aws

開發人員指南

# Amazon API Gateway

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

# Amazon API Gateway: 開發人員指南

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

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

# Table of Contents

什麼是 Amazon API Gateway？ ..... 1
API Gateway 架構 ..... 2
API Gateway 的功能 ..... 2
API Gateway 使用案例 ..... 3
使用 API Gateway 建立 REST API ..... 3
使用 API Gateway 建立 HTTP API ..... 4
使用 API Gateway 來建立 WebSocket API ..... 4
誰使用 API Gateway？ ..... 5
存取 API Gateway ..... 6
無 AWS 伺服器基礎設施的一部分 ..... 6
如何開始使用 Amazon API Gateway ..... 6
API Gateway 概念 ..... 7
在 REST API 與 HTTP API 之間選擇 ..... 11
端點類型 ..... 12
安全 ..... 12
Authorization ..... 12
API 管理 ..... 13
開發 ..... 13
監控 ..... 14
整合 ..... 15
開始使用 REST API 主控台 ..... 15
步驟 1：建立 Lambda 函數 ..... 16
步驟 2：建立 REST API ..... 17
步驟 3：建立 Lambda 代理整合 ..... 17
步驟 4：部署 API ..... 18
步驟 5：調用 API ..... 18
(選用) 步驟 6：清理 ..... 19
先決條件 ..... 21
註冊 AWS 帳戶 ..... 21
開始使用 ..... 22
步驟 1：建立 Lambda 函數 ..... 23
步驟 2：建立 HTTP API ..... 23
步驟 3：測試您的 API ..... 24
(選用) 步驟 4：清理 ..... 25

後續步驟 ..... 26
教學課程和研討會 ..... 28
REST API 教學課程 ..... 29
選擇 Lambda 整合教學課程 ..... 29
教學課程：匯入範例來建立 REST API ..... 51
選擇 HTTP 整合教學課程 ..... 59
教學：建立具私有整合的 REST API ..... 73
教學課程：使用 AWS 整合建立 REST API ..... 78
教學：使用三個整合建立計算器 REST API ..... 84
教學：建立 REST API 做為 Amazon S3 代理 ..... 109
教學課程：建立 REST API 做為 Amazon Kinesis 代理 ..... 152
教學課程：使用 AWS SDKs或 建立 REST API AWS CLI ..... 197
教學：建立私有 REST API ..... 230
HTTP API 教學課程 ..... 236
教學：使用 Lambda 和 DynamoDB 建置 CRUD HTTP API ..... 236
教學：透過私有整合至 Amazon ECS 來建立 HTTP API ..... 249
WebSocket API 教學課程 ..... 256
教學：建立 WebSocket 聊天應用程式 ..... 256
教學課程：使用 AWS 整合建立 WebSocket API ..... 261
API Gateway REST API ..... 274
開發 ..... 274
REST API 的範例資源 ..... 275
用於開發的其他 REST API 功能 ..... 277
API Gateway 端點類型 ..... 279
安全政策 ..... 283
API Gateway 中 REST API 的 IP 位址類型 ..... 303
方法 ..... 305
存取控制 ..... 322
整合 ..... 398
請求驗證 ..... 479
資料轉換 ..... 507
閘道回應 ..... 580
CORS ..... 589
二進位媒體類型 ..... 602
調用 ..... 632
OpenAPI ..... 663

發布 ..... 676
部署 REST API ..... 677
自訂網域名稱 ..... 718
最佳化 ..... 781
快取設定 ..... 781
內容編碼 ..... 793
分配 ..... 799
用量計劃 ..... 799
API 文件 ..... 824
產生軟體開發套件 ..... 885
將 API 當做 SaaS 銷售 ..... 911
保護 ..... 915
需要用戶端憑證 ..... 915
後端用戶端憑證 ..... 921
AWS WAF ..... 969
限流 ..... 972
私有 REST API ..... 974
監控 ..... 1025
CloudWatch 指標 ..... 1025
CloudWatch 日誌 ..... 1033
X-Ray ..... 1058
API Gateway 入口網站 ..... 1070
考量事項 ..... 1070
入口網站產品 ..... 1070
寵物採用入口網站產品範例 ..... 1071
後續步驟 ..... 1072
建立入口網站產品 ..... 1073
建立產品 REST 端點 ..... 1075
建立產品頁面 ..... 1076
更新入口網站產品 ..... 1078
更新產品 REST 端點 ..... 1078
更新產品頁面 ..... 1079
共用入口網站產品 ..... 1079
針對入口網站中的產品 REST 端點啟用試用 ..... 1084
刪除入口網站產品 ..... 1085
建立入口網站 ..... 1086

選擇網域名稱 ..... 1086
建立入口網站 ..... 1086
更新入口網站 ..... 1088
更新入口網站 ..... 1089
預覽入口網站 ..... 1089
考量事項 ..... 1089
預覽入口網站 ..... 1089
發佈入口網站 ..... 1090
考量事項 ..... 1090
發佈入口網站 ..... 1090
使用入口網站 ..... 1091
使用入口網站 ..... 1091
對入口網站進行故障診斷 ..... 1091
停用入口網站 ..... 1092
停用入口網站 ..... 1092
刪除入口網站 ..... 1093
刪除入口網站 ..... 1093
API Gateway HTTP API ..... 1094
開發 ..... 1094
建立 HTTP API ..... 1095
路由 ..... 1096
API Gateway 中 HTTP API 的 IP 位址類型 ..... 1098
存取控制 ..... 1100
整合 ..... 1118
CORS ..... 1135
參數映射 ..... 1138
OpenAPI ..... 1144
發布 ..... 1153
階段 ..... 1154
API Gateway 中 HTTP API 的安全政策 ..... 1156
自訂網域名稱 ..... 1158
保護 ..... 1166
限流 ..... 1166
交互 TLS ..... 1167
監控 ..... 1172
指標 ..... 1173

記錄 ..... 1175
故障診斷 ..... 1184
Lambda 整合 ..... 1184
JWT 授權方 ..... 1186
API Gateway WebSocket API ..... 1188
WebSocket API 的概觀 ..... 1188
使用 WebSocket API 的路由 ..... 1189
將資料傳送至連線的用戶端應用程式 ..... 1189
WebSocket API 狀態碼 ..... 1189
管理連線的使用者和用戶端應用程式 ..... 1190
調用後端整合 ..... 1193
WebSocket 選擇表達式 ..... 1197
開發 ..... 1202
建立和設定 ..... 1203
API Gateway 中 WebSocket API 的 IP 位址類型 ..... 1204
路由 ..... 1206
存取控制 ..... 1213
整合 ..... 1221
請求驗證 ..... 1230
資料轉換 ..... 1233
二進位媒體類型 ..... 1244
調用 ..... 1244
發布 ..... 1247
階段 ..... 1248
部署 WebSocket API ..... 1250
API Gateway 中適用於 WebSocket API 的安全政策 ..... 1252
自訂網域名稱 ..... 1254
保護 ..... 1260
每個區域的帳戶層級調節 ..... 1261
路由層級調節 ..... 1261
監控 ..... 1261
指標 ..... 1262
日誌 ..... 1263
API Gateway ARN ..... 1271
HTTP API 和 WebSocket API 資源 ..... 1271
REST API 資源 ..... 1274

網域名稱資源 ..... 1278
execute-api (HTTP API、WebSocket API 和 REST API) ..... 1279
OpenAPI 延伸 ..... 1280
x-amazon-apigateway-any-method ..... 1281
x-amazon-apigateway-any-method 範例 ..... 1282
x-amazon-apigateway-cors ..... 1283
x-amazon-apigateway-cors 範例 ..... 1283
x-amazon-apigateway-api-key-source ..... 1284
x-amazon-apigateway-api-key-source 範例 ..... 1284
x-amazon-apigateway-auth ..... 1285
x-amazon-apigateway-auth 範例 ..... 1286
x-amazon-apigateway-authorizer ..... 1286
REST API 的 x-amazon-apigateway-authorizer 範例 ..... 1289
HTTP API 的 x-amazon-apigateway-authorizer 範例 ..... 1292
x-amazon-apigateway-authtype ..... 1294
x-amazon-apigateway-authtype 範例 ..... 1294
另請參閱 ..... 1296
x-amazon-apigateway-binary-media-type ..... 1297
x-amazon-apigateway-binary-media-types 範例 ..... 1297
x-amazon-apigateway-documentation ..... 1297
x-amazon-apigateway-documentation 範例 ..... 1297
x-amazon-apigateway-endpoint-access-mode ..... 1298
x-amazon-apigateway-endpoint-access-mode 範例 ..... 1298
x-amazon-apigateway-endpoint-configuration ..... 1298
x-amazon-apigateway-endpoint-configuration 範例 ..... 1299
x-amazon-apigateway-gateway-responses ..... 1300
x-amazon-apigateway-gateway-responses 範例 ..... 1300
x-amazon-apigateway-gateway-responses.gatewayResponse ..... 1300
x-amazon-apigateway-gateway-responses.gatewayResponse 範例 ..... 1301
x-amazon-apigateway-gateway-responses.responseParameters ..... 1301
x-amazon-apigateway-gateway-responses.responseParameters 範例 ..... 1302
x-amazon-apigateway-gateway-responses.responseTemplates ..... 1302
x-amazon-apigateway-gateway-responses.responseTemplates 範例 ..... 1303
x-amazon-apigateway-importexport-version ..... 1303
x-amazon-apigateway-importexport-version example ..... 1303
x-amazon-apigateway-integration ..... 1303

x-amazon-apigateway-integration 範例 ..... 1307
x-amazon-apigateway-integrations ..... 1309
x-amazon-apigateway-integrations 範例 ..... 1309
x-amazon-apigateway-integration.requestTemplates ..... 1311
x-amazon-apigateway-integration.requestTemplates 範例 ..... 1312
x-amazon-apigateway-integration.requestParameters ..... 1312
x-amazon-apigateway-integration.requestParameters 範例 ..... 1313
x-amazon-apigateway-integration.responses ..... 1314
x-amazon-apigateway-integration.responses 範例 ..... 1314
x-amazon-apigateway-integration.response ..... 1315
x-amazon-apigateway-integration.response 範例 ..... 1316
x-amazon-apigateway-integration.responseTemplates ..... 1317
x-amazon-apigateway-integration.responseTemplate 範例 ..... 1317
x-amazon-apigateway-integration.responseParameters ..... 1317
x-amazon-apigateway-integration.responseParameters 範例 ..... 1318
x-amazon-apigateway-integration.tlsConfig ..... 1318
x-amazon-apigateway-integration.tlsConfig examples ..... 1319
x-amazon-apigateway-minimum-compression-size ..... 1320
x-amazon-apigateway-minimum-compression-size example ..... 1320
x-amazon-apigateway-policy ..... 1321
x-amazon-apigateway-policy 範例 ..... 1321
x-amazon-apigateway-request-validator ..... 1322
x-amazon-apigateway-request-validator 範例 ..... 1322
x-amazon-apigateway-request-validators ..... 1322
x-amazon-apigateway-request-validators 範例 ..... 1323
x-amazon-apigateway-request-validators.requestValidator ..... 1324
x-amazon-apigateway-request-validators.requestValidator 範例 ..... 1324
x-amazon-apigateway-security-policy ..... 1324
x-amazon-apigateway-security-policy 範例 ..... 1324
x-amazon-apigateway-tag-value ..... 1325
x-amazon-apigateway-tag-value 範例 ..... 1325
安全 ..... 1326
資料保護 ..... 1326
資料加密 ..... 1327
網際網路流量隱私權 ..... 1329
Identity and Access Management ..... 1329

目標對象 ..... 1330
使用身分驗證 ..... 1330
使用政策管理存取權 ..... 1331
Amazon API Gateway 與 IAM 搭配運作的方式 ..... 1332
身分型政策範例 ..... 1337
資源型政策範例 ..... 1348
故障診斷 ..... 1348
使用服務連結角色 ..... 1350
記錄和監控 ..... 1354
使用 CloudTrail ..... 1355
使用 AWS Config ..... 1358
法規遵循驗證 ..... 1361
恢復能力 ..... 1361
基礎架構安全 ..... 1362
組態與漏洞分析 ..... 1362
最佳實務 ..... 1363
標記 ..... 1365
可標記的 API Gateway 資源 ..... 1365
Amazon API Gateway V1 API 中的標籤繼承 ..... 1367
標籤限制和使用慣例 ..... 1368
屬性型存取控制 ..... 1368
根據資源標籤限制動作 ..... 1369
根據資源標籤允許動作 ..... 1369
拒絕標記操作 ..... 1371
允許標記操作 ..... 1371
API 參考 ..... 1373
API Gateway 服務端點 ..... 1373
IPv4 端點 ..... 1373
雙堆疊 (IPv4 和 IPv6) 端點 ..... 1374
指定端點 ..... 1374
在 IAM 原則中使用 IPv6 地址 ..... 1375
配額和重要說明 ..... 1377
每個區域的 API Gateway 帳戶層級配額 ..... 1377
建立、部署和管理 API 的 API Gateway 配額 ..... 1378
REST API 配額 ..... 1381
減少配額的最佳實務 ..... 1385

HTTP API 配額 ..... 1387
減少配額的最佳實務 ..... 1389
入口網站配額 ..... 1389
WebSocket API 配額 ..... 1390
重要說明 ..... 1392
適用於 HTTP API 的 Amazon API Gateway 重要說明 ..... 1393
適用於 HTTP 和 WebSocket API 的 Amazon API Gateway 重要說明 ..... 1393
適用於 REST API 和 WebSocket API 的重要說明 ..... 1393
適用於 WebSocket API 的重要說明 ..... 1393
適用於 REST API 的重要說明 ..... 1394
文件歷史記錄 ..... 1399
舊版更新 ..... 1408
..... mcdxvi

# 什麼是 Amazon API Gateway？

Amazon API Gateway 是一種 AWS 服務，用於建立、發佈、維護、監控和保護任何規模的 REST、HTTP 和 WebSocket APIs。API 開發人員可以建立 APIs 來存取 AWS 或其他 Web 服務，以及存放在 AWS 雲端中的資料。身為 API Gateway API 開發人員，您也可以建立要在自己用戶端應用程式中使用的 API。或者，您可以讓 API 供第三方應用程式開發人員使用。如需更多詳細資訊，請參閱 the section called “誰使用 API Gateway？”。

API Gateway 所建立的 RESTful API 會：

- 以 HTTP 為基礎。
- 啟用無狀態的用戶端伺服器通訊。
- 實作標準 HTTP 方法，例如 GET、POST、PUT、PATCH 和 DELETE。

如需 API Gateway REST API 和 HTTP API 的詳細資訊，請參閱the section called “在 REST API 與 HTTP API 之間選擇 ”、API Gateway HTTP API、the section called “使用 API Gateway 建立 REST API”，以及the section called “開發”。

API Gateway 所建立的 WebSocket API 會：

- 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” 和the section called “WebSocket API 的概觀”。

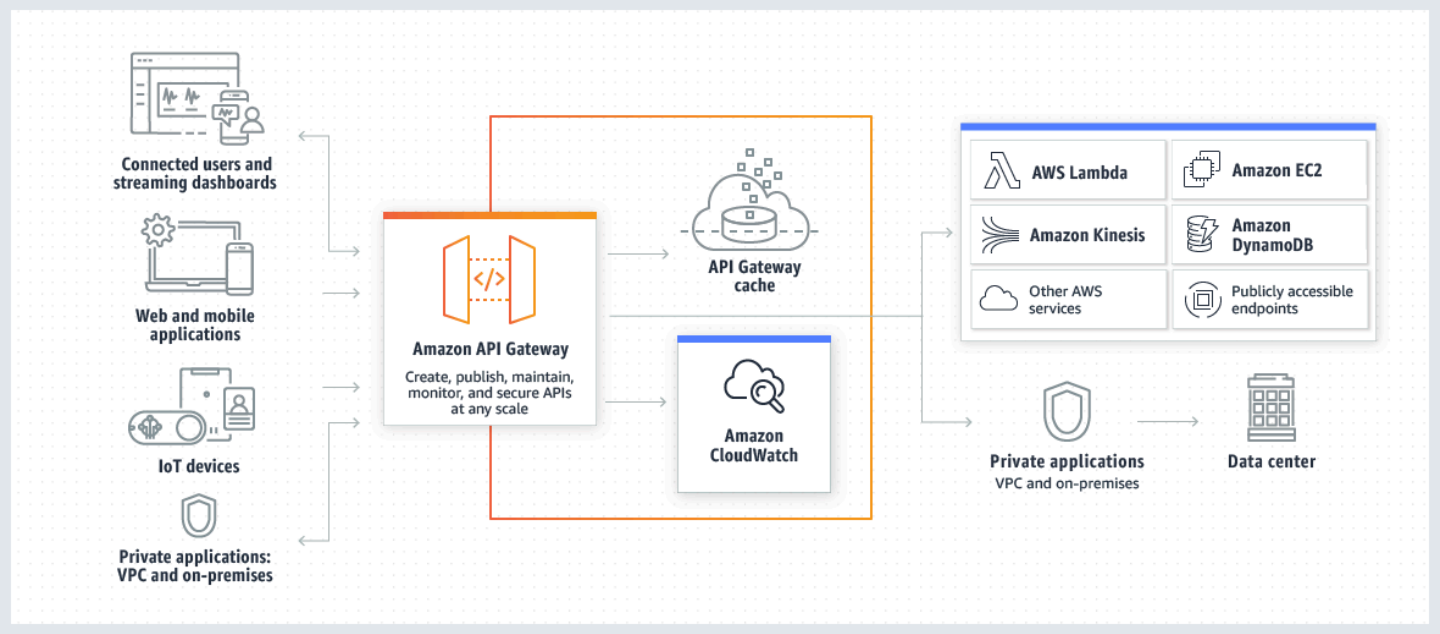
主題

- API Gateway 架構
- API Gateway 的功能
- API Gateway 使用案例
- 存取 API Gateway
- 無 AWS 伺服器基礎設施的一部分
- 如何開始使用 Amazon API Gateway
- Amazon API Gateway 概念

- 在 REST API 與 HTTP API 之間選擇
- 開始使用 REST API 主控台

# API Gateway 架構

下圖顯示的是 API Gateway 架構。




此圖表旨在說明您在 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，享有整合且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 呼叫的所有任務，這些任務包括流量管理、授權和存取控制、監控和 API 版本管理。

API Gateway 可做為應用程式從後端服務存取資料、商業邏輯或功能的「前門」，例如在 Amazon Elastic Compute Cloud (Amazon EC2) 上執行的工作負載 AWS Lambda、在 上執行的程式碼、任何 Web 應用程式或即時通訊應用程式。

# API Gateway 的功能

Amazon API Gateway 提供如下功能：

- 支援狀態 (WebSocket) 和無狀態 (HTTP 及 REST) API。
- 強大且靈活的身分驗證機制，例如 AWS Identity and Access Management 政策、Lambda 授權方函數和 Amazon Cognito 使用者集區。
- Canary 發行部署，可讓您安全地進行變更。

- 記錄和監控 API 使用情況與 API 變更的 CloudTrail。
- CloudWatch 存取記錄和執行記錄，包括設定警示的功能。如需詳細資訊，請參閱the section called “CloudWatch 指標”及the section called “指標”。
- 能夠使用 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。
- 與 AWS WAF 的整合，用於保護 API 避免受到常見的網路攻擊。
- 與 AWS X-Ray 的整合，用於了解和分類效能延遲。

如需 API Gateway 功能發布的完整清單，請參閱文件歷史記錄。

# API Gateway 使用案例

下列使用案例區段呈現不同類型 API Gateway API 的概觀，以及使用 API Gateway 的不同開發人員類型。如需 REST API 與 HTTP API 之間差異的詳細資訊，請參閱 the section called “在 REST API 與 HTTP API 之間選擇 ”。

主題

- 使用 API Gateway 建立 REST API
- 使用 API Gateway 建立 HTTP API
- 使用 API Gateway 來建立 WebSocket API
- 誰使用 API Gateway？

## 使用 API Gateway 建立 REST API

API Gateway REST API 是由資源和方法構成。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，系統會以「方法請求」和「方法回應」封裝前端。API 會使用整合請求和整合回應以與後端互動。

例如，使用 DynamoDB 作為後端時，API 開發人員可以設定整合請求，以將傳入的方法請求轉送至選擇的後端。設定項目包含適當 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 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，以協助設定內文映射範本。

API Gateway 提供如下 REST API 管理功能：

- 支援使用 OpenAPI 的 API Gateway 延伸項目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 建立 HTTP API

HTTP API 可讓您建立比 REST API 延遲更低和成本更低的 RESTful API。

您可以使用 HTTP APIs 將請求傳送至 AWS Lambda 函數或任何可公開路由的 HTTP 端點。

例如，您可以在後端建立與 Lambda 函數整合的 HTTP API。當用戶端呼叫您的 API 時，API Gateway 會將請求傳送到該 Lambda 函數並傳回該函數的回應給用戶端。

HTTP API 支援 OpenID Connect 和 OAuth 2.0 授權。它們提供跨來源資源共享 (CORS) 和自動部署的內建支援。

如需進一步了解，請參閱the section called “在 REST API 與 HTTP API 之間選擇 ”。

## 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 建置無伺服器應用程式 AWS Lambda ，並傳送和接收聊天室中個別使用者或使用者群組的訊息。或者，您可以根據訊息內容叫用後端服務 AWS Lambda，例如 Amazon Kinesis 或 HTTP 端點。

您還能利用 API Gateway WebSocket API 建構安全的即時通訊應用程式，無需佈建或管理任何伺服器，即可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，如下所示：

- 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，如下所示：

- 對連線和訊息進行監控和調節
- 使用 AWS X-Ray 在訊息通過 APIs到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 開發人員可建立和部署 API，以在 API Gateway 中啟用所需功能。API 開發人員必須是擁有 API AWS 之帳戶中的使用者。

應用程式開發人員會建置正常運作的應用程式，透過叫用 API 開發人員在 API Gateway 中建立的 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 AWS 帳戶，前提是 API 不需要 IAM 許可，或支援透過 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 開發人員需要使用名為 `apigateway` 的 API Gateway 服務元件 (用於管理 API) 來建立、設定和部署 API。

身為 API 開發人員，您可以使用 API Gateway 主控台 (如 API Gateway 入門 中所述) 或透過呼叫 API 參考 來建立和管理 API。有數種方式可以呼叫此 API。其中包括使用 AWS Command Line Interface (AWS CLI)，或使用 AWS SDK。此外，您還可以使用 AWS CloudFormation 範本或 (在 REST API 和 HTTP API 的情形下) API Gateway 的 OpenAPI 擴充功能 來建立 API。

如需支援 API Gateway 區域的清單，以及相關聯的控制服務端點，請參閱 Amazon API Gateway 端點和配額。

### 呼叫 API Gateway API

應用程式開發人員可使用名為 `execute-api` (用於執行 API) 的 API Gateway 服務元件來叫用在 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。如需進一步了解，請參閱在 API Gateway 中調用 REST API和調用 WebSocket API。

# 存取 API Gateway

您可以通過以下方式存取 Amazon API Gateway：

- AWS 管理主控台 – AWS 管理主控台 提供用於建立和管理 APIs Web 界面。完成先決條件中的步驟後，您可以前往 https://console.aws.amazon.com/apigateway 存取 API Gateway 主控台。
- AWS SDKs – 如果您使用的是 AWS 提供 開發套件的程式設計語言，您可以使用 開發套件來存取 API Gateway。軟體開發套件可簡化身分驗證、與開發環境輕鬆整合，並讓您存取 API Gateway 命令。如需詳細資訊，請參閱 Amazon Web Services 適用工具。
- API Gateway V1 和 V2 API – 如果您使用的是沒有適用軟體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 使用者指南中的使用 AWS Command Line Interface完成設定。
- AWS Tools for Windows PowerShell – 如需詳細資訊，請參閱 AWS Tools for Windows PowerShell 使用者指南中的設定AWS Tools for PowerShell。

# 無 AWS 伺服器基礎設施的一部分

AWS LambdaAPI Gateway 與 一起組成面向應用程式的無 AWS 伺服器基礎設施部分。若要深入了解如何開始使用無伺服器，請參閱《無伺服器開發人員指南》。

若要讓應用程式呼叫公開可用的 AWS 服務，您可以使用 Lambda 與必要的服務互動，並透過 API Gateway 中的 API 方法公開 Lambda 函數。 會在高可用性的運算基礎設施上執行 AWS Lambda 您的程式碼。它會執行所需的運算資源執行和管理。為了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。

# 如何開始使用 Amazon API Gateway

如需 Amazon API Gateway 的簡介，請參閱下列內容：

- 開始使用，提供用於建立 HTTP API 的演練。
- 無伺服器登陸，提供教學影片。
- Happy Little API Shorts 是一系列簡短的教學影片。

# Amazon API Gateway 概念

下節說明使用 API Gateway 的介紹性概念。

API Gateway

API Gateway 是一項支援下列項目 AWS 的服務：

- 建立、部署和管理 RESTful 應用程式程式設計界面 (API)，以公開後端 HTTP 端點、 AWS Lambda 函數或其他 AWS 服務。
- 建立、部署和管理 WebSocket API，以公開 AWS Lambda 函數或其他 AWS 服務。
- 透過前端 HTTP 及 WebSocket 端點呼叫已公開的 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。您可以在一或多個階段中部署此集合。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。如需詳細資訊，請參閱the section called “在 REST API 與 HTTP API 之間選擇 ”。

API Gateway HTTP API

與後端 HTTP 端點或 Lambda 函數整合的路由和方法集合。您可以在一或多個階段中部署此集合。每個路由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。如需詳細資訊，請參閱the section called “在 REST API 與 HTTP API 之間選擇 ”。

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。您可以在一或多個階段中部署此集合。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 AWS 的帳戶 （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
- 私有 API 端點
- 區域 API 端點

API 金鑰

API Gateway 用來識別使用 REST 或 WebSocket API 之應用程式開發人員的英數字元字串。API Gateway 可以代表您產生 API 金鑰，或從 CSV 檔案匯入它們。您可以同時使用 API 金鑰與 Lambda 授權方或用量計劃，以控制對 API 的存取。

請參閱 API 端點。

API 擁有者

請參閱 API 開發人員。

API 階段

API 生命週期狀態的邏輯參考 (例如，'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

不一定擁有 AWS 帳戶，並與 API 開發人員已部署的 API 互動的應用程式建立者。應用程式開發人員是您的客戶。通常會透過 API 金鑰識別應用程式開發人員。

回呼 URL

當新的用戶端透過 WebSocket 連線連線時，您可以呼叫 API Gateway 中的 整合，以存放用戶端的回呼 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

API 提供者可以與 API 消費者共用其 APIs和 API 文件的應用程式。APIs會分組為產品，這是 REST API 端點、API 文件和補充產品文件的集合。

請參閱 API Gateway 入口網站。

邊緣最佳化的 API 端點

使用 CloudFront 分佈來促進用戶端存取時，部署到指定區域之 API Gateway API 的預設主機名稱。 AWS 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。

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) 表示的指令碼，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 API 方法的公有界面，定義應用程式開發人員必須在請求中傳送以透過 API 存取後端的參數和內文。

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。作為 API 開發人員，您可決定 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。

Portal

請參閱開發人員入口網站。

入口網站產品

您要共用的服務或功能。您的入口網站產品是產品 REST 端點和產品頁面的集合。產品 REST 端點是入口網站產品的存取點，由 REST API 的路徑和方法及其部署的階段組成。產品頁面是您用來說明 API 消費者如何使用您產品端點的文件。您可以跨 AWS 帳戶共用產品，將它們新增至入口網站。

請參閱建立產品。

私有 API

請參閱私有 API 端點。

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，而且只能使用已授與存取權限的 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

簡化的 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 會傳送整個請求作為後端 Lambda 函數的輸入。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 `{proxy+}` 方法，以 Greedy 路徑變數 (如 ANY) 呈現。

快速建立

您可以使用快速建立來簡化 HTTP API 的建立。快速建立會建立具有 Lambda 或 HTTP 整合的 API、預設的全部捕獲路由，以及設定為自動部署變更的預設階段。如需更多詳細資訊，請參閱 the section called “使用 CLI AWS 建立 HTTP API”。

區域 API 端點

部署到指定 區域的 API 主機名稱，旨在為相同區域中的 EC2 執行個體等用戶端提供服務 AWS 。API 請求的目標是直接設為區域特定 API Gateway API，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，以使用相同的區域 API 端點組態來將 API 部署至多個區域、設定每個已部署 API 的相同自訂網域名稱，以及在 Route 53 中設定延遲型 DNS 記錄，以將用戶端請求路由至擁有最低延遲的區域。

請參閱 API 端點。

路由

API Gateway 中的 WebSocket 路由用於根據訊息的內容，將傳入的訊息導向特定整合，例如 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

