷的工具，適用於測試在 API Gateway 中建立和部署的 WebSocket API。您可以如下安裝和使用 `wscat`：

1. 從 <https://www.npmjs.com/package/wscat> 下載 `wscat`。
2. 請執行下列命令進行安裝：

```
npm install -g wscat
```

3. 若要連線到 API，如下列命令範例所示，執行 `wscat` 命令。請注意，這個範例假設 `Authorization` 設定是 `NONE`。

```
wscat -c wss://aabbcdddee.execute-api.us-east-1.amazonaws.com/test/
```

您將需要使用實際 API ID 取代 `aabbcccddee`，該 ID 會顯示在 API Gateway 主控台或由 AWS CLI `create-api` 命令傳回。

此外，若 API 在 us-east-1 以外的區域，您將需要替代正確的區域。

- 為了測試您的 API，請在連線的同時輸入如下訊息：

```
{"${jsonpath-expression}": "${route-key}"}
```

其中 `{jsonpath-expression}` 是一種 JSONPath 表達式和 `{route-key}` 是 API 的路由金鑰。例如：

```
{"action": "action1"
"message": "test response body"}
```

如需 JSONPath 的詳細資訊，請參閱 [JSONPath 或適用於 Java 的 JSONPath](#)。

- 若要中斷與 API 的連線，請輸入 `ctrl-C`。

## 在後端服務使用 @connections 命令

後端服務可以使用以下 WebSocket 連線 HTTP 請求，來將回呼訊息傳送到連線用戶端、取得連線資訊，或中斷用戶端。

### Important

這些請求使用 [IAM 授權 \(p. 566\)](#)，所以您必須使用 [Signature 第 4 版 \(SigV4\)](#) 來進行簽署。

在以下命令中，您將需要使用實際 API ID 取代 `{api-id}`，該 ID 會顯示在 API Gateway 主控台或由 AWS CLI `create-api` 命令傳回。此外，若 API 在 us-east-1 以外的區域，您將需要替代正確的區域。

若要將回呼訊息傳送給用戶端，請使用：

```
POST https://${api-id}.execute-api.us-east-1.amazonaws.com/${stage}/
@connections/${connection_id}
```

您可以使用 [Postman](#) 或呼叫 [awscurl](#) 來測試此請求，如下範例所示：

```
awscurl --service execute-api -X POST -d "hello world" https://${prefix}.execute-api.us-
east-1.amazonaws.com/${stage}/@connections/${connection_id}
```

您將需要經由 URL 將命令編碼，如下範例所示：

```
awscurl --service execute-api -X POST -d "hello world" https://aabbcccddee.execute-api.us-
east-1.amazonaws.com/prod/%40connections/R0oXAdfD0kwCH6w%3D
```

### Note

Postman 不會編碼 @connections URL，因此您需要取代已編碼 URL 中的 = 和 @ 字元。

您可以在整合時使用 `$context` 變數來動態建置回呼 URL。例如，如果您使用 Lambda Proxy 整合搭配 Node.js Lambda 函數，您可以如下所示建置 URL：

```
exports.handler = function(event, context, callback) {
```

```
var domain = event.requestContext.domain;
var stage = event.requestContext.stage;
var connectionId = event.requestContext.connectionId;
var callbackUrl = util.format(util.format('https://%s/%s@connections/%s', domain, stage,
connectionId));
// Do a SigV4 and then make the call
}
```

## 在 API Gateway 中控制 WebSocket API 的存取

### 主題

- [使用 IAM 授權 \(p. 566\)](#)
- [建立 Lambda REQUEST 授權方函數 \(p. 567\)](#)

## 使用 IAM 授權

WebSocket API 及 [REST API \(p. 339\)](#) 具備類似的 IAM 授權，除了以下例外：

- 除了現有動作 (`Invoke`、`InvalidateCache`)，`execute-api` 動作也支援 `ManageConnections`。`ManageConnections` 會控制 @connections API 的存取。
- WebSocket 路由使用不同的 ARN 格式：

```
arn:aws:execute-api:region:account-id:api-id/stage-name/route-key
```

- @connections API 使用的 ARN 格式與 REST API 相同：

```
arn:aws:execute-api:region:account-id:api-id/stage-name/POST/@connections
```

例如，您可針對用戶端設定下列政策。除了 `prod` 階段內的秘密路由，此範例允許所有人針對所有路由傳送訊息 (`Invoke`)，並防止所有人在任何階段回傳訊息給已連線的用戶端 (`ManageConnections`)。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "execute-api:Invoke"
      ],
      "Resource": [
        "arn:aws:execute-api:us-east-1:account-id:api-id/prod/*"
      ]
    },
    {
      "Effect": "Deny",
      "Action": [
        "execute-api:Invoke"
      ],
      "Resource": [
        "arn:aws:execute-api:us-east-1:account-id:api-id/prod/secret"
      ]
    },
    {
      "Effect": "Deny",
      "Action": [
        "execute-api:ManageConnections"
      ],
      "Resource": [
        "arn:aws:execute-api:us-east-1:account-id:api-id/*"
      ]
    }
  ]
}
```

```
        "execute-api:ManageConnections"
    ],
    "Resource": [
        "arn:aws:execute-api:us-east-1:account-id:api-id/*"
    ]
}
}
```

## 建立 Lambda REQUEST 授權方函數

WebSocket API 及 REST API (p. 350) 具備類似的 Lambda 授權方函數，除了以下例外：

- 您不能使用路徑變數 (`event.pathParameters`)，因為路徑已固定。
- `event.methodArn` 與同等的 REST API 不同，因其沒有 HTTP 方法。若是 `$connect`，`methodArn` 會以 `"$connect"` 做為結尾：

```
arn:aws:execute-api:region:account-id:api-id/stage-name/$connect
```

- `event.requestContext` 內的內容變數與 REST API 的不同。

下列範例 Lambda 授權方函數為 the section called “在 Lambda 主控台建立 Lambda 授權方” (p. 350) 內 REST API 之 Lambda 授權方函數的 WebSocket 版本：

```
exports.handler = function(event, context, callback) {
    console.log('Received event:', JSON.stringify(event, null, 2));

    // A simple REQUEST authorizer example to demonstrate how to use request
    // parameters to allow or deny a request. In this example, a request is
    // authorized if the client-supplied HeaderAuth1 header, QueryString1 query parameter,
    // stage variable of StageVar1 and the accountId in the request context all match
    // specified values of 'headerValue1', 'queryValue1', 'stageValue1', and
    // '123456789012', respectively.

    // Retrieve request parameters from the Lambda function input:
    var headers = event.headers;
    var queryStringParameters = event.queryStringParameters;
    var stageVariables = event.stageVariables;
    var requestContext = event.requestContext;

    // Parse the input for the parameter values
    var tmp = event.methodArn.split(':');
    var apiGatewayArnTmp = tmp[5].split('/');
    var awsAccountId = tmp[4];
    var region = tmp[3];
    var restApiId = apiGatewayArnTmp[0];
    var stage = apiGatewayArnTmp[1];
    var route = apiGatewayArnTmp[2];

    // Perform authorization to return the Allow policy for correct parameters and
    // the 'Unauthorized' error, otherwise.
    var authResponse = {};
    var condition = {};
    conditionIpAddress = {};

    if (headers.HeaderAuth1 === "headerValue1"
        && queryStringParameters.QueryString1 === "queryValue1"
        && stageVariables.StageVar1 === "stageValue1"
        && requestContext.accountId === "123456789012") {
        callback(null, generateAllow('me', event.methodArn));
    } else {
```

```
        callback("Unauthorized");
    }

// Help function to generate an IAM policy
var generatePolicy = function(principalId, effect, resource) {
    // Required output:
    var authResponse = {};
    authResponse.principalId = principalId;
    if (effect && resource) {
        var policyDocument = {};
        policyDocument.Version = '2012-10-17'; // default version
        policyDocument.Statement = [];
        var statementOne = {};
        statementOne.Action = 'execute-api:Invoke'; // default action
        statementOne.Effect = effect;
        statementOne.Resource = resource;
        policyDocument.Statement[0] = statementOne;
        authResponse.policyDocument = policyDocument;
    }
    // Optional output with custom properties of the String, Number or Boolean type.
    authResponse.context = {
        "stringKey": "stringval",
        "numberKey": 123,
        "booleanKey": true
    };
    return authResponse;
}

var generateAllow = function(principalId, resource) {
    return generatePolicy(principalId, 'Allow', resource);
}

var generateDeny = function(principalId, resource) {
    return generatePolicy(principalId, 'Deny', resource);
}
```

欲將前述 Lambda 函數設定為 WebSocket API 的 REQUEST 授權方函數，請遵循 [REST API \(p. 355\)](#) 內的相同程序。

欲將 \$connect 路由在主控台內設定為使用此 Lambda 授權方，請選取或建立 \$connect 路由。選擇路由請求，並從 Authorization (授權) 下拉式功能表中選擇您的授權方。

欲測試授權方，必須建立新的連線。變更 \$connect 內的授權方不會影響已經連線的用戶端。連線至您的 WebSocket API 時，必須提供已設定身分來源的值。例如，您可傳送有效查詢字串，標頭使用 wscat，藉此進行連線，如下列範例所示：

```
wscat -c 'wss://myapi.execute-api.us-east-1.amazonaws.com/beta?QueryAuth1=queryValue1' -H HeaderAuth1:headerValue1
```

若您嘗試連線但不具備有效的身分值，將收到 401 回應：

```
wscat -c wss://myapi.execute-api.us-east-1.amazonaws.com/beta
error: Unexpected server response: 401
```

## 使用 CloudWatch 監控 WebSocket API 執行

您可以使用 Amazon CloudWatch 指標和日誌，以監控 WebSocket API 的執行。該組態與用於 REST API 的組態類似。

如需如何設定 CloudWatch 存取和執行記錄的指示，請參閱 [the section called “使用 API Gateway 主控台設定 API 記錄” \(p. 474\)](#)。

當您指定 Log Format (日誌格式)，您可以選擇要記錄哪些環境變數。此為 WebSocket API 環境變數之 JSON 格式清單的範例：

```
{  
    "apiId" : "$context.apiId",  
    "routeKey" : "$context.routeKey",  
    "authorizer" : "$context.authorizer",  
    "messageId" : "$context.messageId",  
    "integrationLatency" : "$context.integrationLatency",  
    "eventType" : "$context.eventType",  
    "error" : "$context.error",  
    "extendedRequestId" : "$context.extendedRequestId",  
    "requestTime" : "$context.requestTime",  
    "stage" : "$context.stage",  
    "connectedAt" : "$context.connectedAt",  
    "requestTimeEpoch" : "$context.requestTimeEpoch",  
    "requestId" : "$context.requestId",  
    "connectionId" : "$context.connectionId"  
}
```

您可以在這裡找到 WebSocket 特定的環境變數：[the section called “WebSocket 映射範本參考” \(p. 575\)](#)

WebSocket API 支援以下指標：

指標	說明
ConnectCount	傳送到 \$connect 路由整合的訊息數。
MessageCount	傳送到 WebSocket API 的訊息數 (來自用戶端或傳送到用戶端)。
IntegrationError	從整合傳回 4XX/5XX 回應的請求數。
ClientError	擁有在整合受到叫用前由 API Gateway 傳回的 4XX 回應之請求數。
ExecutionError	呼叫整合時發生的錯誤。
IntegrationLatency	在 API Gateway 將請求傳送至整合和 API Gateway 從整合接收回應的時間差異。受回呼和偽裝整合抑制。

您可以使用下表中的維度來篩選 API Gateway 指標。

維度	敘述
Apild	篩選所指定 API ID 之 API 的 API Gateway 指標。
Apild、階段	篩選所指定 API ID 與階段 ID 之 API 階段的 API Gateway 指標。

維度	敘述
ApId、階段、路由	<p>篩選所指定 API ID 與路由 ID 之 API 方法的 API Gateway 指標。</p> <p>除非您已明確啟用詳細 CloudWatch 指標，否則 API Gateway 將不會傳送這類指標。您可以透過呼叫 API Gateway V2 REST API 的 UpdateStage 動作，來將 metricsEnabled 屬性更新為「true」。啟用這類指標會產生您帳戶的額外費用。如需定價資訊，請參閱 <a href="#">Amazon CloudWatch 定價</a>。</p>

## API Gateway 內的 WebSocket 選擇表達式

### 主題

- [路由選擇表達式 \(p. 570\)](#)
- [模型選擇表達式 \(p. 571\)](#)
- [範本選擇表達式 \(p. 571\)](#)
- [路由回應選擇表達式 \(p. 572\)](#)
- [API 金鑰選擇表達式 \(p. 572\)](#)
- [API 映射選擇表達式 \(p. 572\)](#)
- [整合回應選擇表達式 \(p. 572\)](#)
- [WebSocket 選擇表達式總結 \(p. 572\)](#)

API Gateway 使用選擇表達式來評估請求及回應內容並產生金鑰。然後，金鑰會用來選擇一組通常由您 (即 API 開發人員) 提供的可能值。所支援的實際變數組將取決於特定表達式，各表達式詳細說明如下。

所有表達式的語言都遵循同一組規則：

- 變數字首會加上 "\$"。
- 大括號可用來明確定義變數邊界，例如 "\${request.body.version}-beta"。
- 支援多個變數，但僅會評估一次 (無遞迴評估)。
- 貨幣符號 (\$) 可用 "\\" 逸出。在定義映射至預留 \$default 金鑰 (如 "\\$default") 的表達式時，這條規則十分實用。
- 有時需要模式格式，此時，表達式前後應以斜線 (" / ") 包裝，例如應符合 2XX 狀態碼的 "/2\d\d/"。

## 路由選擇表達式

本服務針對傳入訊息選擇欲遵循的路由時，將評估路由選擇表達式。本服務將使用 routeKey 與評估值完全相符的路由。若無相符路由，且某路由存在 \$default 路由金鑰，將選取該路由。若沒有路由符合評估值，而且沒有 \$default 路由，本服務將回傳錯誤。以 WebSocket 型的 API 而言，表達式的形式應為 \$request.body.{path\_to\_body\_element}。

例如，假設您正傳送下列 JSON 訊息：

```
{  
    "service" : "chat",
```

```

    "action" : "join",
    "data" : {
        "room" : "room1234"
    }
}

```

您可能想要根據 `action` 屬性來選取您的 API 行為。此時，您可以定義下列路由選擇表達式：

```
$request.body.action
```

此範例中，`request.body` 係指您訊息的 JSON 承載，而 `.action` 則為 [JSONPath 表達式](#)。`request.body` 之後可使用任何 JSON 路徑表達式，但請記住結果會字串化。例如，若您的 JSONPath 表達式回傳兩個元素的陣列，該結果將以字串 "[item1, item2]" 呈現。有鑑於此，理想做法是將表達式的評估結果設為一個值，而非陣列或物件。

您可僅使用靜態值，或者也可以使用多個變數。下表為上述承載的範例及其評估結果。

表達式	評估結果	說明
<code>\$request.body.action</code>	未包裝的變數	
<code> \${request.body.action}</code>	已包裝的變數	
<code> \${request.body.service}/join</code> <code> \${request.body.action}</code>	具備靜態值的多個變數	
<code> \${request.body.action}-</code> <code> \${request.body.invalidPath}</code>	若未找到 JSONPath，則變數將解析為 ""。	
<code>action</code>	靜態值	
<code>\\$default</code>	靜態值	

評估結果將用於尋找路由。若某路由具備相符的路由金鑰，將選取該路由來處理訊息。若找不到相符的路由，則 API Gateway 將嘗試尋找 `$default` 路由（如有）。若未定義 `$default` 路由，則 API Gateway 將回傳錯誤。

## 模型選擇表達式

定義 WebSocket API 的 [路由 \(p. 557\)](#) 時，您可選擇指定模型選擇表達式。評估此表達式後，即可選取接收請求時將用於內文驗證的模型。此表達式的判斷值為路由的 `requestmodels` 其中一個項目。

模型是以 [JSON 結構描述](#) 來表示，並描述請求本文的資料結構。此選擇表達式本身讓您能夠動態選擇欲用來在執行時間驗證特定路由的模型。如需建立模型的資訊，請參閱 [the section called “建立模型與對應範本” \(p. 244\)](#)。

## 範本選擇表達式

定義 WebSocket API 的 [整合請求 \(p. 558\)](#) 或 [整合回應 \(p. 560\)](#) 時，您可選擇指定範本選擇表達式。此表達式的評估結果會判定輸入或輸出範本（如有），輸入範本可將請求本文轉換為整合請求本文，輸出範本則可將整合回應本文轉換為路由回應本文。

`Integration.TemplateSelectionExpression` 支援  `${request.body.jsonPath}` 和靜態值。

`IntegrationResponse.TemplateSelectionExpression` 支援

`${request.body.jsonPath}`、 `${integration.response.statusCode}`、 `${integration.response.headers}` 和靜態值。

## 路由回應選擇表達式

路由回應 (p. 561) 係用來將後端到用戶端的回應建模。以 WebSocket API 而言，路由回應為選用。如有定義，將指示 API Gateway 在接收 WebSocket 訊息時應回傳回應至用戶端。

路由回應選擇表達式的評估將產生路由回應金鑰。此金鑰最終將選擇其中一個與 API 相關聯的 `RouteResponses`。然而，目前僅支援 `$default` 金鑰。

## API 金鑰選擇表達式

本服務判定特定請求只在用戶端提供有效的 API 金鑰 (p. 5) 後才會繼續處理，此時將評估此表達式。

目前僅支援兩個值： `${request.header.x-api-key}` 及  
 `${context.authorizer.usageIdentifierKey}`。

## API 映射選擇表達式

當請求使用自訂網域提出時，須評估此表達式來判定應選取的 API 階段。

目前，僅支援的值為  `${request.basepath}`。

## 整合回應選擇表達式

設定 WebSocket API 的整合回應 (p. 560) 時，您可選擇指定整合回應選擇表達式。此表達式會判定整合回傳時應選取的 `IntegrationResponse`。此表達式的值目前正受 API Gateway 限制，規則如下。請注意，此表達式只與非代理整合有關；代理整合只要將回應承載回傳給發起人即可，無須建模或修改。

此表達式與上述選擇表達式不同，目前支援模式比對格式。此表達式應以斜線包裝。

目前固定的值視 `integrationType` 而異：

- 若是 Lambda 型整合，此值為  `${integration.response.body.errorMessage}`。
- 若是 HTTP 及 MOCK 整合，此值為  `${integration.response.statusCode}`。
- 若是 HTTP\_PROXY 及 AWS\_PROXY，將不會運用此表達式，因為您請求將承載傳遞給發起人。

## WebSocket 選擇表達式總結

下表總結 WebSocket API 內選擇表達式的使用案例：

選擇表達式	判斷值為下列的金鑰	備註	範例使用案例
<code>Api.RouteSelectionKey</code>		所支援的 <code> \$default</code> 為全部截獲路由。	依據用戶端請求內容路由 WebSocket 訊息。

選擇表達式	判斷值為下列的金鑰	備註	範例使用案例
Route.ModelsSelectedFromRequestModels 的金鑰	選用。 若提供 給非 代理 整合， 將出現 模型驗 證。  所支 援的 \$default 為全部 截獲。	在相同 路由內 動態執 行請求 驗證。	
Integration.TemplateIntegrationRequestTemplates 的金鑰	選用。 可供非 代理 整合使 用，藉 此操控 傳入承 載。  支援 \${request.body.jsonPath} 和靜態 值。  所支 援的 \$default 為全部 截獲。	依據請 求的動 態屬性 操控發 起人的 請求。	

選擇表達式	判斷值為下列的金鑰	備註	範例使用案例
	<code>Integration.IntegrationResponseSelectIntegrateResponseKey</code>	選用。 可供 非代理 整合使 用。  做為錯 誤訊息 (來自 Lambda 或狀態 碼 (來自 HTTP 整合) 的 模式比 對。  非代理 整合必 須有 <code>\$default</code> 來全 部截獲 成功回 應。	從後端 操控回 應。  依據後 端的動 態回應 來選擇 欲採取 的動作 (如明 確處理 特定錯 誤)。
	<code>IntegrationResponseTemplateSelectorResponseTemplates</code> 的金鑰	選用。 可供 非代理 整合使 用。  支援 <code>\$default</code> 。	回應的 動態屬 性有時 會在相 同路由 與相關 聯整合 內做出 不同轉 換的決 定。  支援 <code> \${request.body.js}</code> 和靜態 值。  所支 援的 <code>\$default</code> 為全部 截獲。

選擇表達式	判斷值為下列的金鑰	備註	範例使用案例
Route.RouteResponse\$Response\$ResponseKey		應提供，藉以啟動 WebSocket 路由的雙向通訊。 目前此值限制為 \$default。	
RouteResponse.Models\$Response\$Request\$Models	的金鑰	目前不支援。	

## API Gateway WebSocket API 映射範本參考

此章節摘要說明目前 API Gateway 內 WebSocket API 支援的變數組。

參數	描述
\$context.connectionId	連線的唯一 ID，可用來回呼用戶端。
\$context.connectedAt	Epoch 格式化連線時間。
\$context.domainName	WebSocket API 的網域名稱。此可用於回呼用戶端 (非硬式編碼的值)。
\$context.eventType	事件類型：CONNECT、MESSAGE 或 DISCONNECT。
\$context.messageId	訊息的伺服器端唯一 ID。僅在 \$context.eventType 為 MESSAGE 時可用。
\$context.routeKey	所選路由金鑰。
\$context.requestId	與 \$context.extendedRequestId 相同。
\$context.extendedRequestId	API 呼叫的自動產生的 ID，其中包含更多除錯/故障排除的有用的資訊。
\$context.apiId	API Gateway 指派給您 API 的識別符。
\$context.authorizer.principalId	與用戶端所傳送並從 API Gateway Lambda 授權方 (先前稱作自訂授權方) Lambda 函數所傳回之字符建立關聯的主要使用者身分。
\$context.authorizer.property	API Gateway Lambda 授權方函數所傳回 context 映射之指定索引鍵/值對的字串化值。例如，如果授權方傳回下列 context 映射：
	<pre>"context" : {     "key": "value",</pre>

參數	描述
	<pre>"numKey": 1, "boolKey": true }</pre>
	呼叫 \$context.authorizer.key 會傳回 "value" 字串、呼叫 \$context.authorizer.numKey 會傳回 "1" 字串，而呼叫 \$context.authorizer.boolKey 會傳回 "true" 字串。
\$context.error.message	包含 API Gateway 錯誤訊息的字串。此變數只能用於存取記錄。
\$context.error.messageString	\$context.error.message 的引用值，即 "\$context.error.message"。
\$context.error.responseType	錯誤回應類型。此變數只能用於存取記錄。
\$context.error.validationErrorString	字串，其中包含詳細的驗證錯誤訊息。
\$context.identity.accountId	與請求相關聯的 AWS 帳戶 ID。
\$context.identity.apiKey	與啟用金鑰之 API 請求建立關聯的 API 擁有者金鑰。
\$context.identity.apiKeyId	與啟用金鑰之 API 請求建立關聯的 API 金鑰 ID。
\$context.identity.caller	提出請求之發起人的委託人識別符。
\$context.identity.cognitoAuthenticationProvider	提出請求之發起人所使用的 Amazon Cognito 身分驗證提供者。只有在使用 Amazon Cognito 登入資料簽署請求時才能使用。  如需這個和其他 Amazon Cognito \$context 變數的詳細資訊，請參閱 Amazon Cognito 開發人員指南中的 <a href="#">使用聯合身分</a> 。
\$context.identity.sourceIp	對 API Gateway 提出請求之 TCP 連線的來源 IP 地址。
\$context.identity.user	提出請求之使用者的委託人識別符。
\$context.identity.userAgent	API 發起人的使用者代理程式。
\$context.identity.userArn	身分驗證之後識別之有效使用者的 Amazon Resource Name (ARN)。
\$context.integrationLatency	毫秒的整合延遲僅可用於存取記錄。
\$context.requestTime	CLF 格式化請求時間 (dd/MMM/yyyy:HH:mm:ss +-hhmm)。
\$context.requestTimeEpoch	Epoch 格式化請求時間。
\$context.stage	API 呼叫的部署階段 (例如，Beta 或 Prod)。
\$input.body	傳回原始承載作為字串。

參數	描述
<code>\$input.json(x)</code>	<p>此函數會評估 JSONPath 表達式，並傳回結果作為 JSON 字串。</p> <p>例如，<code>\$input.json('\$.pets')</code> 會傳回代表 pets 結構的 JSON 字串。</p> <p>如需 JSONPath 的詳細資訊，請參閱 <a href="#">JSONPath</a> 或 <a href="#">適用於 Java 的 JSONPath</a>。</p>
<code>\$input.path(x)</code>	<p>採用 JSONPath 表達式字串 (x)，並傳回結果的 JSON 物件呈現。這可讓您存取和運用 <a href="#">Apache Velocity 範本語言 (VTL)</a> 中原生承載的元素。</p> <p>例如，如果表達式 <code>\$input.path('\$.pets')</code> 傳回如下物件：</p> <pre>[   {     "id": 1,     "type": "dog",     "price": 249.99   },   {     "id": 2,     "type": "cat",     "price": 124.99   },   {     "id": 3,     "type": "fish",     "price": 0.99   }] ]</pre> <p><code>\$input.path('\$.pets').count()</code> 會傳回 "3"。</p> <p>如需 JSONPath 的詳細資訊，請參閱 <a href="#">JSONPath</a> 或 <a href="#">適用於 Java 的 JSONPath</a>。</p>
<code>\$stageVariables.&lt;variable_name&gt;</code>	<code>&lt;variable_name&gt;</code> 代表階段變數名稱。
<code>\$stageVariables['&lt;variable_name&gt;']</code>	<code>&lt;variable_name&gt;</code> 代表任何階段變數名稱。
<code> \${stageVariables['&lt;variable_name&gt;']}</code>	<code>&lt;variable_name&gt;</code> 代表任何階段變數名稱。

參數	描述
<code>\$util.escapeJavaScript()</code>	<p>使用 JavaScript 字串規則來逸出字串中的字元。</p> <p><b>Note</b></p> <p>此函數會將任何一般單引號 ('') 轉換為逸出單引號 (\')。不過，逸出單引號不適用於 JSON。因此，將此函數的輸出用於 JSON 屬性時，您必須將任何逸出單引號 (\') 轉換為一般單引號 ('')。下列範例顯示這種情況：</p> <pre>\$util.escapeJavaScript(<b>data</b>).replaceAll("\\", "'")</pre>
<code>\$util.parseJson()</code>	<p>採用「字串化」JSON，並傳回結果的物件呈現。您可以使用此函數的結果，來存取和運用 Apache Velocity 範本語言 (VTL) 中原生承載的元素。例如，如果您有下列承載：</p> <pre>{"errorMessage": "{\"key1\": \"var1\", \"key2\": {\"arr\": [1, 2, 3]}}"}</pre> <p>並使用下列映射範本</p> <pre>#set (\$errorMessageObj = \$util.parseJson(\$input.path('\$.errorMessage'))) {     "errorMessageObjKey2ArrVal" : \$errorMessageObj.key2.arr[0] }</pre> <p>您將會收到下列輸出：</p> <pre>{     "errorMessageObjKey2ArrVal" : 1 }</pre>
<code>\$util.urlEncode()</code>	將字串轉換為 "application/x-www-form-urlencoded" 格式。
<code>\$util.urlDecode()</code>	解碼 "application/x-www-form-urlencoded" 字串。
<code>\$util.base64Encode()</code>	將資料編碼為 base64 編碼字串。
<code>\$util.base64Decode()</code>	解碼 base64 編碼字串中的資料。

# 發佈您的 API Gateway API

一旦部署了您的 API，您就可以部署開發人員入口網站來發佈它們，而且可以透過 AWS Marketplace 將您的 API 當作 SaaS 來販售。您也可以針對 API 建立自訂網域名稱。您可以匯出 API 定義，使用支援的程式設計語言產生軟體開發套件，以供使用者呼叫 API。您也可以使用 [Canary Release](#) 來測試新的變更。

## 主題

- [使用無伺服器開發人員入口網站將您的 API Gateway API 編目 \(p. 579\)](#)
- [透過 AWS Marketplace 銷售 API Gateway API \(p. 587\)](#)
- [在 API Gateway 中設定 API 的自訂網域名稱 \(p. 590\)](#)
- [從 API Gateway 匯出 REST API \(p. 604\)](#)
- [設定 API Gateway Canary Release 部署 \(p. 607\)](#)

## 使用無伺服器開發人員入口網站將您的 API Gateway API 編目

開發人員入口網站是您用來讓 API 可供客戶使用的應用程式。無伺服器開發人員入口網站可以用來直接從 API Gateway 發佈 API Gateway 受管 API。您也可以使用它，為非 API Gateway 受管 API 上傳 OpenAPI 定義來發佈這些受管 API。當您 在無伺服器開發人員入口網站中發佈 API 時，您的客戶可以輕鬆執行下列動作：

- 探索可用的 API。
- 瀏覽您的 API 文件。
- 註冊以取得他們可以用來建置應用程式的專屬 API 金鑰，而且可以立即收到此金鑰。
- 在開發人員入口網站 UI 中試試您的 API。
- 監控自己的 API 用量。

Amazon API Gateway 在 [AWS Serverless Application Repository](#) 和 [GitHub 上](#) 發佈定期更新的無伺服器 開發人員入口網站。係依 [Apache 2.0 授權](#) 發佈，讓您自訂和納入到您的建組和部署工具。前端是以 React 撰寫，並設計為可完全自訂。請參閱 <https://github.com/awslabs/aws-api-gateway-developer-portal/wiki/Customization>。

如需 AWS Serverless Application Repository 的詳細資訊，請參閱 [AWS Serverless Application Repository Developer Guide](#)。

### Tip

如果您想要試用範例無伺服器開發人員入口網站，請參閱 <https://developer.exampleapigateway.com/>。

## 主題

- [建立開發人員入口網站 \(p. 580\)](#)
- [開發人員入口網站設定 \(p. 581\)](#)
- [為開發人員入口網站建立管理員使用者 \(p. 582\)](#)
- [將 API Gateway 受管 API 發佈到您的開發人員入口網站 \(p. 583\)](#)
- [更新或刪除 API Gateway 受管 API \(p. 584\)](#)

- 移除非 API Gateway 受管 API (p. 584)
- 將非 API Gateway 受管 API 發佈到您的開發人員入口網站 (p. 584)
- 您的客戶如何使用您的開發人員入口網站 (p. 585)
- 開發人員入口網站的最佳實務 (p. 586)

## 建立開發人員入口網站

您可以依原狀部署 API Gateway 開發人員入口網站，或依品牌需求進行自訂。有兩種方式來部署您的開發人員入口網站：

- 使用 AWS Serverless Application Repository。只需選擇 Deploy (部署) 按鈕，即可啟動 API Gateway 無伺服器開發人員入口網站 AWS CloudFormation 堆疊，然後在 Lambda 主控台中輸入一些堆疊參數。
- 從 AWS GitHub 儲存庫下載 API Gateway 無伺服器開發人員入口網站，並從 AWS SAM CLI 啟動 API Gateway 無伺服器開發人員入口網站 AWS CloudFormation 堆疊。

### 使用 AWS Serverless Application Repository部署無伺服器開發人員入口網站

1. 前往 AWS Serverless Application Repository 中的 [API Gateway 無伺服器開發人員入口網站](#)。
2. 選擇 Deploy (部署)。

#### Note

如果出現系統提示，請登入 AWS Lambda。

3. 在 Lambda 主控台中，您將需要在 Application settings (應用程式設定) 下填寫必要的 [開發人員入口網站堆疊參數 \(p. 581\)](#)。

#### Note

S3 儲存貯體名稱和 Amazon Cognito 網域字首是必要項目；其他設定則為選用項目。

4. 如果您想要啟用 Got an opinion? (取得意見？) 客戶意見回饋按鈕，將需要：
  - 在 DevPortalAdminEmail 方塊中輸入電子郵件地址。當客戶提交意見回饋時，電子郵件將會傳送到這個地址。

#### Note

如果您未提供電子郵件地址，Got an opinion? (取得意見？) 不會出現在開發人員入口網站中。

- 您可以選擇在 DevPortalFeedbackTableName 方塊中輸入資料表名稱。默認名稱為 **DevPortalFeedback**。當客戶提交意見回饋時，它會以此名稱存放在 DynamoDB 資料表中。
5. 選擇 I acknowledge that this app creates custom IAM roles (我認可此應用程式建立自訂的 IAM 角色) 旁的核取方塊。
  6. 選擇 Deploy (部署)。
  7. 如果您在 AdminEmail 堆疊參數中輸入電子郵件地址，Amazon SNS 訂閱電子郵件會傳送到該電子郵件地址，以確認您要訂閱已在 MarketplaceSubscriptionTopicProductCode 設定中指定的 Amazon SNS 主題。

### 使用 AWS SAM下載和部署無伺服器開發人員入口網站

1. 請確定您已安裝並設妥最新 [AWS CLI](#) 和 [AWS SAM CLI](#)。
2. 下載或複製[API Gateway無伺服器開發人員入口網站](#)的儲存庫到本機目錄。
3. 確定您具有不可公開存取的 Amazon S3 儲存貯體，以將壓縮的 Lambda 函數上傳至其中。如果還沒有日誌群組，您可以使用Amazon S3主控台或 CLI 建立一個。

在 SAM CLI 中，從專案根目錄中執行以下命令，將 `{your-lambda-artifacts-bucket-name}` 取代為 Amazon S3 儲存貯體的名稱：

```
sam package --template-file ./cloudformation/template.yaml --output-template-file ./cloudformation/packaged.yaml --s3-bucket {your-lambda-artifacts-bucket-name}
```

- 對於這個步驟，如需後接 `--parameter-overrides` 參數 (例如 `CognitoDomainNameOrPrefix`) 的詳細資訊，請參閱[the section called “開發人員入口網站設定” \(p. 581\)](#)。

Note

確定您具有不可公開存取的 Amazon S3 儲存貯體，以將 SAM 範本上傳至其中。(AWS CloudFormation 會從儲存貯體讀取範本，以部署開發人員入口網站。) 此儲存貯體可以是您在先前步驟中為了上傳壓縮 Lambda 函數而指定的同一個儲存貯體。

從專案根目錄執行下列命令，將：

- `{your-template-bucket-name}` 取代為 Amazon S3 儲存貯體的名稱
- `{custom-prefix}` 取代為全域唯一的字首
- `{cognito-domain-or-prefix}` 取代為唯一字串。

```
sam deploy --template-file ./cloudformation/packaged.yaml --s3-bucket {your-template-bucket-name} --stack-name "{custom-prefix}-dev-portal" --capabilities CAPABILITY_NAMED_IAM --parameter-overrides CognitoDomainNameOrPrefix= "{cognito-domain-or-prefix}" DevPortalSiteS3BucketName="{custom-prefix}-dev-portal-static-assets" ArtifactsS3BucketName="{custom-prefix}-dev-portal-artifacts"
```

一旦您的開發人員入口網站已完全部署，您可以取得其 URL，如下所示。

若要為您新建立的開發人員入口網站取得 URL

1. 開啟 AWS CloudFormation 管理主控台。
2. 選擇堆疊的名稱 (aws-serverless-repository-api-gateway-dev-portal 為預設堆疊名稱)。
3. 開啟「Outputs」一節。在 WebSiteURL 屬性中指定的開發人員入口網站 URL。

## 開發人員入口網站設定

以下是您在部署開發人員入口網站期間和之後需要設定的設定：

Note

在部署了您的開發人員入口網站之後，可以更新 AWS CloudFormation 堆疊來變更底下的多個設定。如需詳細資訊，請參閱[AWS CloudFormation 堆疊更新](#)。

### ArtifactsS3BucketName (必要)

部署程序會建立不可公開存取的 Amazon S3 儲存貯體，其中將存放目錄中繼資料。指定要指派給儲存貯體的名稱。此名稱必須是全域唯一的。

### CognitoDomainNameOrPrefix (必要)

此字串會與 Amazon Cognito 託管的 UI 搭配使用，用於使用者註冊和登入。指定唯一的網域名稱或字首字串。

### CognitoIdentityPoolName

部署程序會建立 Amazon Cognito 身分集區。身分集區的預設名稱是 DevPortalUserPool。

#### CustomDomainNameAcmCertArn

如果已提供與 ACM 憑證相關聯的網域名稱，您亦須在這裡指定 ACM �凭證的 ARN。將此欄位保留空白，來建立沒有自訂網域名稱的開發人員入口網站。

#### DevPortalAdminEmail

指定電子郵件地址，以啟用 Got an opinion? (取得意見？) 客戶意見回饋按鈕。當客戶提交意見回饋時，電子郵件將會傳送到這個地址。

##### Note

如果您未提供電子郵件地址，Got an opinion? (取得意見？) 按鈕不會出現在開發人員入口網站中。

#### DevPortalFeedbackTableName

如果您為 DevPortalAdminEmail 指定電子郵件地址，則部署程序會建立 DynamoDB 資料表，以存放客戶所輸入的意見回饋。默认名称为 **DevPortalFeedback**。您可以選擇性地指定自己的資料表名稱。

#### DevPortalCustomersTableName

部署程序會建立一個 DynamoDB 資料表，其中將存放客戶帳戶。您可以選擇性地指定要指派給此資料表的名稱。在您的帳戶區域內，這個名稱必須是唯一的。資料表的預設名稱為 DevPortalCustomers。

#### DevPortalSiteS3BucketName (必要)

部署程序會建立一個不可公開存取的 Amazon S3 儲存貯體，其中將存放 Web 應用程式碼。指定要指派給此儲存貯體的名稱。此名稱必須是全域唯一的。

#### MarketplaceSubscriptionTopicProductCode

這是訂閱/取消訂閱事件的 Amazon SNS 主題尾碼。輸入所需值。預設值為 **DevPortalMarketplaceSubscriptionTopic**。只在您使用 AWS Marketplace 整合時，此設定才相關。如需詳細資訊，請參閱[將您的 SaaS 產品設定為接受新客戶](#)。

#### StaticAssetRebuildMode

根據預設，靜態資產重建不會覆寫自訂內容。指定 **overwrite-content**，將自訂內容取代為您的本機版本。

##### Important

如果指定 **overwrite-content**，則會失去 Amazon S3 儲存貯體中的所有自訂變更。除非您知道這麼做的用意和後果，否則請不要執行此動作。

#### StaticAssetRebuildToken

提供與上次部署的字符不同的字符，以重新上傳開發人員入口網站的靜態資產。您可以提供每次部署的時間戳記或 GUID，以一律重新上傳資產。

#### UseRoute53Nameservers

只在您為開發人員入口網站建立自訂網域名稱時才適用。指定 **true**，以略過建立 Route 53 HostedZone 和 RecordSet。您將需要提供自己的名稱伺服器託管，以替代 Route 53。否則，將此欄位設定為 **false**。

## 為開發人員入口網站建立管理員使用者

一旦部署了您的開發人員入口網站，您就會想要建立至少一個管理員使用者。做法為建立新使用者，並將該使用者新增至 Amazon Cognito 使用者集區的管理員群組，而此集區是在您部署開發人員入口網站時 AWS CloudFormation 建立的。

### 建立管理員使用者

1. 在您的開發人員入口網站中，選擇 Register (註冊)。

2. 輸入電子郵件地址和密碼，然後選擇 Register (註冊)。
3. 在個別瀏覽器索引標籤中，登入 Amazon Cognito 主控台。
4. 選擇 Manage User Pools (管理使用者集區)。
5. 為您在部署開發人員入口網站時設定的開發人員入口網站選擇使用者集區。
6. 選擇要提升為管理員的使用者。
7. 選擇 Add to group (新增至群組)。
8. 從下拉式功能表中，選擇 `{stackname}-dev-portalAdminsGroup`，其中 `{stackname}` 是您部署開發人員入口網站時的堆疊名稱。
9. 選擇 Add to group (新增至群組)。
10. 在您的開發人員入口網站中，登出並使用相同的登入資料重新登入。您現在應該會在右上角 My Dashboard (我的儀表板) 旁邊看到 Admin Panel (管理面板) 連結。

## 將 API Gateway 受管 API 發佈到您的開發人員入口網站

下列步驟概述如何以 API 擁有者身分將 API 發佈到開發人員入口網站，讓客戶可以進行訂閱。

### 步驟 1：建立可用於發佈的 API Gateway 受管 API

1. 如果您尚未這樣做，請[將 API 部署至階段 \(p. 459\)](#)。
2. 如果您尚未這樣做，請為部署的 API 建立[用量計劃 \(p. 397\)](#)，並將它與 API 階段建立關聯。

#### Note

您不需要將 API 金鑰與用量計劃建立關聯。開發人員入口網站會為您執行此操作。

3. 登入 Developer Portal (開發人員入口網站)，並移至 Admin Panel (管理面板)。
4. 您應該會在 Admin Panel (管理面板) API 清單中看到 API。在 API 清單中，API 會依用量計劃分組。Not Subscribable (無法訂閱) No Usage Plan (無用量計劃) 群組會列出未與用量計劃相關聯的 API。

### 步驟 2：讓 API Gateway 受管 API 可在開發人員入口網站中顯示

1. 在 Admin Panel (管理面板) API 清單中，檢查 API 的 Displayed (顯示) 值。第一次上傳 API 時，此值會設定為 False。
2. 若要讓 API 可在開發人員入口網站中顯示，請選擇 False 按鈕。它的值會變更為 True，表示現在顯示 API。

#### Tip

若要顯示用量計劃中的所有 API，請在用量計劃的標頭列中選擇 False 或 Partial (部分) 按鈕。

3. 導覽至 APIs 面板，以便您的客戶看到開發人員入口網站時加以查看。您應該會看到 API 列示在左側導覽欄中。

如果 API 部署到與用量計劃相關聯的階段，您會看到 API 的 Subscribe (訂閱) 按鈕。這個按鈕會導致客戶的 API 金鑰與用量計劃相關聯，而 API 也與此用量計劃相關聯。

既然顯示 API Gateway受管 API，您可以啟用軟體開發套件產生，讓您的客戶可以為它下載軟體開發套件。

### 步驟 3：啟用軟體開發套件產生

1. 在 Admin Panel (管理面板) API 清單中，選擇 Disabled (已停用) 按鈕。其值會變更為 Enabled (已啟用)，表示現在容許軟體開發套件產生。

2. 導覽至 APIs 面板，然後在左側導覽欄中選擇您的 API。您現在應該會看到它的 Download SDK (下載軟體開發套件) 按鈕。

如果選擇 Download SDK (下載軟體開發套件) 按鈕，您應該會看到一個清單，列出可以下載的軟體開發套件類型。

## 更新或刪除 API Gateway 受管 API

如果您在發佈了 API Gateway 中的 API 之後對其進行變更，則需要重新部署它。

### 更新 API Gateway 受管 API

1. 在 API Gateway 主控台、AWS CLI 或軟體開發套件中，對 API 進行所需的變更。
2. 將 API 重新部署到之前的同一個階段。
3. 在開發人員入口網站的 Admin Panel (管理面板) API 清單中，選擇 Update (更新)。這會更新開發人員入口網站 UI 中顯示的 API。

#### Note

Download SDK (下載軟體開發套件) 按鈕會一律取得 API 的最新部署版本，即使您未在開發人員入口網站中進行更新也一樣。

4. 導覽至 APIs 面板，然後在左側導覽欄中選擇您的 API。您應該會看到您的變更。

若要在開發人員入口網站 API 清單中停止顯示 API (無需撤銷客戶對它的存取)：

1. 在 Admin Panel (管理面板) API 清單中，檢查 API 的 Displayed (顯示) 值。如果顯示 API，此值會設定為 True。
2. 若要不顯示 API，請選擇 True 按鈕。其值會變更為 False，表示現在不顯示 API。
3. 導覽至 APIs 面板。您應該再也看不到 API 列示在左側導覽欄中。

若要撤銷客戶對 API 的存取，而不將其完整刪除，您必須在 [API Gateway 主控台 \(p. 403\)](#) 或 [API Gateway CLI 或 REST API \(p. 408\)](#) 中執行以下其中一項動作：

- 從用量計畫中移除 API 金鑰。
- 從用量計畫中移除 API 階段。

## 移除非 API Gateway 受管 API

若要停止顯示非 API Gateway 受管 API，並移除客戶對它的存取，請選擇 Delete (刪除)。這不會刪除 API，但會從開發人員入口網站中刪除其 OpenAPI 規格檔案。

## 將非 API Gateway 受管 API 發佈到您的開發人員入口網站

下列步驟概述如何將非 API Gateway 受管 (或「一般」) API 發佈到您的客戶。您可以使用 JSON、YAML 或 YAML 檔案，上傳 API 的 OpenAPI 2.0 (Swagger) 或 3.x 定義。

### 在您的開發人員入口網站中發佈非 API Gateway 受管 API

1. 登入 Developer Portal (開發人員入口網站)，並移至 Admin Panel (管理面板)。
2. 在 Generic APIs (一般 API) 下，選擇 Add API (新增 API)。

3. 瀏覽至您 API 的 OpenAPI 檔案所在目錄，然後選擇要上傳的檔案。
4. 選擇 Upload (上傳)。

Note

若有任何檔案無法進行剖析或未包含 API 標題，則您會得到一個警告，而且系統不會上傳那些檔案。將會上傳所有其他檔案。您將需要選擇 x 圖示以關閉對話方塊。

5. 您現在應該會看到 API 列示在 Generic APIs (一般 API) 之下。如果沒有看到，請離開 Admin Panel (管理面板)，然後再次導覽回到其中，以重新整理清單。

## 您的客戶如何使用您的開發人員入口網站

若要建置應用程式和測試您的 API，您的客戶將需要向您的開發人員入口網站註冊，以建立開發人員帳戶。

### 若要建立開發人員帳戶並獲得 API 金鑰

1. 在開發人員入口網站，選擇 Register (註冊)。
2. 輸入電子郵件地址和密碼，然後選擇 Register (註冊)。
3. 若要尋找 API 金鑰，選擇 My Dashboard (我的儀表板)。

開發人員帳戶可給您的客戶 API 金鑰，這通常需要使用和測試您的 API，允許您與客戶用量追蹤。

當客戶先註冊，他們的新 API 金鑰不會與任何 API 建立關聯。

### 若要啟用 API 的 API 金鑰

1. 選擇 API。
2. 從 API 清單中選擇 API，然後選擇 Subscribe (訂閱)。

這會導致 API 金鑰與用量計劃相關聯，而 API 也與此用量計劃相關聯。

客戶現在已訂閱 API，而且他們可對 API 上的方法進行呼叫。每日 API 用量統計資料將會顯示在 MyDashboard (的儀表板) 上。

無需訂閱，客戶即可在開發人員入口網站 UI 中試用 API 方法。

Note

若有任何 API 方法需要 API 金鑰，則 Authorize (授權) 按鈕將出現在方法清單上方。需要 API 金鑰的方法將具有黑色鎖圖示。若要試用需要 API 金鑰的方法，請選擇 Authorize (授權) 按鈕，然後輸入與 API 階段之用量計劃相關聯的有效 API 金鑰。

當 Authorize (授權) 按鈕出現在您已訂閱的 API 上時，您可以放心地忽略此按鈕。

### 在未訂閱的情況下試用 API

1. 導覽至 API 清單中的 API。
2. 選擇 API 的擴展方法。
3. 選擇 Try it out (試用它)。
4. 選擇 Execute (執行)。

您的客戶可以如下所示，將客戶意見回饋提交給您：

### 提交 API 的意見回饋

1. 選擇 Got an opinion? (取得意見？)。

2. 在彈出式視窗中，輸入評論。
3. 選擇 Submit (提交)。

客戶的意見回饋會存放在開發人員入口網站的 DynamoDB 資料表中。此資料表的預設名稱為 DevPortalFeedback。另外，電子郵件會傳送到已在 DevPortalAdminEmail 欄位中指定的電子郵件地址。在部署開發人員入口網站時，如果未指定任何電子郵件地址，則 Got an opinion? (取得意見？) 不會出現。

**Note**

如果變更此資料表的名稱，則您必須使用帳戶區域內的唯一名稱。

如果您已在 Admin Panel (管理面板) 中對 API 啟用軟體開發套件產生，則客戶可以為它下載軟體開發套件。

### 下載 API 的軟體開發套件

1. 為所需 API 選擇 Download SDK (下載軟體開發套件) 按鈕。
2. 選擇所需的軟體開發套件平台或語言，如下所示：

- 若為 Android：

1. 針對 Group ID (群組 ID)，輸入對應專案的唯一識別符。這會用於 pom.xml 檔案 (例如 com.mycompany)。
2. 針對 Invoker package (啟動程式套件)，輸入所產生用戶端類別的命名空間 (例如 com.mycompany.clientsdk)。
3. 針對 Artifact ID (成品 ID)，輸入已編譯 .jar 檔案的名稱 (不含版本)。這會用於 pom.xml 檔案 (例如 aws-apigateway-api-sdk)。
4. 針對 Artifact version (成品版本)，輸入所產生用戶端的成品版本號碼。這會用於 pom.xml 檔案，且應該採用 major.minor.patch 模式 (例如 1.0.0)。

- 若為 JavaScript，可立即下載軟體開發套件。

- 若為 iOS (Objective-C) 或 iOS (Swift)，請在 Prefix (字首) 方塊中輸入唯一字首。

字首的作用如下：如果指派 SIMPLE\_CALC 作為具有 Input、Output 與 Result 模型之 SimpleCalc (p. 487) 的開發套件字首，所產生的開發套件會包含 SIMPLE\_CALCSimpleCalcClient 類別以封裝 API，包括方法請求/回應。此外，所產生的開發套件會包含 SIMPLE\_CALCInput、SIMPLE\_CALCOutput 與 SIMPLE\_CALCResult 類別，分別代表輸入、輸出與結果，以表示請求輸入與回應輸出。如需詳細資訊，請參閱在 Objective-C 或 Swift 中使用 API Gateway 為 REST API 產生的 iOS 開發套件 (p. 508)。

- 若為 Java：

1. 針對 Service Name (服務名稱)，指定您開發套件的名稱。例如，SimpleCalcSdk。這會成為開發套件用戶端類別的名稱。此名稱會對應到 pom.xml 檔案中 <project> 下的 <name> 標籤，而此檔案位於軟體開發套件專案資料夾中。請勿包含連字號。
  2. 針對 Java Package Name (Java 套件名稱)，指定您開發套件的套件名稱。例如，examples.aws.apig.simpleCalc.sdk。此套件名稱會作為您開發套件程式庫的命名空間。請勿包含連字號。
- 若為 Ruby，針對 Service Name (服務名稱)，指定軟體開發套件的名稱。例如，SimpleCalc。這會用來產生您 API 的 Ruby Gem 命名空間。此名稱必須全部都是字母 (a-zA-Z)，不含任何其他特殊字元或數字。

## 開發人員入口網站的最佳實務

以下是在部署開發人員入口網站時建議遵循的最佳實務。

- 在大多數情況下，您只想要針對所有 API 部署一個開發人員入口網站。在某些情況下，您可能針對 API 的開發和生產版本選擇個別的開發人員入口網站。不建議針對生產 API 使用多個開發人員入口網站。

- 當建立用量計劃，並將其與階段建立關聯時，您不需要將任何 API 金鑰與用量計劃建立關聯。開發人員入口網站會為您執行此操作。
- 請注意，如果顯示指定用量計劃中的任何 API，則該特定用量計劃中的所有 API 都是可訂閱的，即使您尚未將它們顯示給客戶也一樣。

## 透過 AWS Marketplace 銷售 API Gateway API

建立、測試及部署 API 之後，您可以將 API 封裝在 API Gateway 用量計劃 (p. 397) 中，並將此計劃當做軟體即服務 (SaaS) 產品透過 AWS Marketplace 銷售。AWS Marketplace 會根據對用量計劃提出的請求數目，向訂閱您產品供應項目的 API 買方收取費用。

若要在 AWS Marketplace 上銷售您的 API，您必須設定銷售管道來整合 AWS Marketplace 與 API Gateway。一般來說，這包括在 AWS Marketplace 上列出您的產品、使用適當的政策設定 IAM 角色來允許 API Gateway 傳送用量指標至 AWS Marketplace、將 AWS Marketplace 產品與 API Gateway 用量計劃相關聯，以及將 AWS Marketplace 買方與 API Gateway API 金鑰相關聯。下列各節將會詳細討論。

為了讓您的客戶在 AWS Marketplace 上購買您的產品，您必須向 AWS Marketplace 註冊開發人員入口網站 (外部應用程式)。開發人員入口網站必須處理從 AWS Marketplace 主控台重新導向的訂閱請求。

如需範例開發人員入口網站應用程式，請參閱 [API Gateway 開發人員入口網站](#)。

如需在 AWS Marketplace 上將 API 當做 SaaS 產品銷售的詳細資訊，請參閱 [AWS Marketplace SaaS 訂閱 - 使用者指南](#)。

### 主題

- [初始化 AWS Marketplace 與 API Gateway 的整合 \(p. 587\)](#)
- [處理客戶的用量計劃訂閱 \(p. 588\)](#)

## 初始化 AWS Marketplace 與 API Gateway 的整合

下列作業適用於 AWS Marketplace 與 API Gateway 整合的一次性初始化，讓您可以將 API 當做 SaaS 產品銷售。

### 在 AWS Marketplace 上列出產品

若要將您的用量計劃列為 SaaS 產品，請透過 [AWS Marketplace](#) 提交產品載入表單。產品必須包含 `apigateway` 類型的維度 `requests`。此維度會定義每個請求的價格，且 API Gateway 會使用它來測量傳送至您 API 的請求。

### 建立計量角色

您可以使用下列執行政策與信任政策來建立名為 `ApiGatewayMarketplaceMeteringRole` 的 IAM 角色。此角色可讓 API Gateway 代替您傳送用量指標至 AWS Marketplace。

### 計量角色的執行政策

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Action": [  
                "aws-marketplace:BatchMeterUsage",  
                "aws-marketplace:ResolveCustomer"  
            ],  
            "Resource": "*"  
        }  
    ]  
}
```

```
        "Resource": "*",
        "Effect": "Allow"
    }
]
```