用量計劃提供所選的 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 會持續維持用戶端和 API Gateway 本身之間的連線。與 API Gateway 和後端整合 (例如 Lambda 函數) 之間沒有持續連線。根據從用戶端接收的郵件內容，視需要呼叫後端服務。

# 在 REST API 與 HTTP API 之間選擇

REST API 和 HTTP API 都是 RESTful API 產品。REST API 支援比 HTTP API 更多的功能，而 HTTP API 的設計具有最少功能，因此它們能以較低的價格提供。如果您需要 API 金鑰、個別用戶端限制、請求驗證、 AWS WAF 整合或私有 API 端點等功能，請選擇 REST API。如果您不需要 REST API 中包含的功能，請選擇 HTTP API。

下節摘要說明REST API 和 HTTP API 中可用的核心功能。必要時會提供其他連結，以在《API Gateway 開發人員指南》的 REST API 和 HTTP API 章節之間導覽。

## 端點類型

端點類型是指 API Gateway 為您的 API 建立的端點。如需詳細資訊，請參閱the section called “API Gateway 端點類型”。

| 端點類型 | REST API | HTTP API |
| --- | --- | --- |
| 邊緣最佳化 | 是 | 否 |
| 區域性 | 是 | 是 |
| 私有 | 是 | 否 |

## 安全

API Gateway 提供多種方法來保護 API 免於遭受特定威脅，例如惡意行為者或流量高峰。如需了解詳細資訊，請參閱 the section called “保護” 和 the section called “保護”。

| 安全性功能 | REST API | HTTP API |
| --- | --- | --- |
| 交互 TLS 驗證 | 是 | 是 |
| 後端身分驗證的憑證 | 是 | 否 |
| AWS WAF | 是 | 否 |

## Authorization

API Gateway 支援多種機制來控制和管理 API 的存取。如需詳細資訊，請參閱the section called “存取控制”及the section called “存取控制”。

| 授權選項 | REST API | HTTP API |
| --- | --- | --- |
| IAM | 是 | 是 |
| 資源政策 | 是 | 否 |

| 授權選項 | REST API | HTTP API |
| --- | --- | --- |
| Amazon Cognito | 是 | 是 [1] |
| 使用 AWS Lambda 函數的自訂授權 | 是 | 是 |
| JSON Web Token (JWT) [2] | 否 | 是 |

[1]您可以搭配 JWT 授權器使用 Amazon Cognito。

[2]您可以使用Lambda 授權器來驗證 REST API 的 JWT。

## API 管理

如果您需要 API 管理功能 (例如 API 金鑰和個別用戶端速率限制)，請選擇 REST API。如需詳細資訊，請參閱the section called “分配”、the section called “自訂網域名稱”及the section called “自訂網域名稱”。

| 功能 | REST API | HTTP API |
| --- | --- | --- |
| 自訂網域 | 是 | 是 |
| API 金鑰 | 是 | 否 |
| 個別用戶端速率限制 | 是 | 否 |
| 個別用戶端使用量限制 | 是 | 否 |
| 開發人員入口網站 | 是 | 否 |

## 開發

在開發 API Gateway API 時，您可以決定 API 的許多特性。這些特性取決於 API 的使用案例。如需詳細資訊，請參閱 the section called “開發” 及 the section called “開發”。

| 功能 | REST API | HTTP API |
| --- | --- | --- |
| CORS 組態 | 是 | 是 |
| 測試叫用 | 是 | 否 |
| 快取 | 是 | 否 |
| 使用者控制的部署 | 是 | 是 |
| 自動部署 | 否 | 是 |
| 自訂閘道回應 | 是 | 否 |
| Canary 版本部署 | 是 | 否 |
| 請求驗證 | 是 | 否 |
| 請求參數轉換 | 是 | 是 |
| 請求主體轉換 | 是 | 否 |

## 監控

API Gateway 支援多個選項來記錄 API 請求和監控 API。如需詳細資訊，請參閱the section called “監控”及the section called “監控”。

| 功能 | REST API | HTTP API |
| --- | --- | --- |
| Amazon CloudWatch 指標 | 是 | 是 |
| CloudWatch Logs 的存取日誌 | 是 | 是 |
| Amazon Data Firehose 的存取日誌 | 是 | 否 |
| 執行日誌 | 是 | 否 |
| AWS X-Ray 追蹤 | 是 | 否 |

## 整合

將 API Gateway API 連接到後端資源的各項整合。如需詳細資訊，請參閱the section called “整合”及the section called “整合”。

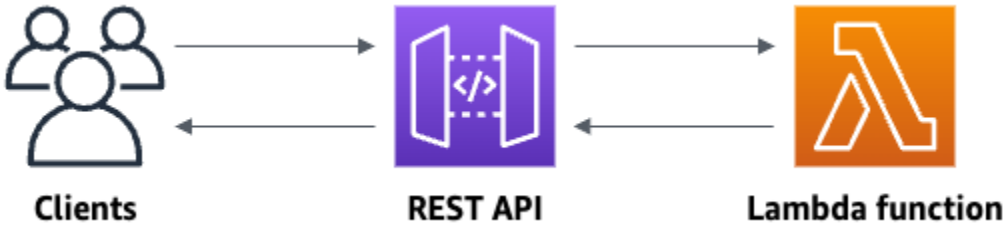
| 功能 | REST API | HTTP API |
| --- | --- | --- |
| 公有 HTTP 端點 | 是 | 是 |
| AWS 服務 | 是 | 是 |
| AWS Lambda 函數 | 是 | 是 |
| 與 Network Load Balancer 的私有整合 | 是 | 是 |
| 與 Application Load Balancer 的私有整合 | 是 | 是 |
| 與 的私有整合 AWS Cloud Map | 否 | 是 |
| 模擬整合 | 是 | 否 |
| 回應串流 | 是 | 否 |

# 開始使用 REST API 主控台

在此入門練習中，您會使用 API Gateway REST API 主控台建立無伺服器 REST API。無伺服器 API 讓您能夠專注於應用程式，而不必花時間佈建和管理伺服器。此練習過程不到 20 分鐘即可完成，而且可以包含在 AWS 免費方案中。

首先，使用 Lambda 主控台建立 Lambda 函數。接著，使用 API Gateway REST API 主控台建立 REST API。然後，建立 API 方法，並使用 Lambda 代理整合將它與 Lambda 函數整合。最後，部署和調用 API。

在您調用 REST API 時，API Gateway 會將請求路由至您的 Lambda 函數。Lambda 會執行函數，並傳回 API Gateway 的回應。API Gateway 隨後將該回應傳回給您。




若要完成此練習，您需要具有主控台存取權的 AWS 帳戶 和 AWS Identity and Access Management (IAM) 使用者。如需詳細資訊，請參閱設定為使用 API Gateway。

主題


- 步驟 1：建立 Lambda 函數
- 步驟 2：建立 REST API
- 步驟 3：建立 Lambda 代理整合
- 步驟 4：部署 API
- 步驟 5：調用 API
- (選用) 步驟 6：清理


## 步驟 1：建立 Lambda 函數

您將使用 Lambda 函數作為您的 API 的後端。Lambda 只有在需要時才會執行程式碼，可自動從每天數項請求擴展成每秒數千項請求。

在本練習中，您將使用 Lambda 主控台中的預設 Node.js 函數。

建立 Lambda 函數

1. 在以下網址登入 Lambda 主控台：https://console.aws.amazon.com/lambda。
2. 選擇 Create function (建立函數)。
3. 在 Basic information (基本資訊) 下，為 Function name (函數名稱) 輸入 **`my-function`**。
4. 其他所有選項請使用預設設定。
5. 選擇建立函數。

預設 Lambda 函數程式碼看起來與下面所示類似：

```
export const handler = async (event) => {
```

```
    const response = {
        statusCode: 200,
        body: JSON.stringify('The API Gateway REST API console is great!'),
    };
    return response;
};
```

您可以修改您的 Lambda 函數來進行本練習，只要函數的回應與 API Gateway 要求的格式相符即可。

將預設回應內文 (`Hello from Lambda!`) 取代為 `The API Gateway REST API console is great!`。當您調用範例函數時，它會將 `200` 回應傳回至用戶端，並且包含更新的回應。

## 步驟 2：建立 REST API

接下來，您要建立具有根資源 (/) 的 REST API。

若要建立 REST API

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 執行以下任意一項：
   - 若要建立您的第一個 API，請針對 REST API 選擇建置。
   - 如果您之前已建立 API，則選擇建立 API，然後針對 REST API 選擇建置。
3. 對於API 名稱，輸入 **my-rest-api**。
4. 在描述，請輸入描述。
5. 將 API 端點類型保持設定為區域。
6. 針對 IP 位址類型，選擇 IPv4。
7. 選擇建立 API。

## 步驟 3：建立 Lambda 代理整合

接下來，您要在根資源 (/) 上為 REST API 建立 API 方法，並使用代理整合將此方法與您的 Lambda 函數整合。在 Lambda 代理整合中，API Gateway 會將來自用戶端的傳入請求直接傳遞至 Lambda 函數。

若要建立 Lambda 代理整合

1. 選取 / 資源，然後選擇建立方法。

2. 針對方法類型，選取 ANY。
3. 針對整合類型，選取 Lambda。
4. 開啟 Lambda 代理整合。
5. 針對 Lambda 函數，輸入 **my-function**，然後選取您的 Lambda 函數。
6. 選擇建立方法。

## 步驟 4：部署 API

接下來，您要建立 API 部署，並將它與階段建立關聯。

部署 API

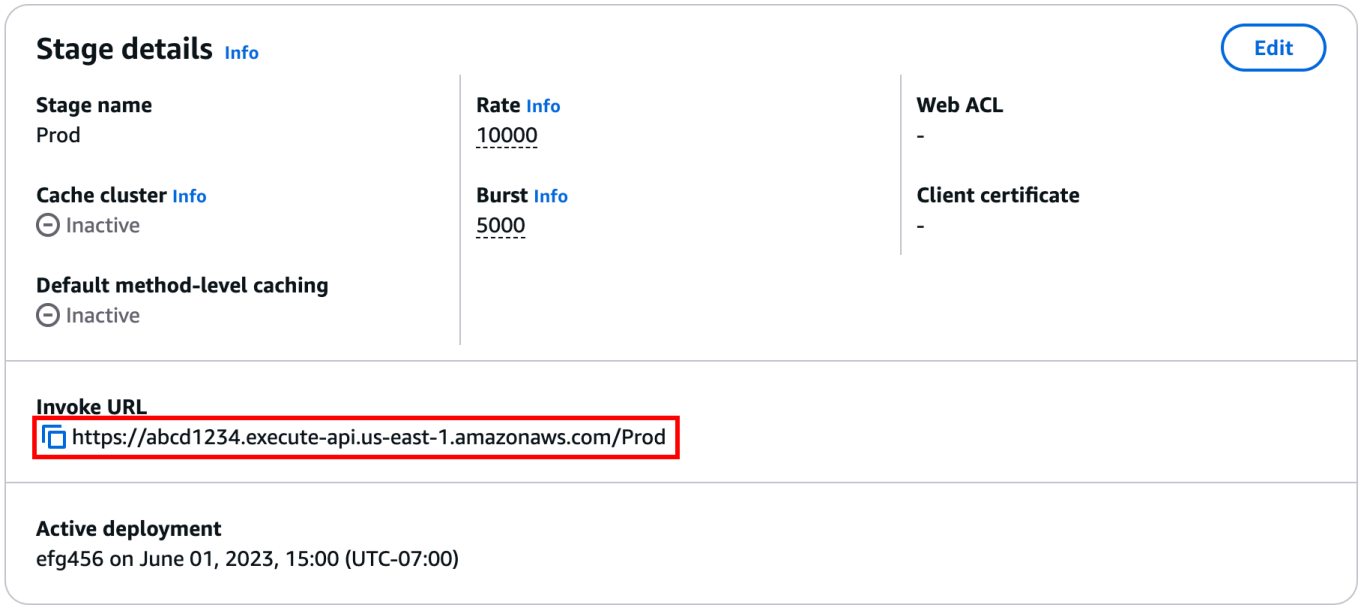
1. 選擇部署 API。
2. 針對階段，選取新階段。
3. 針對階段名稱，輸入 **Prod**。
4. 在描述，請輸入描述。
5. 選擇部署。

現在，用戶端可以呼叫您的 API。若要在部署 API 之前先行測試，您可以選擇性地選擇 ANY 方法，導覽至測試索引標籤，然後選擇測試。

## 步驟 5：調用 API

若要調用您的 API

1. 從主導覽窗格選擇階段。
2. 在階段詳細資訊下，選擇複製圖示以複製 API 的調用 URL。




3. 在 Web 瀏覽器中輸入調用 URL。

   完整 URL 看起來應該會像這樣：`https://abcd123.execute-api.us-east-2.amazonaws.com/Prod`。

   您的瀏覽器將向 API 傳送 GET 請求。

4. 驗證您的 API 的回應。您應該會在瀏覽器中看到文字 `"The API Gateway REST API console is great!"`。

## (選用) 步驟 6：清理

為避免您的 產生不必要的成本 AWS 帳戶，請刪除您在本練習中建立的資源。下列步驟可刪除您的 REST API、Lambda 函數和關聯的資源。

若要刪除 REST API

1. 在資源窗格中，選擇 API 動作、刪除 API。
2. 在刪除 API 對話方塊中，輸入確認，然後選擇刪除。

若要刪除 Lambda 函數

1. 在以下網址登入 Lambda 主控台：https://console.aws.amazon.com/lambda。
2. 在函數頁面上，選取您的函數。選擇 動作、刪除。

3. 在刪除 1 函數對話方塊中輸入 **delete**，然後選擇刪除。

若要刪除 Lambda 函數的日誌群組

1. 開啟 Amazon CloudWatch 主控台的 Log groups (記錄群組) 頁面。
2. 在日誌群組頁面上，選取函數的日誌群組 (`/aws/lambda/my-function`)。然後針對動作，選擇刪除日誌群組。
3. 在 刪除日誌群組 對話方塊中，選擇 刪除 。

若要刪除 Lambda 函數的執行角色

1. 開啟 IAM 主控台中的角色頁面。
2. (選用) 在角色頁面的搜尋方塊中，輸入 **my-function**。
3. 選取函數的角色 (例如 `my-function-31exxmpl`)，然後選擇刪除。
4. 在要刪除 **my-function-*31exxmpl*** 嗎？對話方塊中，輸入角色的名稱，然後選擇刪除。

> **Tip**
>
> 您可以使用 AWS CloudFormation 或 AWS Serverless Application Model () 自動化 AWS 資源的建立和清除AWS SAM。如需一些範例 CloudFormation 範本，請參閱 awsdocs GitHub 儲存庫中 API Gateway 的範例範本。

# 設定為使用 API Gateway

首次使用 Amazon API Gateway 之前，請先完成下列作業：

## 註冊 AWS 帳戶

若要開始使用 AWS，您需要 AWS 帳戶。如需建立 的相關資訊 AWS 帳戶，請參閱《 AWS 帳戶管理參考指南》中的 入門 AWS 帳戶。

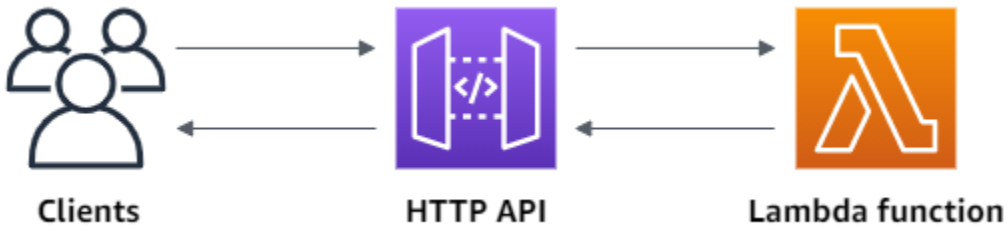
# API Gateway 入門

在此入門練習中，您將建立一個無伺服器 API。無伺服器 API 讓您能夠專注於應用程式，而不必花時間佈建和管理伺服器。此練習需要不到 20 分鐘的時間即可完成，而且可以包含在 AWS 免費方案中。

首先，您需使用 AWS Lambda 主控台建立 Lambda 函數。接著，您可以使用 API Gateway 主控台建立 HTTP API。然後，您將叫用 API。

> **Note**
>
> 本練習使用 HTTP API。API Gateway 也支援 REST API，其中包含更多功能。如需使用 REST API 的教學課程，請參閱 the section called “開始使用 REST API 主控台”。
> 如需 HTTP API 和 REST API 之間差異的詳細資訊，請參閱 the section called “在 REST API 與 HTTP API 之間選擇 ”。

在您調用 HTTP API 時，API Gateway 會將請求路由至您的 Lambda 函數。Lambda 會執行 Lambda 函數，並傳回 API Gateway 的回應。API Gateway 隨後將回應傳回給您。




若要完成此練習，您需要 AWS 帳戶，以及具有主控台存取權的 AWS Identity and Access Management 使用者。如需更多詳細資訊，請參閱 先決條件。

主題

- 步驟 1：建立 Lambda 函數
- 步驟 2：建立 HTTP API
- 步驟 3：測試您的 API
- (選用) 步驟 4：清理
- 後續步驟

# 步驟 1：建立 Lambda 函數

您將使用 Lambda 函數作為您的 API 的後端。Lambda 只有在需要時才會執行程式碼，可自動從每天數項請求擴展成每秒數千項請求。

在此範例中，您將從 Lambda 主控台使用預設 Node.js 函數。

建立 Lambda 函數

1. 在以下網址登入 Lambda 主控台：https://console.aws.amazon.com/lambda。
2. 選擇 Create function (建立函數)。
3. 針對 Function name (函數名稱)，請輸入 **`my-function`**。
4. 其他所有選項請使用預設設定。
5. 選擇 Create function (建立函數)。

範例函數將 `200` 回應傳回給用戶端，以及文字 `Hello from Lambda!`。

您可以修改您的 Lambda 函數，函數的回應與 API Gateway 要求的格式要相符。

預設 Lambda 函數程式碼看起來與下面所示類似：

```
export const handler = async (event) => {
    const response = {
        statusCode: 200,
        body: JSON.stringify('Hello from Lambda!'),
    };
    return response;
};
```

# 步驟 2：建立 HTTP API

接著，您將建立一個 HTTP API。API Gateway 還支援 REST API 和 WebSocket API，但 HTTP API 是此練習的最佳選擇。REST API 支援比 HTTP API 更多的功能，但本練習中不需要那些功能。HTTP API 設計為具有基本功能，因此能以較低的價格提供。WebSocket API 將維持與用戶端的持續連線，以進行全雙工通訊，這並非此範例所必須的。

HTTP API 將針對您的 Lambda 函數提供 HTTP 端點。API Gateway 會將請求路由至您的 Lambda 函數，然後將函數的回應傳回給用戶端。

建立 HTTP API

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 請執行下列其中一項：
   - 若要建立您的第一個 API，針對 HTTP API，選擇 Build (建置)。
   - 如果您之前已建立 API，則選擇 Create API (建立 API)，然後針對 HTTP API 選擇 Build (建置)。
3. 針對 Integrations (整合)，選擇 Add integration (新增整合)。
4. 選擇 Lambda。
5. 對於 Lambda function (Lambda 函數)，請輸入 **`my-function`**。
6. 針對 API name (API 名稱)，請輸入 **`my-http-api`**。
7. 選擇 Next (下一步)。
8. 檢閱 API Gateway 為您建立的 route (路由)，然後選擇 Next (下一步)。
9. 檢閱 API Gateway 為您建立的 stage (階段)，然後選擇 Next (下一步)。
10. 選擇 Create (建立)。

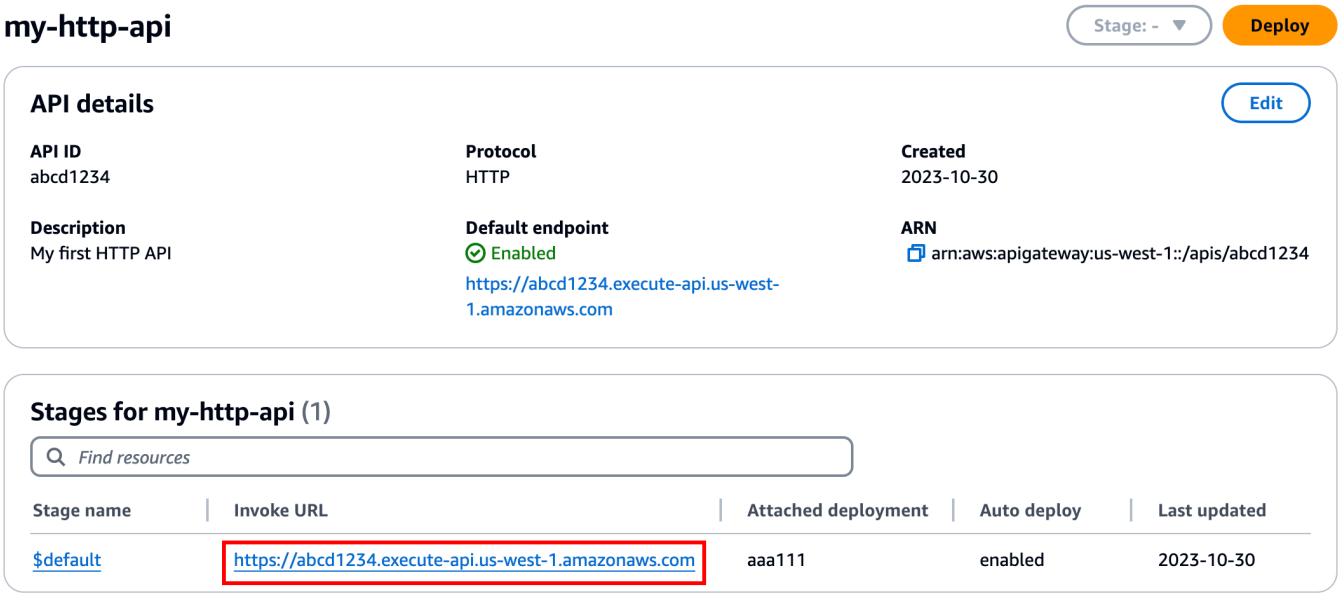
現在，您已建立具有 Lambda 整合的 HTTP API，可隨時接收來自用戶端的請求。

## 步驟 3：測試您的 API

接著，您將測試您的 API 以確保其正常工作。為簡單起見，請使用 Web 瀏覽器來叫用您的 API。

測試您的 API

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 選擇您的 API。
3. 請注意 API 的叫用 URL。




4. 複製 API 的叫用 URL，然後在 Web 瀏覽器中輸入。將 Lambda 函數的名稱附加至叫用 URL，以叫用 Lambda 函數。依預設，API Gateway 主控台會建立與 Lambda 函數相同名稱的路由 `my-function`。

   完整 URL 看起來應該會像這樣：`https://abcdef123.execute-api.us-east-2.amazonaws.com/my-function`。

   您的瀏覽器將向 API 傳送 `GET` 請求。

5. 驗證您的 API 的回應。您應該會在瀏覽器中看到文字 `"Hello from Lambda!"`。

# (選用) 步驟 4：清理

若要避免不必要的成本，請刪除您在此入門練習中建立的資源。採用下列步驟刪除您的 HTTP API、您的 Lambda 函數和相關資源。

刪除 HTTP API

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 在 API 頁面上，選取 API。選擇 Actions (動作)，然後選擇 Delete (刪除)。
3. 選擇 Delete (刪除)。

刪除 Lambda 函數

1. 在以下網址登入 Lambda 主控台：https://console.aws.amazon.com/lambda。
2. 在 Functions (函數) 頁面上，選取函數。選擇 Actions (動作)，然後選擇 Delete (刪除)。
3. 選擇 Delete (刪除)。

刪除 Lambda 函數的日誌群組

1. 在 Amazon CloudWatch 主控台中，開啟 Log groups (日誌群組) 頁面。
2. 在 Log groups (日誌群組) 頁面上，選取函數的日誌群組 (`/aws/lambda/my-function`)。選擇 Actions (動作)，然後選擇 Delete log group (刪除日誌群組)。
3. 選擇 Delete (刪除)。

刪除 Lambda 函數的執行角色

1. 在 AWS Identity and Access Management 主控台中開啟 Roles (角色) 頁面。
2. 選取函數的角色，例如，`my-function-31exxmpl`。
3. 選擇 Delete role (刪除角色)。
4. 選擇 Yes, delete (是，刪除)。

您可以使用 CloudFormation 或 AWS SAM，將 AWS 資源的建立和清理作業自動化。如需 CloudFormation 範本範例，請參閱範例 CloudFormation 範本。

## 後續步驟

在此範例中，您使用了 AWS 管理主控台 來建立簡單的 HTTP API。HTTP API 叫用 Lambda 函數，並將回應傳回給用戶端。

以下是繼續使用 API Gateway 的後續步驟。

- 設定其他類型的 API 整合，包括：
  - HTTP 端點
  - VPC 中的私有資源，例如 Amazon ECS 服務
  - AWS 服務，例如 Amazon Simple Queue Service、AWS Step Functions 和 Kinesis Data Streams

- 控制對您的 API 的存取
- 啟用您的 API 的日誌記錄
- 設定您的 API 節流
- 設定您的 API 的自訂網域

若要從社群取得 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。

如需逕洽 AWS 獲取 API Gateway 的相關協助，請參閱 AWS 支援頁面的支援選項。

另請參閱常見問答集 (FAQ)，或直接聯絡我們。

# Amazon API Gateway 教學課程和研討會

下列教學課程和研討會提供實作練習以協助您了解 API Gateway。若要開始使用，建議您完成 the section called “教學：建立具 Lambda 代理整合的 REST API”。

REST API 教學課程

- 選擇 AWS Lambda 整合教學課程
- 教學課程：匯入範例來建立 REST API
- 選擇 HTTP 整合教學課程
- 教學：建立具私有整合的 REST API
- 教學課程：使用 AWS 整合建立 REST API
- 教學課程：建立具有兩個 AWS 服務整合和一個 Lambda 非代理整合的計算器 REST API
- 教學：建立 REST API 做為 Amazon S3 代理
- 教學課程：建立 REST API 做為 Amazon Kinesis 代理
- 教學課程：使用 AWS SDKs或 建立 REST API AWS CLI
- 教學：建立私有 REST API

HTTP API 教學課程

- 教學：使用 Lambda 和 DynamoDB 建立 CRUD HTTP API
- 教學：透過私有整合至 Amazon ECS 服務來建立 HTTP API

WebSocket API 教學課程

- 教學：使用 WebSocket API、Lambda 和 DynamoDB 建立 WebSocket 聊天應用程式

研討會

- 建置無伺服器 Web 應用程式
- 無伺服器應用程式的 CI/CD
- 無伺服器安全研討會
- 無伺服器身分管理、身分驗證與授權
- Amazon API Gateway 工作坊

# Amazon API Gateway REST API 教學課程

下列教學課程提供實作練習以協助您了解 API Gateway REST API。

主題


- 選擇 AWS Lambda 整合教學課程
- 教學課程：匯入範例來建立 REST API
- 選擇 HTTP 整合教學課程
- 教學：建立具私有整合的 REST API
- 教學課程：使用 AWS 整合建立 REST API
- 教學課程：建立具有兩個 AWS 服務整合和一個 Lambda 非代理整合的計算器 REST API
- 教學：建立 REST API 做為 Amazon S3 代理
- 教學課程：建立 REST API 做為 Amazon Kinesis 代理
- 教學課程：使用 AWS SDKs或 建立 REST API AWS CLI
- 教學：建立私有 REST API


## 選擇 AWS Lambda 整合教學課程

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，Lambda 函數的輸入可以表示為的請求標頭、路徑變數、查詢字串參數和 API 組態資料的任何組合。您只需選擇 Lambda 函數。API Gateway 會設定整合請求和整合回應。設定之後，API 方法可以在不修改現有設定的情況下不斷演進。這是可能的，因為後端 Lambda 函數會剖析傳入的請求資料，並回應用戶端。

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。您必須將用戶端提供為請求參數的任何輸入資料，映射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。

在 Lambda 代理或 Lambda 非代理整合中，您可以在與您建立 API 的帳戶不同的帳戶中使用 Lambda 函數。

主題


- 教學：建立具 Lambda 代理整合的 REST API
- 教學：建立具 Lambda 非代理整合的 REST API

- 教學：建置具 cross-account Lambda 代理整合的 REST API

## 教學：建立具 Lambda 代理整合的 REST API

Lambda 代理整合是一種輕量型彈性 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 函數作為 API 的後端。
- 建立和測試 具有 Lambda 代理整合的 API。