## 計量角色的信任關係政策

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

## 將用量計劃與 AWS Marketplace 產品相關聯

當您 在 AWS Marketplace 上列出產品時，您會收到 AWS Marketplace 產品代碼。若要整合 API Gateway 與 AWS Marketplace，請將您的用量計劃與 AWS Marketplace 產品代碼相關聯。您可以使用 API Gateway 主控台、API Gateway REST API、適用於 API Gateway 的 AWS CLI 或適用於 API Gateway 的 AWS 開發套件，將 API GatewayUsagePlan 的 [productCode](#) productCode 欄位設定為您的 AWS Marketplace 產品代碼來啟用此關聯。下列程式碼範例使用 API Gateway REST API：

```
PATCH /usageplans/USAGE_PLAN_ID
Host: apigateway.region.amazonaws.com
Authorization: ...

{
    "patchOperations" : [{
        "path" : "/productCode",
        "value" : "MARKEPLACE_PRODUCT_CODE",
        "op" : "replace"
    }]
}
```

## 處理客戶的用量計劃訂閱

下列作業是由開發人員入口網站應用程式所處理。

當客戶透過 AWS Marketplace 訂閱您的產品，AWS Marketplace 會轉送 POST 請求至 SaaS 訂閱 URL (也就是在 AWS Marketplace 上列出您的產品時所註冊的 URL)。POST 請求隨附 x-amzn-marketplace-token 標頭參數，其中包含買方資訊。請遵循[設定您的 SaaS 產品接受新客戶](#)的指示，處理開發人員入口網站應用程式中的這個重新導向。

AWS Marketplace回應客戶的訂閱請求，傳送 subscribe-success 通知到您可以訂閱的Amazon SNS 主題。(請參閱[\(Configuring Your SaaS Product to Accept New Customers\)](#) 設定您的 SaaS 產品接受新客戶)。若要接受客戶訂閱請求，您可以透過建立或擷取客戶的 API Gateway API 金鑰、將客戶之 AWS Marketplace 佈建的 customerId 與 API 金鑰相關聯，然後將 API 金鑰新增至您的用量計劃，來處理 subscribe-success 通知。

當客戶的訂閱請求完成時，開發人員入口網站應用程式應該將相關聯的 API 金鑰提供給客戶，並通知客戶必須在 API 請求的 x-api-key 標頭中包含此 API 金鑰。

當客戶取消訂閱用量計劃時，AWS Marketplace 會傳送 `unsubscribe-success` 通知至 SNS 主題。若要完成將客戶取消訂閱的程序，您可以從用量計劃中移除客戶的 API 金鑰，來處理 `unsubscribe-success` 通知。

## 授權客戶存取用量計劃

若要授權指定客戶存取您的用量計劃，請使用 API Gateway API 為客戶擷取或建立 API 金鑰，然後將 API 金鑰新增至用量計劃。

下列範例示範如何呼叫 API Gateway REST API，以使用特定 AWS Marketplace `customerId` 值 (`MARKEPLACE_CUSTOMER_ID`) 來建立新的 API 金鑰。

```
POST apikeys HTTP/1.1
Host: apigateway.region.amazonaws.com
Authorization: ...

{
  "name" : "my_api_key",
  "description" : "My API key",
  "enabled" : "false",
  "stageKeys" : [ {
    "restApiId" : "uycll6xg9a",
    "stageName" : "prod"
  }],
  "customerId" : "MARKEPLACE_CUSTOMER_ID"
}
```

下列範例示範如何使用特定 AWS Marketplace `customerId` 值 (`MARKEPLACE_CUSTOMER_ID`) 來取得 API 金鑰。

```
GET apikeys?customerId=MARKEPLACE_CUSTOMER_ID HTTP/1.1
Host: apigateway.region.amazonaws.com
Authorization: ...
```

若要將 API 金鑰新增至用量計劃，請使用相關用量計劃的 API 金鑰來建立 `UsagePlanKey`。下列範例示範如何使用 API Gateway REST API 來完成此作業，其中 `n371pt` 是用量計劃 ID，而 `q5ugs7qjjh` 是前述範例所傳回的範例 API keyID。

```
POST /usageplans/n371pt/keys HTTP/1.1
Host: apigateway.region.amazonaws.com
Authorization: ...

{
  "keyId": "q5ugs7qjjh",
  "keyType": "API_KEY"
}
```

## 將客戶與 API 金鑰相關聯

您必須將 `ApiKey` 的 `customerId` 欄位更新為客戶的 AWS Marketplace 客戶 ID。這會將 API 金鑰與 AWS Marketplace 客戶相關聯，以啟用買方的計量與計費。下列程式碼範例會呼叫 API Gateway REST API 來執行這項操作。

```
PATCH /apikeys/q5ugs7qjjh
Host: apigateway.region.amazonaws.com
Authorization: ...

{
  "patchOperations" : [ {
```

```
    "path" : "/customerId",
    "value" : "MARKETPLACE_CUSTOMER_ID",
    "op" : "replace"
}
}
```

## 在 API Gateway 中設定 API 的自訂網域名稱

在部署 REST 或 WebSocket API 之後，您 (和您的客戶) 可以使用下列格式的預設基本 URL 來呼叫 API：

```
https://api-id.execute-api.region.amazonaws.com/stage
```

其中 *api-id* 是 API Gateway 所產生，*region* 是您在建立 API 時所指定，而 *stage* 是您在部署 API 時所指定。

### Note

您可以將自訂網域與 REST API 或 Websocket API 建立關聯，但無法同時與兩者建立關聯。  
[私有 API \(p. 183\)](#) 並不支援自訂網域名稱。

URL 的主機名稱部分 (即 *api-id*.execute-api.*region*.amazonaws.com) 指的是 API 端點，這可以是邊緣最佳化的或區域性。預設 API 端點很難取回，而且不方便使用。若要為 API 使用者提供更簡單且更直覺式的 URL，您可以設定自訂網域名稱 (例如，api.example.com) 作為 API 的主機名稱，以及選擇基本路徑 (例如，myservice) 以將替代 URL 對應至此 API。更方便使用者使用的 API 基本 URL 現在成為：

```
https://api.example.com/myservice
```

如果您未在自訂網域名稱下方設定任何基本對應，則產生的 API 基本 URL 會與自訂網域相同 (例如，<https://api.example.com>)。在此情況下，自訂網域名稱只能支援一個 API。

當您部署邊緣最佳化 API 時，API Gateway 會設定 Amazon CloudFront 分佈和 DNS 記錄，以將 API 網域名稱對應至 CloudFront 分佈網域名稱。接著，透過對應的 CloudFront 分佈，將 API 的請求路由至 API Gateway。

當您建立邊緣最佳化 API 的自訂網域名稱時，API Gateway 會設定 CloudFront 分佈。但是，您必須設定 DNS 記錄以將自訂網域名稱對應至 API 請求的 CloudFront 分佈網域名稱，而 API 請求是送往要透過對應的 CloudFront 分佈路由至 API Gateway 的自訂網域名稱。您也必須提供自訂網域名稱的憑證。

當您建立區域 API 的自訂網域名稱時，API Gateway 會建立 API 的區域網域名稱。您必須設定 DNS 記錄以將自訂網域名稱對應至 API 請求的區域網域名稱，而 API 請求是送往要透過對應的區域 API 端點路由至 API Gateway 的自訂網域名稱。您也必須提供自訂網域名稱的憑證。

### Note

API Gateway 所建立的 CloudFront 分佈是由隸屬於 API Gateway 的區域特定帳戶所擁有。在 CloudWatch 日誌中追蹤可建立和更新這類 CloudFront 分佈的操作時，您必須使用此 API Gateway 帳戶 ID。如需詳細資訊，請參閱[CloudTrail 中的日誌自訂網域名稱建立 \(p. 596\)](#)。

若要設定邊緣最佳化自訂網域名稱或更新其憑證，您必須有權限更新 CloudFront 分佈。做法是將下列 IAM 政策陳述式連接至您帳戶中的 IAM 使用者、群組或角色：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowCloudFrontUpdateDistribution",
      "Effect": "Allow",
      "Action": [
        "CloudFront:CreateDistribution",
        "CloudFront:UpdateDistribution"
      ]
    }
  ]
}
```

```
        "cloudfront:updateDistribution"
    ],
    "Resource": [
        "*"
    ]
}
}
```

API Gateway 透過在 CloudFront 分佈上利用伺服器名稱指示 (SNI)，來支援邊緣最佳化自訂網域名稱。如需在 CloudFront 分佈上使用自訂網域名稱的詳細資訊 (包含必要憑證格式和憑證金鑰長度大小上限)，請參閱Amazon CloudFront Developer Guide中的[使用備用網域名稱和 HTTPS](#)。

若要設定自訂網域名稱作為 API 的主機名稱，您身為 API 擁有者所以必須提供自訂網域名稱的 SSL/TLS 憑證。

若要提供邊緣最佳化自訂網域名稱的憑證，您可以請求 [AWS Certificate Manager](#) (ACM) 在 ACM 中產生新的憑證，或將 us-east-1 區域 (維吉尼亞北部) 中第三方憑證授權機構所發出的憑證匯入至 ACM。

若要在支援 ACM 的區域中提供區域自訂網域名稱的憑證，您必須向 ACM 請求憑證。若要在不支援 ACM 的區域中提供區域自訂網域名稱的憑證，您必須將憑證匯入至該區域中的 API Gateway。

若要匯入 SSL/TLS �凭證，您必須提供 PEM 格式化 SSL/TLS �凭證內文、私有金鑰，以及自訂網域名稱的憑證鏈。ACM 中所存放的每個憑證都是透過其 ARN 進行識別。若要使用網域名稱的 AWS 受管憑證，您只需要參考其 ARN。

ACM 可讓您直覺式地設定和使用 API 的自訂網域名稱：在 ACM 中建立所指定網域名稱的憑證或將其匯入 &ACM；使用 ACM 所提供的憑證 ARN 在 API Gateway 中設定網域名稱，以及將自訂網域名稱下的基本路徑對應至 API 的已部署階段。使用 ACM 所發出的憑證，就不需要擔心公開任何敏感的憑證詳細資訊，例如私有金鑰。

您必須具有已註冊的網際網路網域名稱，才能為您的 API 設定自訂網域名稱。需要時，您可以使用 [Amazon Route 53](#) 或使用您選擇的第三方網域註冊機構來註冊網際網路網域。API 的自訂網域名稱可以是已註冊網際網路網域的子網域或根網域 (也稱為 Zone Apex) 的名稱。

在 API Gateway 中建立自訂網域名稱之後，您必須建立或更新網域名稱服務 (DNS) 提供者的資源記錄，以將邊緣最佳化自訂網域名稱對應至其 CloudFront 分佈網域名稱，或將區域自訂網域名稱對應至其區域 API 端點。如果沒有這種對應，送往自訂網域名稱的 API 請求無法到達 API Gateway。

#### Note

邊緣最佳化自訂網域名稱是在特定區域中建立，並由特定 AWS 帳戶所擁有。在區域或 AWS 帳戶之間移動這類自訂網域名稱，包含刪除現有 CloudFront 分佈，以及建立新的 &CF; 分佈。此程序可能需要大約 30 分鐘的時間，新的自訂網域名稱才會變成可用。如需詳細資訊，請參閱[更新 CloudFront 分佈](#)。

下列頁面說明如何使用 ACM 建立自訂網域名稱的 SSL/TLS �凭證、設定 API 的自訂網域名稱、將特定 API 與自訂網域名稱下的基本路徑建立關聯，並續約 (也稱為輪換) 自訂網域名稱之已匯入至 ACM 的到期憑證。

#### 主題

- [在 AWS Certificate Manager 中備妥憑證 \(p. 591\)](#)
- [如何建立邊緣最佳化自訂網域名稱 \(p. 593\)](#)
- [在 API Gateway 中設定區域性 API 的自訂網域名稱 \(p. 599\)](#)
- [將自訂網域名稱遷移至不同的 API 端點 \(p. 601\)](#)

## 在 AWS Certificate Manager 中備妥憑證

必須先在 AWS Certificate Manager 中備妥 SSL/TLS �凭證，再設定 API 的自訂網域名稱。下列步驟說明如何完成這項操作。如需詳細資訊，請參閱 [AWS Certificate Manager User Guide](#)。

#### Note

若要搭配使用 ACM 憑證與 API Gateway 邊緣最佳化自訂網域名稱，您必須在 US East (N. Virginia) (us-east-1) 區域中請求或匯入憑證。針對 API Gateway 區域自訂網域名稱，您必須在與 API 相同的區域中請求或匯入憑證。

#### 取得 ACM 所發出或匯入至其中之指定網域名稱的憑證

- 註冊您的網際網路網域；例如，*myDomain.com*。您可以使用 [Amazon Route 53](#) 或第三方認可的網域註冊機構。如需這類註冊機構的清單，請參閱 ICANN 網站上的[認可的註冊機構目錄](#)。
- 若要在 ACM 中建立網域名稱的 SSL/TLS �凭證或將其匯入至 &ACM；，請執行下列其中一項操作：

#### 請求 ACM 針對網域名稱所提供的憑證

- 登入 [AWS Certificate Manager 主控台](#)。
- 選擇 Request a certificate (請求憑證)。
- 在 Domain name (網域名稱) 中，輸入 API 的自訂網域名稱 (例如 `api.example.com`)。
- (選擇性) 選擇 Add another name to this certificate (將其他名稱新增至此憑證)。
- 選擇 Review and request (檢閱和請求)。
- 選擇 Confirm and request (確認和請求)。
- 針對有效的請求，在 ACM 發出憑證之前，網際網路網域的註冊擁有者必須先同意請求。

#### 將網域名稱的憑證匯入至 ACM

- 取得憑證授權機構中自訂網域名稱的 PEM 編碼 SSL/TLS �凭證。如需這類 CA 的局部清單，請參閱[Mozilla 已包含 CA 清單](#)
  - 使用 OpenSSL 網站上的 [OpenSSL 工具組](#)，來產生憑證的私有金鑰，並將輸出儲存至檔案：

```
openssl genrsa -out private-key-file 2048
```

#### Note

Amazon API Gateway 利用 Amazon CloudFront 支援自訂網域名稱的憑證。因此，自訂網域名稱 SSL/TLS �凭證的需求和限制是由 [CloudFront](#) 所指定。例如，公有金鑰大小上限是 2048，而私有金鑰大小可以是 1024、2048 和 4096。公有金鑰大小是由您使用的憑證授權機構所決定。請求憑證授權機構傳回大小與預設長度不同的金鑰。如需詳細資訊，請參閱[保護物件的存取和建立已簽署 URL 和已簽署 Cookie](#)。

- 使用 OpenSSL，透過先前產生的私有金鑰來產生憑證簽署請求 (CSR)：

```
openssl req -new -sha256 -key private-key-file -out CSR-file
```

- 將 CSR 提交至憑證授權機構，並儲存產生的憑證。
- 從憑證授權機構下載憑證鏈。

#### Note

如果您使用另一種方式來取得私有金鑰並加密金鑰，則可以使用下列命令先解密金鑰，再將它提交至 API Gateway 來設定自訂網域名稱。