主題

- 建立 "Hello, World!" Lambda 函式
- 建立 "Hello, World!" API
- 部署和測試 API

建立 "Hello, World!" Lambda 函式

建立 "Hello, World!" Lambda 控制台中的 Lambda 函數

1. 在以下網址登入 Lambda 主控台：https://console.aws.amazon.com/lambda。
2. 在 AWS 導覽列上，選擇 AWS 區域。

   > **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. 在函數名稱 中，輸入 **`GetStartedLambdaProxyIntegration`**。

   b. 針對執行期，請選擇最新支援的 Node.js 或 Python 執行期。

c. 對於架構，請保留預設設定。

d. 在許可下，展開變更預設執行角色。從執行角色下拉式清單中，選擇從 AWS 政策範本建立新角色。

e. 在角色名稱 中，輸入 **GetStartedLambdaBasicExecutionRole**。

f. 將 Policy templates (政策範本) 欄位留白。

g. 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：

Node.js

```
export const handler = async(event, context) => {
    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;
        }
    } 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 + "!";
    return res
};
```

Python

```
import json


def lambda_handler(event, context):
    print(event)

    greeter = 'World'

    try:
        if (event['queryStringParameters']) and (event['queryStringParameters']
['greeter']) and (
                event['queryStringParameters']['greeter'] is not None):
            greeter = event['queryStringParameters']['greeter']
    except KeyError:
        print('No greeter')

    try:
        if (event['multiValueHeaders']) and (event['multiValueHeaders']
['greeter']) and (
                event['multiValueHeaders']['greeter'] is not None):
            greeter = " and ".join(event['multiValueHeaders']['greeter'])
    except KeyError:
        print('No greeter')

    try:
        if (event['headers']) and (event['headers']['greeter']) and (
                event['headers']['greeter'] is not None):
            greeter = event['headers']['greeter']
    except KeyError:
        print('No greeter')

    if (event['body']) and (event['body'] is not None):
        body = json.loads(event['body'])
        try:
            if (body['greeter']) and (body['greeter'] is not None):
                greeter = body['greeter']
        except KeyError:
            print('No greeter')

    res = {
        "statusCode": 200,
```

```
        "headers": {
            "Content-Type": "*/*"
        },
        "body": "Hello, " + greeter + "!"
    }

    return res
```

8. 選擇 Deploy (部署)。

建立 "Hello, World!" API

立即為您的 "Hello, World!" 建立 API 使用 API Gateway 主控台的 Lambda 函數。

建立 "Hello, World!" API

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 如果這是您第一次使用 API Gateway，您會看到服務功能的介紹頁面。在 REST API 下方，選擇 Build (組建)。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)。在 REST API 下方，選擇組建。
3. 對於API 名稱，輸入 **LambdaProxyAPI**。
4. 在描述，請輸入描述。
5. 將 API 端點類型保持設定為區域。
6. 針對 IP 位址類型，選擇 IPv4。
7. 選擇建立 API。

在建立 API 之後，請建立資源。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。在此範例中，您會建立 /helloworld 資源。

建立資源

1. 選擇建立資源。
2. 讓代理資源保持關閉。
3. 將資源路徑保持為 /。
4. 針對資源名稱，輸入 **helloworld**。

5. 讓 CORS (跨來源資源分享) 保持關閉。
6. 選擇建立資源。

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。

建立 **ANY** 方法

1. 選取 /helloworld 資源，然後選擇建立方法。
2. 針對方法類型，選取 ANY。
3. 針對整合類型，選取 Lambda 函數。
4. 開啟 Lambda 代理整合。
5. 針對 Lambda 函數，選取您建立 Lambda 函數 AWS 區域 的 ，然後輸入函數名稱。
6. 若要使用 29 秒的預設逾時值，請將預設逾時保持開啟。若要設定自訂逾時，請選擇預設逾時，然後輸入介於 50 和 29000 毫秒之間的逾時值。
7. 選擇建立方法。

部署和測試 API

部署 API

1. 選擇部署 API。
2. 針對階段，選取新階段。
3. 針對階段名稱，輸入 **test**。
4. 在描述，請輸入描述。
5. 選擇部署。
6. 在階段詳細資訊下，選擇複製圖示以複製 API 的調用 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。

若要建立適用於 API `helloworld` 資源的 URL，請將資源 `helloworld` 和查詢字串參數 `?greeter=John` 附加至您的調用 URL。您的 URL 看起來應該如下所示。

```
https://r275xc9bmd.execute-api.us-east-1.amazonaws.com/test/helloworld?greeter=John
```

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視窗，然後將調用 URL 取代為您在上一個步驟中複製的 URL，再將 **`/helloworld`** 新增至該 URL 的結尾。

   > **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" -H "content-type: application/json" -d "{ \"greeter\":
   >  \"John\" }"
   > ```

   a. 使用查詢字串參數 `?greeter=John` 呼叫 API：

   ```
   curl -X GET 'https://r275xc9bmd.execute-api.us-east-1.amazonaws.com/test/
   helloworld?greeter=John'
   ```

   b. 使用標頭參數 `greeter:John` 呼叫 API：

   ```
   curl -X GET https://r275xc9bmd.execute-api.us-east-1.amazonaws.com/test/
   helloworld \
     -H 'content-type: application/json' \
     -H 'greeter: John'
   ```

   c. 使用內文 `{"greeter":"John"}` 呼叫 API：

   ```
   curl -X POST https://r275xc9bmd.execute-api.us-east-1.amazonaws.com/test/
   helloworld \
     -H 'content-type: application/json' \
   ```

```
  -d '{ "greeter": "John" }'
```

在所有情況下，輸出都是具有下列回應內文的 200 回應：

```
Hello, John!
```

## 教學：建立具 Lambda 非代理整合的 REST 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。如需有關 AWS Lambda 和 Lambda 函數的詳細資訊，請參閱 AWS Lambda 開發人員指南。

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：建立具有兩個 AWS 服務整合和一個 Lambda 非代理整合的計算器 REST API。

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
- 建立具有 Lambda 非代理整合的 API
- 測試呼叫 API 方法
- 部署 API
- 在部署階段測試 API
- 清除

### 為 Lambda 非代理整合建立 Lambda 函數

> **Note**
>
> 建立 Lambda 函數可能會導致 AWS 您的帳戶產生費用。

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 函數的實作：

Node.js

```
'use strict';
var days = ['Sunday', 'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday',
 'Saturday'];
var times = ['morning', 'afternoon', 'evening', 'night', 'day'];

export const handler = async(event) => {
  console.log(event);
  // Parse the input for the name, city, time and day property values
  let name = event.name === null || event.name === undefined || event.name === "" ?
 '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
  return {"greeting": greeting}
};
```

Python

```
import json

days = {
    'Sunday',
    'Monday',
    'Tuesday',
    'Wednesday',
    'Thursday',
    'Friday',
    'Saturday'}
times = {'morning', 'afternoon', 'evening', 'night', 'day'}
```

```
def lambda_handler(event, context):
    print(event)
    # parse the input for the name, city, time, and day property values
    name = event.get("name") or 'you'
    city = event.get("city") or 'World'
    try:
        if event['time'] in times:
            time = event['time']
        else:
            time = 'day'
    except KeyError:
        time = 'day'
    try:
        if event['day'] in days:
            day = event['day']
        else:
            day = ''
    except KeyError:
        day = ''
    # Generate a greeting
    greeting = 'Good ' + time + ', ' + name + ' of ' + \
        city + '.' + ['', ' Happy ' + day + '!'][day != '']
    # Log the greeting to CloudWatch
    print(greeting)

    # Return a greeting to the caller
    return {"greeting": greeting}
```

對於 Lambda 自訂整合，API Gateway 會從用戶端傳遞 Lambda 函數的輸入作為整合請求內文。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) 給發起人。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 開發人員指南。

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) 並設定叫用角色與政策，API Gateway 才能叫用 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 許可權控制 REST API 的存取。

相較於 `GetStartedLambdaProxyIntegration` (Lambda 代理整合的 Lambda 函數)，`GetStartedLambdaIntegration` (Lambda 自訂整合的 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 請求參數作為輸入，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. 在 https：//https://console.aws.amazon.com/lambda/ 開啟 AWS Lambda 主控台。
2. 執行以下任意一項：
   - 出現歡迎頁面時，請選擇 Get Started Now (立即開始使用)，然後選擇 Create function (建立函數)。
   - 出現 Lambda > Functions (Lambda > 函數) 清單頁面時，請選擇 Create 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. 在名稱，輸入 **`GetStartedLambdaIntegration`** 做為 Lambda 函數名稱。
   b. 針對執行期，請選擇最新支援的 Node.js 或 Python 執行期。
   c. 對於架構，請保留預設設定。

d. 在許可下，展開變更預設執行角色。從執行角色下拉式清單中，選擇從 AWS 政策範本建立新角色。

e. 在角色名稱中輸入角色名稱 (例如 **GetStartedLambdaIntegrationRole**)。

f. 對於 Policy templates (政策範本)，選擇 Simple microservice permissions (簡易微服務許可)。

g. 選擇 Create function (建立函數)。

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. 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。

b. 對於此區段中的所有其他欄位，則保留預設選項。

c. 選擇 Deploy (部署)。

6. 若要測試新建立的函數，請選擇測試索引標籤。

a. 事件名稱輸入 **HelloWorldTest**。

b. 對於事件 JSON，請使用下列命令取代預設程式碼。

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

c. 選擇測試以呼叫函數。執行結果：成功區段會隨即顯示。展開詳細資訊，您會看到下列輸出。

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

該輸出也會寫入 CloudWatch Logs。

作為附帶練習，您可使用 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 角色會附加兩個內嵌政策。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 `GetStartedLambdaIntegration` Lambda 函數的 CloudWatch 串流與記錄日誌事件。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "logs:CreateLogGroup",
            "Resource": "arn:aws:logs:us-east-1:111111111111:*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogStream",
                "logs:PutLogEvents"
            ],
            "Resource": [
                "arn:aws:logs:us-east-1:111111111111:log-group:/aws/lambda/
GetStartedLambdaIntegration:*"
            ]
        }
    ]
}
```

另一個政策文件適用於叫用此範例中 AWS 未使用的其他服務。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：

JSON

```
{
  "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 函數叫用 `console.log()` 函數來將事件記錄到 CloudWatch Logs。

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，我們會透過泛型 HTTP 方法公開 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。

建立具有 Lambda 非代理整合的 API

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 如果這是您第一次使用 API Gateway，您會看到服務功能的介紹頁面。在 REST API 下方，選擇 Build (組建)。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)。在 REST API 下方，選擇組建。
3. 對於API 名稱，輸入 **`LambdaNonProxyAPI`**。
4. 在描述，請輸入描述。
5. 將 API 端點類型保持設定為區域。
6. 針對 IP 位址類型，選擇 IPv4。
7. 選擇建立 API。

建立 API 之後，請建立 /{city} 資源。這是具有路徑變數的資源範例，其會接受來自用戶端的輸入。稍後，您會使用對應範本將此路徑變數對應到 Lambda 函數輸入。

建立資源

1. 選擇建立資源。
2. 讓代理資源保持關閉。
3. 將資源路徑保持為 `/`。
4. 針對資源名稱，輸入 **`{city}`**。

5. 讓 CORS (跨來源資源分享) 保持關閉。
6. 選擇建立資源。

在建立 /{city} 資源後，請建立 ANY 方法。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。

建立 **ANY** 方法

1. 選取 /{city} 資源，然後選擇建立方法。
2. 針對方法類型，選取 ANY。
3. 針對整合類型，選取 Lambda 函數。
4. 將 Lambda 代理整合保持關閉。
5. 針對 Lambda 函數，選取您建立 Lambda 函數 AWS 區域 的 ，然後輸入函數名稱。
6. 選擇方法請求設定。

   此時，請開啟 URL 路徑變數、查詢字串參數與標頭的請求驗證程式，以確保所有必要的資料都已定義。在此範例中，您會建立 `time` 查詢字串參數和 `day` 標頭。
7. 對於請求驗證程式，選取驗證查詢字串參數與標頭。
8. 選擇 URL 查詢字串參數，然後執行下列動作：

   a. 選擇新增查詢字串。

   b. 對於名稱，輸入 **`time`**。

   c. 開啟必要。

   d. 讓快取保持關閉。
9. 選擇 HTTP 請求標頭，然後執行下列動作：

   a. 選擇新增標頭。

   b. 對於名稱，輸入 **`day`**。

   c. 開啟必要。

   d. 讓快取保持關閉。
10. 選擇建立方法。

開啟請求驗證程式之後，請根據後端 Lambda 函數的要求，新增內文對應範本將傳入請求轉換為 JSON 承載，以設定 ANY 方法的整合請求。

設定整合請求

1. 在整合請求標籤上，於整合請求設定下，選擇編輯。
2. 針對請求內文傳遞，選取未定義範本時 (建議)。
3. 選擇對應範本。
4. 選擇新增對應範本。
5. 針對內容類型，輸入 **application/json**。
6. 針對範本內文，輸入下列程式碼：

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

7. 選擇儲存。

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 **ANY** 方法

1. 選擇測試標籤。您可能需要選擇向右箭頭按鈕才能顯示此索引標籤。
2. 針對方法類型，選取 POST。
3. 針對路徑，在城市下輸入 **Seattle**。

4. 針對查詢字串，輸入 **time=morning**。
5. 針對標頭，輸入 **day:Wednesday**。
6. 針對請求內文，輸入 **{ "callerName": "John" }**。
7. 選擇測試。

遵循下列方式驗證所傳回的回應承載：

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

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=beags1mnid, X-Amz-
Source-Arn=arn:aws:execute-api:us-west-2:123456789012:beags1mnid/null/POST/
{city}, Accept=application/json, User-Agent=AmazonAPIGateway_beags1mnid,
 X-Amz-Security-Token=FQoDYXdzELL//////////wEaDMHGzEdEOT/VvGhabiK3AzgKrJw
+3zLqJZG4PhOq12K6W21+QotY2rrZyOzqhLoiuRg3CAYNQ2eqgL5D54+63ey9bIdtwHGoyBdq8ecWxJK/
YUnT2Rau0L9HCG5p7FC05h3IvwlFfvcidQNXeYvsKJTLXI05/
yEnY3ttIAnpNYLOezD9Es8rBfyruHfJfOqextKlsC8DymCcqlGkig8qLKcZ0hWJWVwiPJiFgL7laabXs+
+ZhCa4hdZo4iqlG729DE4gaV1mJVdoAagIUwLMo+y4NxFDu0r7I0/
EO5nYcCrppGVVBYiGk7H4T6sXuhTkbNNqVmXtV3ch5bOlh7 [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

1. 選擇部署 API。
2. 針對階段，選取新階段。
3. 針對階段名稱，輸入 **test**。

> **Note**
>
> 輸入必須為 UTF-8 編碼 (例如未本地化的) 文字。

4. 在描述，請輸入描述。
5. 選擇部署。

在階段詳細資訊下，選擇複製圖示以複製 API 的調用 URL。API 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

1. 在連線至網際網路的本機電腦上，開啟終端機視窗。
2. 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。

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。IAM 資源一經刪除，即無法復原。所以若要再次使用該 IAM 資源，必須加以重新建立。

若要刪除 Lambda 函數

1. 登入 AWS 管理主控台 並在 https：//https://console.aws.amazon.com/lambda/ 開啟 AWS Lambda 主控台。
2. 從函數清單中選擇 GetStartedLambdaIntegration，然後依序選擇動作和刪除函數。出現提示時，再次選擇 Delete (刪除)。

刪除相關聯的 IAM 資源

1. 在以下網址開啟 IAM 主控台：https://console.aws.amazon.com/iam/。
2. 從 Details (詳細資訊) 中，選擇 Roles (角色)。
3. 從角色清單中選擇 GetStartedLambdaIntegrationRole，然後依序選擇角色動作和刪除角色。在主控台依照以下步驟刪除角色。

## 教學：建置具 cross-account Lambda 代理整合的 REST API

您現在可以使用來自不同 AWS 帳戶的 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。

建立 API Gateway 跨帳戶 Lambda 整合的 API

建立 API

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 如果這是您第一次使用 API Gateway，您會看到服務功能的介紹頁面。在 REST API 下方，選擇 Build (組建)。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)。在 REST API 下方，選擇組建。
3. 對於API 名稱，輸入 **`CrossAccountLambdaAPI`**。
4. 在描述，請輸入描述。
5. 將 API 端點類型保持設定為區域。
6. 針對 IP 位址類型，選擇 IPv4。
7. 選擇建立 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 (建立函數)。
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. 針對 Function name (函數名稱)，輸入名稱。

   b. 從 Runtime (執行時間) 下拉式清單中，選擇支援的 Node.js 執行時間。

   c. 對於架構，請保留預設設定。

   d. 在 Permissions (許可) 下，展開 Choose or create an execution role (選擇或建立執行角色)。您可以建立角色或選擇現有角色。

   e. 選擇 Create function (建立函數) 繼續。
5. 向下捲動到函數程式碼 窗格中。
6. 從 the section called “教學：建立具 Lambda 代理整合的 REST API” 輸入 Node.js 函數實作。

7. 選擇 Deploy (部署)。
8. 請注意適用於您函數的完整 ARN (在 Lambda 函數窗格的右上角)。在建立跨帳戶 Lambda 整合時會需要用到。

設定跨帳戶 Lambda 整合

在不同的帳戶有 Lambda 整合函數後，您可以使用 API Gateway 主控台在您的第一個帳戶將其新增至 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。

在建立 API 之後，請建立資源。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。在此範例中，您會建立 /helloworld 資源。

建立資源

1. 選擇建立資源。
2. 讓代理資源保持關閉。
3. 將資源路徑保持為 `/`。
4. 針對資源名稱，輸入 **`helloworld`**。
5. 讓 CORS (跨來源資源分享) 保持關閉。
6. 選擇建立資源。

在建立資源之後，請建立 GET 方法。將 GET 方法與另一個帳戶中的 Lambda 函數整合。

建立 **GET** 方法

1. 選取 /helloworld 資源，然後選擇建立方法。
2. 針對方法類型，選取 GET。
3. 針對整合類型，選取 Lambda 函數。
4. 開啟 Lambda 代理整合。

5. 針對 Lambda 函數，輸入步驟 1 中 Lambda 函數的完整 ARN。

   在 Lambda 主控台中，您可以在主控台視窗的右上角找到適用於您函數的 ARN。

6. 當您輸入 ARN 時，便會出現 `aws lambda add-permission` 命令字串。這個政策會授予您的第一個帳戶對第二個帳戶 Lambda 函數的存取權。將`aws lambda add-permission`命令字串複製並貼到為第二個帳戶設定的 AWS CLI 視窗中。
7. 選擇建立方法。

您可以在 Lambda 主控台中查看您的函數適用的更新政策。

(選用) 查看更新的政策

1. 登入 AWS 管理主控台 並在 https：//https://console.aws.amazon.com/lambda/ 開啟 AWS Lambda 主控台。
2. 選擇 Lambda 函數。
3. 選擇許可。

   您現在應該會看到內含 `Allow` 子句的 `Condition` 政策，其中 `AWS:SourceArn` 為您 API `GET` 方法的 ARN。

## 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。

> **Note**
>
> 本教學課程使用 HTTP 端點作為範例。當您建立自己的 API 時，我們建議您使用 HTTPS 端點進行 HTTP 整合。

- `GET /`：適用於未與任何後端端點整合之 API 根資源的讀取存取。API Gateway 會回應 PetStore 網站的概觀。這是 `MOCK` 整合類型範例。

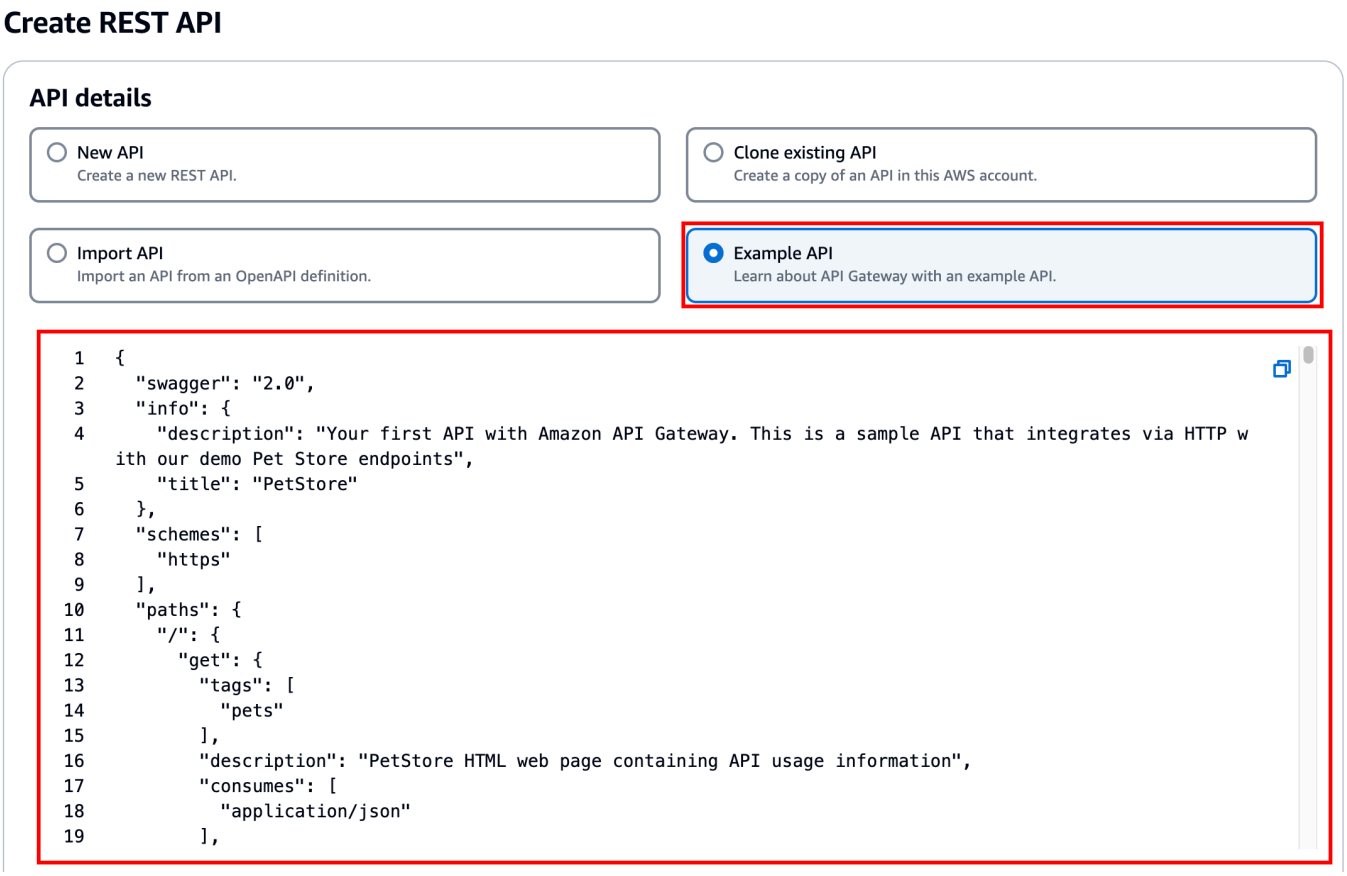
- `GET /pets`：適用於與相同具名後端 `/pets` 資源整合之 API `/pets` 資源的讀取存取。後端會傳回 PetStore 中可用寵物的頁面。這是 HTTP 整合類型範例。整合端點的 URL 是 `http://petstore-demo-endpoint.execute-api.com/petstore/pets`。
- `POST /pets`：適用於與後端 `/pets` 資源整合之 API `/petstore/pets` 資源的寫入存取。在收到正確的請求時，後端會將指定的寵物新增至 PetStore，並將結果傳回給發起人。此整合也是 HTTP。
- `GET /pets/{petId}`：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。

API 支援透過 `OPTIONS` 整合類型之 `MOCK` 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。

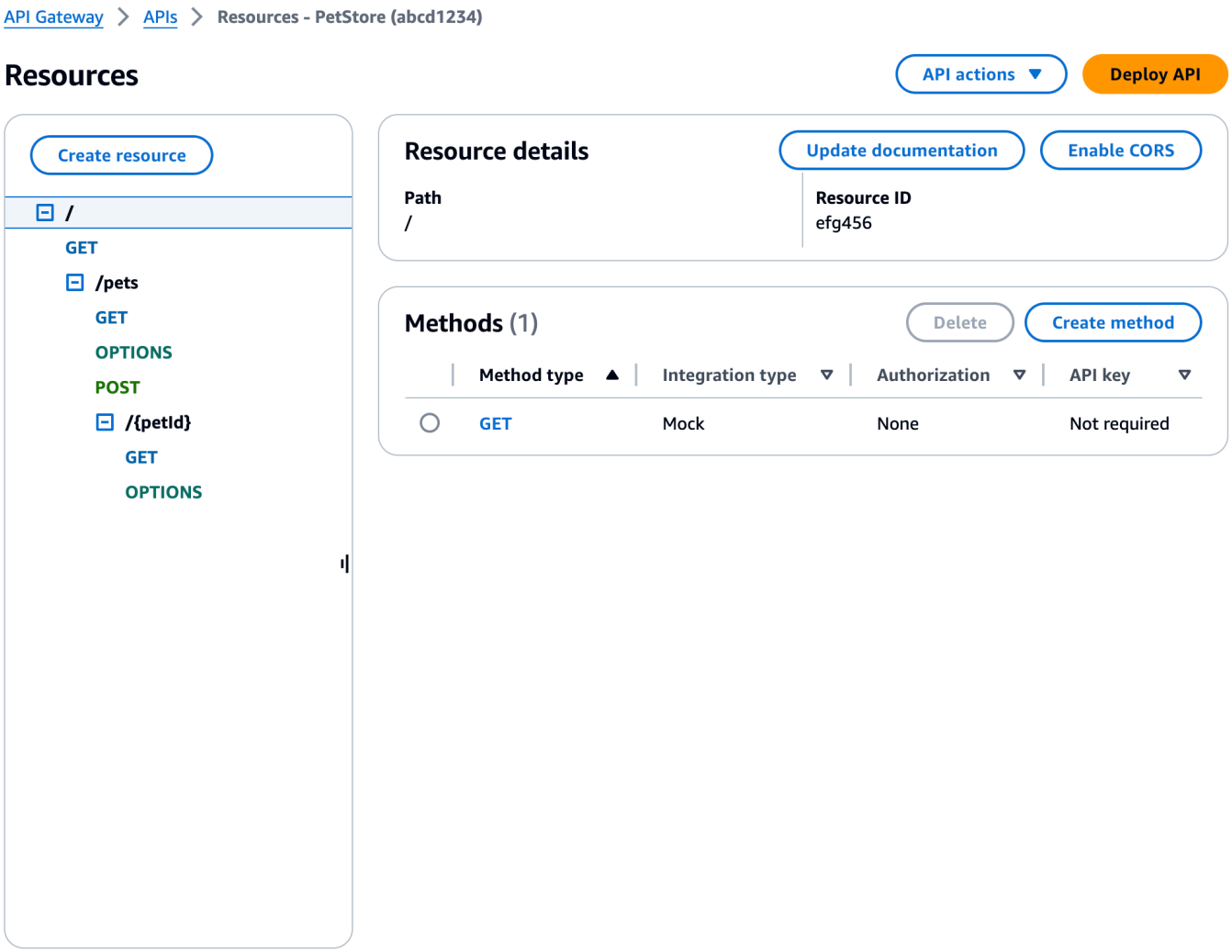
匯入、建置和測試範例 API

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 執行以下任意一項：
   - 若要建立您的第一個 API，請針對 REST API 選擇建置。
   - 如果您之前已建立 API，則選擇建立 API，然後針對 REST API 選擇建置。
3. 在建立 REST API 下，選擇範例 API，然後選擇建立 API 來建立範例 API。