```
openssl pkcs8 -topk8 -inform pem -in MyEncryptedKey.pem -outform pem -nocrypt -out MyDecryptedKey.pem
```

- 將憑證上傳至 AWS Certificate Manager：

- a. 登入 [AWS Certificate Manager 主控台](#)。
- b. 選擇 Import a certificate (匯入憑證)。
- c. 針對 Certificate body (憑證內文)，輸入或貼上您憑證授權機構中的 PEM 格式化伺服器憑證內文。以下示範這類憑證的縮寫範例。

```
-----BEGIN CERTIFICATE-----  
EXAMPLECA+KgAwIBAgIQJ1XxJ8P1++gOfQtj0IBoqDANBgkqhkiG9w0BAQUFADBB  
...  
az8Cg1aicxLBQ7EaWIhhgEXAMPLE  
-----END CERTIFICATE-----
```

- d. 針對 Certificate private key (憑證私有金鑰)，輸入或貼上您 PEM 格式化憑證的私有金鑰。以下示範這類金鑰的縮寫範例。

```
-----BEGIN RSA PRIVATE KEY-----  
EXAMPLEBAAKCAQEA2Qb3LDHD7StY7Wj6U2/opV6Xu37qUCCkeDWhwpZMYJ9/nETO  
...  
1qGvJ3u04vdnzaYN5WoyN5LFckrlA71+CszD1CGSqbVDWEXAMPLE  
-----END RSA PRIVATE KEY-----
```

- e. 針對 Certificate chain (憑證鏈)，輸入或貼上 PEM 格式化中繼憑證，以及選擇性地逐一輸入或貼上根憑證，不能有任何空白行。如果您包含根憑證，則憑證鏈的開頭必須是中繼憑證，而結尾必須是根憑證。使用憑證授權機構所提供的中繼憑證。請不要包含不在信任路徑鏈中的任何中介。以下示範縮寫範例。

```
-----BEGIN CERTIFICATE-----  
EXAMPLECA4ugAwIBAgIQWrYdrB5NogYUx1U9Pamy3DANBgkqhkiG9w0BAQUFADCB  
...  
8/ifBLIK3se2e4/hEfccEejX/arxbx1BJCHBv1EPNnsdw8EXAMPLE  
-----END CERTIFICATE-----
```

以下是另一個範例。

```
-----BEGIN CERTIFICATE-----  
Intermediate certificate 2  
-----END CERTIFICATE-----  
-----BEGIN CERTIFICATE-----  
Intermediate certificate 1  
-----END CERTIFICATE-----  
-----BEGIN CERTIFICATE-----  
Optional: Root certificate  
-----END CERTIFICATE-----
```

- f. 選擇 Review and import (檢閱和匯入)。
3. 成功建立或匯入憑證之後，請記下憑證 ARN。之後，在設定自訂網域名稱時，您會需要它。

## 如何建立邊緣最佳化自訂網域名稱

### 主題

- [設定 API Gateway API 的邊緣最佳化自訂網域名稱 \(p. 594\)](#)
- [CloudTrail 中的日誌自訂網域名稱建立 \(p. 596\)](#)
- [使用自訂網域名稱作為其主機名稱來設定 API 的基本路徑對應 \(p. 597\)](#)
- [輪換匯入至 ACM 的憑證 \(p. 597\)](#)
- [呼叫具有自訂網域名稱的 API \(p. 598\)](#)

## 設定 API Gateway API 的邊緣最佳化自訂網域名稱

下列程序說明如何使用 API Gateway 主控台來設定 API 的自訂網域名稱。

### 使用 API Gateway 主控台設定自訂網域名稱

1. 登入位於 <https://console.aws.amazon.com/apigateway> 的 API Gateway 主控台。
2. 從主要導覽窗格中，選擇 Custom Domain Names (自訂網域名稱)。
3. 接下來，選擇 Create Custom Domain Name (建立自訂網域名稱)。
4. a. 在 New Custom Domain Name (新的自訂網域名稱) 的 Domain Name (網域名稱) 中，輸入您的網域名稱，例如 `api.example.com`。

#### Note

請不要針對自訂網域名稱使用萬用字元 (即 \*)。API Gateway 不支援它，即使 API Gateway 主控台 (或 AWS CLI) 接受它而且可以將它對應至 CloudFront 分佈也是一樣。不過，您可以使用萬用字元憑證。

- b. 從 ACM Certificate (&ACM; 憑證) 清單中，選擇憑證。

#### Note

若要搭配使用 ACM 憑證與 API Gateway 邊緣最佳化自訂網域名稱，您必須在 us-east-1 區域 (維吉尼亞北部) 區域中請求或匯入憑證。

- c. 選擇 Base Path Mappings (基本路徑對應) 下的 Add mapping (新增對應) 以設定指定階段中已部署 API 的基本路徑 (Path (路徑)) (從 Destination (目標) 下拉式清單中選取)。在建立自訂網域名稱之後，您也可以設定基本路徑對應。如需詳細資訊，請參閱[使用自訂網域名稱作為其主機名稱來設定 API 的基本路徑對應 \(p. 597\)](#)。
- d. 選擇 Save (儲存)。
5. 建立自訂網域名稱之後，主控台會顯示相關聯的 CloudFront 分佈網域名稱 (格式為 `distribution-id.cloudfront.net`) 以及憑證 ARN。請記下輸出中所顯示的 CloudFront 分佈網域名稱。在後續步驟中，您需要它來設定自訂網域之 DNS 中的 CNAME 值或 A 記錄別名目標。

#### Note

新建立的自訂網域名稱大約需要 40 分鐘的時間準備。同時，您可以設定 DNS 記錄別名，以將自訂網域名稱對應至相關聯的 CloudFront 分佈網域名稱，以及在初始化自訂網域名稱時設定自訂網域名稱的基本路徑對應。

6. 在此步驟中，我們使用 Amazon Route 53 做為範例 DNS 提供者，示範如何設定您網際網路網域的 A 記錄別名，以將自訂網域名稱對應至相關聯的 CloudFront 分佈名稱。這些說明可能會針對其他 DNS 提供者進行調整。
  - a. 登入 Route 53 主控台。
  - b. 建立自訂網域的 A-IPv4 address 記錄集 (例如，`api.example.com`)。A 記錄會將自訂網域名稱對應至 IP4 地址。
  - c. 針對 Alias (別名) 選擇 Yes (是)，並在 Alias Target (別名目標) 中輸入 CloudFront 網域名稱 (例如 `d3boq9ikothtgw.cloudfront.net`)，然後選擇 Create (建立)。這裡的 A 記錄別名會將您的自訂網域名稱對應至本身對應至 IP4 地址的指定 CloudFront 網域名稱。

**Create Record Set**

**Name:** api.haymuto.com.

**Type:** A – IPv4 address

**Alias:**  Yes  No

**Alias Target:** d3boq9ikothtgw.cloudfront.net !

**Alias Hosted Zone ID:** Z2FDTNDATAQYW2

You can also type the domain name for the resource. Examples:  
- CloudFront distribution domain name: d111111abcdef8.cloudfront.net  
- Elastic Beanstalk environment CNAME: example.elasticbeanstalk.com  
- ELB load balancer DNS name: example-1.us-east-1.elb.amazonaws.com  
- S3 website endpoint: s3-website.us-east-2.amazonaws.com  
- Resource record set in this hosted zone: www.example.com

[Learn More](#)

**Routing Policy:** Simple

Route 53 responds to queries based only on the values in this record. [Learn More](#)

**Evaluate Target Health:**  Yes  No !

**Create**

Tip

Alias Hosted Zone ID (別名託管區域 ID) 識別所指定 Alias Target (別名目標) 的託管區域。在您輸入 Alias Target (別名目標) 的有效網域名稱時，Route 53 主控台會自動填入值。若要建立 A 記錄別名，而不使用 Route 53 主控台 (例如，使用 AWS CLI 時)，您必須指定所需的託管

區域 ID。針對任何 CloudFront 分佈網域名稱，託管區域 ID 值一律是 Z2FDTNDATAQYW2，如 [CloudFront 的 AWS 區域和端點](#) 中所記錄。

針對大多數的 DNS 提供者，自訂網域名稱會新增至託管區域作為 CNAME 資源記錄集。CNAME 記錄名稱指定您先前在 Domain Name (網域名稱) 中輸入的自訂網域名稱 (例如，api.example.com)。CNAME 記錄值指定 CloudFront 分佈的網域名稱。不過，如果您的自訂網域是 Zone Apex (即 example.com，而不是 api.example.com)，則無法使用 CNAME 記錄。Zone Apex 通常也稱為您組織的根網域。針對 Zone Apex，您需要使用 A 記錄別名，但前提是 DNS 提供者予以支援。

使用 Route 53，您可以建立自訂網域名稱的 A 記錄別名，並指定 CloudFront 分佈網域名稱做為別名目標，如上所示。這表示 Route 53 可以路由自訂網域名稱，即使是 Zone Apex 也是一樣。如需詳細資訊，請參閱 [Amazon Route 53 Developer Guide](#) 中的 [選擇別名與非別名資源記錄集](#)。

使用 A 記錄別名也不需要公開基礎 CloudFront 分佈網域名稱，因為網域名稱對應只會發生在 Route 53 內。基於這些原因，建議您盡可能使用 Route 53 A 記錄別名。

除了使用 API Gateway 主控台之外，您還可以使用 API Gateway REST API、AWS CLI 或其中一個 AWS 開發套件來設定 API 的自訂網域名稱。作為說明，下列程序概述使用 REST API 呼叫進行這項操作的步驟。

#### 使用 API Gateway REST API 設定自訂網域名稱

1. 呼叫 [domainname:create](#)，並指定 AWS Certificate Manager 中所存放憑證的自訂網域名稱和 ARN。  
成功 API 呼叫會傳回 201 Created 回應，其中包含憑證 ARN，以及其承載中的相關聯 CloudFront 分佈名稱。
2. 請記下輸出中所顯示的 CloudFront 分佈網域名稱。在後續步驟中，您需要它來設定自訂網域之 DNS 中的 CNAME 值或 A 記錄別名目標。
3. 遵循先前程序中的步驟 6 來設定 A 記錄別名，以將自訂網域名稱對應至其 CloudFront 分佈名稱。

如需此 REST API 呼叫的程式碼範例，請參閱 [domainname:create](#)。

## CloudTrail 中的日誌自訂網域名稱建立

啟用 CloudTrail 來記錄您帳戶所進行的 API Gateway 呼叫時，API Gateway 會在建立或更新 API 的自訂網域名稱時記錄相關聯 CloudFront 分佈更新。因為這些 CloudFront 分佈是 API Gateway 所擁有，所以所有這些報告的 CloudFront 分佈都是透過下列其中一個區域特定 API Gateway 帳戶 ID 來識別，而不是 API 擁有者的帳戶 ID。

#### 與自訂網域名稱建立關聯之 CloudFront 分佈的區域特定 API Gateway 帳戶 ID

區域	帳戶 ID
us-east-1	392220576650
us-east-2	718770453195
us-west-1	968246515281
us-west-2	109351309407
eu-west-1	631144002099
eu-west-2	544388816663
eu-central-1	474240146802
ap-northeast-1	969236854626

區域	帳戶 ID
ap-northeast-2	020402002396
ap-southeast-1	195145609632
ap-southeast-2	798376113853

## 使用自訂網域名稱作為其主機名稱來設定 API 的基本路徑對應

您可以使用單一自訂網域名稱作為多個 API 的主機名稱。做法是在自訂網域名稱上設定基本路徑對應。使用基本路徑對應，自訂網域下的 API 可以透過自訂網域名稱與相關聯基本路徑的組合進行存取。

例如，如果您已建立名為 PetStore 的 API 和名為 PetShop 的另一個 API，並在 API Gateway 中設定自訂網域名稱 api.example.com，則可以將 PetStore API 的 URL 設定為 https://api.example.com 或 https://api.example.com/myPetStore。PetStoreAPI 是與自訂網域名稱 myPetStore 下具有空字串或 api.example.com 的基本路徑建立關聯。同樣地，您可以指派 yourPetShop API 的基本路徑 PetShop。https://api.example.com/yourPetShop URL 接著會是 PetShop API 的根 URL。

設定 API 的基本路徑之前，請完成[設定 API Gateway API 的邊緣最佳化自訂網域名稱 \(p. 594\)](#)中的步驟。

### 使用 API Gateway 主控台設定 API 對應的基本路徑

- 從您帳戶下可用的 Custom Domain Names (自訂網域名稱) 清單中，選擇自訂網域名稱。
- 選擇 Show Base Path Mappings (顯示基本路徑對應) 或 Edit (編輯)。
- 選擇 Add mapping (新增對應)。
- (選用) 針對 Path (路徑) 輸入基本路徑名稱，並從 Destination (目標) 中選擇 API，然後選擇階段。

#### Note

Destination (目標) 清單會顯示您帳戶下的已部署 API。

- 選擇 Save (儲存)，完成 API 的基本路徑對應設定。

#### Note

若要在建立對應之後將其刪除，請選擇您要刪除之對應旁邊的垃圾桶圖示。

此外，您還可以呼叫 API Gateway REST API、AWS CLI 或其中一個 AWS 開發套件，來設定自訂網域名稱作為其主機名稱之 API 的基本路徑對應。作為說明，下列程序概述使用 REST API 呼叫進行這項操作的步驟。

### 使用 API Gateway REST API 設定 API 的基本路徑對應

- 在特定自訂網域名稱上呼叫 [basepathmapping:create](#)，並在請求承載中指定 basePath、restApiId 和部署 stage 屬性。

成功 API 呼叫會傳回 201 Created 回應。

如需 REST API 呼叫的程式碼範例，請參閱 [basepathmapping:create](#)。

## 輪換匯入至 ACM 的憑證

ACM 會自動處理它所發出之憑證的續約。您不需要為自訂網域名稱輪換任何 ACM 發出的憑證。CloudFront 會代表您處理這項操作。

不過，如果您將憑證匯入至 ACM，並將它用於自訂網域名稱，則您必須在憑證過期之前輪換憑證。這包含匯入網域名稱的新第三方憑證，並將現有憑證輪換為新的憑證。新匯入的憑證過期時，您需要重複進行此程

序。或者，您也可以請求 ACM 發出網域名稱的新憑證，並將現有憑證輪換為新的 ACM 發出憑證。之後，您可以讓 ACM 和 CloudFront 自動處理憑證輪換。若要建立或匯入新的 ACM 憑證，請遵循所指定網域名稱的 [請求或匯入新 ACM �凭證 \(p. 591\)](#)步驟。

若要輪換網域名稱的憑證，您可以使用 API Gateway 主控台、API Gateway REST API、AWS CLI 或其中一個 AWS 開發套件。

#### 使用 API Gateway 主控台輪換匯入至 ACM 的過期憑證

1. 在 ACM 中請求或匯入憑證。
2. 回到 API Gateway 主控台。
3. 從 API Gateway 主控台主要導覽窗格中，選擇 Custom Domain Names (自訂網域名稱)。
4. 在 Custom Domain Names (自訂網域名稱) 窗格下，選取您所選擇的自訂網域名稱。
5. 選擇 Edit (編輯)。
6. 從 ACM Certificate (ACM 憑證) 下拉式清單中，選擇所要的憑證。
7. 選擇 Save (儲存)，開始輪換自訂網域名稱的憑證。

#### Note

大約需要 40 分鐘的時間，程序才能完成。輪換完成之後，您可以選擇 ACM Certificate (ACM �凭證) 旁邊的雙向箭頭圖示來復原為原始憑證。

為了說明如何以程式設計方式輪換自訂網域名稱的已匯入憑證，我們概述使用 API Gateway REST API 的步驟。

#### 使用 API Gateway REST API 輪換已匯入的憑證

- 呼叫 [domainname:update](#) 動作，並指定所指定網域名稱之新 ACM 憑證的 ARN。

## 呼叫具有自訂網域名稱的 API

呼叫具有自訂網域名稱的 API 與呼叫具有其預設網域名稱的 API 相同，但前提是使用正確的 URL。

下列範例會比較和對比所指定區域 (udxjef) 中兩個 API (qf3duz 和 us-east-1) 以及所指定自訂網域名稱 (api.example.com) 的一組預設 URL 和對應自訂 URL。

#### 具有預設和自訂網域名稱之 API 的根 URL

API ID	階段	預設 URL	基本路徑	自訂 URL
udxjef	pro	<code>https://udxjef.execute-api.us-east-1.amazonaws.com/pro</code>	/petstore	<code>https://api.example.com/petstore</code>
udxjef	tst	<code>https://udxjef.execute-api.us-east-1.amazonaws.com/tst</code>	/petdepot	<code>https://api.example.com/petdepot</code>
qf3duz	dev	<code>https://qf3duz.execute-api.us-</code>	/bookstore	<code>https://api.example.com/bookstore</code>

API ID	階段	預設 URL	基本路徑	自訂 URL
		east-1.amazonaws.com/dev		
qf3duz	tst	https://qf3duz.execute-api.us-east-1.amazonaws.com/tst	/bookstand	https://api.example.com/bookstand

API Gateway 使用[伺服器名稱指示 \(SNI\)](#)來支援 API 的自訂網域名稱。您可以使用支援 SNI 的瀏覽器或用戶端程式庫，以使用自訂網域名稱來呼叫 API。

API Gateway 會在 CloudFront 分佈上強制執行 SNI。如需 CloudFront 如何使用自訂網域名稱的資訊，請參閱[Amazon CloudFront 自訂 SSL](#)。

## 在 API Gateway 中設定區域性 API 的自訂網域名稱

如同邊緣最佳化的 API 端點，您可以為區域性 API 端點建立自訂網域名稱。若要支援區域性自訂網域名稱，您必須提供憑證。如果使用 AWS Certificate Manager (ACM) 憑證，此憑證必須為區域特定。如果區域中可以使用 ACM，您必須提供該區域特定的 ACM �凭證。如果該區域不支援 ACM，您必須在建立區域性自訂網域名稱時，將憑證上傳至該區域中的 API Gateway。如需建立或上傳自訂網域名稱憑證的詳細資訊，請參閱「[在 AWS Certificate Manager 中備妥憑證 \(p. 591\)](#)」。

### Important

針對 API Gateway 區域自訂網域名稱，您必須在與 API 相同的區域中請求或匯入憑證。

當您使用 ACM �凭證建立 (或遷移) 區域性自訂網域名稱，API Gateway 會在您的帳戶中建立服務連結角色 (如果該角色尚不存在)。需要此服務連結角色，才能將您的 ACM �凭證附加至您的區域性端點。該角色的名稱為 AWSServiceRoleForAPIGateway，而且將會連接 APIGatewayServiceRolePolicy 受管政策。如需使用服務連結角色的詳細資訊，請參閱[使用服務連結角色](#)。

### Important

您必須建立 DNS 記錄，將自訂網域名稱指向區域性網域名稱。如此可讓繫結至自訂網域名稱的流量路由至 API 的區域性主機名稱。DNS 記錄可以是 CNAME 或 A 類型。

### 主題

- [使用 API Gateway 主控台設定區域性自訂網域名稱 \(p. 599\)](#)
- [使用 AWS CLI 設定區域性自訂網域名稱 \(p. 600\)](#)

## 使用 API Gateway 主控台設定區域性自訂網域名稱

若要使用 API Gateway 主控台來設定區域性自訂網域名稱，請使用下列程序。

### 使用 API Gateway 主控台設定區域性自訂網域名稱

1. 登入 API Gateway 主控台，然後在主導覽窗格中選擇 Custom Domain Names (自訂網域名稱)。
2. 選擇 Custom Domain Names (自訂網域名稱) 表格上方的 +Create New Custom Domain Name (+建立新的自訂網域名稱)。
3. 在 New Custom Domain Name (新的自訂網域名稱) 的 Domain Name (網域名稱) 中，輸入自訂網域名稱，例如 my-api.example.com。

4. 針對 Endpoint Configuration (端點組態) , 選擇 Regional (區域性)。
5. 從 ACM Certificate (ACM 憑證) 下拉式清單中 , 選擇一個憑證。此憑證必須來自 API 部署所在的同一個區域。
6. 如果您已建立並部署 API 來使用此自訂網域名稱 , 請選擇 Add mapping (新增對應) , 在 Path (路徑) 中輸入自訂網域名稱下的基底路徑 , 從 Destination (目標) 下的 API 下拉式清單中選擇 API , 然後從 Stage (階段) 下拉式清單中選擇階段。若要新增另一個基底路徑對應 , 請重複此步驟。
7. 選擇 Save (儲存)。
8. 依照 Route 53 文件 , [設定 Route 53 將流量路由至 API Gateway](#)。

## 使用 AWS CLI 設定區域性自訂網域名稱

若要使用 AWS CLI 來設定區域性 API 的自訂網域名稱 , 請使用下列程序。

1. 呼叫 `create-domain-name` , 並指定 REGIONAL 類型的自訂網域名稱與區域性憑證的 ARN。

```
aws apigateway create-domain-name \
--domain-name 'regional.example.com' \
--endpoint-configuration types=REGIONAL \
--regional-certificate-arn 'arn:aws:acm:us-west-2:123456789012:certificate/c19332f0-3be6-457f-a244-e03a423084e6'
```

請注意 , 指定的憑證是來自 us-west-2 區域 , 在此範例中 , 我們假設基礎 API 是來自相同的區域。

如果成功 , 呼叫會傳回類似如下的結果 :

```
{
  "certificateUploadDate": "2017-10-13T23:02:54Z",
  "domainName": "regional.example.com",
  "endpointConfiguration": {
    "types": "REGIONAL"
  },
  "regionalCertificateArn": "arn:aws:acm:us-west-2:123456789012:certificate/c19332f0-3be6-457f-a244-e03a423084e6",
  "regionalDomainName": "d-numh1z56v6.execute-api.us-west-2.amazonaws.com"
}
```

regionalDomainName 屬性值會傳回區域性 API 的主機名稱。您必須建立 DNS 記錄 , 將您的自訂網域名稱指向此區域性網域名稱。如此可讓繫結至自訂網域名稱的流量路由至此區域性 API 的主機名稱。

2. 建立 DNS 記錄 , 將自訂網域名稱與區域性網域名稱相關聯。如此可讓繫結至自訂網域名稱的請求路由至 API 的區域性主機名稱。
3. 新增基底路徑對應 , 以根據指定的自訂網域名稱 (例如 0qzs2sy7bh) , 在部署階段 (例如 test) 中公開指定的 API (例如 regional.example.com)。