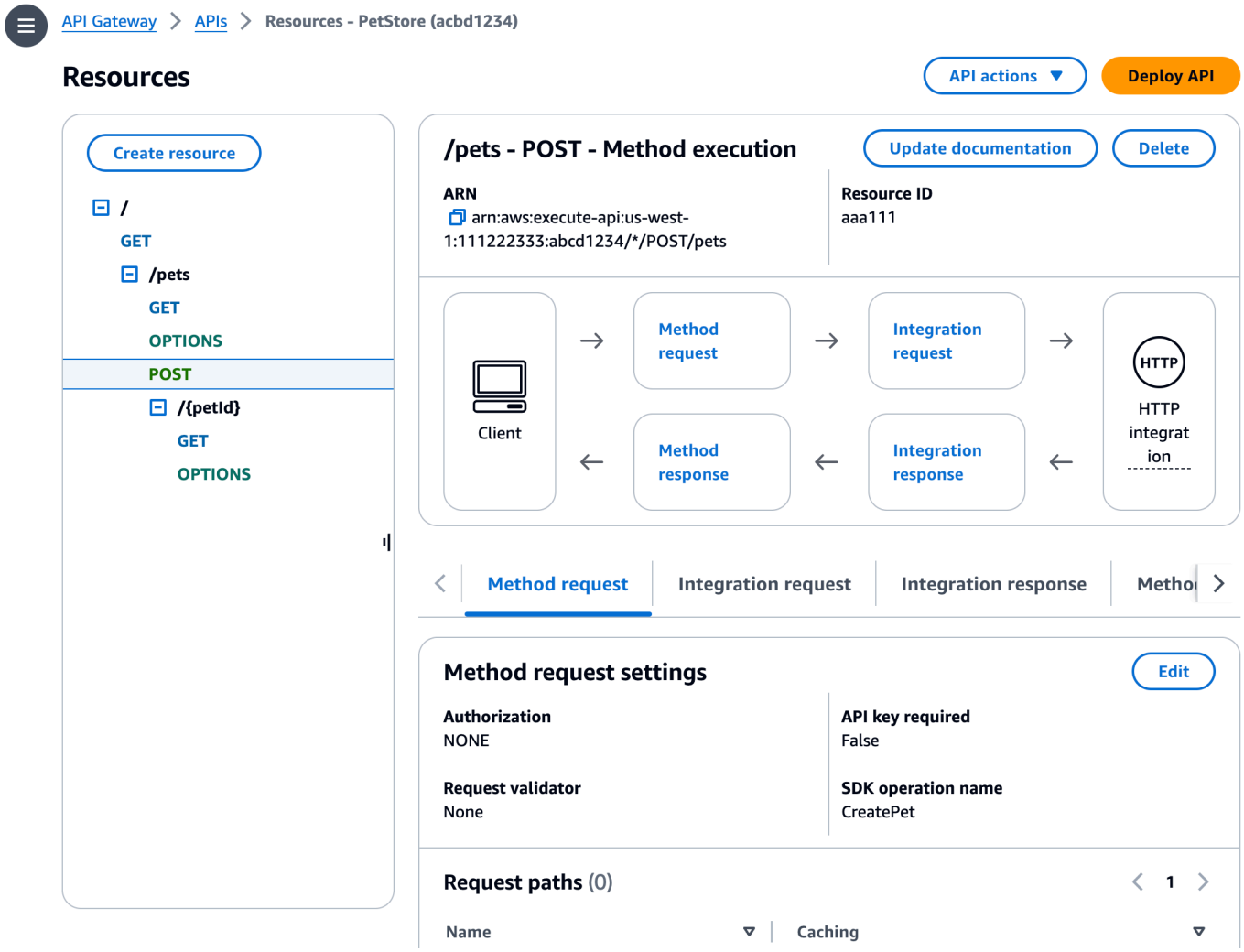

您可以先捲動 OpenAPI 定義以取得此範例 API 的詳細資訊，再選擇建立 API。

4. 在主導覽窗格中，選擇資源。新建立的 API 如下所示：




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方法表中。每一種方法都會顯示方法類型、整合類型、授權類型和 API 金鑰需求。

5. 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：




產生的方法執行窗格會顯示所選 (POST /pets) 方法結構和行為的邏輯檢視。

方法請求和方法回應代表 API 與前端的介面，而整合請求和整合回應代表 API 與後端的介面。

用戶端會使用 API 透過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轉譯為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。

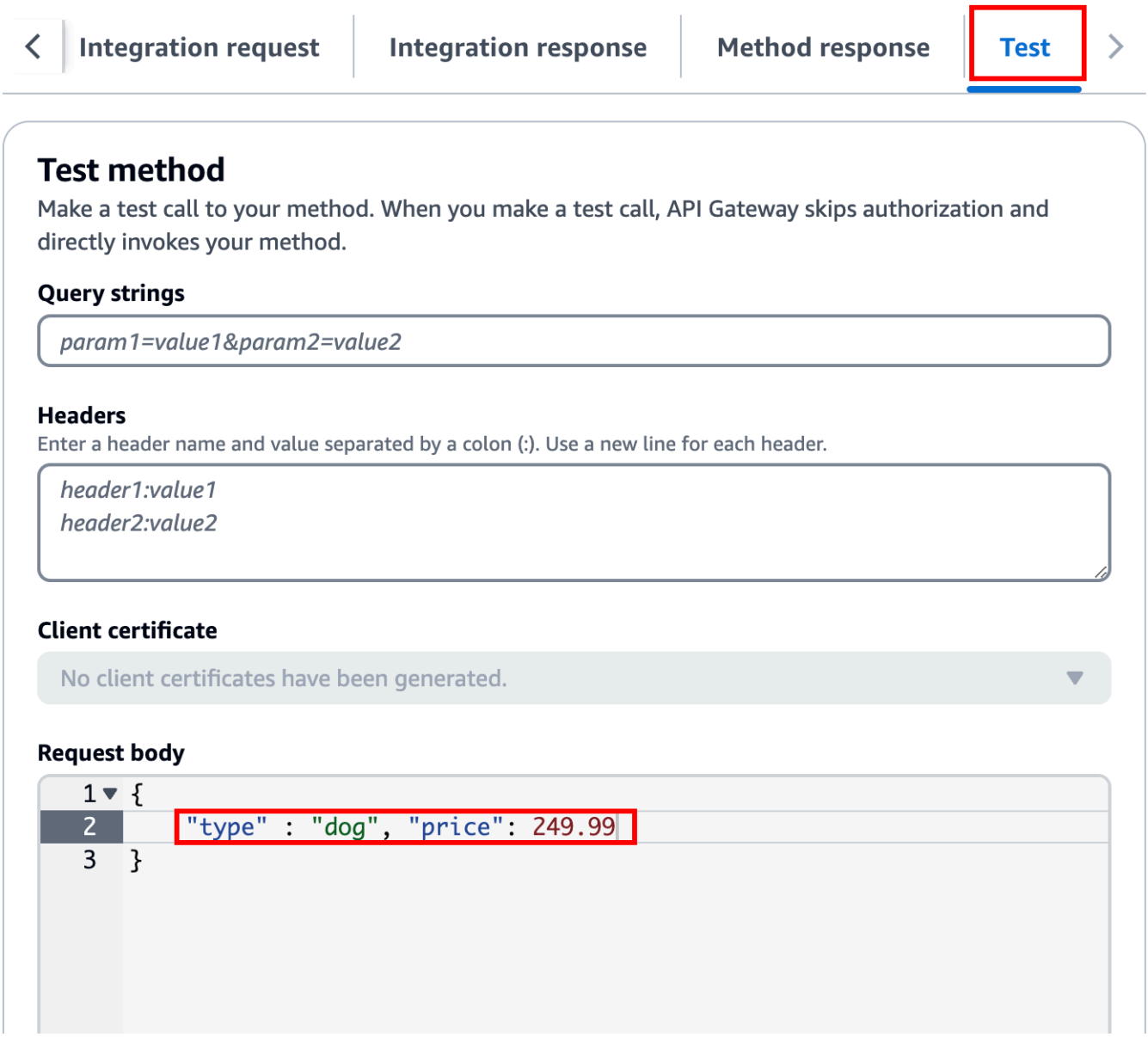
針對 API 資源上的 POST 方法，如果方法請求承載的格式與整合請求承載的格式相同，則可以將方法請求承載傳遞至整合請求，而不需要修改。

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。這些涉及方法與其映射整合之間的資料映射。現在，我們專注於測試 API，以提供端對端使用者體驗。

6. 選取測試索引標籤。您可能需要選擇向右箭頭按鈕才能顯示此索引標籤。
7. 例如，若要測試 POST /pets 方法，請將下列 **{"type": "dog","price": 249.99}** 承載輸入至請求內文，然後選擇測試。




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：

**/pets - POST method test results**

| Request | Latency ms | Status |
|---|---|---|
| /pets | 13 | 200 |

**Response body**

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

**Response headers**

```
{
  "Access-Control-Allow-Origin": "*",
  "Content-Type": "application/json",
  "X-Amzn-Trace-Id": "Root=1-67928ab3-ade6b7bdc102614983b74a5f"
}
```

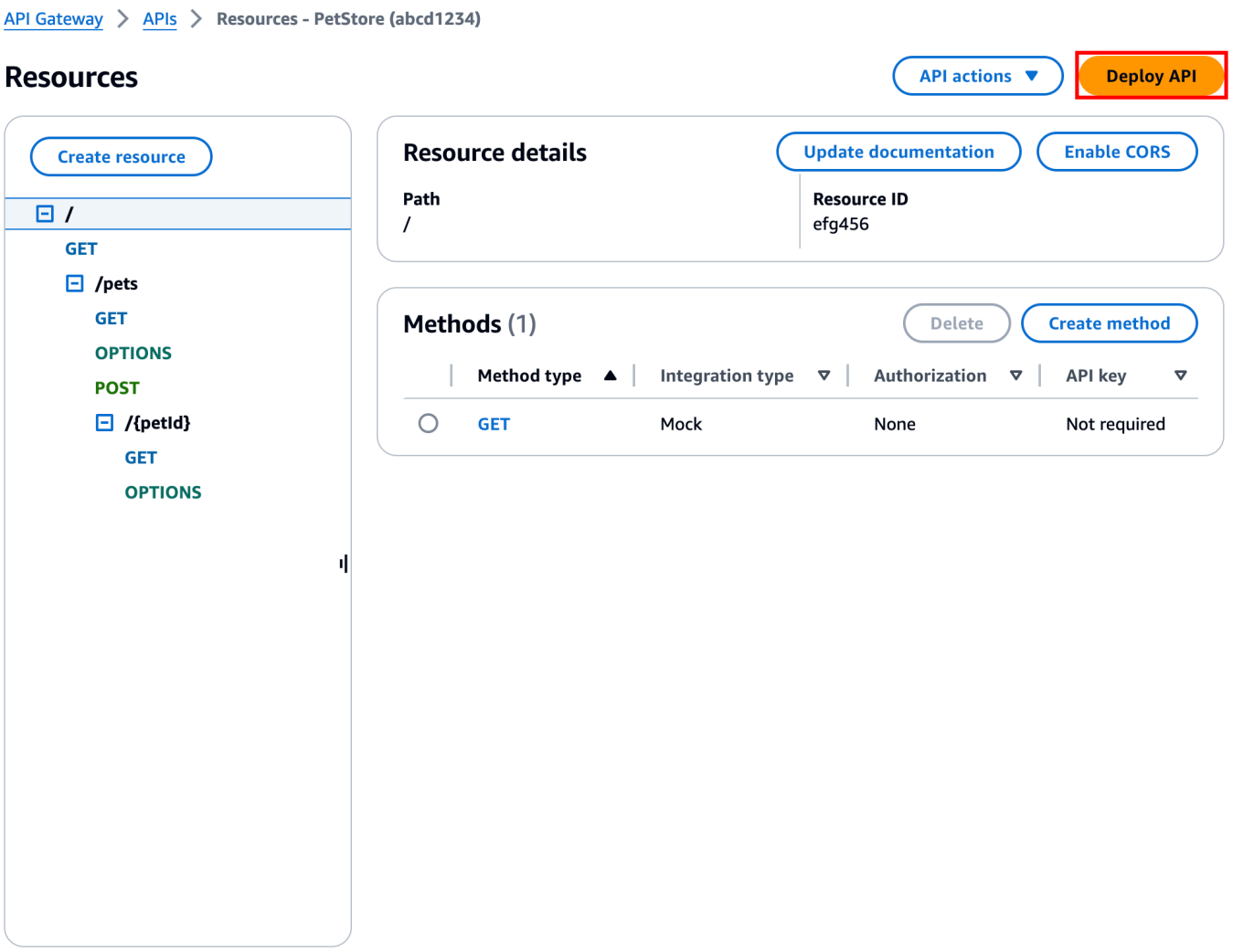
**Logs**

```
Execution log for request 06089002-b249-47e9-b485-5530e4843290
Thu Jan 23 18:30:11 UTC 2025 : Starting execution for request: 06089002-b249-47e9-b485-
```

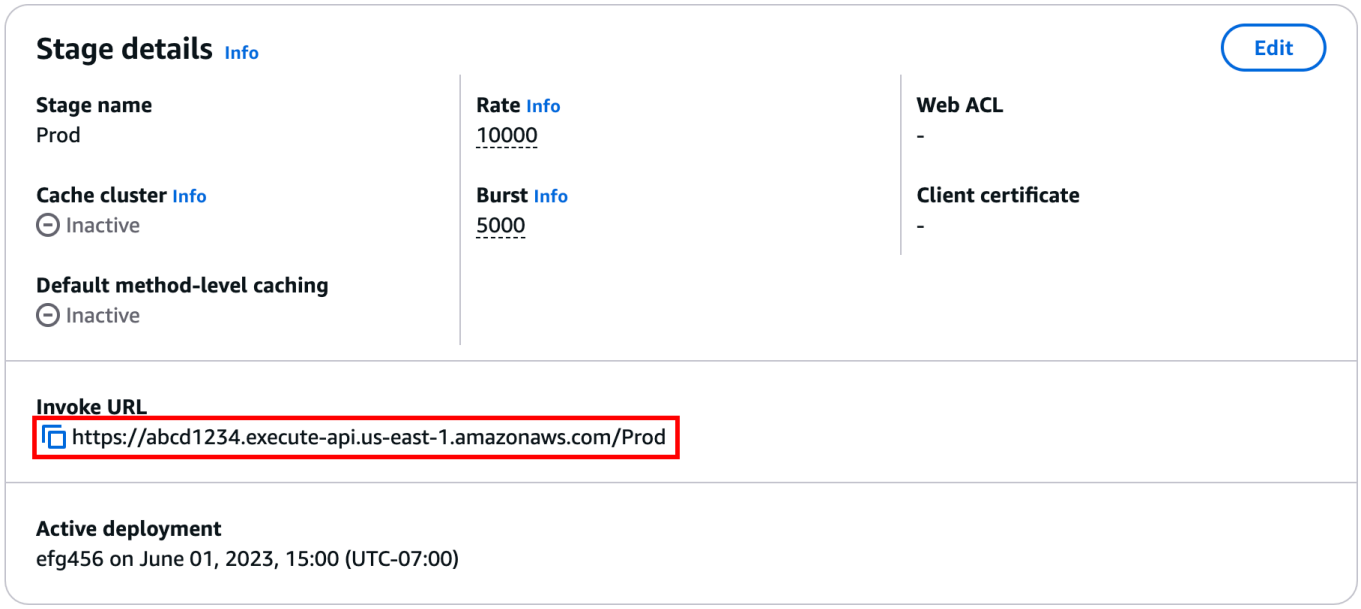
輸出的日誌項目會顯示從方法請求到整合請求，以及從整合回應到方法回應的狀態變更。這適用於故障診斷任何會導致請求失敗的映射錯誤問題。在這個範例中，不會套用任何映射：方法請求承載會透過整合請求傳遞至後端；且同樣地，後端回應會透過整合回應傳遞至方法回應。

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. 選擇部署 API以部署範例 API。




10. 針對階段，選取新增階段，然後輸入 **`test`**。
11. 在描述，請輸入描述。
12. 選擇部署。
13. 在產生的階段窗格中的階段詳細資訊下，調用 URL 會顯示調用 API 的 GET / 方法請求的 URL。




14. 選擇複製圖示以複製您的 API 的調用 URL，然後在 Web 瀏覽器中輸入您的 API 調用 URL。成功回應會傳回整合回應中從映射範本產生的結果。

15. 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 `https://api-id.execute-api.region.amazonaws.com/test/pets/{petId}` 的 Invoke URL (叫用 URL) 值。`{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，請使用第 4 版簽署程序 (SigV4) 或第 4a 版簽署程序 (SigV4a) 通訊協定來簽署請求。如需這類請求的範例，請參閱the section called “教學：建立具有 HTTP 非代理整合的 REST API”。

## 選擇 HTTP 整合教學課程

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。

在 HTTP 代理整合中，您只需根據後端需求來設定 HTTP 方法和 HTTP 端點 URI。建議您盡可能使用 HTTP 代理整合，以充分善用簡化的 API 設定。

如果您需要為後端轉換用戶端請求資料，或為用戶端轉換後端回應資料，您可能會想要使用 HTTP 自訂整合。

主題

- 教學：建立具有 HTTP 代理整合的 REST API
- 教學：建立具有 HTTP 非代理整合的 REST API

## 教學：建立具有 HTTP 代理整合的 REST 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”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。在教學課程中，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 主控台來建立具有 HTTP 代理整合的 API。
- 測試具有 HTTP 代理整合的 API

使用 API Gateway 主控台來建立具有 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「PetStore」中的 `http://petstore-demo-endpoint.execute-api.com/petstore/pets` 網站 (教學：建立具有 HTTP 非代理整合的 REST API)；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。

建立 API

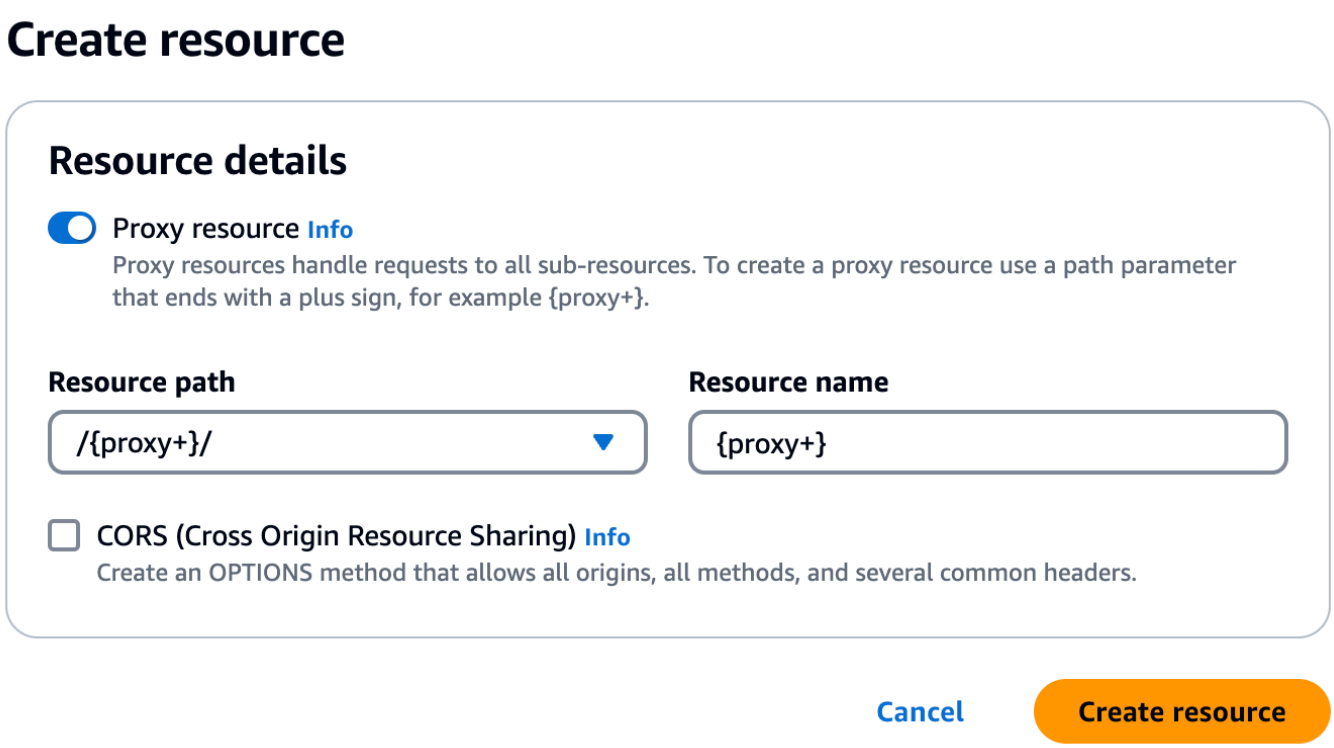
1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 如果這是您第一次使用 API Gateway，您會看到服務功能的介紹頁面。在 REST API 下方，選擇 Build (組建)。當 Create Example API (建立範例 API) 快顯出現時，選擇 OK (確定)。

   如果這不是第一次使用 API Gateway，請選擇 Create API (建立 API)。在 REST API 下方，選擇組建。
3. 對於API 名稱，輸入 **HTTPProxyAPI**。
4. 在描述，請輸入描述。
5. 將 API 端點類型保持設定為區域。
6. 針對 IP 位址類型，選擇 IPv4。
7. 選擇建立 API。

在此步驟中，您會建立 {proxy+} 的代理資源路徑。這是 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}。API Gateway 會在您建立 {proxy+} 資源時建立 ANY 方法，並在執行時間將該方法作為任何受支援 HTTP 動詞的預留位置。

建立 /{proxy+} 資源

1. 選擇您的 API。
2. 在主導覽窗格中，選擇資源。
3. 選擇建立資源。
4. 開啟代理資源。
5. 將資源路徑保持為 /。
6. 針對資源名稱，輸入 **{proxy+}**。
7. 讓 CORS (跨來源資源分享) 保持關閉。
8. 選擇建立資源。




在此步驟中，您會使用代理整合將 ANY 方法與後端 HTTP 端點整合。在代理整合中，API Gateway 會將用戶端提交的方法請求傳遞至後端，而不需要 API Gateway 介入。

建立 **ANY** 方法

1. 選擇 /{proxy+} 資源。
2. 選擇 ANY 方法。
3. 在警告符號下，選擇編輯整合。如果 API 的方法沒有整合，便無法部署該 API。
4. 針對整合類型，選取 HTTP。
5. 開啟 HTTP 代理整合。
6. 針對 HTTP 方法，選取 ANY。
7. 針對端點 URL，輸入 **`http://petstore-demo-endpoint.execute-api.com/{proxy}`**。
8. 選擇儲存。

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：

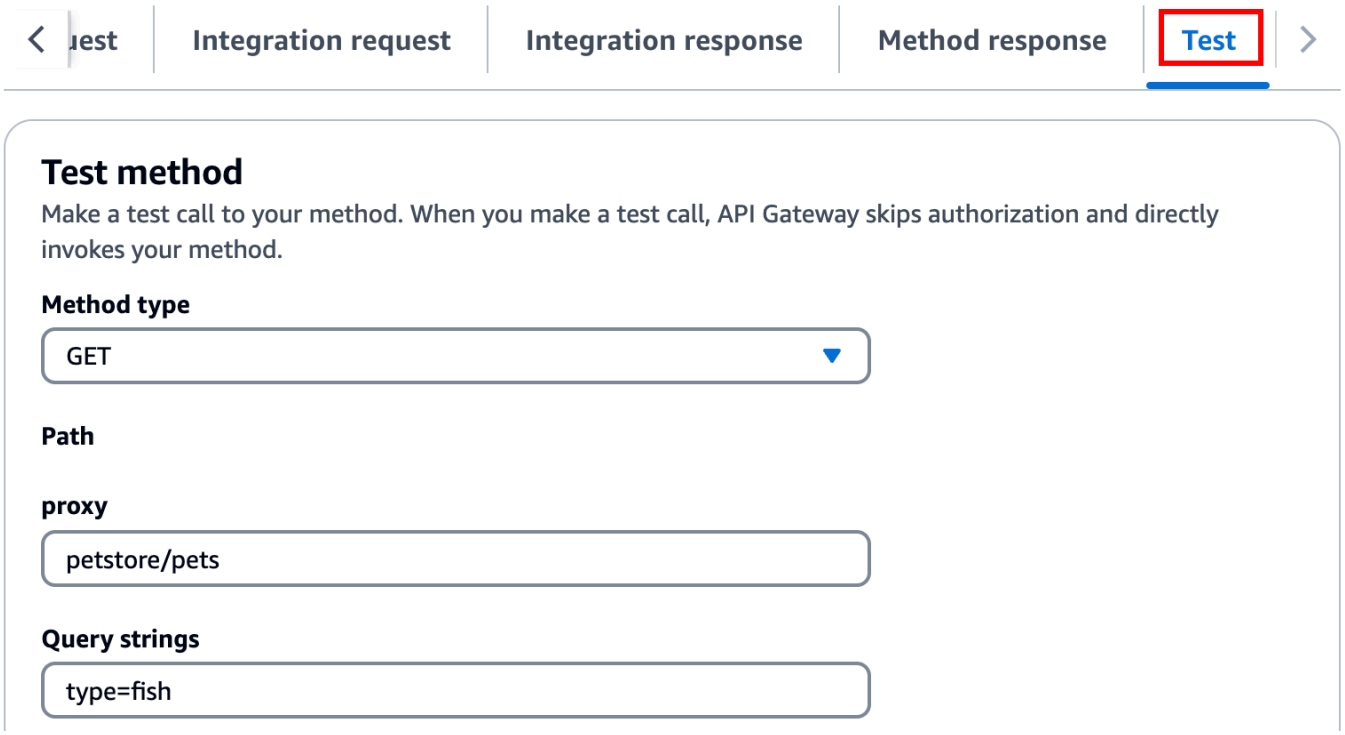
- 如果後端已讓對應的後端端點可供使用，因此已授予所需的存取許可。
- 如果用戶端提供正確的輸入。

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. 選取測試索引標籤。您可能需要選擇向右箭頭按鈕才能顯示此索引標籤。
2. 針對方法類型，選取 GET。
3. 針對路徑，在代理下輸入 **petstore/pets**。
4. 針對查詢字串，輸入 **type=fish**。
5. 選擇測試。