```
aws apigateway create-base-path-mapping \
--domain-name 'regional.example.com' \
--base-path 'RegionalApiTest' \
--rest-api-id 0qzs2sy7bh \
--stage 'test'
```

因此 , 使用階段中所部署 API 之自訂網域名稱的基底 URL 會變成 <https://regional.example.com/RegionalApiTest>。

4. 請設定 DNS 記錄 , 將區域性自訂網域名稱對應到其指定託管區域 ID 的主機名稱。首先建立 JSON 檔案 , 其中包含用於設定區域性網域名稱之 DNS 記錄的組態。下列範例示範如何建立 DNS A 記錄 , 將區域性自訂網域名稱 (regional.example.com) 映射到建立自訂網域名稱時所佈建的區域性主機名稱 (d-numh1z56v6.execute-api.us-west-2.amazonaws.com)。DNSName 的 HostedZoneId 與

AliasTarget 屬性可分別接受自訂網域名稱的 regionalDomainName 與 regionalHostedZoneId 值。您也可以在 [API Gateway 區域與端點](#) 中取得區域性 Route 53 託管區域 ID。

```
{  
    "Changes": [  
        {  
            "Action": "CREATE",  
            "ResourceRecordSet": {  
                "Name": "regional.example.com",  
                "Type": "A",  
                "AliasTarget": {  
                    "DNSName": "d-numh1z56v6.execute-api.us-west-2.amazonaws.com",  
                    "HostedZoneId": "Z2OJLYMU09EFXC",  
                    "EvaluateTargetHealth": false  
                }  
            }  
        }  
    ]  
}
```

5. 執行以下 CLI 命令：

```
aws route53 change-resource-record-sets \  
    --hosted-zone-id {your-hosted-zone-id} \  
    --change-batch file://path/to/your/setup-dns-record.json
```

其中 *{your-hosted-zone-id}* 是您帳戶中所設定之 DNS 記錄的 Route 53 託管區域 ID。change-batch 參數值指向資料夾 (*path/to/your*) 中的 JSON 檔案 (*setup-dns-record.json*)。

## 將自訂網域名稱遷移至不同的 API 端點

您可以在邊緣最佳化與區域端點之間遷移自訂網域名稱。您會先將新的端點組態類型新增至自訂網域名稱的現有 endpointConfiguration.types 清單。接著，您設定 DNS 記錄，以將自訂網域名稱指向新佈建的端點。選用的最後一個步驟是移除過時自訂網域名稱組態資料。

規劃遷移時，請記住，針對邊緣最佳化 API 的自訂網域名稱，ACM 所提供的必要憑證必須來自 US East (N. Virginia) Region (us-east-1)。此憑證會分發至所有地理位置。不過，針對區域 API，區域網域名稱的 ACM 憑證必須來自託管 API 的相同區域。您可以將不在 us-east-1 區域中的邊緣最佳化自訂網域名稱遷移至區域自訂網域名稱，方法是先請求來自 API 之本機區域的新 ACM 憑證。

這可能需要最多 60 秒的時間才能完成邊緣最佳化自訂網域名稱與 API Gateway 中區域自訂網域名稱之間的遷移。若要讓新建立的端點準備好接受流量，遷移時間也取決於何時更新 DNS 記錄。

### 主題

- [使用 API Gateway 主控台遷移區域性與邊緣最佳化的網域名稱 \(p. 601\)](#)
- [使用 AWS CLI 遷移自訂網域名稱 \(p. 602\)](#)

## 使用 API Gateway 主控台遷移區域性與邊緣最佳化的網域名稱

若要使用 API Gateway 主控台將區域性自訂網域名稱遷移到邊緣最佳化的自訂網域名稱 (反之亦然)，請使用下列程序。

### 使用 API Gateway 主控台遷移區域性或邊緣最佳化的自訂網域名稱

1. 登入 API Gateway 主控台，然後在主導覽窗格中選擇 Custom Domain Names (自訂網域名稱)。
2. 從 Custom Domain Names (自訂網域名稱) 選擇現有的網域名稱，然後選擇 Edit (編輯)。

3. 根據現有的端點類型，執行下列操作：
  - a. 針對邊緣最佳化的網域名稱，選擇 Add Regional Configuration (新增區域性組態)。
  - b. 針對區域性網域名稱，選擇 Add Edge Configuration (新增邊緣組態)。
4. 從下拉式清單中選擇憑證。
5. 選擇 Save (儲存)。
6. 選擇 Proceed (繼續) 以確認加入新的端點。
7. 更新 DNS 記錄，將新的網域名稱指向新佈建的目標網域名稱。

## 使用 AWS CLI 遷移自訂網域名稱

若要使用 AWS CLI 將自訂網域名稱從邊緣最佳化端點遷移至區域端點 (或反之亦然)，請呼叫 [update-domain-name](#) 命令來新增端點類型，並選擇性地呼叫 [update-domain-name](#) 命令來移除舊端點類型。

### 主題

- [將邊緣最佳化自訂網域名稱遷移至區域 \(p. 602\)](#)
- [將區域自訂網域名稱遷移至邊緣最佳化 \(p. 603\)](#)

### 將邊緣最佳化自訂網域名稱遷移至區域

若要將邊緣最佳化自訂網域名稱遷移至區域自訂網域名稱，請呼叫 `update-domain-name` CLI 命令，如下所示：

```
aws apigateway update-domain-name \
--domain-name 'api.example.com' \
--patch-operations [ \
    { op:'add', path: '/endpointConfiguration/types', value: 'REGIONAL' }, \
    { op:'add', path: '/regionalCertificateArn', value: 'arn:aws:acm:us-west-2:123456789012:certificate/cd833b28-58d2-407e-83e9-dce3fd852149' } \
]
```

區域憑證必須與區域 API 位在相同的區域。

成功回應會有 200 OK 狀態碼以及與下列類似的內文：

```
{
    "certificateArn": "arn:aws:acm:us-east-1:123456789012:certificate/34a95aa1-77fa-427caa07-3a88bd9f3c0a",
    "certificateName": "edge-cert",
    "certificateUploadDate": "2017-10-16T23:22:57Z",
    "distributionDomainName": "d1frvgze7vy1bf.cloudfront.net",
    "domainName": "api.example.com",
    "endpointConfiguration": {
        "types": [
            "EDGE",
            "REGIONAL"
        ],
        "regionalCertificateArn": "arn:aws:acm:us-west-2:123456789012:certificate/cd833b28-58d2-407e-83e9-dce3fd852149",
        "regionalDomainName": "d-fdisjghyn6.execute-api.us-west-2.amazonaws.com"
    }
}
```

針對已遷移的區域自訂網域名稱，產生的 `regionalDomainName` 屬性會傳回區域 API 主機名稱。您必須設定 DNS 記錄，將區域自訂網域名稱指向此區域主機名稱。如此可讓繫結至自訂網域名稱的流量路由至區域主機。

設定 DNS 記錄之後，您可以呼叫 AWS CLI 的 [update-domain-name](#) 命令來移除邊緣最佳化自訂網域名稱：

```
aws apigateway update-domain-name \
--domain-name api.example.com \
--patch-operations [ \
    {op:'remove', path:'/endpointConfiguration/types', value:'EDGE'}, \
    {op:'remove', path:'certificateName'}, \
    {op:'remove', path:'certificateArn'} \
]
```

## 將區域自訂網域名稱遷移至邊緣最佳化

若要將區域自訂網域名稱遷移至邊緣最佳化自訂網域名稱，請呼叫 AWS CLI 的 [update-domain-name](#) 命令，如下所示：

```
aws apigateway update-domain-name \
--domain-name 'api.example.com' \
--patch-operations [ \
    { op:'add', path:'/endpointConfiguration/types',value: 'EDGE' }, \
    { op:'add', path:'/certificateName', value:'edge-cert'}, \
    { op:'add', path:'/certificateArn', value: 'arn:aws:acm:us- \
east-1:123456789012:certificate/34a95aa1-77fa-427c-aa07-3a88bd9f3c0a' } \
]
```

在 us-east-1 區域中，必須建立邊緣最佳化網域憑證。

成功回應會有 200 OK 狀態碼以及與下列類似的內文：

```
{
    "certificateArn": "arn:aws:acm:us-east-1:738575810317:certificate/34a95aa1-77fa-427c- \
aa07-3a88bd9f3c0a",
    "certificateName": "edge-cert",
    "certificateUploadDate": "2017-10-16T23:22:57Z",
    "distributionDomainName": "d1frvgze7vy1bf.cloudfront.net",
    "domainName": "api.example.com",
    "endpointConfiguration": {
        "types": [
            "EDGE",
            "REGIONAL"
        ],
        "regionalCertificateArn": "arn:aws:acm:us- \
east-1:123456789012:certificate/3d881b54-851a-478a-a887-f6502760461d",
        "regionalDomainName": "d-cgkq2qwgf.execute-api.us-east-1.amazonaws.com"
    }
}
```

針對指定的自訂網域名稱，API Gateway 會傳回邊緣最佳化 API 主機名稱做為 `distributionDomainName` 屬性值。您必須設定 DNS 記錄，將邊緣最佳化自訂網域名稱指向此分佈網域名稱。如此可讓繫結至邊緣最佳化自訂網域名稱的流量路由至邊緣最佳化 API 主機名稱。

設定 DNS 記錄之後，您可以移除自訂網域名稱的 REGION 端點類型：

```
aws apigateway update-domain-name \
--domain-name api.example.com \
--patch-operations [ \
    {op:'remove', path:'/endpointConfiguration/types', value:'REGIONAL'}, \
    {op:'remove', path:'regionalCertificateArn'} \
]
```

此命令的結果類似下列輸出，而且只有邊緣最佳化網域名稱組態資料：

```
{  
    "certificateArn": "arn:aws:acm:us-east-1:738575810317:certificate/34a95aa1-77fa-427c-  
    aa07-3a88bd9f3c0a",  
    "certificateName": "edge-cert",  
    "certificateUploadDate": "2017-10-16T23:22:57Z",  
    "distributionDomainName": "d1frvgze7vy1bf.cloudfront.net",  
    "domainName": "regional.haymuto.com",  
    "endpointConfiguration": {  
        "types": "EDGE"  
    }  
}
```

## 從 API Gateway 匯出 REST API

使用 API Gateway 主控台或其他方式在 API Gateway 中建立和設定 REST API 之後，即可使用 API Gateway 匯出 API (Amazon API Gateway 控制服務的一部分) 將其匯出至 OpenAPI 檔案。您可以選擇在匯出的 OpenAPI 定義檔中包含 API Gateway 整合延伸和 Postman 延伸。

### Note

當使用 AWS CLI 匯出 API 時，請務必包含以下範例中所示的延伸模組參數，以確保包含 `x-amazon-apigateway-request-validator` 延伸模組：

```
aws apigateway get-export --parameters extensions='apigateway' --rest-api-id  
abcdefg123 --stage-name dev --export-type swagger latestswagger2.json
```

如果 API 的承載類型不是 `application/json`，則無法匯出 API。如果您嘗試，則會收到錯誤回應，指出找不到 JSON 內文模型。

## 匯出 REST API 的請求

使用匯出 API 時，您可以提交 GET 請求並將欲匯出的 API 指定為 URL 路徑的一部分，藉此匯出現有 REST API。請求 URL 的格式如下：

### OpenAPI 3.0

```
https://<host>/restapis/<restapi_id>/stages/<stage_name>/exports/oas30
```

### OpenAPI 2.0

```
https://<host>/restapis/<restapi_id>/stages/<stage_name>/exports/swagger
```

您可以附加 `extensions` 查詢字串，以指定要包含 API Gateway 延伸 (具有 `integration` 值) 還是 Postman 延伸 (具有 `postman` 值)。

此外，您也可以將 `Accept` 標頭設定為 `application/json` 或 `application/yaml`，分別接收 JSON 或 YAML 格式的 API 定義輸出。

如需使用 API Gateway 匯出 API 提交 GET 請求的詳細資訊，請參閱[提出 HTTP 請求](#)。

Note

如果您在 API 中定義模型，則模型必須適用於內容類型 "application/json"，API Gateway 才會匯出模型。否則，API Gateway 會擲回例外狀況，而且錯誤訊息為「只發現 ... 的非 JSON 內文模型」。

模型必須包含屬性，或是定義為特定 JSONSchema 類型。

## 下載 JSON 格式的 REST API OpenAPI 定義

以 JSON 格式的 OpenAPI 定義匯出並下載 REST API：

OpenAPI 3.0

```
GET /restapis/<restapi_id>/stages/<stage_name>/exports/oas30
Host: apigateway.<region>.amazonaws.com
Accept: application/json
```

OpenAPI 2.0

```
GET /restapis/<restapi_id>/stages/<stage_name>/exports/swagger
Host: apigateway.<region>.amazonaws.com
Accept: application/json
```

例如，在這裡，<region> 可以是 us-east-1。如需 API Gateway 可用的所有區域，請參閱[區域和端點](#)

## 下載 YAML 格式的 REST API OpenAPI 定義

以 YAML 格式的 OpenAPI 定義匯出並下載 REST API：

OpenAPI 3.0

```
GET /restapis/<restapi_id>/stages/<stage_name>/exports/oas30
Host: apigateway.<region>.amazonaws.com
Accept: application/yaml
```

OpenAPI 2.0

```
GET /restapis/<restapi_id>/stages/<stage_name>/exports/swagger
Host: apigateway.<region>.amazonaws.com
Accept: application/yaml
```

## 下載 JSON 格式且具有 Postman 延伸的 REST API OpenAPI 定義

以 JSON 格式的 OpenAPI 定義搭配 Postman 匯出並下載 REST API：

OpenAPI 3.0

```
GET /restapis/<restapi_id>/stages/<stage_name>/exports/oas30?extensions=postman
Host: apigateway.<region>.amazonaws.com
Accept: application/json
```

OpenAPI 2.0

```
GET /restapis/<restapi_id>/stages/<stage_name>/exports/swagger?extensions=postman
Host: apigateway.<region>.amazonaws.com
Accept: application/json
```

## 下載 YAML 格式且具有 API Gateway 整合的 REST API OpenAPI 定義

以 YAML 格式的 OpenAPI 定義搭配 API Gateway 整合匯出並下載 REST API：

OpenAPI 3.0

```
GET /restapis/<restapi_id>/stages/<stage_name>/exports/oas30?extensions=integrations
Host: apigateway.<region>.amazonaws.com
Accept: application/yaml
```

OpenAPI 2.0

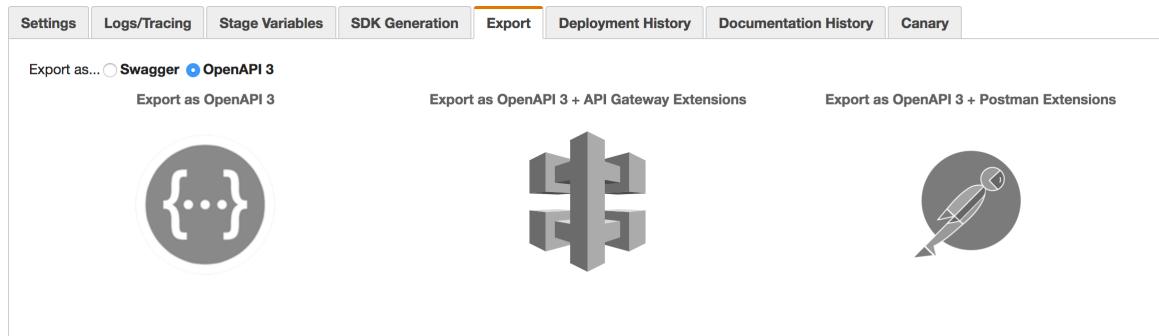
```
GET /restapis/<restapi_id>/stages/<stage_name>/exports/swagger?extensions=integrations
Host: apigateway.<region>.amazonaws.com
Accept: application/yaml
```

## 使用 API Gateway 主控台匯出 REST API

將 REST API 部署至階段 (p. 459) 之後，即可繼續使用 API Gateway 主控台，將階段中的 API 匯出至 OpenAPI 檔案。

從 API Gateway 主控台的 stage configuration (階段組態) 頁面中，選擇 Export (匯出) 標籤，然後選擇其中一個可用的選項 (Export as OpenAPI (匯出為 OpenAPI)、Export as OpenAPI + API Gateway Integrations

(匯出為 OpenAPI + API Gateway 整合) 和 Export as Postman (匯出為 Postman)) 來下載 API 的 OpenAPI 定義。



## 設定 API Gateway Canary Release 部署

[Canary Release](#) 是一種軟體開發策略，其中，新版本的 API (和其他軟體) 會部署為 Canary Release 以進行測試，而且基本版本仍會部署為相同階段上正常操作的生產版本。為了進行討論，我們在這份文件中將基本版本參照為生產版本。雖然這十分合理，但是您可以在任何非生產版本上自由地套用 Canary Release 來進行測試。

在 Canary Release 部署中，總 API 流量會隨機分為生產版本以及具有預先設定比率的 Canary Release。一般而言，Canary Release 會收到一小部分的 API 流量，而生產版本則會佔用其餘流量。透過 Canary，只有 API 流量才能看到更新過的 API 功能。您可以調整 Canary 流量百分比，以最佳化測試涵蓋範圍或效能。

透過保持較小的 Canary 流量以及隨機選取，新版本中的潛在錯誤在任何時間都不會對使用者造成嚴重影響，而且隨時都不會對單一使用者造成嚴重影響。

測試指標通過您的需求之後，即可將 Canary Release 提升至生產版本，並停用 Canary 進行部署。這讓新功能可在生產階段中使用。

### 主題

- [API Gateway 中的 Canary Release 部署 \(p. 607\)](#)
- [建立 Canary Release 部署 \(p. 608\)](#)
- [更新 Canary Release \(p. 612\)](#)
- [提升 Canary Release \(p. 614\)](#)
- [停用 Canary Release \(p. 616\)](#)

## API Gateway 中的 Canary Release 部署

在 API Gateway 中，Canary Release 部署會使用 API 基本版本之生產版本的部署階段，並連接至 API 新版本之 Canary Release 的階段 (相較於基本版本)。階段是與初始部署以及具有後續部署的 Canary 建立關聯。一開始，階段和 Canary 都指向相同的 API 版本。在本節中，我們會交換使用階段和生產版本，並交換使用 Canary 和 Canary Release。

若要部署具有 Canary Release 的 API，請將 [Canary 設定](#)新增至一般部署的階段，以建立 Canary Release 部署。Canary 設定說明基礎 Canary Release，而階段代表此部署內 API 的生產版本。若要新增 Canary 設定，請在部署階段上設定 `canarySettings`，並指定下列項目：

- 部署 ID，一開始與階段上設定的基本版本部署的 ID 完全相同。
- Canary Release 的 [API 流量百分比](#)，介於 0.0 與 100.0 (含) 之間。

- Canary Release 的階段變數，可以覆寫生產版本階段變數。
- 使用階段快取 (屬於 Canary 請求)，如果設定 `useStageCache` 並在階段上啟用 API 快取。

啟用 Canary Release 之後，除非停用 Canary Release 並從階段移除 Canary 設定，否則部署階段無法與其他非 Canary Release 部署建立關聯。

當您啟用 API 執行記錄時，Canary Release 會有針對所有 Canary 請求產生的專屬日誌和指標。會將它們報告到生產階段 CloudWatch Logs 日誌群組，以及 Canary 特定 CloudWatch Logs 日誌群組。這也適用於存取記錄。不同的 Canary 特定日誌適用於驗證新的 API 變更，以及決定是否接受變更，並將 Canary Release 提升至生產階段，或是捨棄變更，並從生產階段還原 Canary Release。

生產階段執行日誌群組命名為 `API-Gateway-Execution-Logs/{rest-api-id}/{stage-name}`，而 Canary Release 執行日誌群組命名為 `API-Gateway-Execution-Logs/{rest-api-id}/{stage-name}/Canary`。對於存取記錄，您必須建立新的日誌群組，或選擇現有日誌群組。Canary Release 存取日誌群組名稱會有附加至所選取日誌群組名稱的 `/Canary` 尾碼。

在預先設定的存活期 (TTL) 內，Canary Release 可以使用階段快取 (啟用時) 存放回應，以及使用快取項目將結果傳回至下一個 Canary 請求。

在 Canary Release 部署中，API 的生產版本和 Canary Release 可以與相同的版本或不同的版本建立關聯。當它們與不同版本建立關聯時，會分別快取生產和 Canary 請求的回應，而且階段快取會傳回生產和 Canary 請求的對應結果。當生產版本和 Canary Release 與相同部署建立關聯時，階段快取會將單一快取金鑰用於這兩種類型的請求，並傳回生產版本和 Canary Release 中相同請求的相同回應。

## 建立 Canary Release 部署

部署將 [Canary 設定](#) 做為部署建立操作額外輸入的 API 時，請建立 Canary Release 部署。

您也可以提出在階段上新增 Canary 設定的 `stage:update` 請求，以從現有非 Canary 部署建立 Canary Release 部署。

建立非 Canary Release 部署時，您可以指定非現有階段名稱。如果指定的階段不存在，則 API Gateway 會建立階段。不過，建立 Canary Release 部署時，您無法指定任何非現有階段名稱。您將會收到錯誤，而且 API Gateway 不會建立 Canary Release 部署。

您可以使用 API Gateway 主控台、AWS CLI 或 AWS 開發套件，在 API Gateway 中建立 Canary Release 部署。

### 主題

- [使用 API Gateway 主控台建立 Canary 部署 \(p. 608\)](#)
- [使用 AWS CLI 建立 Canary 部署 \(p. 609\)](#)

## 使用 API Gateway 主控台建立 Canary 部署

若要使用 API Gateway 主控台來建立 Canary Release 部署，請遵循下面的說明：

### 建立初始 Canary Release 部署

1. 登入 API Gateway 主控台。
2. 選擇現有 API，或建立新的 API。
3. 變更 API (必要時)，或設定所需的 API 方法和整合。
4. 從 Actions (動作) 下拉式選單中，選擇 Deploy API (部署 API)。請遵循 Deploy API (部署 API) 中的畫面說明，將 API 部署至新階段。

到目前為止，您已將 API 部署至生產版本階段。接著，您在階段上設定 Canary 設定，如果需要，也可以啟用快取、設定階段變數，或設定 API 執行或存取日誌。

5. 若要啟用 API 快取，請選擇 Stage Editor (階段編輯器) 中的 Settings (設定) 標籤，並遵循畫面說明。如需詳細資訊，請參閱[the section called “啟用 API 快取” \(p. 463\)](#)。
6. 若要設定階段變數，請選擇 Stage Editor (階段編輯器) 中的 Stage Variables (階段變數) 標籤，並遵循畫面說明來新增或修改階段變數。如需詳細資訊，請參閱[the section called “設定 REST API 的階段變數” \(p. 475\)](#)。
7. 若要設定執行或存取記錄，請選擇 Stage Editor (階段編輯器) 中的 Logs (日誌) 標籤，並遵循畫面說明。如需詳細資訊，請參閱[在 API Gateway 中設定 CloudWatch API 記錄 \(p. 472\)](#)。
8. 在 Stage Editor (階段編輯器) 中，選擇 Canary 標籤，然後選擇 Create Canary (建立 Canary)。
9. 在 Stage's Request Distribution (階段的請求分佈) 區段下，選擇 Percentage of requests to Canary (Canary 請求百分比) 旁邊的鉛筆圖示，並在輸入文字欄位中輸入數字 (例如，5.0)。選擇核取記號圖示來儲存設定。
10. 若要將 AWS WAF web ACL 與該階段建立關聯，請從 Web ACL 下拉式清單中選擇 web ACL。

#### Note

如果有需要，選擇 Create Web ACL (建立 Web ACL)，即可在新的瀏覽器分頁中開啟 AWS WAF 主控台，並返回 API Gateway 主控台來建立 web ACL 與階段的關聯。

11. 如有需要，選擇 Block API Request if WebACL cannot be evaluated (Fail- Close) (如果無法評估 WebACL，則阻擋 API 請求 (失敗 - 關閉))。
12. 如有需要，請選擇 Add Stage Variables (新增階段變數)，將它們新增至 Canary Stage Variables (Canary 階段變數) 區段，以覆寫現有階段變數或是新增 Canary Release 的新階段變數。
13. 如有需要，請選擇 Enable use of stage cache (啟用階段快取)，以啟用 Canary Release 的快取並儲存您的選擇。除非啟用 API 快取，否則快取不適用於 Canary Release。

在部署階段初始化 Canary Release 之後，您變更 API 而且想要測試變更。您可以將 API 重新部署至相同階段，以透過相同的階段存取更新過的版本和基本版本。下列步驟說明如何執行這項操作。

#### 將最新的 API 版本部署至 Canary

1. 使用 API 的每個更新，從 Resources (資源) 清單旁邊的 Actions (動作) 下拉式選單中選擇 Deploy API (部署 API)。
2. 在 Deploy API (部署 API) 中，從 Deployment stage (部署階段) 下拉式清單中選擇現在已啟用 Canary 的階段。
3. (選擇性) 在 Deployment description (部署說明) 中輸入說明。
4. 選擇 Deploy (部署)，將最新的 API 版本推送至 Canary Release。
5. 如有需要，請重新設定階段設定、日誌或 Canary 設定，如「[建立初始 Canary Release 部署 \(p. 608\)](#)」中所述。

因此，Canary Release 會指向最新版本，而生產版本仍然指向 API 的初始版本。`canarySettings` 現在有新的 `deploymentId` 值，而階段仍然有初始 `deploymentId` 值。在幕後，主控台會呼叫 `stage:update`。

## 使用 AWS CLI 建立 Canary 部署

先建立具有兩個階段變數的基準部署，但不需要任何 Canary：

```
aws apigateway create-deployment
--variables sv0=val0,sv1=val1
--rest-api-id 4wk1k4onj3
--stage-name prod
```

此命令會傳回所產生 Deployment 的呈現，與下列類似：

```
{  
    "id": "du4ot1",  
    "createdDate": 1511379050  
}
```

產生的部署 id 識別 API 的快照 (或版本)。

現在在 prod 階段上建立 Canary 部署：

```
aws apigateway create-deployment  
    --canary-settings '{ \  
        "percentTraffic":10.5, \  
        "useStageCache":false, \  
        "stageVariableOverrides":{ \  
            "sv1":"val2", \  
            "sv2":"val3" \  
        } \  
    }' \  
    --rest-api-id 4wk1k4onj3 \  
    --stage-name prod
```

如果指定的階段 (prod) 不存在，上述命令會傳回錯誤。否則，它會傳回新建的部署資源呈現，與下列類似：

```
{  
    "id": "a6rox0",  
    "createdDate": 1511379433  
}
```

產生的部署 id 識別 Canary Release 的 API 測試版本。因此，相關聯的階段是已啟用 Canary。您可以呼叫 get-stage 命令來檢視此階段呈現，與下列類似：

```
aws apigateway get-stage --rest-api-id 4wk1k4onj3 --stage-name prod
```

以下顯示 Stage 作為命令輸出的呈現：

```
{  
    "stageName": "prod",  
    "variables": {  
        "sv0": "val0",  
        "sv1": "val1"  
    },  
    "cacheClusterEnabled": false,  
    "cacheClusterStatus": "NOT_AVAILABLE",  
    "deploymentId": "du4ot1",  
    "lastUpdatedDate": 1511379433,  
    "createdDate": 1511379050,  
    "canarySettings": {  
        "percentTraffic": 10.5,  
        "deploymentId": "a6rox0",  
        "useStageCache": false,  
        "stageVariableOverrides": {  
            "sv2": "val3",  
            "sv1": "val2"  
        }  
    },  
    "methodSettings": {}  
}
```

}

在此範例中，API 基本版本會使用階段變數 `{"sv0":val0", "sv1":val1"}`，而測試版本使用階段變數 `{"sv1":val2", "sv2":val3"}`。生產版本和 Canary Release 使用相同的階段變數 sv1，但分別具有不同的值 val1 和 val2。階段變數 sv0 單獨用於生產版本，而階段變數 sv2 用於 Canary Release 中。

您可以更新階段以啟用 Canary，以從現有一般部署建立 Canary Release 部署。為了示範這項操作，請先建立一般部署：

```
aws apigateway create-deployment \
--variables sv0=val0,sv1=val1 \
--rest-api-id 4wk1k4onj3 \
--stage-name beta
```

此命令會傳回基本版本部署呈現：

```
{
  "id": "cifeiw",
  "createdDate": 1511380879
}
```

相關聯的 Beta 階段沒有任何 Canary 設定：

```
{
  "stageName": "beta",
  "variables": {
    "sv0": "val0",
    "sv1": "val1"
  },
  "cacheClusterEnabled": false,
  "cacheClusterStatus": "NOT_AVAILABLE",
  "deploymentId": "cifeiw",
  "lastUpdatedDate": 1511380879,
  "createdDate": 1511380879,
  "methodSettings": {}
}
```

現在，在階段上連接 Canary，以建立新的 Canary Release 部署：

```
aws apigateway update-stage \
--patch-operations '[{ \
  "op":"replace", \
  "path":"/canarySettings/percentTraffic", \
  "value":"10.5" \
}, { \
  "op":"replace", \
  "path":"/canarySettings/useStageCache", \
  "value":"false" \
}, { \
  "op":"replace", \
  "path":"/canarySettings/stageVariableOverrides/sv1", \
  "value":"val2" \
}, { \
  "op":"replace", \
  "path":"/canarySettings/stageVariableOverrides/sv2", \
  "value":"val3" \
}]' \
--rest-api-id 4wk1k4onj3 \
--stage-name beta
```

更新過的階段呈現如下：

```
{  
    "stageName": "beta",  
    "variables": {  
        "sv0": "val0",  
        "sv1": "val1"  
    },  
    "cacheClusterEnabled": false,  
    "cacheClusterStatus": "NOT_AVAILABLE",  
    "deploymentId": "cifeiw",  
    "lastUpdatedDate": 1511381930,  
    "createdDate": 1511380879,  
    "canarySettings": {  
        "percentTraffic": 10.5,  
        "deploymentId": "cifeiw",  
        "useStageCache": false,  
        "stageVariableOverrides": {  
            "sv2": "val3",  
            "sv1": "val2"  
        }  
    },  
    "methodSettings": {}  
}
```

因為我們只會在現有 API 版本上啟用 Canary，所以生產版本 (Stage) 和 Canary Release (canarySettings) 指向相同的部署，即 API 的相同版本 (deploymentId)。在您變更 API 並將它再次部署至此階段之後，新的版本將會在 Canary Release 中，而基本版本仍然在生產版本中。Canary Release 中的 deploymentId 更新為新部署 id 而生產版本中的 deploymentId 保持不變時，這是在階段發展中體現。

## 更新 Canary Release

部署 Canary Release 之後，建議您調整 Canary 流量百分比，或者啟用或停用階段快取來最佳化測試效能。更新執行內容時，您也可以修改 Canary Release 中所使用的階段變數。若要進行這類更新，請呼叫 canarySettings 上具有新值的 stage:update 操作。

您可以使用 API Gateway 主控台、AWS CLI update-stage 命令和 AWS 開發套件來更新 Canary Release。

### 主題

- [使用 API Gateway 主控台更新 Canary Release \(p. 612\)](#)
- [使用 AWS CLI 更新 Canary Release \(p. 613\)](#)

## 使用 API Gateway 主控台更新 Canary Release

若要使用 API Gateway 主控台來更新階段上的現有 Canary 設定，請執行下列操作：

1. 登入 API Gateway 主控台，然後在主導覽窗格中選擇現有 API。
2. 選擇 API 下的 Stages (階段)，然後選擇 Stages (階段) 清單下的現有階段來開啟 Stage Editor (階段編輯器)。
3. 在 Stage Editor (階段編輯器) 中，選擇 Canary 標籤。
4. 增加或減少 0.0 和 100.0 (含) 百分比數字，以更新 Percentage of requests directed to Canary (導向 Canary 的請求百分比)。
5. 更新 Canary Stage Variables (Canary 階段變數)，包含新增、移除或修改所需的階段變數。
6. 選取或清除核取方塊，以更新 Enable use of stage cache (啟用階段快取) 選項。
7. 儲存變更。

## 使用 AWS CLI 更新 Canary Release

若要使用 AWS CLI 更新 Canary，請呼叫 [update-stage](#) 命令。

若要啟用或停用 Canary 的階段快取，請呼叫 [update-stage](#) 命令，如下所示：

```
aws apigateway update-stage \
--rest-api-id {rest-api-id} \
--stage-name '{stage-name}' \
--patch-operations op=replace,path=/canarySettings/useStageCache,value=true
```

若要調整 Canary 流量百分比，請呼叫 [update-stage](#) 來取代階段上的 /canarySettings/percentTraffic 值。

```
aws apigateway update-stage \
--rest-api-id {rest-api-id} \
--stage-name '{stage-name}' \
--patch-operations op=replace,path=/canarySettings/percentTraffic,value=25.0
```

更新 Canary 階段變數，包含新增、取代或移除 Canary 階段變數：

```
aws apigateway update-stage \
--rest-api-id {rest-api-id} \
--stage-name '{stage-name}' \
--patch-operations '[{
    "op": "replace",
    "path": "/canarySettings/stageVariableOverrides/newVar",
    "value": "newVal",
},
{
    "op": "replace",
    "path": "/canarySettings/stageVariableOverrides/var2",
    "value": "val4",
},
{
    "op": "remove",
    "path": "/canarySettings/stageVariableOverrides/var1"
}]'
```

您可以將操作結合為單一 patch-operations 值，來更新上述所有項目：

```
aws apigateway update-stage \
--rest-api-id {rest-api-id} \
--stage-name '{stage-name}' \
--patch-operations '[{
    "op": "replace",
    "path": "/canary/percentTraffic",
    "value": "20.0"
},
{
    "op": "replace",
    "path": "/canary/useStageCache",
    "value": "true"
},
{
    "op": "remove",
    "path": "/canarySettings/stageVariableOverrides/var1"
},
{
    "op": "replace",
    "path": "/canarySettings/stageVariableOverrides/newVar",
    "value": "newVal"
},
{
    "op": "replace",
    "path": "/canarySettings/stageVariableOverrides/val2",
    "value": "val2"
}]'
```

```
        "value": "val4"
    }]' \
```

## 提升 Canary Release

若要提升 Canary Release，請讓進行測試的 API 版本可在生產階段中使用。此操作包含下列任務：

- 重設具有 Canary 部署 ID 設定之階段的部署 ID。這會更新具有 Canary 快照之階段的 API 快照，同時讓測試版本成為生產版本。
- 使用 Canary 階段變數更新階段變數 (如果有的話)。這會更新具有 Canary API 執行內容之階段的 API 執行內容。沒有這項更新時，如果測試版本使用不同的階段變數或不同的現有階段變數值，則新的 API 版本可能會產生意外的結果。
- 將 Canary 流量百分比設定為 0.0%。

提升 Canary Release 並不會停用階段上的 Canary。若要停用 Canary，您必須移除階段上的 Canary 設定。

### 主題

- [使用 API Gateway 主控台提升 Canary Release \(p. 614\)](#)
- [使用 AWS CLI 提升 Canary Release \(p. 614\)](#)

## 使用 API Gateway 主控台提升 Canary Release

若要使用 API Gateway 主控台來提升 Canary Release 部署，請執行下列操作：

1. 登入 API Gateway 主控台，然後在主導覽窗格中選擇現有 API。
2. 選擇 API 下的 Stages (階段)，然後選擇 Stages (階段) 清單下的現有階段來開啟 Stage Editor (階段編輯器)。
3. 在 Stage Editor (階段編輯器) 中，選擇 Canary 標籤。
4. 選擇 Promote Canary (提升 Canary)。
5. 確認要進行的變更，然後選擇 Update (更新)。

在提升之後，生產版本會參考相同的 API 版本 (deploymentId) 作為 Canary Release。您可以使用 AWS CLI 進行驗證。如需範例，請參閱 [the section called “使用 AWS CLI 提升 Canary Release” \(p. 614\)](#)。

## 使用 AWS CLI 提升 Canary Release

若要使用 AWS CLI 命令將 Canary Release 提升為生產版本，請呼叫 update-stage 命令將 Canary 相關聯的 deploymentId 複製至與階段相關聯的 deploymentId、將 Canary 流量百分比重設為零 (0.0)，以及將任何 Canary 繫結階段變數複製至對應的階段繫結階段變數。

假設我們有 Canary Release 部署，如與以下類似的階段所描述：

```
{
  "_links": {
    ...
  },
  "accessLogSettings": {
    ...
  },
  "cacheClusterEnabled": false,
  "cacheClusterStatus": "NOT_AVAILABLE",
  "canarySettings": {
    "deploymentId": "eh1sby",
```

```
"useStageCache": false,  
"stageVariableOverrides": {  
    "sv2": "val3",  
    "sv1": "val2"  
},  
"percentTraffic": 10.5  
},  
"createdDate": "2017-11-20T04:42:19Z",  
"deploymentId": "nfcn0x",  
"lastUpdatedDate": "2017-11-22T00:54:28Z",  
"methodSettings": {  
    ...  
},  
"stageName": "prod",  
"variables": {  
    "sv1": "val1"  
}  
}
```

我們呼叫下列 update-stage 請求將其提升：

```
aws apigateway update-stage  
--rest-api-id {rest-api-id}  
--stage-name '{stage-name}'  
--patch-operations '[{  
    "op": "replace",  
    "value": "0.0"  
    "path": "/canarySettings/percentTraffic",  
}, {  
    "op": "copy",  
    "from": "/canarySettings/stageVariableOverrides",  
    "path": "/variables",  
}, {  
    "op": "copy",  
    "from": "/canarySettings/deploymentId",  
    "path": "/deploymentId"  
}]'
```

提升後，該階段現如下所示：

```
{  
    "_links": {  
        ...  
    },  
    "accessLogSettings": {  
        ...  
    },  
    "cacheClusterEnabled": false,  
    "cacheClusterStatus": "NOT_AVAILABLE",  
    "canarySettings": {  
        "deploymentId": "ehlsby",  
        "useStageCache": false,  
        "stageVariableOverrides": {  
            "sv2": "val3",  
            "sv1": "val2"  
        },  
        "percentTraffic": 0  
    },  
    "createdDate": "2017-11-20T04:42:19Z",  
    "deploymentId": "ehlsby",  
    "lastUpdatedDate": "2017-11-22T05:29:47Z",  
    "methodSettings": {  
        ...  
    }  
}
```

```
},
"stageName": "prod",
"variables": {
    "sv2": "val3",
    "sv1": "val2"
}
}
```

如您所見，將 Canary Release 提升到該階段不會停用 Canary，而部署仍然是 Canary Release 部署。為了讓該部署成為一般生產版本部署，您必須停用 Canary 設定。如需有關如何停用 Canary Release 部署的詳細資訊，請參閱 [the section called “停用 Canary Release” \(p. 616\)](#)。

## 停用 Canary Release

停用 Canary Release 部署是將 `canarySettings` 設定為 `null`，以將它從階段中移除。

您可以使用 API Gateway 主控台、AWS CLI 或 AWS 開發套件，以停用 Canary Release 部署。

### 主題

- [使用 API Gateway 主控台停用 Canary Release \(p. 616\)](#)
- [使用 AWS CLI 停用 Canary Release \(p. 616\)](#)

## 使用 API Gateway 主控台停用 Canary Release

若要使用 API Gateway 主控台來停用 Canary Release 部署，請使用下列步驟：

1. 登入 API Gateway 主控台，然後在主導覽窗格中選擇現有 API。
2. 選擇 API 下的 Stages (階段)，然後選擇 Stages (階段) 清單下的現有階段來開啟 Stage Editor (階段編輯器)。
3. 在 Stage Editor (階段編輯器) 中，選擇 Canary 標籤。
4. 選擇 Delete Canary (刪除 Canary)。
5. 選擇 Delete (刪除)，確認您要刪除 Canary。

因此，`canarySettings` 屬性會成為 `null`，並從部署階段中予以移除。您可以使用 AWS CLI 進行驗證。如需範例，請參閱 [the section called “使用 AWS CLI 停用 Canary Release” \(p. 616\)](#)。

## 使用 AWS CLI 停用 Canary Release

若要使用 AWS CLI 來停用 Canary Release 部署，請呼叫 `update-stage` 命令，如下所示：

```
aws apigateway update-stage \
--rest-api-id 4wk1k4onj3 \
--stage-name canary \
--patch-operations '[{"op":"remove", "path":"/canarySettings"}]
```

成功回應會傳回與下列類似的承載：

```
{
    "stageName": "prod",
    "accessLogSettings": {
        ...
    },
    "cacheClusterEnabled": false,
    "cacheClusterStatus": "NOT_AVAILABLE",
    "deploymentId": "nfcn0x",
```

```
"lastUpdatedDate": 1511309280,  
"createdDate": 1511152939,  
"methodSettings": {  
    ...  
}  
}
```

如輸出中所示，[canarySettings](#) 屬性於已停用 Canary 部署的階段中不再存在。

# 使用 AWS Config 監控 API Gateway API 組態

您可以使用 [AWS Config](#)，來記錄對 API Gateway API 資源所做的組態變更，並根據資源變更傳送通知。保留 API Gateway 資源的組態變更歷史記錄很實用，可用於解決操作問題、稽核和合規使用案例。

AWS Config 可以追蹤對下列組態所做的變更：

- API 階段組態，例如：
  - 快取叢集設定
  - 調節設定
  - 存取日誌設定
  - 階段上使用中的部署設定
- API 組態，例如：
  - 端點組態
  - version
  - 通訊協定
  - tags

此外，AWS Config Rules 功能可讓您定義組態規則，以及自動偵測、追蹤和提醒這些規則的違規。透過追蹤這些資源組態屬性的變更，您也可以為 API Gateway 資源撰寫變更觸發的 AWS Config 規則，並根據最佳實務測試您的資源組態。

您可以使用 AWS Config 主控台或 AWS CLI，在您的帳戶中啟用 AWS Config。選取您要追蹤變更的資源類型。如果您之前已將 AWS Config 設定為記錄所有資源類型，則這些 API Gateway 資源會自動記錄在您的帳戶中。AWS Config 中的 Amazon API Gateway 支援功能已在所有 AWS 公有區域和 AWS GovCloud (US) 提供使用。如需受支援區域的完整清單，請參閱 AWS General Reference 中的 [區域與端點](#)。

## 主題

- [支援的資源類型 \(p. 618\)](#)
- [設定 AWS Config \(p. 619\)](#)
- [設定 AWS Config 以記錄 API Gateway 資源 \(p. 619\)](#)
- [在 AWS Config 主控台檢視 API Gateway 組態詳細資訊 \(p. 619\)](#)
- [使用 AWS Config 規則評估 API Gateway 資源 \(p. 620\)](#)

## 支援的資源類型

以下 API Gateway 資源類型會與 AWS Config 整合，並記載於 [AWS Config 支援的 AWS 資源類型和資源關係](#) 中：

- [AWS::ApiGatewayV2::Api \(WebSocket API\)](#)
- [AWS::ApiGateway::RestApi \(REST API\)](#)
- [AWS::ApiGatewayV2::Stage \(WebSocket API 階段\)](#)
- [AWS::ApiGateway::Stage \(REST API 階段\)](#)

如需 AWS Config 的詳細資訊，請參閱 [AWS Config Developer Guide](#)。如需定價資訊，請參閱[AWS Config 定價資訊頁面](#)。

**Important**

如果您在部署 API 之後變更任何以下 API 屬性，則必須[重新部署 \(p. 457\)](#) API 以傳播變更。否則，您將會在 AWS Config 主控台中看到屬性變更，但先前的屬性設定仍然有效；API 的執行時間行為將保持不變。

- **AWS::ApiGateway::RestApi** –  
binaryMediaTypes、minimumCompressionSize、apiKeySource
- **AWS::ApiGatewayV2::Api** – apiKeySelectionExpression

## 設定 AWS Config

若要初始設定 AWS Config，請參閱下列 [AWS Config Developer Guide](#) 的主題。

- [使用主控台設定 AWS Config](#)
- [使用 AWS CLI 設定 AWS Config](#)

## 設定 AWS Config 以記錄 API Gateway 資源

在預設情況下，AWS Config 會記錄所有區域資源支援類型的組態變更，它會探索環境執行所在的區域。您可以自訂 AWS Config 以記錄變更，其僅適用於特定的資源類型，或全域資源的變化。

若要了解區域與全域資源，以及了解如何自訂您的 AWS Config 組態，請參閱[選取 AWS Config 記錄的資源](#)。

## 在 AWS Config 主控台檢視 API Gateway 組態詳細資訊

您可以使用 AWS Config 主控台尋找 API Gateway 資源，並取得其組態的目前和歷史詳細資訊。以下程序顯示如何尋找 API Gateway API 的相關資訊。

### 在 AWS Config 主控台尋找 API Gateway 資源

1. 開啟 [AWS Config 主控台](#)。
2. 選擇 Resources (資源)。
3. 在 Resource (資源) 清單頁面上，選擇 Resources (資源)。
4. 開啟 Resource type (資源類型) 功能表，捲動到 APIGateway 或 APIGatewayV2，然後選擇一或多個 API Gateway 資源類型。
5. 選擇 Look up (查閱)。
6. 在 AWS Config 顯示的資源清單中選擇資源 ID。AWS Config 會顯示組態詳細資訊，以及其他有關您所選取資源的資訊。
7. 若要查看記錄組態的完整詳細資訊，請選擇 View Details (檢視詳細資訊)。

若要進一步了解尋找資源以及檢視此頁面的資訊，請參閱《AWS Config Developer Guide》中的[檢視 AWS 資源組態和歷史記錄](#)。

## 使用 AWS Config 規則評估 API Gateway 資源

您可以建立 AWS Config 規則，這代表您的 API Gateway 資源的理想組態設定。您可以使用預先定義的 [AWS Config 受管規則](#)，或是定義自訂規則。AWS Config 會持續追蹤您資源組態的變更，以判斷變更是否會違反您的規則條件。AWS Config 主控台會顯示您規則和資源的合規狀態。

如果資源違反規則，並標記為不合規，AWS Config 會使用 [Amazon Simple Notification Service Developer Guide](#) (Amazon SNS) 主題提醒您。若要透過程式設計方式使用這些 AWS Config 提醒資料，請使用 Amazon Simple Queue Service (Amazon SQS) 併列做為 Amazon SNS 主題的通知端點。

若要進一步了解設定及使用規則，請參閱《[AWS Config Developer Guide](#)》中的[使用規則評估資源](#)。

# 標記 API Gateway 資源

標籤是您或 AWS 指派給 AWS 資源的中繼資料標籤。每個標籤有兩個部分：

- 標籤鍵 (例如，CostCenter、Environment 或 Project)。標籤鍵會區分大小寫。
- 一個選用欄位，稱為標籤值 (例如 111122223333 或 Production)。忽略標籤值基本上等同於使用空字串。與標籤鍵相同，標籤值會區分大小寫。

標籤可協助您執行以下操作：

- 根據資源獲派的標籤，控制資源的存取權。您可以透過在 AWS Identity and Access Management (IAM) 政策條件中指定標籤金鑰和值，控制存取。如需有關以標籤為基礎的存取控制的詳細資訊，請參閱 IAM 使用者指南中的[使用標籤控制存取](#)。
- 追蹤您的 AWS 成本。您會在 AWS Billing and Cost Management 儀表板上啟用這些標籤。AWS 會使用標籤來分類您的成本，並提供每月成本分配報告給您。如需詳細資訊，請參閱 [AWS Billing and Cost Management User Guide](#) 中的[使用成本分配標籤](#)。
- 識別和組織您的 AWS 資源。許多 AWS 服務支援標籤，因此您可以對來自不同服務的資源指派相同的標籤，以指出資源是相關的。例如，您可以將您指派給 CloudWatch Events 規則的相同標籤指派給 API Gateway 階段。

如需使用標籤的提示，請參閱 AWS Answers 部落格上的 [AWS 標籤策略](#) 文章。

目前標記僅適用於 REST API 資源，而不適用於 WebSocket API 資源。

下列各節提供 Amazon API Gateway 標籤的詳細資訊。

## 主題

- [可以標記的API Gateway 資源 \(p. 621\)](#)
- [使用標籤來控制對 API Gateway 資源的存取 \(p. 623\)](#)

## 可以標記的API Gateway 資源

目前標記僅適用於 [Amazon API Gateway V1 API](#) 中的 REST API 中的資源。

可在以下資源上設定標籤：

- ApiKey
- ClientCertificate
- DomainName
- RestApi
- Stage
- UsagePlan
- VpcLink

無法直接在其他資源上設定標籤。不過，子資源會繼承父資源上設定的標籤。例如：

- 如果在 RestApi 資源上設定標籤，該 RestApi 的以下子資源會繼承該標籤：
  - Authorizer

- Deployment
  - Documentation
  - GatewayResponse
  - Integration
  - Method
  - Model
  - Resource
  - ResourcePolicy
  - Setting
  - Stage
- 如果在 DomainName 上設定標籤，它之下的任何 BasePathMapping 資源會繼承該標籤。
  - 如果在 UsagePlan 上設定標籤，它之下的任何 UsagePlanKey 資源會繼承該標籤。

## 標籤繼承

在過去，只能在階段上設定標籤。現在，您也可以在其他資源上設定標籤，Stage 可以透過兩種方式接收標籤：

- 可以直接在 Stage 上設定的標籤。
- 階段可以繼承其父系 RestApi 的標籤。

如果階段以這兩種方式接收標籤，則以直接在階段上設定的標籤為優先。例如，假設階段從其父系 REST API 繼承以下標籤：