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
}
```

您也可以使用瀏覽器來測試您的 API。部署您的 API 並將其關聯到某個階段，以建立 API 的調用 URL。

部署 API

1. 選擇**部署 API**。
2. 針對**階段**，選取**新階段**。
3. 針對**階段名稱**，輸入 **test**。

4. 在描述，請輸入描述。
5. 選擇部署。

現在，用戶端可以呼叫您的 API。

若要調用您的 API

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 選擇您的 API。
3. 在主導覽窗格中，選擇階段。
4. 在階段詳細資訊下，選擇複製圖示以複製 API 的調用 URL。

   在 Web 瀏覽器中輸入 API 的調用 URL。

   完整 URL 看起來應該會像這樣：`https://abcdef123.execute-api.us-east-2.amazonaws.com/test/petstore/pets?type=fish`。

   您的瀏覽器將向 API 傳送 `GET` 請求。
5. 結果應該與在 API Gateway 主控台中使用測試時所傳回的結果相同。

## 教學：建立具有 HTTP 非代理整合的 REST API

在本教學課程中，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
- (選用) 映射請求參數

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。

在此步驟中，您將建立空白 API。在以下步驟中，您會建立資源和方法，以使用非代理 HTTP 整合將 API 連線到 `http://petstore-demo-endpoint.execute-api.com/petstore/pets` 端點。

建立 API

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 如果這是您第一次使用 API Gateway，您會看到服務功能的介紹頁面。在 REST API 下方，選擇 Build (組建)。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)。在 REST API 下方，選擇組建。
3. 對於API 名稱，輸入 **HTTPNonProxyAPI**。
4. 在描述，請輸入描述。
5. 將 API 端點類型保持設定為區域。
6. 針對 IP 位址類型，選擇 IPv4。
7. 選擇建立 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。

建立 /pets 資源

1. 選擇建立資源。
2. 讓代理資源保持關閉。
3. 將資源路徑保持為 `/`。
4. 針對資源名稱，輸入 **`pets`**。
5. 讓 CORS (跨來源資源分享) 保持關閉。
6. 選擇建立資源。

在此步驟中，您會在 /pets 資源上建立 GET 方法。GET 方法會與 `http://petstore-demo-endpoint.execute-api.com/petstore/pets` 網站整合。API 方法的其他選項包括以下項目：

- 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。

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` 來叫用函數，而方法請求則可根據 Lambda 函數的邏輯來使用任何 HTTP 動詞。

在 /pets 資源上建立 **`GET`** 方法

1. 選取 /pets 資源。
2. 選擇建立方法。
3. 針對方法類型，選取 GET。
4. 針對整合類型，選取 HTTP 整合。
5. 讓 HTTP 代理整合保持關閉。
6. 針對 HTTP 方法，選取 GET。
7. 針對端點 URL，輸入 **`http://petstore-demo-endpoint.execute-api.com/petstore/pets`**。

   PetStore 網站可讓您依指定頁面上的寵物類型 (例如 "Dog" 或 "Cat") 擷取 `Pet` 項目清單。
8. 針對內容處理，選取傳遞。
9. 選擇 URL 查詢字串參數。

   PetStore 網站使用 `type` 與 `page` 查詢字串參數來接受輸入內容。您會將查詢字串參數新增至方法請求，然後將其映射至整合請求的映射查詢字串。
10. 若要將查詢字串參數，請執行下列操作：

    a. 選擇新增查詢字串。

    b. 對於名稱，輸入 **`type`**。

    c. 將必要和快取保持關閉。

    重複上述步驟，另外建立名稱為 **`page`** 的查詢字串。
11. 選擇建立方法。

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。

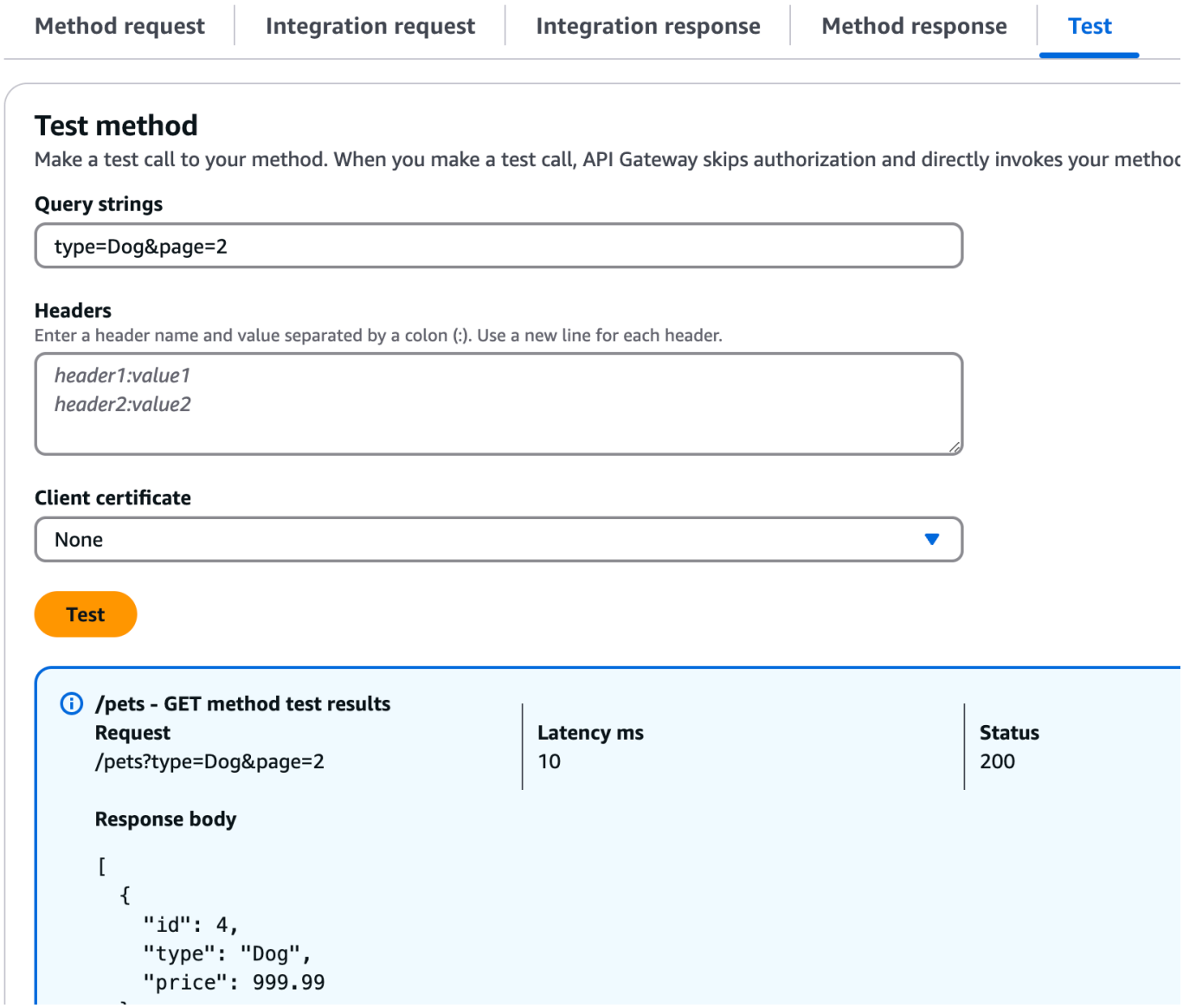
將輸入參數映射至整合請求

1. 在整合請求索引標籤上，於整合請求設定下，選擇編輯。
2. 選擇 URL 查詢字串參數，然後執行下列動作：
   a. 選擇新增查詢字串參數。
   b. 對於名稱，輸入 **type**。
   c. 對於映射自，輸入 **method.request.querystring.type**。
   d. 讓快取保持關閉。
   e. 選擇新增查詢字串參數。
   f. 對於名稱，輸入 **page**。
   g. 對於映射自，輸入 **method.request.querystring.page**。
   h. 讓快取保持關閉。
3. 選擇儲存。

若要測試 API

1. 選擇測試標籤。您可能需要選擇向右箭頭按鈕才能顯示此索引標籤。
2. 針對查詢字串，輸入 **type=Dog&page=2**。
3. 選擇測試。

   結果類似如下：




現在測試成功，我們可以部署 API 來公開提供使用。

4. 選擇部署 API。
5. 針對階段，選取新階段。
6. 針對階段名稱，輸入 **Prod**。
7. 在描述，請輸入描述。
8. 選擇部署。
9. (選用) 針對階段詳細資訊下的調用 URL，您可以選擇複製圖示以複製 API 的調用 URL。您可以使用此項與 Postman 和 cURL 這類工具搭配，來測試您的 API。

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，請參閱您選擇的 AWS 軟體開發套件。

> **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。

(選用) 映射請求參數

對應 API Gateway API 的請求參數

本教學課程顯示如何在 API 的方法請求上建立 `{petId}` 的路徑參數，以指定項目 ID、將它映射至整合請求 URL 中的 `{id}` 路徑參數，然後將請求傳送到 HTTP 端點。

> **Note**
>
> 如果您輸入大小寫錯誤的字母 (例如輸入小寫字母而非大寫字母)，這可能會在稍後的演練中造成錯誤。

步驟 1：建立資源

在此步驟中，您會建立路徑參數為 {petId} 的資源。

建立 {petId} 資源

1. 選取 /pets 資源，然後選擇建立資源。
2. 讓代理資源保持關閉。
3. 針對資源路徑，選取 /pets。
4. 針對資源名稱，輸入 **`{petId}`**。

   使用大括號 ({ }) 括住 `petId`，以便顯示 /pets/{petId}。
5. 讓 CORS (跨來源資源分享) 保持關閉。
6. 選擇建立資源。

步驟 2：建立及測試方法

在此步驟中，您會建立路徑參數為 `{petId}` 的 `GET` 方法。

設定 GET 方法

1. 選取 /{petId} 資源，然後選擇建立方法。
2. 針對方法類型，選取 GET。
3. 針對整合類型，選取 HTTP 整合。
4. 讓 HTTP 代理整合保持關閉。
5. 針對 HTTP 方法，選取 GET。
6. 針對端點 URL，輸入 **http://petstore-demo-endpoint.execute-api.com/petstore/pets/{id}**。
7. 針對內容處理，選取傳遞。
8. 讓預設逾時保持開啟。
9. 選擇建立方法。

此時您將剛建立的 {petId} 路徑參數映射至整合請求之 HTTP 端點 URL 中的 {id} 路徑參數。HTTP 端點 URL 為 **http://petstore-demo-endpoint.execute-api.com/petstore/pets/{id}**。

映射 **{petId}** 路徑參數

1. 在整合請求索引標籤上，於整合請求設定下，選擇編輯。
2. 選擇 URL 路徑參數。
3. API Gateway 會為名為 petId 的整合請求建立路徑參數，但此路徑參數對於您設定為後端整合的 HTTP 端點 URL 無效。HTTP 端點會使用 {id} 作為路徑參數。針對名稱，刪除 petId 並輸入 **id**。

   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。
4. 選擇儲存。

現在，請測試該方法。

測試該方法

1. 選擇測試標籤。您可能需要選擇向右箭頭按鈕才能顯示此索引標籤。
2. 在 petId 的路徑下，輸入 **4**。
3. 選擇測試。

若成功，回應內文會顯示如下：

```
{
  "id": 4,
  "type": "bird",
  "price": 999.99
}
```

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. 選擇部署 API。
2. 對於階段，選取生產。
3. 在描述，請輸入描述。
4. 選擇部署。

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. 在主導覽窗格中，選擇階段。
2. 在階段詳細資訊下，選擇複製圖示以複製 API 的調用 URL。

   此 URL 看起來如下：

   ```
   https://my-api-id.execute-api.region-id.amazonaws.com/prod
   ```
3. 在新瀏覽器標籤的地址方塊中輸入此 URL，並為此 URL 附加 `/pets/4`，然後才提交請求。
4. 瀏覽器會傳回以下內容：

   ```
   {
     "id": 4,
     "type": "bird",
     "price": 999.99
   }
   ```

### 後續步驟

您可以開啟請求驗證、轉換資料或建立自訂閘道回應，以進一步自訂 API。

若要探索更多自訂 API 的方法，請參閱下列教學課程：

- 如需請求驗證的詳細資訊，請參閱「在 API Gateway 中設定基本請求驗證」。
- 如需如何轉換請求和回應承載的相關資訊，請參閱教學課程：修改 AWS 服務整合的整合請求和回應。
- 如需如何建立自訂閘道回應的相關資訊，請參閱使用 API Gateway 主控台設定 REST API 的閘道回應。

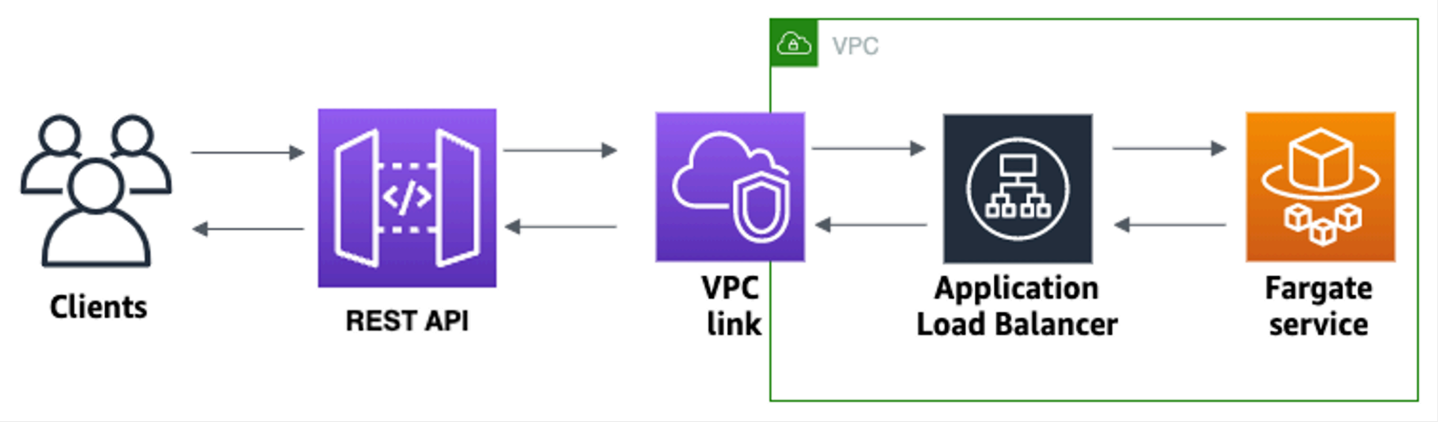
## 教學：建立具私有整合的 REST API

在本教學課程中，您會建立 REST API，連線至在 Amazon VPC 中執行的 Amazon ECS 服務。不在您 Amazon VPC 上的客戶可以使用該 API 來存取您的 Amazon ECS 服務。

此教學課程需約 1 小時方能完成。首先，您使用 CloudFormation 範本來建立 Amazon VPC 和 Amazon ECS 服務。然後，您可以使用 API Gateway 主控台來建立 VPC 連結 V2。VPC 連結允許 API Gateway 存取在您 Amazon VPC 中執行的 Amazon ECS 服務。接著，您建立使用 VPC 連結 V2 連線至 Amazon ECS 服務的 REST API。最後，測試您的 API。

當您叫用 REST API 時，API Gateway 會透過 VPC 連結 V2 將請求路由到您的 Amazon ECS 服務，然後從服務傳回回應。

> **Note**
>
> 此教學課程先前支援 HTTP APIs，現在支援使用 VPC 連結 V2 APIs。




若要完成本教學課程，您需要 AWS 帳戶和具有主控台存取權 AWS Identity and Access Management 的使用者。如需詳細資訊，請參閱先決條件。

主題

- 步驟 1：建立 Amazon ECS 服務
- 步驟 2：建立 VPC 連結
- 步驟 3：建立 REST API
- 步驟 4：測試您的 API
- 步驟 5：部署 API
- 步驟 6：呼叫您的 API
- 步驟 7：清除

## 步驟 1：建立 Amazon ECS 服務

Amazon ECS 是一種容器管理服務，可以在叢集上輕鬆執行、停止及管理 Docker 容器。在本教學課程中，您可以在 Amazon ECS 管理的無伺服器基礎設施上執行叢集。

下載並解壓縮此 CloudFormation 範本，這會建立服務的所有相依性，包括 Amazon VPC。您可以使用範本來建立使用 Application Load Balancer 的 Amazon ECS 服務。

建立 CloudFormation 堆疊

1. 在 https：//https://console.aws.amazon.com/cloudformation 開啟 CloudFormation 主控台。
2. 選擇 Create stack (建立堆疊)，然後選擇 With new resources (standard) (使用新資源 (標準))。
3. 對於 Specify template (指定範本)，選擇 Upload a template file (上傳範本檔案)。
4. 選取您下載的範本。

5. 選擇 Next (下一步)。
6. 針對 Stack name (堆疊名稱)，輸入 **`rest-api-private-integrations-tutorial`**，然後選擇 Next (下一步)。
7. 針對 Configure stack options (設定堆疊選項)，選擇 Next (下一步)。
8. 針對 功能，確認 CloudFormation 可以在您的帳戶中建立 IAM 資源。
9. 選擇下一步，然後選擇提交。

CloudFormation 會佈建 ECS 服務，這可能需要幾分鐘的時間。當您的 CloudFormation 堆疊狀態為 CREATE_COMPLETE 時，您就可以繼續進行下一個步驟。

## 步驟 2：建立 VPC 連結

VPC 連結允許 API Gateway 存取 Amazon VPC 中的私有資源。您可以使用 VPC 連結，允許用戶端透過 REST API 存取您的 Amazon ECS 服務。

建立 VPC 連結

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 在主要導覽窗格中，選擇 VPC 連結，然後選擇建立。

   您可能需要選擇選單圖示，才能開啟主要導覽窗格。
3. 針對選擇 VPC 連結版本，選取 VPC 連結 V2。
4. 針對 Name (名稱)，輸入 **`private-integrations-tutorial`**。
5. 在 VPC 欄位選擇您在步驟 1 建立的 VPC。名稱應以 RestApiStack 開頭。
6. 對於 Subnets (子網路)，請在您的 VPC 中選取兩個私有子網路。它們的名稱要以 `PrivateSubnet` 結尾。
7. 對於安全群組，選取以 `private-integrations-tutorial` 開頭且描述為 `RestApiStack/RestApiTutorialService/Service/SecurityGroup` 的群組 ID。
8. 選擇建立。

建立 VPC 連結 V2 之後，API Gateway 會佈建彈性網路界面來存取 VPC。該程序需要幾分鐘的時間。同時，您可以創建您的 API。

## 步驟 3：建立 REST API

REST API 為您的 Amazon ECS 服務提供 HTTP 端點。

若要建立 REST API

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 選擇 Create API (建立 API)，然後針對 REST API，選擇 Build (建置)。
3. 對於名稱，輸入 **`private-integration-api`**。
4. 針對 IP 位址類型，選擇 IPv4。
5. 選擇建立 API。

   建立 API 之後，您可以建立 方法。
6. 選擇建立方法，然後執行下列動作：

   a. 針對方法類型，選取 GET。
   b. 針對整合類型，選取 VPC 連結。
   c. 開啟 VPC 代理整合。
   d. 針對 HTTP 方法，選取 GET。
   e. 針對 VPC 連結，選擇您在上一個步驟中建立的 VPC 連結 V2。
   f. 針對整合目標，輸入您在步驟 1 中使用範本建立的 CloudFormation 負載平衡器。其名稱應以 rest- 開頭。
   g. 針對端點 URL，輸入 `http://private-integrations-tutorial.com`。

      URL 用於設定整合請求的`Host`標頭。在此情況下，主機標頭為 **`private-integrations-tutorial`**。
   h. 選擇建立方法。

      透過代理整合，API 已準備好進行測試。

## 步驟 4：測試您的 API

接下來，您將測試叫用 API 方法。

測試您的 API

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 選擇您的 API。
3. 選擇測試標籤。您可能需要選擇向右箭頭按鈕才能顯示此索引標籤。
4. 選擇測試

確認 API 的回應是一則歡迎訊息，告知您應用程式正在 Amazon ECS 上執行。

## 步驟 5：部署 API

接著，部署您的 API。

部署 API

1. 選擇部署 API。
2. 針對階段，選取新階段。
3. 針對階段名稱，輸入 **Prod**。
4. 在描述，請輸入描述。
5. 選擇部署。

## 步驟 6：呼叫您的 API

部署 API 之後，您可以呼叫它。

呼叫您的 API

1. 在 Web 瀏覽器中輸入調用 URL。

   完整 URL 看起來應該會像這樣：`https://abcd123.execute-api.us-east-2.amazonaws.com/Prod`。

   您的瀏覽器將向 API 傳送 `GET` 請求。
2. 確認 API 的回應是一則歡迎訊息，告知您應用程式正在 Amazon ECS 上執行。

   如果您看到歡迎訊息，表示您已成功建立在 Amazon VPC 中執行的 Amazon ECS 服務，並使用 API Gateway REST API 搭配 VPC 連結 V2 來存取 Amazon ECS 服務。

## 步驟 7：清除

若要避免不必要的成本，請刪除您在此教學課程中建立的資源。下列步驟會刪除您的 VPC 連結 V2、CloudFormation stack 和 REST API。

刪除 REST API

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 在 API 頁面上，選取 API。依次選擇 Actions (動作)、Delete (刪除)，然後確認您的選擇。

刪除 VPC 連結

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 選擇 VPC link (VPC 連結)。
3. 選取您的 VPC 連結、選擇 Delete (刪除)，然後確認您的選擇。

刪除 CloudFormation 堆疊

1. 在 https：//https://console.aws.amazon.com/cloudformation 開啟 CloudFormation 主控台。
2. 選取您的 CloudFormation 堆疊。
3. 選擇刪除，然後確認您的選擇。

# 教學課程：使用 AWS 整合建立 REST API

如需了解如何建立 API Gateway API 以公開整合的 Lambda 函數，請參閱教學：建立具 Lambda 代理整合的 REST API 和教學：建立具 Lambda 非代理整合的 REST API 主題。此外，您可以建立 API Gateway API 來公開其他服務 AWS ，例如 Amazon SNS、Amazon S3、Amazon Kinesis 甚至 AWS Lambda。這可透過 AWS 整合進行。Lambda 整合或 Lambda 代理整合則屬於特殊案例，這些項目會透過 API Gateway API 來公開叫用 Lambda 函數。

所有 AWS 服務都支援專用 APIs來公開其功能。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 可為您的用戶端提供一致的應用程式通訊協定，以存取不同的 AWS 服務。

在本演練中，我們會建立 API 來公開 Amazon SNS。如需將 API 與其他 AWS 服務整合的更多範例，請參閱 Amazon API Gateway 教學課程和研討會。

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 方法與單一 AWS 動作整合。如需更多的彈性，您可以設定 Lambda 代理整合，這種方法與代理整合類似。Lambda 函數接著會剖析和處理其他 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。

此演練採用 教學：建立具 Lambda 非代理整合的 REST API 中的說明與概念為基礎。若您尚未完成該演練，建議先行完成。

主題


- 先決條件
- 步驟 1：建立 AWS 服務代理執行角色
- 步驟 2：建立資源
- 步驟 3：建立 GET 方法
- 步驟 4：指定方法設定並測試方法
- 步驟 5：部署 API
- 步驟 6：測試 API
- 步驟 7：清除


## 先決條件

開始此演練前，請執行下列操作：

1. 完成「設定為使用 API Gateway」中的步驟。
2. 建立名為 `MyDemoAPI` 的新 API。如需詳細資訊，請參閱 教學：建立具有 HTTP 非代理整合的 REST API。
3. 至少將 API 部署至名為 `test` 的階段一次。如需詳細資訊，請參閱「」中的「部署 API選擇 AWS Lambda 整合教學課程」。
4. 完成 中的其餘步驟選擇 AWS Lambda 整合教學課程
5. 在 Amazon Simple Notification Service (Amazon SNS) 中建立至少一個主題。您將使用部署的 API 來取得與您 AWS 帳戶相關聯的 Amazon SNS 主題清單。如需了解如何在 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 AWS 服務代理執行角色

若要允許 API 調用 Amazon SNS 動作，您必須將適當的 IAM 政策附加至 IAM 角色。在此步驟中，您會建立新的 IAM 角色。

建立 AWS 服務代理執行角色

1. 登入 AWS 管理主控台 ，並在 https：//https://console.aws.amazon.com/iam/ 開啟 IAM 主控台。

2. 選擇角色。
3. 選擇建立角色。
4. 在選取信任的實體類型下選擇 AWS 服務，然後選取 API Gateway 並選取允許 API Gateway 將日誌推送到 CloudWatch Logs。
5. 選擇下一步，然後選擇下一步。
6. 針對角色名稱，輸入 **APIGatewaySNSProxyPolicy**，然後選擇建立角色。
7. 在角色清單中，選擇您剛剛建立的角色。您可能需要捲動或使用搜尋列來尋找該角色。
8. 針對選取的角色，選取 許可 索引標籤。
9. 在下拉式清單中，選擇 連接政策。
10. 在搜尋列中輸入 **AmazonSNSReadOnlyAccess**，並選擇新增許可。

> **Note**
>
> 本教學課程為了簡單起見，使用受管理政策。最佳實務是，您應建立自己的 IAM 政策以授予所需的最低許可。

11. 記下新建的角色 ARN，以在稍後使用。

## 步驟 2：建立資源

在此步驟中，您會建立資源，讓 AWS 服務代理能夠與服務互動 AWS 。

建立資源

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 選擇您的 API。
3. 選取根資源 / (其以一條正斜線 (/) 表示)，然後選擇建立資源。
4. 讓代理資源保持關閉。
5. 將資源路徑保持為 /。
6. 針對資源名稱，輸入 **mydemoawsproxy**。
7. 讓 CORS (跨來源資源分享) 保持關閉。
8. 選擇建立資源。

## 步驟 3：建立 GET 方法

在此步驟中，您會建立 GET 方法，讓 AWS 服務代理能夠與服務互動 AWS 。

**建立 GET 方法**

1. 選取 /mydemoawsproxy 資源，然後選擇建立方法。
2. 針對方法類型，選取 GET。
3. 針對整合類型，選取 AWS 服務。
4. 針對 AWS 區域，選取您建立 Amazon SNS 主題 AWS 區域 的 。
5. 針對 AWS 服務，選取 Amazon SNS。
6. 讓 AWS 子網域保持空白。
7. 針對 HTTP 方法，選取 GET。
8. 針對動作類型，選取使用動作名稱。
9. 針對動作名稱，輸入 **ListTopics**。
10. 針對執行角色，輸入 **APIGatewaySNSProxyPolicy** 的角色 ARN。
11. 選擇建立方法。

## 步驟 4：指定方法設定並測試方法

您現在可以測試您的 GET 方法，確認其已正確設定，可以列出您的 Amazon SNS 主題。

**測試 GET 方法**

1. 選擇測試標籤。您可能需要選擇向右箭頭按鈕才能顯示此索引標籤。
2. 選擇測試。

   結果會顯示類似以下內容的回應：

```
{
  "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，您將部署 API 以從 API Gateway 主控台外部呼叫 API。

部署 API

1. 選擇部署 API。
2. 針對階段，選取新階段。
3. 針對階段名稱，輸入 **test**。
4. 在描述，請輸入描述。
5. 選擇部署。

## 步驟 6：測試 API

在此步驟中，您會離開 API Gateway 主控台，並使用 AWS 服務代理與 Amazon SNS 服務互動。

1. 在主導覽窗格中，選擇階段。
2. 在階段詳細資訊下，選擇複製圖示以複製 API 的調用 URL。

   它應該如下所示：

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

3. 將 URL 輸入到新瀏覽器標籤的網址方塊中。

4. 附加 /mydemoawsproxy，讓 URL 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 服務代理需要運作的 IAM 資源。

> **Warning**
>
> 如果您刪除 AWS 服務代理依賴的 IAM 資源，則該 AWS 服務代理及其依賴的任何 APIs 將無法再運作。IAM 資源一經刪除，即無法復原。如果您要再次使用該 IAM 資源，就必須予以重建。

刪除相關聯的 IAM 資源

1. 在以下網址開啟 IAM 主控台：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) 和 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. 選取 APIGatewayAWSProxyExecPolicy，然後依序選擇 Policy Actions (政策動作) 和 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：建立 REST API 做為 Amazon S3 代理、 教學課程：建立具有兩個 AWS 服務整合和一個 Lambda 非代理整合的計算器 REST API或 教學課程：建立 REST API 做為 Amazon Kinesis 代理。

# 教學課程：建立具有兩個 AWS 服務整合和一個 Lambda 非代理整合的計算器 REST API

the section called “教學：建立具 Lambda 非代理整合的 REST API” 僅使用 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，例如自動新增調用 Lambda 函數所需的資源型許可權。在這裡，三種整合中的兩種會使用 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。

在本教學課程中，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 GET 資源上公開 /calc 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 POST 資源上公開 /calc 方法來叫用 Lambda 函數，並在方法請求酬載中提供輸入。(AWS Service 整合)
3. 在巢狀 GET 資源上公開 /calc/{operand1}/{operand2}/{operator} 來叫用 Lambda 函數，並提供輸入做為路徑參數 (Lambda Function 整合)

除了試做本教學課程之外，您可能還想要研究 Calc API 的 OpenAPI 定義檔，您可以依照 the section called “OpenAPI” 中的指示，將此定義檔匯入至 API Gateway。

主題

- 建立可擔任的 IAM 角色
- 建立 Calc Lambda 函數
- 測試 Calc Lambda 函式
- 建立 Calc API
- 整合 1：建立 GET 方法與查詢參數搭配來呼叫 Lambda 函數
- 整合 2：建立 POST 方法與 JSON 承載搭配來呼叫 Lambda 函數
- 整合 3：建立 GET 方法與路徑參數搭配來呼叫 Lambda 函數
- 與 Lambda 函數整合之範例 API 的 OpenAPI 定義

## 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 可擔任的 IAM 角色，這是具有以下信任關係的 IAM 角色：

JSON

```
{
  "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：

JSON

```
{
    "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 可擔任的 IAM 角色

1. 登入 IAM 主控台。
2. 選擇角色。
3. 選擇 Create Role (建立角色)。
4. 在選取可信任執行個體類型下，選取 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 (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。

      JSON

```
{
    "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：

    JSON

```
{
  "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，您會使用 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. 針對名稱，輸入 **Calc**。
4. 針對執行期，請選擇最新支援的 Node.js 或 Python 執行期。
5. 其他所有選項請使用預設設定。
6. 選擇建立函數。
7. 在 Lambda 主控台中，複製下列 Lambda 函數，並將其貼入程式碼編輯器。

   Node.js

```
export const handler = async function (event, context) {
  console.log("Received event:", JSON.stringify(event));

  if (
    event.a === undefined ||
    event.b === undefined ||
    event.op === undefined
  ) {
    return "400 Invalid Input";
  }

  const res = {};
  res.a = Number(event.a);
  res.b = Number(event.b);
  res.op = event.op;
  if (isNaN(event.a) || isNaN(event.b)) {
    return "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) {
        return "400 Divide by Zero";
      } else {
        res.c = res.a / res.b;
      }
      break;
    default:
      return "400 Invalid Operator";
  }

  return res;
};
```

Python

```
import json


def lambda_handler(event, context):
    print(event)

    try:
        (event['a']) and (event['b']) and (event['op'])
    except KeyError:
        return '400 Invalid Input'

    try:
        res = {
            "a": float(
                event['a']), "b": float(
                event['b']), "op": event['op']}
    except ValueError:
        return '400 Invalid Operand'
```

```
    if event['op'] == '+':
        res['c'] = res['a'] + res['b']
    elif event['op'] == '-':
        res['c'] = res['a'] - res['b']
    elif event['op'] == '*':
        res['c'] = res['a'] * res['b']
    elif event['op'] == '/':
        if res['b'] == 0:
            return '400 Divide by Zero'
        else:
            res['c'] = res['a'] / res['b']
    else:
        return '400 Invalid Operator'

    return res
```

8. 在執行角色下，選擇選擇現有的角色。
9. 為您先前建立的 **lambda_invoke_function_assume_apigw_role** 角色輸入角色 ARN。
10. 選擇 Deploy (部署)。

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 函式

以下是在 Lambda 主控台測試 `Calc` 函數的方式：

1. 選擇測試標籤。
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 API

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 如果這是您第一次使用 API Gateway，您會看到服務功能的介紹頁面。在 REST API 下方，選擇 Build (組建)。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)。在 REST API 下方，選擇組建。

3. 對於API 名稱，輸入 **LambdaCalc**。
4. 在描述，請輸入描述。
5. 將 API 端點類型保持設定為區域。
6. 針對 IP 位址類型，選擇 IPv4。
7. 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。

在建立 API 之後，請建立資源。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。在此步驟中，您會建立 /calc 資源。

建立 /calc 資源

1. 選擇建立資源。
2. 讓代理資源保持關閉。
3. 將資源路徑保持為 /。
4. 針對資源名稱，輸入 **calc**。
5. 讓 CORS (跨來源資源分享) 保持關閉。
6. 選擇建立資源。

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，讓其可透過瀏覽器調用。這種方法很有用，尤其適用於允許開放存取的 API。

在此方法中，Lambda 需要使用 POST 請求來叫用任何 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。

建立 **GET** 方法

1. 選取 /calc 資源，然後選擇建立方法。
2. 針對方法類型，選取 GET。

3. 針對整合類型，選取 AWS 服務。
4. 針對 AWS 區域，選取您建立 Lambda 函數 AWS 區域 的 。
5. 針對 AWS 服務，選取 Lambda。
6. 讓 AWS 子網域保持空白。
7. 針對 HTTP 方法，選取 POST。
8. 針對動作類型，選取使用路徑覆寫。這個選項可讓我們指定 Invoke 動作的 ARN，來執行 `Calc` 函數。
9. 針對路徑覆寫，輸入 **`2015-03-31/functions/arn:aws:lambda:`*`us-east-2`*`:`*`account-id`*`:function:Calc/invocations`**。針對 ***`account-id`***，輸入您的 AWS 帳戶 ID。針對 ***`us-east-2`***，輸入您建立 Lambda 函數 AWS 區域 的 。
10. 針對執行角色，輸入 **`lambda_invoke_function_assume_apigw_role`** 的角色 ARN。
11. 請勿變更憑證快取和預設逾時的設定。
12. 選擇方法請求設定。
13. 對於請求驗證程式，選取驗證查詢字串參數與標頭。

    如果用戶端未指定所需參數，則此設定會造成系統傳回錯誤訊息。
14. 選擇 URL 查詢字串參數。

    現在，您會在 /calc 資源上設定 GET 方法的查詢字串參數，用以代表後端 Lambda 函數接收輸入。

    若要建立查詢字串參數，請進行下列操作：

    a. 選擇新增查詢字串。
    b. 對於名稱，輸入 **`operand1`**。
    c. 開啟必要。
    d. 讓快取保持關閉。

    重複相同的步驟，並建立名為 **`operand2`** 的查詢字串和名為 **`operator`** 的查詢字串。
15. 選擇建立方法。

現在，請建立對應範本，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。此範本會將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，作為後端 Lambda 函數的輸入。轉換過的 JSON 物件將會包含為整合請求承載。

將輸入參數對應至整合請求

1. 在整合請求索引標籤上，於整合請求設定下，選擇編輯。
2. 針對請求內文傳遞，選取未定義範本時 (建議)。
3. 選擇對應範本。
4. 選擇新增對應範本。
5. 針對內容類型，輸入 **application/json**。
6. 針對範本內文，輸入下列程式碼：

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

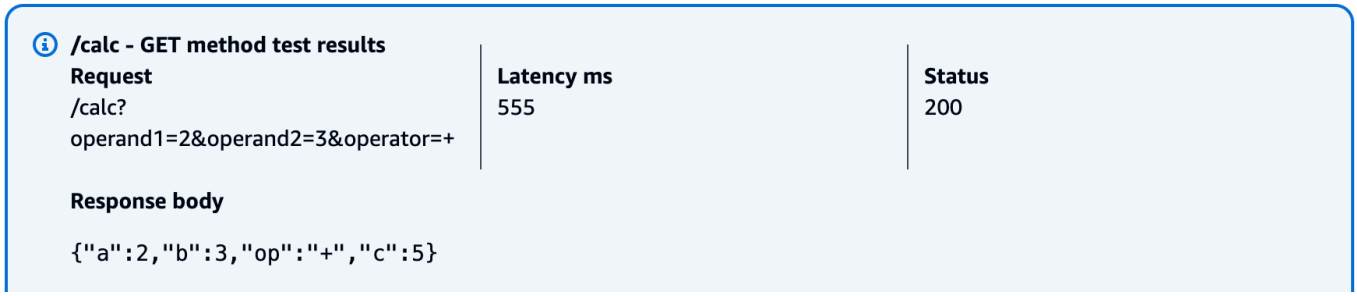
7. 選擇儲存。

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。

測試 **GET** 方法

1. 選擇測試標籤。您可能需要選擇向右箭頭按鈕才能顯示此索引標籤。
2. 針對查詢字串，輸入 **operand1=2&operand2=3&operator=+**。
3. 選擇測試。

   結果看起來會與下列類似：




## 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。

建立具有 JSON 承載的 **POST** 方法

1. 選取 /calc 資源，然後選擇建立方法。
2. 針對方法類型，選取 POST。
3. 針對整合類型，選取 AWS 服務。
4. 針對 AWS 區域，選取您建立 Lambda 函數 AWS 區域 的 。
5. 針對 AWS 服務，選取 Lambda。
6. 讓 AWS 子網域保持空白。
7. 針對 HTTP 方法，選取 POST。
8. 針對動作類型，選取使用路徑覆寫。這個選項可讓我們指定 Invoke 動作的 ARN，來執行 `Calc` 函數。
9. 針對路徑覆寫，輸入 **2015-03-31/functions/arn:aws:lambda:*us-east-2*:*account-id*:function:Calc/invocations**。針對 ***account-id***，輸入您的 AWS 帳戶 ID。針對 ***us-east-2***，輸入您建立 Lambda 函數 AWS 區域 的 。
10. 針對執行角色，輸入 **lambda_invoke_function_assume_apigw_role** 的角色 ARN。
11. 請勿變更憑證快取和預設逾時的設定。
12. 選擇建立方法。

現在，請建立輸入模型來說明輸入資料結構，並驗證傳入請求內文。

建立輸入模型

1. 在主導覽窗格中，選擇模型。
2. 選擇建立模型。
3. 對於名稱，輸入 **input**。
4. 針對內容類型，輸入 **application/json**。

   如果找不到相符的內容類型，則不會執行請求驗證。若要使用相同的模型，而不論內容類型為何，請輸入 **$default**。
5. 針對模型結構描述，輸入下列模型：

```
{
    "type":"object",
    "properties":{
        "a":{"type":"number"},
```

```
        "b":{"type":"number"},
        "op":{"type":"string"}
    },
    "title":"input"
}
```

6. 選擇建立模型。

您現在要建立輸出模型。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。

建立輸出模型

1. 選擇建立模型。
2. 對於名稱，輸入 **output**。
3. 針對內容類型，輸入 **application/json**。

   如果找不到相符的內容類型，則不會執行請求驗證。若要使用相同的模型，而不論內容類型為何，請輸入 **$default**。
4. 針對模型結構描述，輸入下列模型：

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

5. 選擇建立模型。

您現在會建立結果模型。此模型說明所傳回回應資料的資料結構。其會參考 API 中所定義的輸入和輸出結構描述。

建立結果模型

1. 選擇建立模型。
2. 對於名稱，輸入 **result**。
3. 針對內容類型，輸入 **application/json**。

如果找不到相符的內容類型，則不會執行請求驗證。若要使用相同的模型，而不論內容類型為何，請輸入 **$default**。

4. 針對模型結構描述，使用您的 *restapi-id* 輸入以下模型。您的 *restapi-id* 會以括號列在以下流程中的主控台頂端：API Gateway > APIs > LambdaCalc (*abc123*).

```
{
    "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":"result"
}
```

5. 選擇建立模型。

您現在要設定 POST 方法的方法請求，以在傳入請求內文上啟用請求驗證。

若要針對 POST 方法啟用請求驗證

1. 在主導覽窗格中，選擇資源，然後從資源樹狀目錄中選取 POST 方法。
2. 在方法請求索引標籤的方法請求設定下，選擇編輯。
3. 對於請求驗證程式，選取驗證內文。
4. 選擇請求內文，然後選擇新增模型。
5. 針對內容類型，輸入 **application/json**。

   如果找不到相符的內容類型，則不會執行請求驗證。若要使用相同的模型，而不論內容類型為何，請輸入 **$default**。
6. 針對模型，選取輸入。
7. 選擇儲存。

您現在可以測試您的 POST 方法，確認它已正確設定，可以叫用 Lambda 函數。

**測試 POST 方法**

1. 選擇測試標籤。您可能需要選擇向右箭頭按鈕才能顯示此索引標籤。
2. 針對請求內文，輸入下列 JSON 承載。

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

3. 選擇測試。

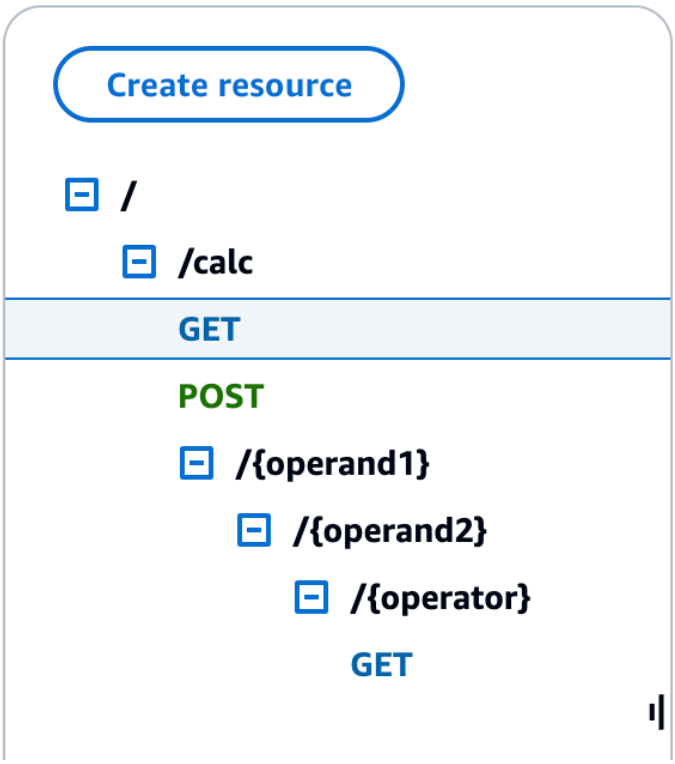
   您應該會看到下列輸出：

```
{
  "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。您需要使用映射範本將傳入路徑參數值映射至所需的整合請求承載。

產生的 API 資源結構看起來像這樣：




建立 /{operand1}/{operand2}/{operator} 資源

1. 選擇建立資源。
2. 針對資源路徑，選取 `/calc`。
3. 針對資源名稱，輸入 **`{operand1}`**。
4. 讓 CORS (跨來源資源分享) 保持關閉。
5. 選擇建立資源。
6. 針對資源路徑，選取 `/calc/{operand1}/`。
7. 針對資源名稱，輸入 **`{operand2}`**。
8. 讓 CORS (跨來源資源分享) 保持關閉。
9. 選擇建立資源。
10. 針對資源路徑，選取 `/calc/{operand1}/{operand2}/`。
11. 針對資源名稱，輸入 **`{operator}`**。
12. 讓 CORS (跨來源資源分享) 保持關閉。
13. 選擇建立資源。

此時，您要在 API Gateway 主控台中使用內建的 Lambda 整合來設定方法整合。

設定方法整合

1. 選取 /{operand1}/{operand2}/{operator} 資源，然後選擇建立方法。
2. 針對方法類型，選取 GET。
3. 針對整合類型，選取 Lambda。
4. 將 Lambda 代理整合保持關閉。
5. 針對 Lambda 函數，選取您建立 Lambda 函數 AWS 區域 的 ，然後輸入 **`Calc`**。
6. 讓預設逾時保持開啟。
7. 選擇建立方法。

您現在要建立對應範本，以將建立 /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。

建立映射範本

1. 在整合請求索引標籤上，於整合請求設定下，選擇編輯。
2. 針對請求內文傳遞，選取未定義範本時 (建議)。
3. 選擇對應範本。
4. 針對內容類型，輸入 **`application/json`**。
5. 針對範本內文，輸入下列程式碼：

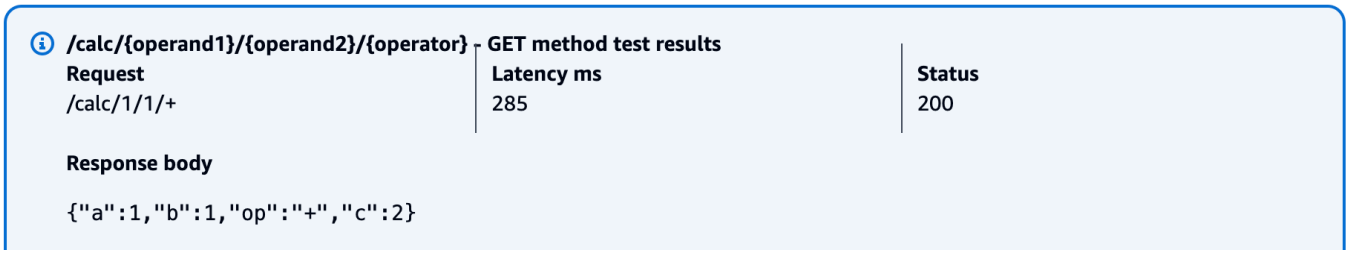
```
{
   "a": "$input.params('operand1')",
   "b": "$input.params('operand2')",
   "op":
 #if($input.params('operator')=='%2F')"/"#{else}"$input.params('operator')"#end
}
```

6. 選擇儲存。

您現在可以測試您的 GET 方法，確認其已正確設定，可以叫用 Lambda 函數，並透過整合回應傳遞原始輸出，而不需要對應。

測試 **GET** 方法

1. 選擇測試標籤。您可能需要選擇向右箭頭按鈕才能顯示此索引標籤。
2. 對於路徑，執行下列動作：
   a. 對於 operand1，輸入 **1**。
   b. 對於 operand2，輸入 **1**。
   c. 對於運算子，輸入 **+**。
3. 選擇 Test (測試)。
4. 結果應如下所示：




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。

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。

建立方法回應

1. 在方法回應索引標籤上的回應 200 下，選擇編輯。
2. 在請求內文下，選擇新增模型。
3. 針對內容類型，輸入 **`application/json`**。
4. 針對模型，選取結果。
5. 選擇儲存。

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。

建立映射範本

1. 在整合回應索引標籤上，於預設 - 回應下，選擇編輯。
2. 選擇映射範本。
3. 針對內容類型，輸入 **application/json**。
4. 針對範本內文，輸入下列程式碼：

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

5. 選擇儲存。

測試對應範本

1. 選擇測試標籤。您可能需要選擇向右箭頭按鈕才能顯示此索引標籤。
2. 對於路徑，執行下列動作：
   a. 對於 operand1，輸入 **1**。
   b. 對於 operand2，輸入 **2**。
   c. 對於運算子，輸入 **+**。
3. 選擇測試。
4. 結果如下所示：

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

```
    "c": 3
  }
}
```

此時只能透過 API Gateway 主控台中的測試功能來呼叫 API。若要讓 API 可供用戶端使用，您需要部署 API。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，如下所示：

部署 API

1. 選擇部署 API。
2. 針對階段，選取新階段。
3. 針對階段名稱，輸入 **Prod**。
4. 在描述，請輸入描述。
5. 選擇部署。
6. (選用) 針對階段詳細資訊下的調用 URL，您可以選擇複製圖示以複製 API 的調用 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": "{\n   \"a\": \"$input.params('operand1')\",
\n   \"b\": \"$input.params('operand2')\",\n   \"op\":
 #if($input.params('operator')=='%2F')\"/\"#{else}\"$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-request-validators": {
    "Validate body": {
      "validateRequestParameters": false,
      "validateRequestBody": true
    },
    "Validate query string parameters and headers": {
      "validateRequestParameters": true,
      "validateRequestBody": false
    }
  }
}
```

## 教學：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/Item 資源上公開 GET，以檢視或下載 Amazon S3 儲存貯體中的物件。

建議您匯入範例 API 作為 Amazon S3 代理，如做為 Amazon S3 代理之範例 API 的 OpenAPI 定義中所示。此範例包含更多已公開的方法。如需如何使用 OpenAPI 定義匯入 API 的說明，請參閱在 API Gateway 中使用 OpenAPI 開發 REST API。

> **Note**
>
> 若要整合 API Gateway API 與 Amazon S3，您必須選擇可使用 API Gateway API 與 Amazon S3 服務的區域。如需區域可用性，請參閱 Amazon API Gateway 端點和配額。

主題


- 設定 API 叫用 Amazon S3 動作的 IAM 許可
- 建立 API 資源以代表 Amazon S3 資源
- 公開 API 方法來列出發起人的 Amazon S3 儲存貯體
- 公開 API 方法來存取 Amazon S3 儲存貯體
- 公開 API 方法來存取儲存貯體中的 Amazon S3 物件
- 做為 Amazon S3 代理之範例 API 的 OpenAPI 定義
- 使用 REST API 用戶端呼叫 API


## 設定 API 叫用 Amazon S3 動作的 IAM 許可

若要允許 API 調用 Amazon S3 動作，您必須將適當的 IAM 政策附加至 IAM 角色。在此步驟中，您會建立新的 IAM 角色。

建立 AWS 服務代理執行角色

1. 登入 AWS 管理主控台 ，並在 https：//https://console.aws.amazon.com/iam/ 開啟 IAM 主控台。
2. 選擇角色。
3. 選擇建立角色。
4. 在選取信任的實體類型下選擇 AWS 服務，然後選取 API Gateway 並選取允許 API Gateway 將日誌推送到 CloudWatch Logs。
5. 選擇下一步，然後選擇下一步。
6. 針對角色名稱，輸入 **`APIGatewayS3ProxyPolicy`**，然後選擇建立角色。
7. 在角色清單中，選擇您剛剛建立的角色。您可能需要捲動或使用搜尋列來尋找該角色。
8. 針對選取的角色，選取 許可 索引標籤。

9. 在下拉式清單中，選擇 連接政策。
10. 在搜尋列中輸入 **AmazonS3FullAccess**，並選擇新增許可。

> **Note**
>
> 本教學課程為了簡單起見，使用受管理政策。最佳實務是，您應建立自己的 IAM 政策以授予所需的最低許可。

11. 記下新建的角色 ARN，以在稍後使用。

## 建立 API 資源以代表 Amazon S3 資源

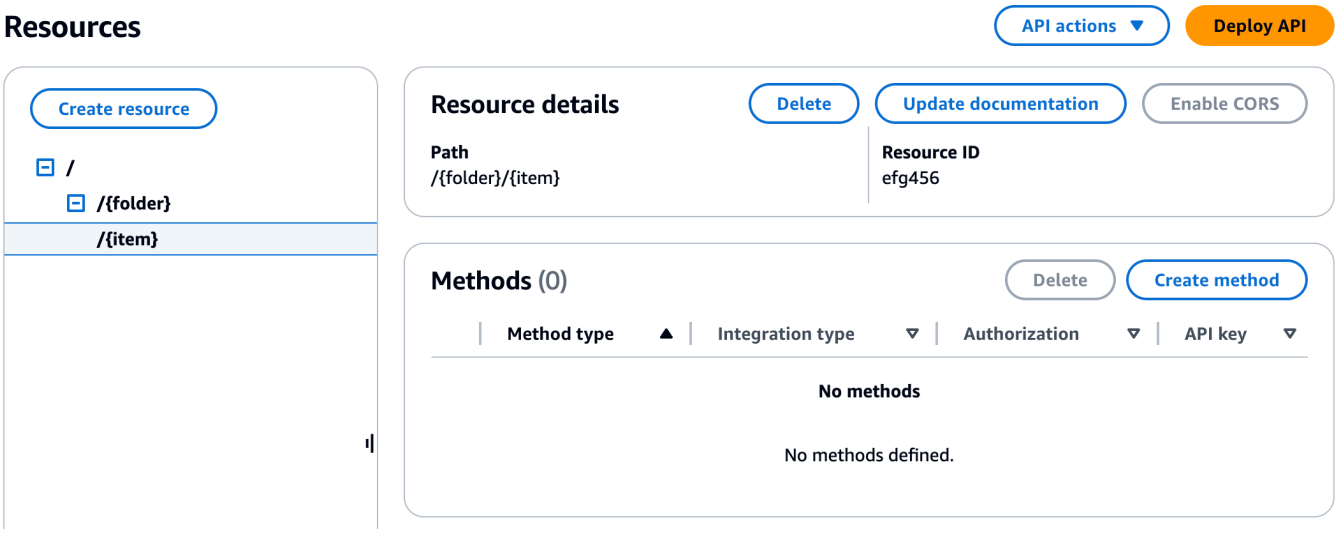
您使用 API 的根 (/) 資源做為已驗證發起人之 Amazon S3 儲存貯體的容器。您也會建立 `Folder` 和 `Item` 資源，分別代表特定 Amazon S3 儲存貯體和特定 Amazon S3 物件。發起人會將資料夾名稱和物件金鑰指定為請求 URL 一部分的路徑參數。

> **Note**
>
> 當所存取物件的物件金鑰包含 / 或任何其他特殊字元時，該字元必須以 URL 編碼。例如，`test/test.txt` 應編碼為 `test%2Ftest.txt`。

建立可公開 Amazon S3 服務功能的 API 資源

1. 在 AWS 區域 建立 Amazon S3 儲存貯體的相同 中，建立名為 MyS3 的 API。這個 API 的根資源 (/) 代表 Amazon S3 服務。在此步驟中，您會另外建立兩個資源 /{folder} 和 /{item}。
2. 選擇建立資源。
3. 讓代理資源保持關閉。
4. 針對資源路徑，選取 /。
5. 針對資源名稱，輸入 **{folder}**。
6. 讓 CORS (跨來源資源分享) 保持未核取狀態。
7. 選擇建立資源。
8. 選取 /{folder} 資源，然後選擇建立資源。
9. 使用上述步驟建立名為 **{item}** 的 /{folder} 子資源。

   您的最終 API 應會如下所示：




## 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。

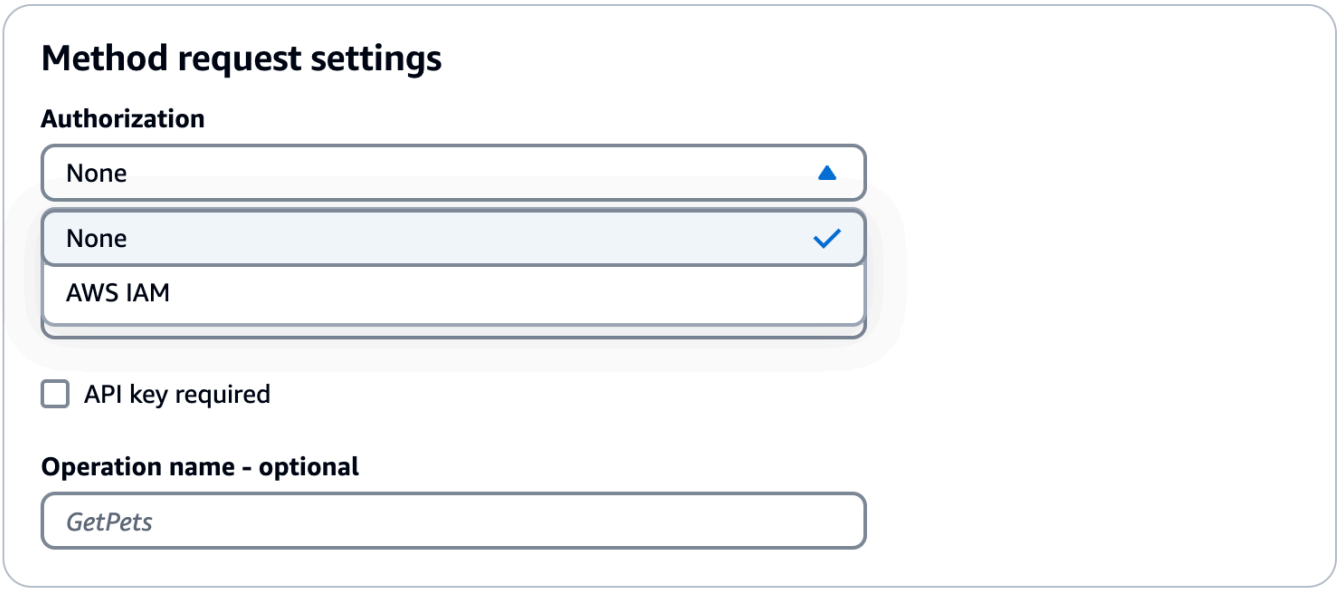
建立和初始化 API 的 **`GET /`** 方法

1. 選取 / 資源，然後選擇建立方法。
2. 針對方法類型，選取 GET。
3. 針對整合類型，選取 AWS 服務。
4. 針對 AWS 區域，選取您建立 Amazon S3 儲存貯 AWS 區域 體的 。
5. 針對 AWS 服務，選取 Amazon Simple Storage Service。
6. 讓 AWS 子網域保持空白。
7. 針對 HTTP 方法，選取 GET。
8. 針對動作類型，選取使用路徑覆寫。

   使用路徑覆寫，API Gateway 會將用戶端請求轉送給 Amazon S3 以作為對應的 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 會設定 `s3-host-name`，並將用戶端指定的 `bucket` 和 `key` 從用戶端傳遞給 Amazon S3。
9. 針對路徑覆寫，輸入 /。
10. 針對執行角色，輸入 **`APIGatewayS3ProxyPolicy`** 的角色 ARN。
11. 選擇方法請求設定。

您可以使用方法請求設定來控制誰可以呼叫此 API 方法。

12. 針對授權，從下拉式功能表中選取 AWS_IAM。




13. 選擇建立方法。

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

為了讓您的 API 將成功回應和例外狀況正確地傳回給發起人，請在方法回應中宣告 200、400 和 500 回應。您使用 200 回應做為預設映射；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. 在方法回應索引標籤上的回應 200 下，選擇編輯。
2. 選擇新增標頭，然後執行下列動作：

   a. 針對標頭名稱，輸入 **Content-Type**。

   b. 選擇新增標頭。

   重複這些步驟以建立 **Timestamp** 標頭和 **Content-Length** 標頭。
3. 選擇儲存。

4. 在方法回應索引標籤上的方法回應下，選擇建立回應。
5. 針對 HTTP 狀態碼，輸入 400。

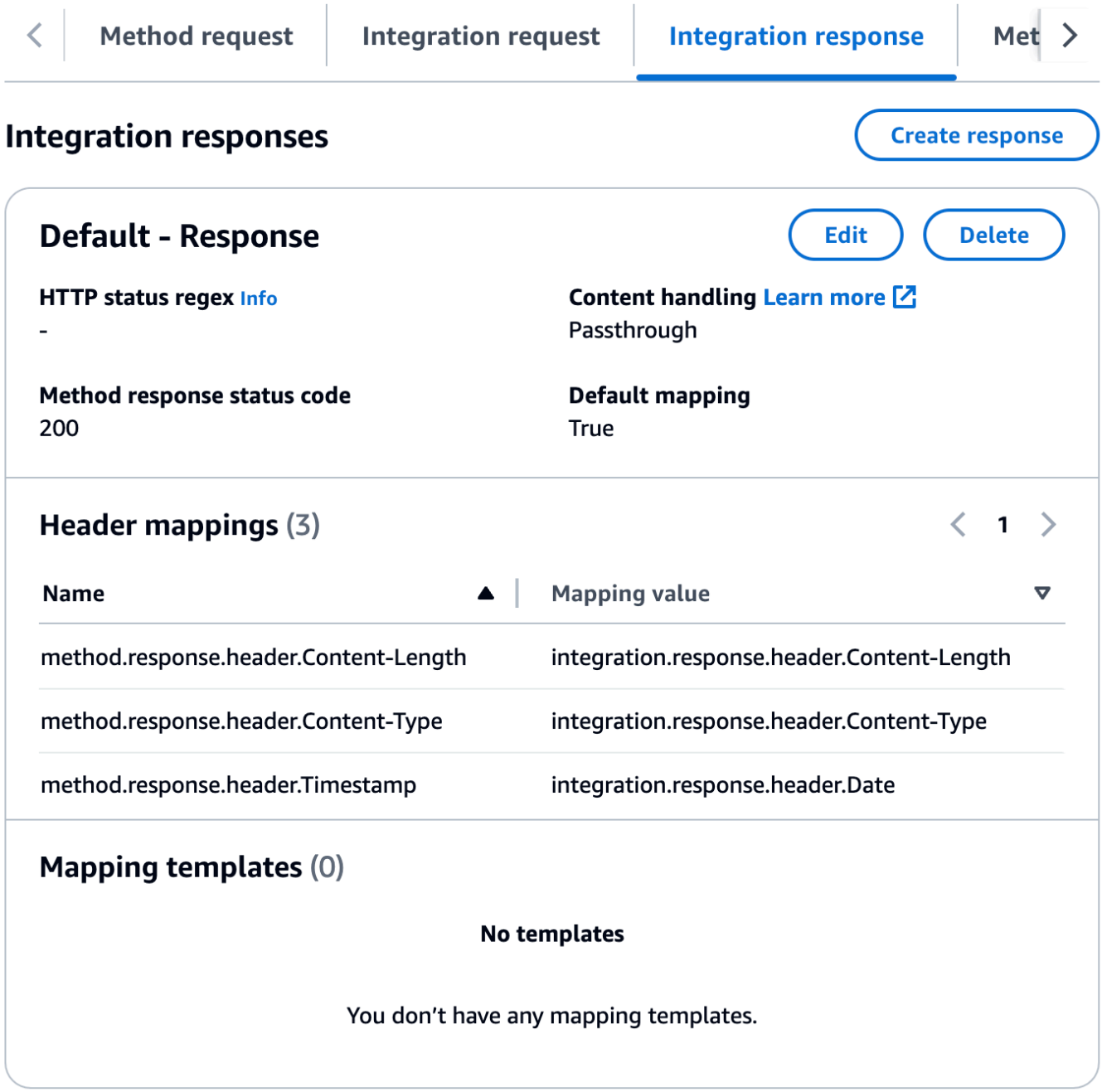
   您未針對此回應設定任何標頭。
6. 選擇儲存。
7. 重複以下步驟以建立 500 回應。