```
{  
  'foo': 'bar',  
  'x':'y'  
}
```

假設它本身也直接設定以下標籤：

```
{  
  'foo': 'bar2',  
  'hello': 'world'  
}
```

最後結果是階段會有以下標籤和以下的值：

```
{  
  'foo': 'bar2',  
  'hello': 'world'  
  'x':'y'  
}
```

## 標籤限制和使用慣例

下列限制和使用慣例適用於搭配 API Gateway 資源使用標籤：

- 目前標記僅適用於 REST API 資源，而不適用於 WebSocket API 資源。
- 每個資源的上限為 50 個標籤。
- 對於每一個資源，每個標籤金鑰必須是唯一的，且每個標籤金鑰只能有一個值。

- 最大標籤索引鍵長度為 128 個 UTF-8 形式的 Unicode 字元。
- 最大標籤值長度為 256 個 UTF-8 形式的 Unicode 字元。
- 索引和值允許的字元包括可用 UTF-8 表示的英文字母、數字、空格，還有以下字元 : . : + = @ \_ / - (連字號)。Amazon EC2 資源允許任何字元。
- 標籤索引鍵和值皆區分大小寫。做為最佳實務，請決定大寫標籤的策略，並一致地在所有資源類型中實作該策略。例如，決定要使用 Costcenter、costcenter 還是 CostCenter，並針對所有標籤使用相同的慣例。避免針對相似的標籤使用不一致的大小寫處理。
- 標籤禁止使用 aws: 字首，它保留給 AWS 使用。您不可編輯或刪除具此字首的標籤索引鍵或值。具此字首的標籤，不算在受資源限制的標籤計數內。

## 使用標籤來控制對 API Gateway 資源的存取

AWS Identity and Access Management 政策中的條件是您指定對 API Gateway 資源的許可時所使用語法的一部分。如需有關指定 IAM 政策的詳細資訊，請參閱[the section called “使用 IAM 許可” \(p. 333\)](#)。在 API Gateway 中，資源可以擁有標籤，也有一些動作會包含標籤。在建立 IAM 政策時，可使用標記條件鍵來控制以下項目：

- 可在 API Gateway 資源上執行動作的使用者 (根據資源已具有的標籤)。
- 可在動作請求中傳遞的標籤。
- 請求中是否可使用特定的標籤索引鍵。

使用標籤型存取控制可達到比 API 層級控制更精確的控制，以及比資源型存取控制更動態的控制。您可以建立 IAM 政策，以根據請求中提供的標籤 (請求標籤) 或所操作資源上的標籤 (資源標籤)，決定允許或不允許操作。一般而言，資源標籤適用於已存在的資源。請求標籤適用於當您建立新的資源時。

關於標籤條件索引鍵的完整語法和語義，請參閱 IAM 使用者指南中的[使用標籤控制存取](#)。

以下範例顯示如何指定 API Gateway 使用者政策中的標籤條件。

### 範例 1：根據資源標籤限制動作

以下範例政策授與使用者在所有資源上執行 GET 動作的許可。此外，如果資源有一個標籤名為 iamrole，而值為 readWrite，此政策會授與使用者在資源上執行所有動作的許可。

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": "apigateway:GET",  
            "Resource": "*"  
        },  
        {  
            "Effect": "Allow",  
            "Action": "apigateway:*",  
            "Resource": "*",  
            "Condition": {  
                "StringEquals": {  
                    "aws:ResourceTag/iamrole": "readWrite"  
                }  
            }  
        }  
    ]  
}
```

## 範例 2：根據請求中的標籤限制動作

以下範例政策指定：

- 當使用者建立新的階段時，為了建立階段而提出的請求必須包含名為 stage 的標籤。
- stage 標籤的值必須是 beta、gamma 或 prod。否則會拒絕建立階段的請求。

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "apigateway:*"  
            ],  
            "Resource": [  
                "*"  
            ]  
        },  
        {  
            "Effect": "Deny",  
            "Action": "apigateway:POST",  
            "Resource": "arn:aws:apigateway:*/restapis/*/stages",  
            "Condition": {  
                "Null": {  
                    "aws:RequestTag/stage": "true"  
                }  
            }  
        },  
        {  
            "Effect": "Deny",  
            "Action": "apigateway:POST",  
            "Resource": "arn:aws:apigateway:*/restapis/*/stages",  
            "Condition": {  
                "ForAnyValue:StringNotEquals": {  
                    "aws:RequestTag/stage": [  
                        "beta",  
                        "gamma",  
                        "prod"  
                    ]  
                }  
            }  
        }  
    ]  
}
```

## 範例 3：根據資源標籤拒絕動作

以下範例政策預設允許使用者在 API Gateway 資源上執行所有動作。如果資源有一個標籤名為 stage，而值為 prod，則會拒絕使用者在資源上執行修改的許可 (PATCH、PUT、POST、DELETE)。

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": "apigateway:*,  
            "Resource": "*"  
        },  
        {  
            "Effect": "Deny",  
            "Action": "apigateway:PUT,  
            "Resource": "arn:aws:apigateway:*/restapis/*/  
        }  
    ]  
}
```

```
"Effect": "Deny",
"Action": [
    "apigateway:PATCH",
    "apigateway:PUT",
    "apigateway:POST",
    "apigateway:DELETE"
],
"Resource": "*",
"Condition": {
    "StringEquals": {
        "aws:ResourceTag/stage": "prod"
    }
}
]
}
```

## 範例 4：根據資源標籤允許動作

此範例政策預設允許使用者在所有 API Gateway 資源上執行所有動作。如果資源有一個標籤名為 environment，而值為 prod，則不允許使用者在資源上執行任何操作。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "apigateway:*",
            "Resource": "*"
        },
        {
            "Effect": "Deny",
            "Action": [
                "apigateway:*
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/environment": "prod"
                }
            }
        }
    ]
}
```

# Amazon API Gateway 版本 1 和版本 2 API 參考

Amazon API Gateway 提供的 API 可用來建立並部署您自己的 HTTP 和 WebSocket API。此外，標準的 AWS 開發套件也提供 API Gateway API。

如果您使用 AWS 開發套件擁有的語言，您可能會比較希望使用開發套件，而非直接使用 API Gateway REST API。開發套件可簡化身份驗證、與您的開發環境輕鬆整合，並可輕鬆存取 API Gateway 命令。

下列各處可看到 AWS 開發套件及 API Gateway REST API 參考文件：

- [Amazon Web Services 的工具](#) (適用於開發人員工具、開發套件和 IDE 工具組)
- [Amazon API Gateway 版本 1 API 參考](#) (適用於建立和部署 HTTP API)
- [Amazon API Gateway 版本 2 API 參考](#) (適用於建立和部署 WebSocket API)

# Amazon API Gateway 限制和重要說明

## 主題

- [API Gateway 限制 \(p. 627\)](#)
- [Amazon API Gateway 重要說明 \(p. 630\)](#)

## API Gateway 限制

除非另有說明，否則可以在請求時提高限制。若要請求提高限制，請與 [AWS Support 中心](#) 聯絡。

在方法上啟用授權時，方法 ARN 的長度上限（例如 `arn:aws:execute-api:{region-id}:{account-id}:{api-id}/{stage-id}/{method}/{resource}/{path}`）是 1600 個位元組。路徑參數值是在執行時間所決定的大小，可能會導致 ARN 長度超過限制。發生此情況時，API 用戶端會收到 414 Request URI too long 回應。

標頭值限制為 10240 個位元組。

## API Gateway 帳戶等級限制

Amazon API Gateway 內的帳戶等級適用下列限制。

資源或操作	預設限制	可以提高
每個區域所有 REST API、WebSocket API 及 WebSocket 回呼 API 的調節限制	每秒 10,000 個請求 (RPS)，加上 <a href="#">字符儲存貯體演算法</a> 提供的額外爆量容量，最多使用 5,000 個請求的儲存貯體容量。  Note  該爆量限制是由 API Gateway 服務團隊根據該帳戶整體 RPS 限制而決定。客戶無法針對該限制進行控制或請求變更。	是
區域 API	600	否
邊緣最佳化的 API	120	否

## 設定和執行 WebSocket API 的 API Gateway 限制

下列限制適用於在 Amazon API Gateway 中設定和執行 WebSocket API。

資源或操作	預設限制	可以提高
每個帳戶（所有 WebSocket API）在每個區域每秒的新連線	500	否

資源或操作	預設限制	可以提高
每 API AWS Lambda 授權方	10	是
每 API 的路由	300	是
每 API 的整合	300	是
每個 API 階段	10	是
WebSocket 框架大小	32 KB	否
訊息承載大小	128 KB  <b>Note</b>  由於 WebSocket 框架大小限制為 32 KB，因此大於 32 KB 的訊息必須分割成多個框架，每個最多 32 KB。如果接收到更大的訊息（或更大的框架大小），則該連線會關閉，並出現代碼 1009。	否
WebSocket API 連線持續時間	2 小時	否
閒置連線逾時	10 分鐘	否

## 設定和執行 REST API 的 API Gateway 限制

下列限制適用於在 Amazon API Gateway 中設定和執行 REST API。

資源或操作	預設限制	可以提高
每個區域每個帳戶的自訂網域名稱	30	是
每個區域每個帳戶的私有 API	600	否
API Gateway 資源政策的長度（以字元為單位）	8092	是
每個區域每個帳戶的 API 金鑰	500	是
每個區域每個帳戶的用戶端憑證	60	是
每個 API 的授權方（AWS Lambda 及 Amazon Cognito）	10	是
每個 API 的文件部分	2000	是
每個 API 資源	300	是
每個 API 階段	10	是

資源或操作	預設限制	可以提高
每個區域每個帳戶的用量計劃	300	是
每個 API 金鑰的用量計畫	10	是
每個 API 階段每個方法的調節限制設定	20	是
每個區域每個帳戶的 VPC 連結	20	是
API 快取 TTL	預設值為 300 秒，而且 API 擁有者可以設定為 0 與 3600 之間的值。	不適用於上限 (3600)
快取的回應大小	1048576 個位元組。快取資料加密可能增加正在快取的項目大小。	否
整合逾時	50 毫秒 - 所有整合類型都是 29 秒 (包含 Lambda、Lambda 代理、HTTP、HTTP 代理和 AWS 整合)。	不適用於下限或上限。
標頭值大小	10240 個位元組	否
承載大小	10 MB	否
每個階段的標籤	50	否
映射範本中 <code>#foreach ... #end</code> 迴圈的反覆運算數目	1000	否
具有授權之方法的 ARN 長度	1600 個位元組	否

針對 `restapi:import` 或 `restapi:put`，API 定義檔案的大小上限為 2 MB。

每個 API 的所有限制只有在特定 API 上才能提高。

## 建立、部署和管理 API 的 API Gateway 限制

以下的固定限制適用於使用 AWS CLI、API Gateway 主控台或 API Gateway REST API 及其開發套件，在 API Gateway 中建立、部署和管理 API。這些限制無法提高。

動作	預設限制	可以提高
<a href="#">CreateApiKey</a>	每個帳戶每秒 5 個請求	否
<a href="#">CreateDeployment</a>	每個帳戶每 5 秒 1 個請求。	否
<a href="#">CreateDocumentationVersion</a>	每個帳戶每 20 秒 1 個請求。	否
<a href="#">CreateDomainName</a>	每個帳戶每 30 秒 1 個請求。	否
<a href="#">CreateResource</a>	每個帳戶每秒 5 個請求	否

動作	預設限制	可以提高
CreateRestApi	區域或私有 API  • 每個帳戶每 3 秒 1 個請求。  邊緣最佳化的 API  • 每個帳戶每 30 秒 1 個請求	否
DeleteApiKey	每個帳戶每秒 5 個請求	否
DeleteResource	每個帳戶每秒 5 個請求	否
DeleteRestApi	每個帳戶每 30 秒 1 個請求。	否
GetResources	每個帳戶每 2 秒 5 個請求。	否
ImportDocumentationParts	每個帳戶每 30 秒 1 個請求	否
ImportRestApi	區域或私有 API  • 每個帳戶每 3 秒 1 個請求。  邊緣最佳化的 API  • 每個帳戶每 30 秒 1 個請求	否
PutRestApi	每個帳戶每秒 1 個請求	否
UpdateAccount	每個帳戶每 20 秒 1 個請求。	否
UpdateUsagePlan	每個帳戶每 20 秒 1 個請求。	否
其他操作	帳戶限制總計沒有上限。	否
操作總計	每秒 10 個請求 (rps)，爆量限制為每秒 40 個請求。	否

## Amazon API Gateway 重要說明

### 主題

- 適用於 REST 和 WebSocket API 的 Amazon API Gateway 重要說明 (p. 630)
- 適用於 WebSocket API 的 Amazon API Gateway 重要說明 (p. 631)
- 適用於 REST API 的 Amazon API Gateway 重要說明 (p. 631)

## 適用於 REST 和 WebSocket API 的 Amazon API Gateway 重要說明

- API Gateway 不支援萬用字元子網域名稱 (\*.domain 格式)。然而它支援萬用字元憑證 (萬用字元子網域名稱的憑證)。
- API Gateway 不支援在 REST 和 WebSocket API 間共用自訂網域名稱。

- 階段名稱僅可含有英數字元、連字號與底線。長度上限為 128 字元。
- 系統會為服務運作狀態檢查保留 /ping 和 /sping 的路徑。針對具有自訂網域的 API 根層級資源使用這些項目，將會無法產生預期的結果。
- API Gateway 目前將日誌事件限制為 1024 個位元組。API Gateway 會先截斷大於 1024 個位元組的日誌事件 (例如請求和回應內文)，再提交至 CloudWatch Logs。
- CloudWatch 指標目前會將維度名稱和值限制為 255 有效的 XML 字元。(如需詳細資訊，請參閱 [CloudWatch 使用者指南](#)。) 維度值是使用者定義的名稱的函數，包括 API 名稱、標籤 (階段) 名稱和資源名稱。選擇這些名稱時，請小心不要超過 CloudWatch 指標限制。

## 適用於 WebSocket API 的 Amazon API Gateway 重要說明

- API Gateway 支援高達 128 KB 的訊息承載，影格大小上限為 32 KB。如果訊息超過 32 KB，則必須分割成多個影格，每個為 32 KB 或以下。如果接收到更大的訊息，則該連線會關閉，並出現程式碼 1009。

## 適用於 REST API 的 Amazon API Gateway 重要說明

- 任何請求 URL 查詢字串不支援純文字管道字元 (|)，而且必須對其進行 URL 編碼。
- 任何請求 URL 查詢字串不支援分號字元 (;) 且會導致資料分割。
- 使用 API Gateway 主控台測試 API 時，如果向後端呈現自我簽署憑證、憑證鏈遺失中繼憑證，或後端擲回任何其他無法辨識憑證的相關例外狀況，則您可能會收到「不明端點錯誤」回應。
- 針對具有私有整合的 API [Resource](#) 或 [Method](#) 實體，您應該在移除 [VpcLink](#) 的任何硬式編碼參考之後將其刪除。否則，您會有一個懸置整合，並收到錯誤，指出仍在使用 VPC 連結，即使刪除 Resource 或 Method 實體也是一樣。私有整合透過階段變數參考 [VpcLink](#) 時，此行為不適用。
- 下列後端可能不支援使用與 API Gateway 相容的方式來進行 SSL 用戶端身分驗證：
  - [NGINX](#)
  - [Heroku](#)
- API Gateway 支援大部分 [OpenAPI 2.0 規格](#) 和 [OpenAPI 3.0 規格](#)，除了以下例外：
  - 路徑區段只能包含英數字元、連字號、句號、逗號和大括號。路徑參數必須是個別的路徑區段。例如，"resource/{path\_parameter\_name}" 有效，"resource{path\_parameter\_name}" 却無效。
  - 模型名稱只能包含英數字元。
  - securitySchemes 類型 (若已使用) 必須是 apiKey。不過，透過 [Lambda 授權方 \(p. 348\)](#) 支援 OAuth 2 和 HTTP 基本驗證；透過 [供應商延伸模組 \(p. 535\)](#) 完成 OpenAPI 組態。
  - 不支援 deprecated 欄位，且會在匯出的 API 中刪除此欄位。
  - 使用 [JSON 結構描述草稿第 4 版](#) 定義 API Gateway 模型，而非 OpenAPI 所使用的 JSON 結構描述。
  - 模型中不支援 additionalProperties 和 anyOf 欄位。
  - 在任何結構描述物件中，不支援 discriminator 參數。
  - 不支援 example 標籤。
  - API Gateway 不支援 exclusiveMinimum。
  - 簡單請求驗證不包含 maxItems 和 minItems 標籤。若要解決這個問題，請在執行驗證之前於匯入之後更新模型。
  - API Gateway 不支援 oneOf。
  - API Gateway 不支援 pattern。
  - 不支援 readOnly 欄位。
  - 在 OpenAPI 文件根中，不支援 "500": {"\$ref": "#/responses/UnexpectedError"} 形式的回應定義。若要解決這個問題，請將參考取代為內嵌結構描述。
  - 不支援 Int32 或 Int64 類型的數字。範例顯示如下：

```
"elementId": {  
    "description": "Working Element Id",  
    "format": "int32",  
    "type": "number"  
}
```

- 在結構描述定義中，不支援小數格式類型 ("format": "decimal")。
- 在方法回應中，結構描述定義必須為物件類型，而且不能是基本類型。例如，不支援 "schema": { "type": "string"}。不過，您可以使用下列物件類型來解決這個問題：