   您未針對此回應設定任何標頭。

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，所以您必須將後端 Content-Type 標頭參數值映射至前端相應實物。否則，回應內文實際上是 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，您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

1. 在整合回應索引標籤上，於預設 - 回應下，選擇編輯。
2. 針對 Content-Length 標頭，輸入 **`integration.response.header.Content-Length`** 作為對應值。
3. 針對 Content-Type 標頭，輸入 **`integration.response.header.Content-Type`** 作為對應值。
4. 針對時間戳記標頭，輸入 **`integration.response.header.Date`** 作為對應值。
5. 選擇儲存。結果應類似以下內容：

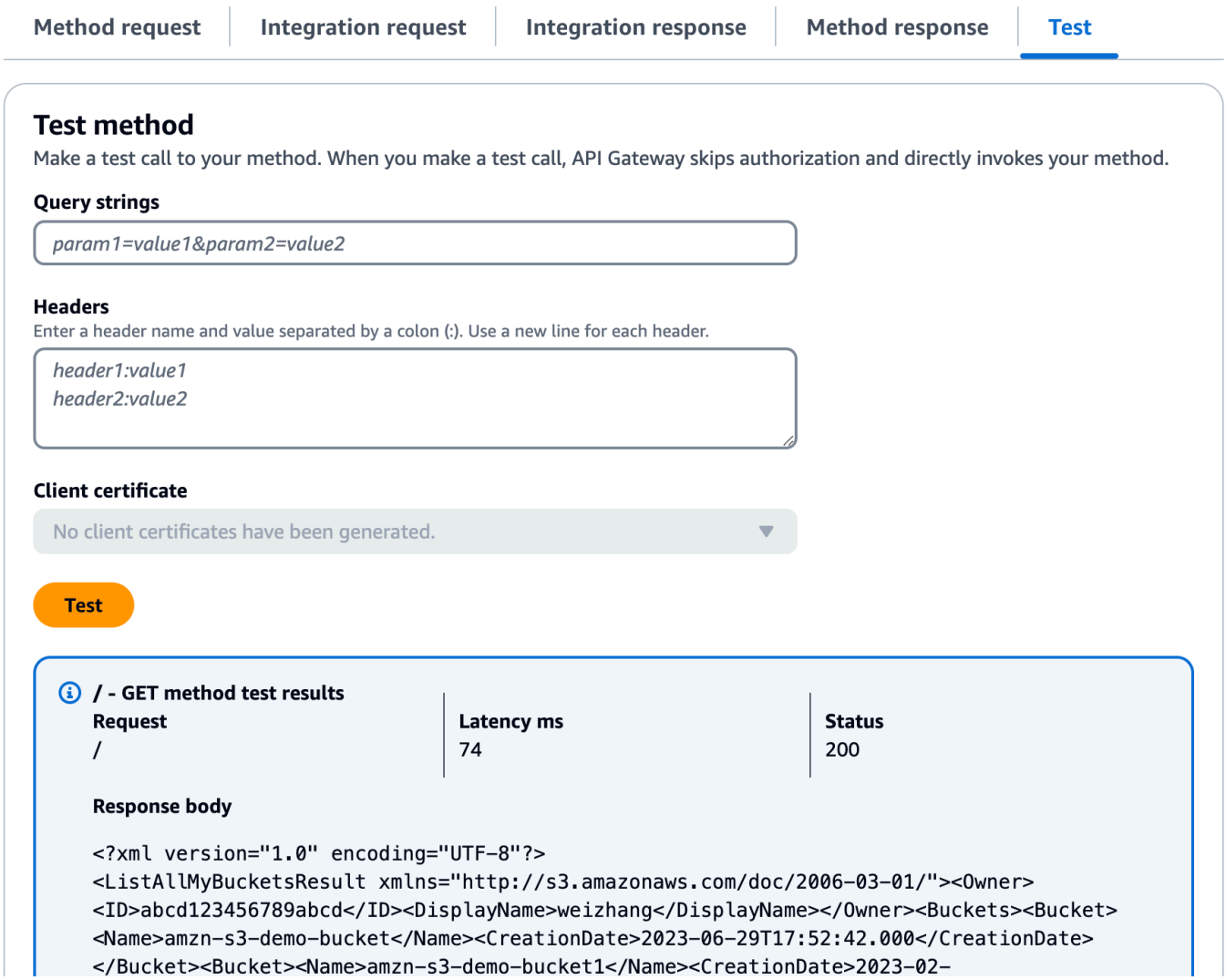



6. 在整合回應索引標籤上的整合回應下，選擇建立回應。
7. 對於 HTTP 狀態 regex，輸入 **`4\d{2}`**。這會將所有 4xx HTTP 回應狀態碼對應到方法回應。
8. 針對方法回應狀態碼，選取 **`400`**。
9. 選擇建立。
10. 重複下列步驟，為 500 方法回應建立整合回應。對於 HTTP 狀態 regex，輸入 **`5\d{2}`**。

您最好測試到目前為止已設定的 API。

### 測試 GET / 方法

1. 選擇測試標籤。您可能需要選擇向右箭頭按鈕才能顯示此索引標籤。
2. 選擇測試。結果應如下圖所示：




## 公開 API 方法來存取 Amazon S3 儲存貯體

為了使用 Amazon S3 儲存貯體，您在/{folder} 資源上公開 GET 方法，以列出儲存貯體中的物件。這些說明與「公開 API 方法來列出發起人的 Amazon S3 儲存貯體」中所述的說明類似。如需更多方法，您可以在此處 (做為 Amazon S3 代理之範例 API 的 OpenAPI 定義) 匯入範例 API。

在資料夾資源上公開 GET 方法

1. 選取 /{folder} 資源，然後選擇建立方法。
2. 針對方法類型，選取 GET。
3. 針對整合類型，選取 AWS 服務。
4. 針對 AWS 區域，選取您建立 Amazon S3 儲存貯 AWS 區域 體的 。
5. 針對 AWS 服務，選取 Amazon Simple Storage Service。
6. 讓 AWS 子網域保持空白。
7. 針對 HTTP 方法，選取 GET。
8. 針對動作類型，選取使用路徑覆寫。
9. 針對路徑覆寫，輸入 **{bucket}**。
10. 針對執行角色，輸入 **APIGatewayS3ProxyPolicy** 的角色 ARN。
11. 選擇建立方法。

您在 Amazon S3 端點 URL 中設定 {folder} 路徑參數。您需要將方法請求的 {folder} 路徑參數對應至整合請求的 {bucket} 路徑參數。

將 **{folder}** 對應至 **{bucket}**

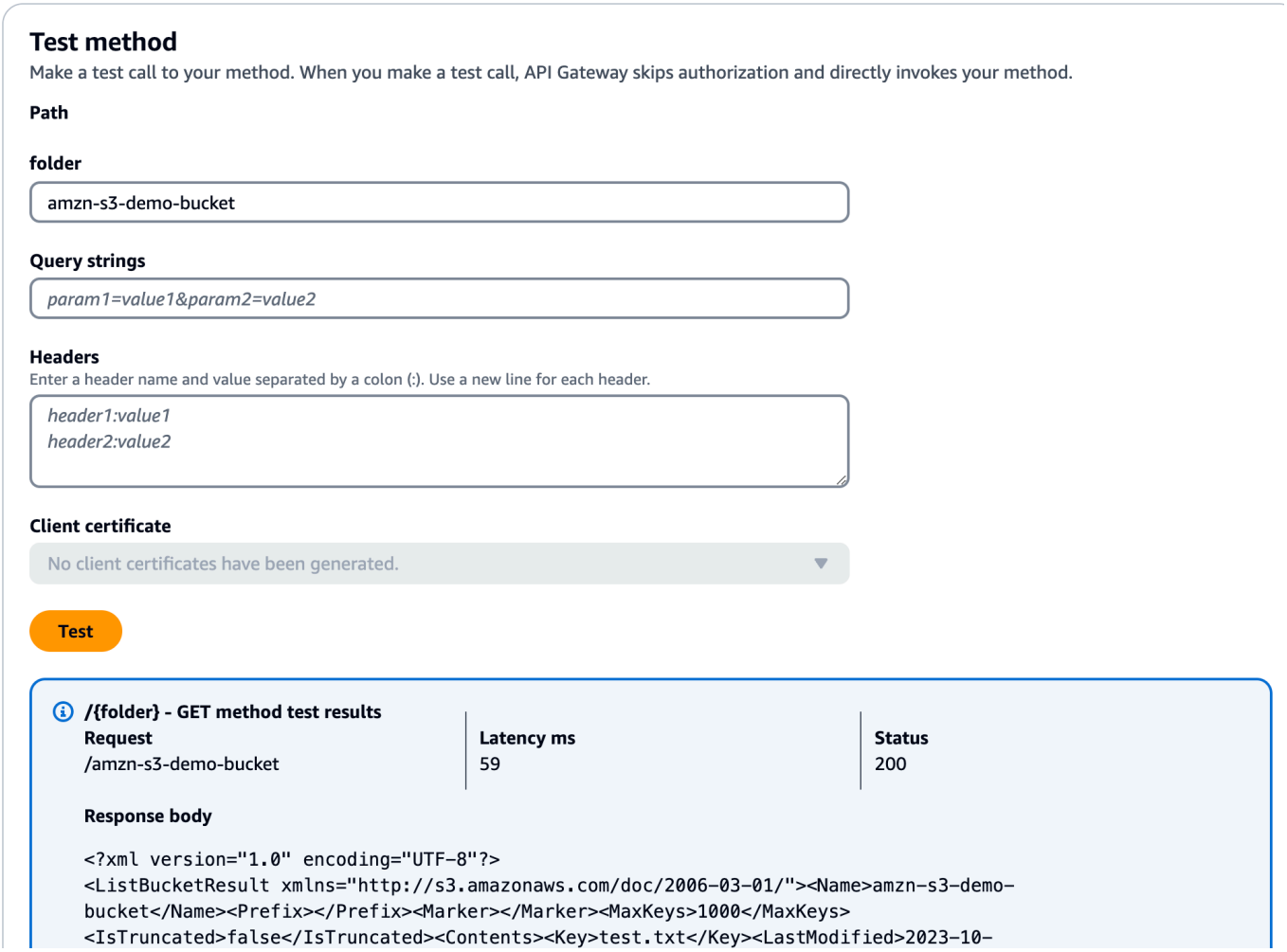
1. 在整合請求索引標籤上，於整合請求設定下，選擇編輯。
2. 選擇 URL 路徑參數，然後選擇新增路徑參數。
3. 對於名稱，輸入 **bucket**。
4. 對於映射自，輸入 **method.request.path.folder**。
5. 選擇儲存。

現在，測試您的 API。

測試 **/{folder} GET** 方法。

1. 選擇測試標籤。您可能需要選擇向右箭頭按鈕才能顯示此索引標籤。
2. 在路徑下，針對資料夾，輸入儲存貯體的名稱。
3. 選擇測試。

   測試結果將包含儲存貯體中的物件清單。




## 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。在本教學課程中，您會在 `{folder}/{item}` 資源上公開 GET 方法，以從儲存貯體取得映像。如需 `{folder}/{item}` 資源的更多應用，請參閱範例 API (做為 Amazon S3 代理之範例 API 的 OpenAPI 定義)。

在項目資源上公開 GET 方法

1. 選取 /{item} 資源，然後選擇建立方法。
2. 針對方法類型，選取 GET。
3. 針對整合類型，選取 AWS 服務。
4. 針對 AWS 區域，選取您建立 Amazon S3 儲存貯 AWS 區域 體的 。
5. 針對 AWS 服務，選取 Amazon Simple Storage Service。

6. 讓 AWS 子網域保持空白。
7. 針對 HTTP 方法，選取 GET。
8. 針對動作類型，選取使用路徑覆寫。
9. 針對路徑覆寫，輸入 {bucket}/{object}。
10. 針對執行角色，輸入 **APIGatewayS3ProxyPolicy** 的角色 ARN。
11. 選擇建立方法。

您在 Amazon S3 端點 URL 中設定 {folder} 和 {item} 路徑參數。您需要將方法請求的 路徑參數映射至整合請求的路徑參數。

請於本步驟執行以下操作：

- 將方法請求的 {folder} 路徑參數對應至整合請求的 {bucket} 路徑參數。
- 將方法請求的 {item} 路徑參數對應至整合請求的 {object} 路徑參數。

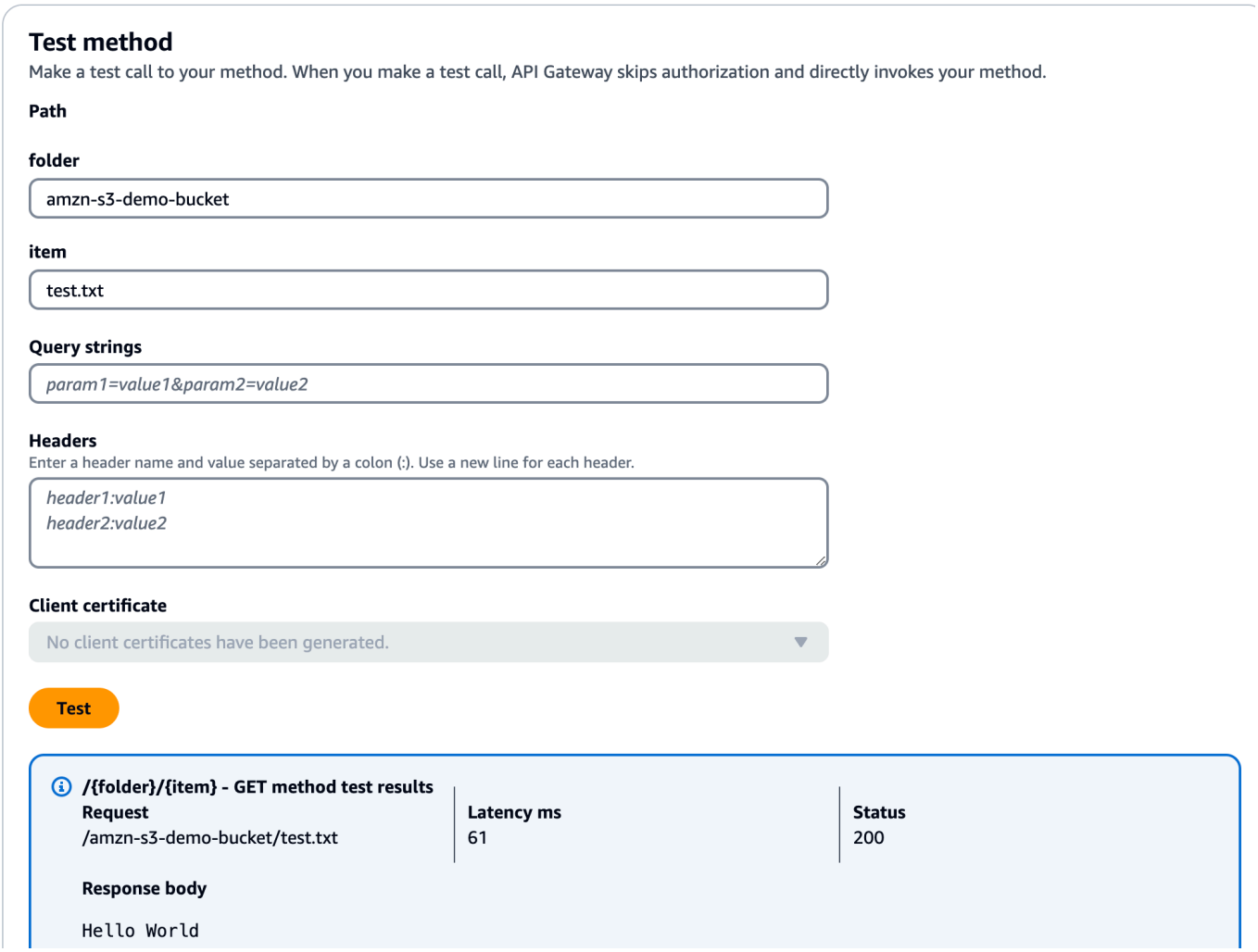
### 將 **{folder}** 對應至 **{bucket}** 並將 **{item}** 對應至 **{object}**

1. 在整合請求索引標籤上，於整合請求設定下，選擇編輯。
2. 選擇 URL 路徑參數。
3. 選擇新增路徑參數。
4. 對於名稱，輸入 **bucket**。
5. 對於映射自，輸入 **method.request.path.folder**。
6. 選擇新增路徑參數。
7. 對於名稱，輸入 **object**。
8. 對於映射自，輸入 **method.request.path.item**。
9. 選擇儲存。

### 測試 **/{folder}/{item} GET** 方法。

1. 選擇測試標籤。您可能需要選擇向右箭頭按鈕才能顯示此索引標籤。
2. 在路徑下，針對資料夾，輸入儲存貯體的名稱。
3. 在路徑下，針對項目，輸入項目的名稱。
4. 選擇測試。

回應內文會包含該項目的內容。




請求會正確地傳回 ("Hello world") 的純文字作為指定 Amazon S3 儲存貯體 (amzn-s3-demo-bucket) 中所指定檔案 (test.txt) 的內容。

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. 選擇 API 的 API 設定。
   b. 在二進位媒體類型下，選擇管理媒體類型。

c. 選擇新增二進位媒體類型，然後輸入所需的媒體類型，例如 `image/png`。

d. 選擇儲存變更來儲存設定。

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the section called “資料轉換”。

承載大小上限為 10 MB。請參閱在 API Gateway 中設定和執行 REST API 的配額。

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。

## 做為 Amazon S3 代理之範例 API 的 OpenAPI 定義

下列 OpenAPI 定義會說明以 Amazon S3 代理形式來運作的 API。此 API 包含的 Amazon S3 操作比您在教學課程中所建立的 API 還多。下列方法會在 OpenAPI 定義中公開：

- 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。

如需如何使用 OpenAPI 定義匯入 API 的說明，請參閱在 API Gateway 中使用 OpenAPI 開發 REST API。

如需如何建立類似 API 的指示，請參閱教學：建立 REST API 做為 Amazon S3 代理。

若要了解如何使用支援 IAM 授權的 Postman AWS 叫用此 API，請參閱 使用 REST API 用戶端呼叫 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::123456789012: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::123456789012: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",
```

```
                "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::123456789012: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.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::123456789012: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::123456789012: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::123456789012: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.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": 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::123456789012: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.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::123456789012: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"
    }
  }
}
```

## OpenAPI 3.0

```
{
  "openapi" : "3.0.1",
  "info" : {
    "title" : "MyS3",
    "version" : "2016-10-13T23:04:43Z"
```

```
  },
  "servers" : [ {
    "url" : "https://9gn28ca086.execute-api.{region}.amazonaws.com/{basePath}",
    "variables" : {
      "basePath" : {
        "default" : "S3"
      }
    }
  } ],
  "paths" : {
    "/{folder}" : {
      "get" : {
        "parameters" : [ {
          "name" : "folder",
          "in" : "path",
          "required" : true,
          "schema" : {
            "type" : "string"
          }
        } ],
        "responses" : {
          "400" : {
            "description" : "400 response",
            "content" : { }
          },
          "500" : {
            "description" : "500 response",
            "content" : { }
          },
          "200" : {
            "description" : "200 response",
            "headers" : {
              "Content-Length" : {
                "schema" : {
                  "type" : "string"
                }
              },
              "Date" : {
                "schema" : {
                  "type" : "string"
                }
              },
              "Content-Type" : {
                "schema" : {
```

```
                "type" : "string"
              }
            }
          },
          "content" : {
            "application/json" : {
              "schema" : {
                "$ref" : "#/components/schemas/Empty"
              }
            }
          }
        }
      },
      "x-amazon-apigateway-integration" : {
        "credentials" : "arn:aws:iam::123456789012:role/apigAwsProxyRole",
        "httpMethod" : "GET",
        "uri" : "arn:aws:apigateway:us-west-2:s3:path/{bucket}",
        "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",
              "method.response.header.Content-Length" :
"integration.response.header.content-length"
            }
          },
          "5\\d{2}" : {
            "statusCode" : "500"
          }
        },
        "requestParameters" : {
          "integration.request.path.bucket" : "method.request.path.folder"
        },
        "passthroughBehavior" : "when_no_match",
        "type" : "aws"
      }
    },
    "put" : {
      "parameters" : [ {
```

```
        "name" : "Content-Type",
        "in" : "header",
        "schema" : {
          "type" : "string"
        }
      }, {
        "name" : "folder",
        "in" : "path",
        "required" : true,
        "schema" : {
          "type" : "string"
        }
      } ],
      "responses" : {
        "400" : {
          "description" : "400 response",
          "content" : { }
        },
        "500" : {
          "description" : "500 response",
          "content" : { }
        },
        "200" : {
          "description" : "200 response",
          "headers" : {
            "Content-Length" : {
              "schema" : {
                "type" : "string"
              }
            },
            "Content-Type" : {
              "schema" : {
                "type" : "string"
              }
            }
          },
          "content" : {
            "application/json" : {
              "schema" : {
                "$ref" : "#/components/schemas/Empty"
              }
            }
          }
        }
```

```
        },
        "x-amazon-apigateway-integration" : {
          "credentials" : "arn:aws:iam::123456789012:role/apigAwsProxyRole",
          "httpMethod" : "PUT",
          "uri" : "arn:aws:apigateway:us-west-2:s3:path/{bucket}",
          "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.bucket" : "method.request.path.folder",
            "integration.request.header.Content-Type" :
"method.request.header.Content-Type"
          },
          "passthroughBehavior" : "when_no_match",
          "type" : "aws"
        }
      },
      "delete" : {
        "parameters" : [ {
          "name" : "folder",
          "in" : "path",
          "required" : true,
          "schema" : {
            "type" : "string"
          }
        } ],
        "responses" : {
          "400" : {
            "description" : "400 response",
            "content" : { }
```

```
          },
          "500" : {
            "description" : "500 response",
            "content" : { }
          },
          "200" : {
            "description" : "200 response",
            "headers" : {
              "Date" : {
                "schema" : {
                  "type" : "string"
                }
              },
              "Content-Type" : {
                "schema" : {
                  "type" : "string"
                }
              }
            },
            "content" : {
              "application/json" : {
                "schema" : {
                  "$ref" : "#/components/schemas/Empty"
                }
              }
            }
          }
        },
        "x-amazon-apigateway-integration" : {
          "credentials" : "arn:aws:iam::123456789012:role/apigAwsProxyRole",
          "httpMethod" : "DELETE",
          "uri" : "arn:aws:apigateway:us-west-2:s3:path/{bucket}",
          "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"
          },
          "passthroughBehavior" : "when_no_match",
          "type" : "aws"
        }
      }
    },
    "/{folder}/{item}" : {
      "get" : {
        "parameters" : [ {
          "name" : "item",
          "in" : "path",
          "required" : true,
          "schema" : {
            "type" : "string"
          }
        }, {
          "name" : "folder",
          "in" : "path",
          "required" : true,
          "schema" : {
            "type" : "string"
          }
        } ],
        "responses" : {
          "400" : {
            "description" : "400 response",
            "content" : { }
          },
          "500" : {
            "description" : "500 response",
            "content" : { }
          },
          "200" : {
            "description" : "200 response",
            "headers" : {
              "content-type" : {
                "schema" : {
                  "type" : "string"
```

```
              }
            },
            "Content-Type" : {
              "schema" : {
                "type" : "string"
              }
            }
          },
          "content" : {
            "application/json" : {
              "schema" : {
                "$ref" : "#/components/schemas/Empty"
              }
            }
          }
        }
      },
      "x-amazon-apigateway-integration" : {
        "credentials" : "arn:aws:iam::123456789012:role/apigAwsProxyRole",
        "httpMethod" : "GET",
        "uri" : "arn:aws:apigateway:us-west-2:s3:path/{bucket}/{object}",
        "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"
        },
        "passthroughBehavior" : "when_no_match",
        "type" : "aws"
```

```
        }
      },
      "put" : {
        "parameters" : [ {
          "name" : "Content-Type",
          "in" : "header",
          "schema" : {
            "type" : "string"
          }
        }, {
          "name" : "item",
          "in" : "path",
          "required" : true,
          "schema" : {
            "type" : "string"
          }
        }, {
          "name" : "folder",
          "in" : "path",
          "required" : true,
          "schema" : {
            "type" : "string"
          }
        } ],
        "responses" : {
          "400" : {
            "description" : "400 response",
            "content" : { }
          },
          "500" : {
            "description" : "500 response",
            "content" : { }
          },
          "200" : {
            "description" : "200 response",
            "headers" : {
              "Content-Length" : {
                "schema" : {
                  "type" : "string"
                }
              },
              "Content-Type" : {
                "schema" : {
                  "type" : "string"
```

```
              }
            }
          },
          "content" : {
            "application/json" : {
              "schema" : {
                "$ref" : "#/components/schemas/Empty"
              }
            }
          }
        }
      },
      "x-amazon-apigateway-integration" : {
        "credentials" : "arn:aws:iam::123456789012:role/apigAwsProxyRole",
        "httpMethod" : "PUT",
        "uri" : "arn:aws:apigateway:us-west-2:s3:path/{bucket}/{object}",
        "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.path.bucket" : "method.request.path.folder",
          "integration.request.header.Content-Type" :
"method.request.header.Content-Type"
        },
        "passthroughBehavior" : "when_no_match",
        "type" : "aws"
      }
    },
    "delete" : {
```

```
        "parameters" : [ {
          "name" : "item",
          "in" : "path",
          "required" : true,
          "schema" : {
            "type" : "string"
          }
        }, {
          "name" : "folder",
          "in" : "path",
          "required" : true,
          "schema" : {
            "type" : "string"
          }
        } ],
        "responses" : {
          "400" : {
            "description" : "400 response",
            "content" : { }
          },
          "500" : {
            "description" : "500 response",
            "content" : { }
          },
          "200" : {
            "description" : "200 response",
            "headers" : {
              "Content-Length" : {
                "schema" : {
                  "type" : "string"
                }
              },
              "Content-Type" : {
                "schema" : {
                  "type" : "string"
                }
              }
            },
            "content" : {
              "application/json" : {
                "schema" : {
                  "$ref" : "#/components/schemas/Empty"
                }
              }
```

```
            }
          }
        },
        "x-amazon-apigateway-integration" : {
          "credentials" : "arn:aws:iam::123456789012:role/apigAwsProxyRole",
          "httpMethod" : "DELETE",
          "uri" : "arn:aws:apigateway:us-west-2:s3:path/{bucket}/{object}",
          "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"
          },
          "passthroughBehavior" : "when_no_match",
          "type" : "aws"
        }
      },
      "head" : {
        "parameters" : [ {
          "name" : "item",
          "in" : "path",
          "required" : true,
          "schema" : {
            "type" : "string"
          }
        }, {
          "name" : "folder",
          "in" : "path",
          "required" : true,
          "schema" : {
            "type" : "string"
          }
        } ],
        "responses" : {
          "400" : {
```