```
"schema": {  
    "$ref": "#/definitions/StringResponse"  
}  
  
"definitions": {  
    "StringResponse": {  
        "type": "string"  
    }  
}
```

- 使用 Lambda 整合或 HTTP 整合處理方法時，API Gateway 制定下列限制。
  - 標頭名稱和查詢參數是以區分大小寫的方式進行處理。
  - 傳送至整合端點或由整合端點送回時，可能會將下列標頭捨棄或修改：

標頭名稱	請求 ( <a href="#">http/http_proxy/lambda</a> )	回應 ( <a href="#">http/http_proxy/lambda</a> )
Age	傳遞	傳遞
Accept	傳遞	已捨棄/傳遞/傳遞
Accept-Charset	傳遞	傳遞
Accept-Encoding	傳遞	傳遞
Authorization	已捨棄 (400)	已重新對應
Connection	傳遞/傳遞/已捨棄	已重新對應
Content-Encoding	傳遞/已捨棄/傳遞	傳遞
Content-Length	傳遞 (依據內文產生)	傳遞
Content-MD5	已捨棄	已重新對應
Content-Type	傳遞	傳遞
Date	傳遞	已重新對應覆寫

標頭名稱	請求 ( <a href="#">http/http_proxy/lambda</a> )	回應 ( <a href="#">http/http_proxy/lambda</a> )
Expect	已捨棄	已捨棄
Host	被 Lambda5XX/5XX/覆寫	已捨棄
Max-Forwards	已捨棄	已重新對應
Pragma	傳遞	傳遞
Proxy-Authenticate	已捨棄	已捨棄
Range	傳遞	傳遞
Referer	傳遞	傳遞
Server	已捨棄	已重新對應/覆寫
TE	已捨棄	已捨棄
Transfer-Encoding	已捨棄/已捨棄/例外	已捨棄
Trailer	已捨棄	已捨棄
Upgrade	已捨棄	已捨棄
User-Agent	傳遞	已重新對應
Via	已捨棄/已捨棄/傳遞	傳遞/已捨棄/已捨棄
Warn	傳遞	傳遞
WWW-Authenticate	已捨棄	已重新對應

- API Gateway 所產生之 API 的 Android 開發套件使用 `java.net.HttpURLConnection` 類別。針對將 `WWW-Authenticate` 標頭重新對應至 `X-Amzn-Remapped-WWW-Authenticate` 的 401 回應，在執行 Android 4.4 和之前版本的裝置上，此類別將會擲回未處理的例外狀況。
- 與 API Gateway 所產生之 API 的 Java、Android 和 iOS 開發套件，API Gateway 所產生之 API 的 JavaScript 開發套件不支援 500 層級錯誤重試。
- 方法的測試呼叫使用 `application/json` 的預設內容類型，並忽略任何其他內容類型的規格。

# 文件歷史記錄

下表說明自上次發行 Amazon API Gateway 後，文件的重要變更。如需有關此文件更新的通知，您可以在頂部功能面板中選擇 RSS 按鈕來訂閱 RSS 摘要。

- 文件最近更新時間：2019 年 5 月 29 日

update-history-change	update-history-description	update-history-date
<a href="#">文件已更新 (p. 634)</a>	重寫 Amazon API Gateway 入門。將教學課程移到 Amazon API Gateway 教學課程。	May 29, 2019
<a href="#">標記型存取控制 (p. 634)</a>	新增支援標籤型存取控制。如需詳細資訊，請參閱 <a href="#">使用標籤搭配 IAM 政策以控制對 API Gateway 資源的存取</a> 。	May 23, 2019
<a href="#">文件已更新 (p. 634)</a>	已重寫 6 個主題：何謂 Amazon API Gateway？、教學：建置具有 HTTP 代理整合的 API、教學：建立具有三個非代理整合的計算 REST API、API Gateway 映射範本和存取記錄變數參考、使用 API Gateway Lambda 授權方，以及為 API Gateway REST API 資源啟用 CORS。	April 5, 2019
<a href="#">無伺服器開發人員入口網站改善功能 (p. 634)</a>	已新增管理員面板和其他改善功能，讓您可在 Amazon API Gateway 開發人員入口網站中更輕鬆地發佈 API。如需詳細資訊，請參閱 <a href="#">使用開發人員入口網站將您的 API 編目</a> 。	March 28, 2019
<a href="#">支援 AWS Config (p. 634)</a>	新增對 AWS Config 的支援。如需詳細資訊，請參閱 <a href="#">使用 AWS Config 監控 API Gateway API 狀態</a> 。	March 20, 2019
<a href="#">支援 AWS CloudFormation (p. 634)</a>	已將 API Gateway V2 API 新增至 AWS CloudFormation 範本參考。如需詳細資訊，請參閱 <a href="#">Amazon API Gateway V2 資源類型參考</a> 。	February 7, 2019
<a href="#">支援 WebSocket API (p. 634)</a>	新增對 WebSocket API 的支援。如需詳細資訊，請參閱 <a href="#">在 Amazon API Gateway 中 WebSocket API</a> 。	December 18, 2018
<a href="#">無伺服器開發人員入口網站可以透過 AWS Serverless Application Repository 取得 (p. 634)</a>	此 Amazon API Gateway 開發人員入口網站無伺服器應用程式現在可從 <a href="#">AWS Serverless Application Repository</a> (除了 GitHub 以外) 取得。如需詳細資訊，請參閱 <a href="#">使</a>	November 16, 2018

用開發人員入口網站將您的 API  
Gateway API 編目。

支援 AWS WAF (p. 634)	新增對 AWS WAF (Web 應用程 式防火牆) 的支援。如需詳細資 訊，請參閱 <a href="#">使用 AWS WAF 控制 對 API 的存取</a> 。	November 5, 2018
無伺服器開發人員入口網 站 (p. 634)	Amazon API Gateway 現在提供 完全自訂的開發人員入口網站，做 為無伺服器應用程式，您可以部署 以發行您的 API Gateway API。如 需詳細資訊，請參閱 <a href="#">使用開發人 員入口網站將您的 API Gateway API 編目</a> 。	October 29, 2018
支援多值標頭和查詢字串參 數 (p. 634)	Amazon API Gateway 現在支援 具有相同名稱的多個標頭和查詢字 串參數。如需詳細資訊，請參閱 <a href="#">支 援多值標頭和查詢字串參數</a> 。	October 4, 2018
文件已更新 (p. 634)	新增主題： <a href="#">Amazon API Gateway 資源政策如何影響授權工作流程</a> 。	September 27, 2018
OpenAPI 支援 (p. 634)	Amazon API Gateway 現在支 援 OpenAPI 3.0 以及 OpenAPI (Swagger) 2.0。	September 27, 2018
Active AWS X-Ray 整 合 (p. 634)	您現在可以使用 AWS X-Ray，在 使用者請求通過 API 進入基礎服 務時追蹤和分析延遲。如需詳細資 訊，請參閱 <a href="#">使用 AWS X-Ray 追蹤 API Gateway API 執行</a> 。	September 6, 2018
快取功能改良 (p. 634)	當您啟用 API 階段的快取時，預 設只有 GET 方法會啟用快取。這 有助於確保 API 的安全。您可以 透過覆寫方法設定，來啟用其他 方法的快取。如需詳細資訊，請 參閱 <a href="#">啟用 API 快取以提升回應能 力</a> 。	August 20, 2018
Service Limits 已修訂 (p. 634)	許多限制已修訂：提升每個帳戶 的 API 數。為建立/匯入/部署 API 提升 API 速率限制。將部分費率 從每分鐘修正為每秒。如需詳細資 訊，請參閱 <a href="#">限制</a> 。	July 13, 2018
覆寫 API 請求和回應參數和標 頭 (p. 634)	新增支援覆寫請求標頭、查詢字 串和路徑，以及回應標頭和狀態代 碼。如需詳細資訊，請參閱 <a href="#">使用映 射範本來覆寫 API 請求和回應參 數和標頭</a> 。	July 12, 2018

用量計劃的方法層級調節 (p. 634)	新增對設定預設調節限制的支援， 以及在用量計劃設定中個別的 API 方法的設定調節限制。這些是您可 以在階段設定中設定的預設方法層 級設定調節限制 (除了設定現有帳 戶層級調節)。如需詳細資訊，請 參閱 <a href="#">調節 API 請求以達到更高的輸送量</a> 。	July 11, 2018
現在可透過 RSS 提供《API Gateway Developer Guide》更新通知 (p. 634)	API Gateway Developer Guide 的 HTML 版本現在支援在文件歷史記錄頁面中記錄的 RSS 摘要更新。RSS 摘要包括 2018 年 6 月 27 日以後所做的更新。先前發佈的更新仍可在此頁面中取得。使用頂部選單面板中的 RSS 按鈕來訂閱摘要。	June 27, 2018

## 舊版更新

下表說明在 2018 年 6 月 27 日前，每個 API Gateway Developer Guide 版本的重要變更。

變更	描述	變更日期
私有 API	為您透過 <a href="#">界面 VPC 端點</a> 公開的 <a href="#">私有 API (p. 183)</a> 新增支援。通往您私有 API 的流量都會留在 Amazon 網路中，並與公有網際網路隔離。	2018 年 6 月 14 日
跨帳戶 Lambda 授權方和整合，以及跨帳戶 Amazon Cognito 使用者集區授權方	從不同的 AWS 帳戶使用 AWS Lambda 函數做為 Lambda 授權方函數或為 API 整合後端。或使用 Amazon Cognito 使用者集區做為授權方。其他帳戶可以位於 Amazon API Gateway 可用的任何區域。如需詳細資訊，請參閱 <a href="#">the section called “設定跨帳戶 Lambda 授權方” (p. 362)</a> 、 <a href="#">the section called “教學：建置具有跨帳戶 Lambda 代理整合的 API” (p. 29)</a> 及 <a href="#">the section called “為 REST API 設定跨帳戶 Amazon Cognito 授權方” (p. 370)</a> 。	2018 年 4 月 2 日
API 的資源政策	使用 API Gateway 資源政策，讓使用者可安全存取您的 API，或僅允許從指定的來源 IP 地址範圍或 CIDR 區塊呼叫 API。如需詳細資訊，請參閱 <a href="#">the section called “使用 API Gateway 資源政策” (p. 324)</a> 。	2018 年 4 月 2 日
API Gateway 資源的標記	為 API Gateway 中 API 請求和快取的成本分配使用多達 50 個標籤標記 API 階段。如需詳細資訊，請參閱 <a href="#">「the section called “設定標籤” (p. 470)」</a> 。	2017 年 12 月 19 日
承載壓縮和解壓縮	提供您使用以其中一個支援之內容編碼壓縮的承載呼叫 API。如果指定內文對應範本，則壓縮承載也會進行對應。如需詳細資訊，請參閱 <a href="#">the section called “啟用承載壓縮功能” (p. 304)</a> 。	2017 年 12 月 19 日
源自自訂授權方的 API 金鑰	將 API 金鑰從自訂授權方傳回給 API Gateway，以套用需要金鑰之 API 方法的用量方案。如需詳細資訊，請參閱 <a href="#">the section called “選擇 API 金鑰來源” (p. 398)</a> 。	2017 年 12 月 19 日
使用 OAuth 2 範圍的授權	讓您可搭配 COGNITO_USER_POOLS 授權方使用 OAuth 2 範圍，來授權方法呼叫。如需詳細資訊，請參閱 <a href="#">the section</a>	2017 年 12 月 14 日

變更	描述	變更日期
	called “針對 REST API 使用 Cognito 使用者集區做為授權方” (p. 363)。	
私有整合和 VPC 連結	建立具有 API Gateway 私有整合的 API，讓用戶端可以透過 <a href="#">VpcLink</a> 資源在 VPC 外部存取 Amazon VPC 中的 HTTP/HTTPS 資源。如需詳細資訊，請參閱 <a href="#">the section called “教學：建置具有私有整合的 API” (p. 80)</a> 及 <a href="#">the section called “設定私有整合” (p. 229)</a> 。	2017 年 11 月 30 日
部署 Canary 以測試 API	將 Canary Release 新增至現有 API 部署，以測試較新的 API 版本，同時保留同一個階段上操作中目前的版本。您可以設定 Canary Release 的階段流量百分比，並啟用記錄在不同 CloudWatch Logs 日誌中的 Canary 專屬執行和存取。如需詳細資訊，請參閱 <a href="#">the section called “設定 Canary Release 部署” (p. 607)</a> 。	2017 年 11 月 28 日
存取記錄	使用您所選格式之 <a href="#">\$context 變數 (p. 274)</a> 產生的資料，記錄用戶端對 API 的存取。如需詳細資訊，請參閱 <a href="#">the section called “設定 CloudWatch API 記錄” (p. 472)</a> 。	2017 年 11 月 21 日
API 的 Ruby 開發套件	產生您 API 的 Ruby 開發套件，並使用它來呼叫您的 API 方法。如需詳細資訊，請參閱 <a href="#">the section called “產生 API 的 Ruby 開發套件” (p. 484)</a> 及 <a href="#">the section called “使用 API Gateway 為 REST API 所產生的 Ruby 開發套件” (p. 506)</a> 。	2017 年 11 月 20 日
區域 API 端點	指定區域 API 端點，以建立非行動用戶端的 API。非行動用戶端 (例如，EC2 執行個體) 會在 API 部署所在的同一個 AWS 區域中執行。如同邊緣最佳化的 API，您也可以為區域 API 建立自訂網域名稱。如需詳細資訊，請參閱 <a href="#">the section called “設定區域 API” (p. 181)</a> 及 <a href="#">the section called “設定區域性自訂網域名稱” (p. 599)</a> 。	2017 年 11 月 2 日
自訂請求授權方	使用自訂請求授權方在請求參數中提供使用者驗證資訊，以授權 API 方法呼叫。請求參數包含標頭和查詢字串參數，以及階段和內容變數。如需詳細資訊，請參閱 <a href="#">使用 API Gateway Lambda 授權方 (p. 348)</a> 。	2017 年 9 月 15 日
自訂閘道回應	對無法送達整合後端的 API 請求，自訂 API Gateway 產生的閘道回應。自訂的閘道訊息可以為發起人提供 API 專屬自訂錯誤訊息 (包括傳回所需的 CORS 標頭)，也可以將閘道回應資料轉換為外部交換的格式。如需詳細資訊，請參閱 <a href="#">設定閘道回應以自訂錯誤回應 (p. 237)</a> 。	2017 年 6 月 6 日
將 Lambda 自訂錯誤屬性對應至方法回應標頭	使用 <code>integration.response.body</code> 參數，將 Lambda 傳回的個別自訂錯誤屬性對應至方法回應標頭參數，並由 API Gateway 在執行時間將字串化自訂錯誤物件還原序列化。如需詳細資訊，請參閱 <a href="#">在 API Gateway 中處理自訂 Lambda 錯誤 (p. 223)</a> 。	2017 年 6 月 6 日
調節限制提高	將帳戶層級穩定狀態請求速率限制增加為每秒 10,000 個請求 (rps)，並將爆增限制增加為 5000 個並行請求。如需詳細資訊，請參閱 <a href="#">調節 API 請求以獲得較佳輸送 (p. 468)</a> 。	2017 年 6 月 6 日
驗證方法請求	在 API 層級或方法層級設定基本請求驗證程式，讓 API Gateway 可以驗證傳入請求。API Gateway 會驗證所需參數已設定且不是空白，並驗證適用承載的格式符合設定的模型。如需詳細資訊，請參閱 <a href="#">在 API Gateway 中啟用請求驗證 (p. 307)</a> 。	2017 年 4 月 11 日

變更	描述	變更日期
與 ACM 整合	為您 API 的自訂網域名稱使用 ACM 憑證。您可以在 AWS Certificate Manager 中建立憑證，或將現有 PEM 格式的憑證匯入 ACM。然後在設定您 API 的自訂網域名稱時，參考憑證的 ARN。如需詳細資訊，請參閱 <a href="#">在 API Gateway 中設定 API 的自訂網域名稱 (p. 590)</a> 。	2017 年 3 月 9 日
產生和呼叫 API 的 Java 開發套件	讓 API Gateway 產生您 API 的 Java 開發套件，並在您的 Java 用戶端中使用開發套件呼叫 API。如需詳細資訊，請參閱 <a href="#">使用 API Gateway 為 REST API 所產生的 Java 開發套件 (p. 499)</a> 。	2017 年 1 月 13 日
與 AWS Marketplace 整合	透過 AWS Marketplace 將用量方案中的 API 做為 SaaS 產品銷售。使用 AWS Marketplace 推廣您的 API。由 AWS Marketplace 代您進行客戶計費。讓 API Gateway 處理使用者授權和用量量測。如需詳細資訊，請參閱 <a href="#">將 API 當做 SaaS 銷售 (p. 587)</a> 。	2016 年 12 月 1 日
提供您 API 的文件支援	新增 API Gateway 之 <a href="#">DocumentationPart</a> 資源中 API 實體的文件。將集合 DocumentationPart 執行個體的快照與 API 階段建立關聯，以建立 <a href="#">DocumentationVersion</a> 。將文件版本匯出至外部檔案 (例如 Swagger 檔案)，來發佈 API 文件。如需詳細資訊，請參閱 <a href="#">記錄 API Gateway 中的 REST API (p. 411)</a> 。	2016 年 12 月 1 日
更新了自訂授權方	客戶授權方 Lambda 函數現在會傳回發起人的委託人識別符。此函數也可以將其他資訊做為 context 對應和 IAM 政策的鍵值對傳回。如需詳細資訊，請參閱 <a href="#">Amazon API Gateway Lambda 授權方的輸出 (p. 358)</a> 。	2016 年 12 月 1 日
支援二進位承載	設定您 API 上的 <a href="#">binaryMediaTypes</a> ，以支援請求或回應的二進位承載。設定 <a href="#">Integration</a> 或 <a href="#">IntegrationResponse</a> 上的 <a href="#">contentHandling</a> 屬性，以指定是否將二進位承載處理為原生二進位 Blob、Base64 編碼字串，或傳遞而不修改。如需詳細資訊，請參閱 <a href="#">支援 API Gateway 中的二進位承載 (p. 283)</a> 。	2016 年 11 月 17 日
提供您透過 API 的代理資源與 HTTP 或 Lambda 後端進行代理整合。	使用 {proxy+} 形式的 Greedy 路徑參數以及全部截獲的 ANY 方法建立代理資源。代理資源 分別使用 HTTP 或 Lambda 代理整合，與 HTTP 或 Lambda 後端整合。如需詳細資訊，請參閱 <a href="#">設定代理整合與代理資源 (p. 202)</a> 。	2016 年 9 月 20 日
提供一或多個用量方案，以將 API Gateway 中選取的 API 擴展為您客戶的產品供應項目。	在 API Gateway 中建立用量方案，以讓選取的 API 用戶端依同意的請求速率和配額來存取指定的 API 階段。如需詳細資訊，請參閱 <a href="#">建立和使用 API 金鑰的用量計劃 (p. 397)</a> 。	2016 年 8 月 11 日
提供您使用 Amazon Cognito 中使用者集區的方法層級授權	在 Amazon Cognito 中建立使用者集區，並將其做為您自己的身分提供者使用。您可以設定使用者集區做為方法層級授權方，以授予向使用者集區註冊之使用者的存取權。如需詳細資訊，請參閱 <a href="#">使用 Amazon Cognito User Pools 做為授權方來控制 REST API 的存取 (p. 363)</a> 。	2016 年 7 月 28 日
提供 AWS/ApiGateway 命名空間下的 Amazon CloudWatch 指標和維度。	API Gateway 指標現已在 AWS/ApiGateway 的 CloudWatch 命名空間下標準化。您可同時在 API Gateway 主控台和 Amazon CloudWatch 主控台中加以檢視。如需詳細資訊，請參閱 <a href="#">Amazon API Gateway 維度與指標 (p. 529)</a> 。	2016 年 7 月 28 日

變更	描述	變更日期
提供自訂網域名稱的憑證輪換	憑證輪換可讓您上傳和續約自訂網域名稱的到期憑證。如需詳細資訊，請參閱 <a href="#">輪換匯入至 ACM 的憑證 (p. 597)</a> 。	2016 年 4 月 27 日
記錄已更新 Amazon API Gateway 主控台的變更。	了解如何使用已更新的 API Gateway 主控台建立和設定 API。如需詳細資訊，請參閱 <a href="#">教學課程：匯入範例來建立 REST API (p. 40)</a> 及 <a href="#">教學：建置具有 HTTP 非代理整合的 API (p. 52)</a> 。	2016 年 4 月 5 日
提供 Import API (匯入 API) 功能，以透過外部 API 定義建立新的 API 或更新現有的 API。	使用 Import API (匯入 API) 功能，您可以上傳以具有 API Gateway 延伸之 Swagger 2.0 表示的外部 API 定義，來建立新的 API 或更新現有的 API。如需 Import API (匯入 API) 的詳細資訊，請參閱「 <a href="#">將 REST API 匯入 API Gateway (p. 319)</a> 」。	2016 年 4 月 5 日
在對應範本中公布 \$input.body 變數，用以存取字串形式的原始承載，以及公布 \$util.parseJson() 函數，用以將 JSON 字串轉為 JSON 物件。	如需 \$input.body 和 \$util.parseJson() 的詳細資訊，請參閱「 <a href="#">API Gateway 映射範本和存取記錄變數參考 (p. 274)</a> 」。	2016 年 4 月 5 日
提供方法層級快取失效的用戶端請求，並改善請求調節管理。	排清 API 階段層級快取，並使個別快取項目失效。如需詳細資訊，請參閱 <a href="#">在 API Gateway 中排清 API 階段快取 (p. 466)</a> 及 <a href="#">使 API Gateway 快取項目失效 (p. 466)</a> 。改善用於管理 API 請求調節的主控台體驗。如需詳細資訊，請參閱 <a href="#">調節 API 請求以獲得較佳輸送 (p. 468)</a> 。	2016 年 3 月 25 日
使用自訂授權啟用和呼叫 API Gateway API	建立和設定 AWS Lambda 函數來實作自訂授權。該函數會傳回 IAM 政策文件，該文件會將允許或拒絕許可授予 API Gateway API 的用戶端請求。如需詳細資訊，請參閱 <a href="#">使用 API Gateway Lambda 授權方 (p. 348)</a> 。	2016 年 2 月 11 日
使用 Swagger 定義檔和延伸來匯入和匯出 API Gateway API	搭配使用 Swagger 規定與 API Gateway 延伸來建立和更新 API Gateway API。使用 API Gateway Importer 匯入 Swagger 定義。使用 API Gateway 主控台或 API Gateway Export API，將 API Gateway API 匯出至 Swagger 定義檔。如需詳細資訊，請參閱 <a href="#">將 REST API 匯入 API Gateway (p. 319)</a> 及 <a href="#">匯出 REST API (p. 604)</a> 。	2015 年 12 月 18 日
將請求或回應內文或內文的 JSON 欄位對應至請求或回應參數。	將方法請求內文或其 JSON 欄位對應至整合請求的路徑、查詢字串或標頭。將整合回應內文或其 JSON 欄位對應至請求回應的標頭。如需詳細資訊，請參閱 <a href="#">Amazon API Gateway API 請求和回應資料對應參考 (p. 270)</a> 。	2015 年 12 月 18 日
在 Amazon API Gateway 中使用階段變數	了解如何在 Amazon API Gateway 中將組態屬性與 API 的部署階段建立關聯。如需詳細資訊，請參閱 <a href="#">設定 REST API 部署的階段變數 (p. 475)</a> 。	2015 年 11 月 5 日
如何：為方法啟用 CORS	現在可以更輕鬆地針對 Amazon API Gateway 中的方法啟用跨來源資源共享 (CORS)。如需詳細資訊，請參閱 <a href="#">為 REST API 資源啟用 CORS (p. 371)</a> 。	2015 年 3 月 11 日
如何：使用用戶端 SSL 身分驗證	使用 Amazon API Gateway 產生 SSL 憑證，讓您可用來驗證對 HTTP 後端的呼叫。如需詳細資訊，請參閱 <a href="#">後端使用用戶端 SSL 憑證進行驗證 (p. 377)</a> 。	2015 年 9 月 22 日

變更	描述	變更日期
方法的模擬整合	了解如何 <a href="#">模擬整合 API 與 Amazon API Gateway (p. 234)</a> 。此功能讓開發人員可直接從 API Gateway 產生 API 回應，而不需事先具有最終整合後端。	2015 年 9 月 1 日
Amazon Cognito Identity 支援	Amazon API Gateway 已擴展 \$context 變數的範圍，因此當請求以 Amazon Cognito 登入資料簽署時，該變數現在會傳回 Amazon Cognito Identity 的相關資訊。此外，我們也新增了 \$util 變數，用以逸出 JavaScript 中的字元以及編碼 URL 和字串。如需詳細資訊，請參閱 <a href="#">API Gateway 映射範本和存取記錄變數參考 (p. 274)</a> 。	2015 年 8 月 28 日
Swagger 整合	使用 <a href="#">GitHub 上的 Swagger 匯入工具</a> ，將 Swagger API 定義匯入 Amazon API Gateway。進一步了解 <a href="#">OpenAPI 的 API Gateway 延伸 (p. 533)</a> ，以使用匯入工具來建立和部署 API 和方法。使用 Swagger 匯入程式工具，您也可以更新現有的 API。	2015 年 7 月 21 日
對應範本參考	參閱「\$input」中的 <a href="#">API Gateway 映射範本和存取記錄變數參考 (p. 274)</a> 參數和其函數。	2015 年 7 月 18 日
初始公有版本	這是API Gateway Developer Guide的初始公有版本。	2015 年 7 月 9 日

# AWS Glossary

For the latest AWS terminology, see the [AWS Glossary](#) in the AWS General Reference.