```
            "description" : "400 response",
            "content" : { }
          },
          "500" : {
            "description" : "500 response",
            "content" : { }
          },
          "200" : {
            "description" : "200 response",
            "headers" : {
              "Content-Length" : {
                "schema" : {
                  "type" : "string"
                }
              },
              "Content-Type" : {
                "schema" : {
                  "type" : "string"
                }
              }
            },
            "content" : {
              "application/json" : {
                "schema" : {
                  "$ref" : "#/components/schemas/Empty"
                }
              }
            }
          }
        },
        "x-amazon-apigateway-integration" : {
          "credentials" : "arn:aws:iam::123456789012:role/apigAwsProxyRole",
          "httpMethod" : "HEAD",
          "uri" : "arn:aws:apigateway:us-west-2:s3:path/{bucket}/{object}",
          "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.path.bucket" : "method.request.path.folder"
          },
          "passthroughBehavior" : "when_no_match",
          "type" : "aws"
        }
      }
    },
    "/" : {
      "get" : {
        "responses" : {
          "400" : {
            "description" : "400 response",
            "content" : { }
          },
          "500" : {
            "description" : "500 response",
            "content" : { }
          },
          "200" : {
            "description" : "200 response",
            "headers" : {
              "Content-Length" : {
                "schema" : {
                  "type" : "string"
                }
              },
              "Timestamp" : {
                "schema" : {
                  "type" : "string"
                }
              },
              "Content-Type" : {
                "schema" : {
                  "type" : "string"
```

```
              }
            }
          },
          "content" : {
            "application/json" : {
              "schema" : {
                "$ref" : "#/components/schemas/Empty"
              }
            }
          }
        }
      },
      "x-amazon-apigateway-integration" : {
        "credentials" : "arn:aws:iam::123456789012:role/apigAwsProxyRole",
        "httpMethod" : "GET",
        "uri" : "arn:aws:apigateway:us-west-2:s3:path//",
        "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"
          }
        },
        "passthroughBehavior" : "when_no_match",
        "type" : "aws"
      }
    }
  },
  "components" : {
    "schemas" : {
      "Empty" : {
```

```
        "title" : "Empty Schema",
        "type" : "object"
      }
    }
  }
}
```

## 使用 REST API 用戶端呼叫 API

為了提供end-to-end教學課程，我們現在示範如何使用支援 IAM 授權的 Postman AWS 呼叫 API。

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 顯示在階段編輯器頂端的呼叫 URL 旁邊。
2. 啟動 Postman。
3. 選擇授權，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 區域文字方塊中輸入要部署 API 的 AWS 區域 。在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 `amzn-s3-demo-bucket` 的儲存貯體新增至 *`{region}`* 區域中的 Amazon S3 帳戶：

   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，請遵循上述說明操作。如果您在 URL 中指定 **`amzn-s3-demo-bucket`** 的儲存貯體名稱 `{folder}` 和 **`Readme.txt`** 的檔案名稱 `{item}`，並提供字串 **`Hello, World!`** 做為檔案內容 (因此其會成為請求承載)，則請求會如下所示：

```
PUT /S3/amzn-s3-demo-bucket/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，您應該會收到含有空白承載的 200 OK 回應。

6. 若要取得剛剛新增至 Readme.txt 儲存貯體之 amzn-s3-demo-bucket 檔案的內容，請執行 GET 請求，如下所示：

```
GET /S3/amzn-s3-demo-bucket/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: d60fcb59-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。

7. 若要列出 amzn-s3-demo-bucket 儲存貯體中的項目，請提交下列請求：

```
GET /S3/amzn-s3-demo-bucket 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...603addd12ee</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。

## 教學課程：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 API 與 Kinesis 服務的區域。如需區域可用性，請參閱服務端點和配額。

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 的 `/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 的 `/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「在 API Gateway 中使用 OpenAPI 開發 REST API」。

## 建立 API 的 IAM 角色和政策來存取 Kinesis

若要允許 API 叫用 Kinesis 動作，您必須將適當的 IAM 政策附加至 IAM 角色。在此步驟中，您會建立新的 IAM 角色。

建立 AWS 服務代理執行角色

1. 登入 AWS 管理主控台 並開啟位於 https：//https://console.aws.amazon.com/iam/ 的 IAM 主控台。
2. 選擇角色。
3. 選擇建立角色。
4. 在選取信任的實體類型下選擇 AWS 服務，然後選取 API Gateway 並選取允許 API Gateway 將日誌推送到 CloudWatch Logs。
5. 選擇下一步，然後選擇下一步。
6. 針對角色名稱，輸入 **APIGatewayKinesisProxyPolicy**，然後選擇建立角色。
7. 在角色清單中，選擇您剛剛建立的角色。您可能需要捲動或使用搜尋列來尋找該角色。
8. 針對選取的角色，選取 許可 索引標籤。
9. 在下拉式清單中，選擇 連接政策。
10. 在搜尋列中輸入 **AmazonKinesisFullAccess**，並選擇新增許可。

> **Note**
>
> 本教學課程為了簡單起見，使用受管理政策。最佳實務是，您應建立自己的 IAM 政策以授予所需的最低許可。

11. 記下新建的角色 ARN，以在稍後使用。

## 建立 API 以做為 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 AWS 的服務代理

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 如果這是您第一次使用 API Gateway，您會看到服務功能的介紹頁面。在 REST API 下方，選擇 Build (組建)。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)。在 REST API 下方，選擇 Build (組建)。
3. 選擇 New API (新增 API)。
4. 在 API name (API 名稱) 中，輸入 **KinesisProxy**。對於所有其他欄位，請保留預設值。
5. 在描述，請輸入描述。
6. 針對 IP 位址類型，選擇 IPv4。
7. 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 主控台來列出 Kinesis 串流

1. 選取 / 資源，然後選擇建立資源。
2. 針對資源名稱，輸入 **`streams`**。
3. 讓 CORS (跨來源資源分享) 保持關閉。
4. 選擇建立資源。
5. 選擇 `/streams` 資源，然後選擇建立方法，然後執行下列動作：
   a. 針對方法類型，選取 GET。

      > **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。我們在這裡選取 `GET`，因為列出串流直覺上就是 READ 操作。

   b. 針對整合類型，選取 AWS 服務。
   c. 針對 AWS 區域，選取您建立 Kinesis 串流 AWS 區域 的 。
   d. 針對 AWS 服務，選取 Kinesis。
   e. 讓 AWS 子網域保持空白。
   f. 針對 HTTP method (HTTP 方法)，選擇 POST。

> **Note**
>
> 我們在這裡選擇 POST，因為 Kinesis 需要使用它來叫用 `ListStreams` 動作。

g. 針對動作類型，選擇使用動作名稱。

h. 針對動作名稱，輸入 **`ListStreams`**。

i. 針對執行角色，輸入執行角色的 ARN。

j. 保留內容處理之傳遞的預設值。

k. 選擇建立方法。

6. 在整合請求索引標籤上，於整合請求設定下，選擇編輯。

7. 針對請求內文傳遞，選取未定義範本時 (建議)。

8. 選擇 URL 請求標頭參數，然後執行下列動作：

   a. 選擇新增請求標頭參數。

   b. 對於名稱，輸入 **`Content-Type`**。

   c. 對於映射自，輸入 **`'application/x-amz-json-1.1'`**。

   我們會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。

9. 選擇對應範本，然後選擇新增對應範本，並執行下列動作：

   a. 針對 Content-Type，輸入 **`application/json`**。

   b. 針對範本內文，輸入 **`{}`**。

   c. 選擇儲存。

   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。

10. 測試 /streams 資源上的 GET 方法，以在 Kinesis 中叫用 `ListStreams` 動作：

    選擇測試標籤。您可能需要選擇向右箭頭按鈕才能顯示此索引標籤。

    選擇測試以測試您的方法。

    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

{
    "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 的 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。

建立 {stream-name} 資源

1. 選擇 /streams 資源，然後選擇建立資源。
2. 讓代理資源保持關閉。
3. 針對資源路徑，選取 /streams。
4. 針對資源名稱，輸入 **{stream-name}**。
5. 讓 CORS (跨來源資源分享) 保持關閉。
6. 選擇建立資源。

設定和測試串流資源上的 GET 方法

1. 選擇 /{stream-name} 資源，然後選擇建立方法。
2. 針對方法類型，選取 GET。
3. 針對整合類型，選取 AWS 服務。
4. 針對 AWS 區域，選取您建立 Kinesis 串流 AWS 區域 的 。
5. 針對 AWS 服務，選取 Kinesis。
6. 讓 AWS 子網域保持空白。
7. 針對 HTTP 方法，請選擇 POST。
8. 針對動作類型，選擇使用動作名稱。
9. 針對動作名稱，輸入 **DescribeStream**。
10. 針對執行角色，輸入執行角色的 ARN。
11. 保留內容處理之傳遞的預設值。
12. 選擇建立方法。
13. 在整合請求區段中，新增下列 URL 請求標頭參數：

    ```
    Content-Type: 'x-amz-json-1.1'
    ```

    任務所遵循的程序與設定 GET /streams 方法的請求參數映射相同。
14. 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 DescribeStream 路徑參數值，來產生 Kinesis 之 stream-name 動作所需的整合請求承載。
15. 若要測試在 Kinesis 中叫用 DescribeStream 動作的 GET /stream/{stream-name} 方法，請選擇測試索引標籤。
16. 針對路徑，在 stream-name 下，輸入現有 Kinesis 串流的名稱。
17. 選擇測試。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": "680564733841876926926749214863536422 90",
          "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/stage/streams/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. 選擇 /{stream-name} 資源，然後選擇建立方法。
2. 針對方法類型，選取 POST。
3. 針對整合類型，選取 AWS 服務。
4. 針對 AWS 區域，選取您建立 Kinesis 串流 AWS 區域 的 。
5. 針對 AWS 服務，選取 Kinesis。
6. 讓 AWS 子網域保持空白。
7. 針對 HTTP 方法，請選擇 POST。
8. 針對動作類型，選擇使用動作名稱。
9. 針對動作名稱，輸入 **CreateStream**。
10. 針對執行角色，輸入執行角色的 ARN。
11. 保留內容處理之傳遞的預設值。
12. 選擇建立方法。
13. 在整合請求區段中，新增下列 URL 請求標頭參數：

    ```
    Content-Type: 'x-amz-json-1.1'
    ```

    任務所遵循的程序與設定 `GET /streams` 方法的請求參數映射相同。

14. 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。

15. 若要測試在 Kinesis 中叫用 `CreateStream` 動作的 `POST /stream/{stream-name}` 方法，請選擇測試索引標籤。
16. 針對路徑，在 stream-name 下，輸入新 Kinesis 串流的名稱。
17. 選擇測試。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 中叫用 `CreateStream` 動作：

```
POST 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

{
    "ShardCount": 5
}
```

設定和測試串流資源上的 DELETE 方法

1. 選擇 /{stream-name} 資源，然後選擇建立方法。
2. 針對方法類型，選取 DELETE。
3. 針對整合類型，選取 AWS 服務。
4. 針對 AWS 區域，選取您建立 Kinesis 串流 AWS 區域 的 。
5. 針對 AWS 服務，選取 Kinesis。
6. 讓 AWS 子網域保持空白。
7. 針對 HTTP 方法，請選擇 POST。
8. 針對動作類型，選擇使用動作名稱。
9. 針對動作名稱，輸入 **`DeleteStream`**。
10. 針對執行角色，輸入執行角色的 ARN。
11. 保留內容處理之傳遞的預設值。
12. 選擇建立方法。
13. 在整合請求區段中，新增下列 URL 請求標頭參數：

```
Content-Type: 'x-amz-json-1.1'
```

任務所遵循的程序與設定 GET /streams 方法的請求參數映射相同。

14. 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。

15. 若要測試在 Kinesis 中叫用 DeleteStream 動作的 DELETE /stream/{stream-name} 方法，請選擇測試索引標籤。
16. 針對路徑，在 stream-name 下，輸入現有 Kinesis 串流的名稱。
17. 選擇測試。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：

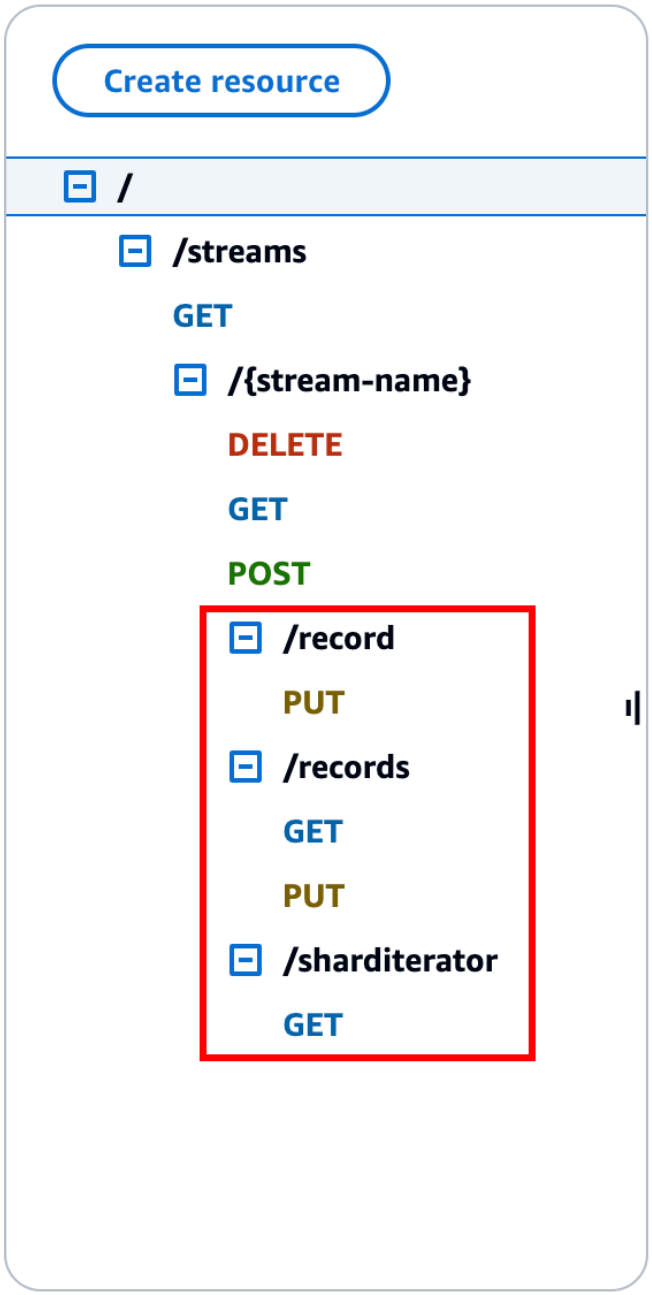
```
POST /?Action=GetShardIterator HTTP/1.1
Host: kinesis.region.domain
Authorization: AWS4-HMAC-SHA256 Credential=..., ...
...
Content-Type: application/x-amz-json-1.1
Content-Length: PayloadSizeBytes

{
    "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)。

建立 /record、/records 和 /sharditerator 資源

1. 選擇 /{stream-name} 資源，然後選擇建立資源。
2. 讓代理資源保持關閉。
3. 針對資源路徑，選取 `/{stream-name}`。
4. 針對資源名稱，輸入 **`record`**。
5. 讓 CORS (跨來源資源分享) 保持關閉。
6. 選擇建立資源。
7. 重複前面的步驟，以建立 /records 和 /sharditerator 資源。最終 API 應如下所示：




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 中叫用 **`PutRecord`**：

1. 選擇 /record，然後選擇建立方法。
2. 針對方法類型，選取 PUT。
3. 針對整合類型，選取 AWS 服務。

4. 針對 AWS 區域，選取您建立 Kinesis 串流 AWS 區域 的 。
5. 針對 AWS 服務，選取 Kinesis。
6. 讓 AWS 子網域保持空白。
7. 針對 HTTP 方法，請選擇 POST。
8. 針對動作類型，選擇使用動作名稱。
9. 針對動作名稱，輸入 **PutRecord**。
10. 針對執行角色，輸入執行角色的 ARN。
11. 保留內容處理之傳遞的預設值。
12. 選擇建立方法。
13. 在整合請求區段中，新增下列 URL 請求標頭參數：

```
Content-Type: 'x-amz-json-1.1'
```

任務所遵循的程序與設定 GET /streams 方法的請求參數映射相同。

14. 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。

15. 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-000000000004"
}
```

### 設定和測試 **PUT /streams/{stream-name}/records** 方法以在 Kinesis 中叫用 **PutRecords**

1. 選擇 /records 資源，然後選擇建立方法。
2. 針對方法類型，選取 PUT。
3. 針對整合類型，選取 AWS 服務。
4. 針對 AWS 區域，選取您建立 Kinesis 串流 AWS 區域 的 。
5. 針對 AWS 服務，選取 Kinesis。
6. 讓 AWS 子網域保持空白。
7. 針對 HTTP 方法，請選擇 POST。
8. 針對動作類型，選擇使用動作名稱。
9. 針對動作名稱，輸入 **PutRecords**。
10. 針對執行角色，輸入執行角色的 ARN。
11. 保留內容處理之傳遞的預設值。
12. 選擇建立方法。
13. 在整合請求區段中，新增下列 URL 請求標頭參數：

```
Content-Type: 'x-amz-json-1.1'
```

任務所遵循的程序與設定 GET /streams 方法的請求參數映射相同。

14. 新增下列對應範本，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。

15. 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 中叫用 **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. 選擇 /sharditerator 資源，然後選擇建立方法。
2. 針對方法類型，選取 GET。
3. 針對整合類型，選取 AWS 服務。
4. 針對 AWS 區域，選取您建立 Kinesis 串流 AWS 區域 的 。
5. 針對 AWS 服務，選取 Kinesis。
6. 讓 AWS 子網域保持空白。
7. 針對 HTTP 方法，請選擇 POST。
8. 針對動作類型，選擇使用動作名稱。
9. 針對動作名稱，輸入 **GetShardIterator**。
10. 針對執行角色，輸入執行角色的 ARN。
11. 保留內容處理之傳遞的預設值。
12. 選擇 URL 查詢字串參數。

    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，如下列步驟所示。
13. 選擇新增查詢字串。
14. 對於名稱，輸入 **shard-id**。
15. 將必要和快取保持關閉。
16. 選擇建立方法。
17. 在整合請求區段中，新增下列對應範本，以從方法請求的 shard-id 和 stream-name 參數產生 GetShardIterator 動作所需的輸入 (ShardId 和 StreamName)。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')"
}
```

18. 使用 API Gateway 主控台中的 Test (測試) 選項，輸入現有串流名稱作為 `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 中叫用 **GetRecords** 動作

1. 選擇 /records 資源，然後選擇建立方法。
2. 針對方法類型，選取 GET。
3. 針對整合類型，選取 AWS 服務。
4. 針對 AWS 區域，選取您建立 Kinesis 串流 AWS 區域 的 。
5. 針對 AWS 服務，選取 Kinesis。
6. 讓 AWS 子網域保持空白。
7. 針對 HTTP 方法，請選擇 POST。
8. 針對動作類型，選擇使用動作名稱。
9. 針對動作名稱，輸入 **GetRecords**。
10. 針對執行角色，輸入執行角色的 ARN。
11. 保留內容處理之傳遞的預設值。
12. 選擇 HTTP 請求標頭。

    `GetRecords` 動作需要 `ShardIterator` 值的輸入。若要傳遞用戶端提供的 `ShardIterator` 值，我們會將 `Shard-Iterator` 標頭參數新增至方法請求。
13. 選擇新增標頭。
14. 對於名稱，輸入 **Shard-Iterator**。
15. 將必要和快取保持關閉。
16. 選擇建立方法。

17. 在整合請求區段中，新增下列內文對應範本，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')"
}
```

18. 使用 API Gateway 主控台中的測試選項，輸入現有串流名稱做為 stream-name 路徑變數值，並將 Shard-Iterator 標頭設定為取自 ShardIterator 方法 (上方) 測試回合的 GET /streams/{stream-name}/sharditerator 值，然後選擇測試。

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

以下是用於本教學課程中做為 Kinesis 代理之範例 API 的 OpenAPI 定義。

OpenAPI 3.0

```
{
  "openapi": "3.0.0",
  "info": {
    "title": "KinesisProxy",
    "version": "2016-03-31T18:25:32Z"
  },
  "paths": {
    "/streams/{stream-name}/sharditerator": {
      "get": {
        "parameters": [
          {
            "name": "stream-name",
            "in": "path",
            "required": true,
            "schema": {
              "type": "string"
```

```
            }
          },
          {
            "name": "shard-id",
            "in": "query",
            "schema": {
              "type": "string"
            }
          }
        ],
        "responses": {
          "200": {
            "description": "200 response",
            "content": {
              "application/json": {
                "schema": {
                  "$ref": "#/components/schemas/Empty"
                }
              }
            }
          }
        },
        "x-amazon-apigateway-integration": {
          "type": "aws",
          "credentials": "arn:aws:iam::123456789012:role/apigAwsProxyRole",
          "uri": "arn:aws:apigateway:us-east-1:kinesis:action/GetShardIterator",
          "responses": {
            "default": {
              "statusCode": "200"
            }
          },
          "requestParameters": {
            "integration.request.header.Content-Type": "'application/x-amz-
json-1.1'"
          },
          "requestTemplates": {
            "application/json": "{\n    \"ShardId\": \"$input.params('shard-
id')\",\n    \"ShardIteratorType\": \"TRIM_HORIZON\",\n    \"StreamName\":
 \"$input.params('stream-name')\"\n}"
          },
          "passthroughBehavior": "when_no_match",
          "httpMethod": "POST"
        }
      }
```

```
    },
    "/streams/{stream-name}/records": {
      "get": {
        "parameters": [
          {
            "name": "stream-name",
            "in": "path",
            "required": true,
            "schema": {
              "type": "string"
            }
          },
          {
            "name": "Shard-Iterator",
            "in": "header",
            "schema": {
              "type": "string"
            }
          }
        ],
        "responses": {
          "200": {
            "description": "200 response",
            "content": {
              "application/json": {
                "schema": {
                  "$ref": "#/components/schemas/Empty"
                }
              }
            }
          }
        },
        "x-amazon-apigateway-integration": {
          "type": "aws",
          "credentials": "arn:aws:iam::123456789012:role/apigAwsProxyRole",
          "uri": "arn:aws:apigateway:us-east-1:kinesis:action/GetRecords",
          "responses": {
            "default": {
              "statusCode": "200"
            }
          },
          "requestParameters": {
            "integration.request.header.Content-Type": "'application/x-amz-
json-1.1'"
```

```
          },
          "requestTemplates": {
            "application/json": "{\n    \"ShardIterator\": \"$input.params('Shard-
Iterator')\"\n}"
          },
          "passthroughBehavior": "when_no_match",
          "httpMethod": "POST"
        }
      },
      "put": {
        "parameters": [
          {
            "name": "Content-Type",
            "in": "header",
            "schema": {
              "type": "string"
            }
          },
          {
            "name": "stream-name",
            "in": "path",
            "required": true,
            "schema": {
              "type": "string"
            }
          }
        ],
        "requestBody": {
          "content": {
            "application/json": {
              "schema": {
                "$ref": "#/components/schemas/PutRecordsMethodRequestPayload"
              }
            },
            "application/x-amz-json-1.1": {
              "schema": {
                "$ref": "#/components/schemas/PutRecordsMethodRequestPayload"
              }
            }
          },
          "required": true
        },
        "responses": {
          "200": {
```

```
            "description": "200 response",
            "content": {
              "application/json": {
                "schema": {
                  "$ref": "#/components/schemas/Empty"
                }
              }
            }
          }
        },
        "x-amazon-apigateway-integration": {
          "type": "aws",
          "credentials": "arn:aws:iam::123456789012:role/apigAwsProxyRole",
          "uri": "arn:aws:apigateway:us-east-1:kinesis:action/PutRecords",
          "responses": {
            "default": {
              "statusCode": "200"
            }
          },
          "requestParameters": {
            "integration.request.header.Content-Type": "'application/x-amz-
json-1.1'"
          },
          "requestTemplates": {
            "application/json": "{\n    \"StreamName\": \"$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}",
            "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}"
          },
          "passthroughBehavior": "when_no_match",
          "httpMethod": "POST"
        }
      }
    },
    "/streams/{stream-name}": {
      "get": {
        "parameters": [
          {
            "name": "stream-name",
            "in": "path",
```

```
            "required": true,
            "schema": {
              "type": "string"
            }
          }
        ],
        "responses": {
          "200": {
            "description": "200 response",
            "content": {
              "application/json": {
                "schema": {
                  "$ref": "#/components/schemas/Empty"
                }
              }
            }
          }
        },
        "x-amazon-apigateway-integration": {
          "type": "aws",
          "credentials": "arn:aws:iam::123456789012:role/apigAwsProxyRole",
          "uri": "arn:aws:apigateway:us-east-1:kinesis:action/DescribeStream",
          "responses": {
            "default": {
              "statusCode": "200"
            }
          },
          "requestTemplates": {
            "application/json": "{\n    \"StreamName\": \"$input.params('stream-
name')\"\n}"
          },
          "passthroughBehavior": "when_no_match",
          "httpMethod": "POST"
        }
      },
      "post": {
        "parameters": [
          {
            "name": "stream-name",
            "in": "path",
            "required": true,
            "schema": {
              "type": "string"
            }
```

```
          }
        ],
        "responses": {
          "200": {
            "description": "200 response",
            "content": {
              "application/json": {
                "schema": {
                  "$ref": "#/components/schemas/Empty"
                }
              }
            }
          }
        },
        "x-amazon-apigateway-integration": {
          "type": "aws",
          "credentials": "arn:aws:iam::123456789012:role/apigAwsProxyRole",
          "uri": "arn:aws:apigateway:us-east-1:kinesis:action/CreateStream",
          "responses": {
            "default": {
              "statusCode": "200"
            }
          },
          "requestParameters": {
            "integration.request.header.Content-Type": "'application/x-amz-
json-1.1'"
          },
          "requestTemplates": {
            "application/json": "{\n    \"ShardCount\": 5,\n    \"StreamName\":
 \"$input.params('stream-name')\"\n}"
          },
          "passthroughBehavior": "when_no_match",
          "httpMethod": "POST"
        }
      },
      "delete": {
        "parameters": [
          {
            "name": "stream-name",
            "in": "path",
            "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"
                }
              }
            },
            "content": {}
          },
          "500": {
            "description": "500 response",
            "headers": {
              "Content-Type": {
                "schema": {
                  "type": "string"
                }
              }
            },
            "content": {}
          }
        },
        "x-amazon-apigateway-integration": {
```

```
          "type": "aws",
          "credentials": "arn:aws:iam::123456789012:role/apigAwsProxyRole",
          "uri": "arn:aws:apigateway:us-east-1:kinesis:action/DeleteStream",
          "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"
              }
            }
          },
          "requestParameters": {
            "integration.request.header.Content-Type": "'application/x-amz-
json-1.1'"
          },
          "requestTemplates": {
            "application/json": "{\n    \"StreamName\": \"$input.params('stream-
name')\"\n}"
          },
          "passthroughBehavior": "when_no_match",
          "httpMethod": "POST"
        }
      }
    },
    "/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": {
                  "$ref": "#/components/schemas/Empty"
                }
              }
            }
          }
        },
        "x-amazon-apigateway-integration": {
          "type": "aws",
          "credentials": "arn:aws:iam::123456789012:role/apigAwsProxyRole",
          "uri": "arn:aws:apigateway:us-east-1:kinesis:action/PutRecord",
          "responses": {
            "default": {
              "statusCode": "200"
            }
          },
          "requestParameters": {
            "integration.request.header.Content-Type": "'application/x-amz-
json-1.1'"
          },
          "requestTemplates": {
            "application/json": "{\n    \"StreamName\": \"$input.params('stream-
name')\",\n    \"Data\": \"$util.base64Encode($input.json('$.Data'))\",\n
 \"PartitionKey\": \"$input.path('$.PartitionKey')\"\n}"
          },
          "passthroughBehavior": "when_no_match",
          "httpMethod": "POST"
        }
      }
    },
    "/streams": {
      "get": {
```

```
        "responses": {
          "200": {
            "description": "200 response",
            "content": {
              "application/json": {
                "schema": {
                  "$ref": "#/components/schemas/Empty"
                }
              }
            }
          }
        },
        "x-amazon-apigateway-integration": {
          "type": "aws",
          "credentials": "arn:aws:iam::123456789012:role/apigAwsProxyRole",
          "uri": "arn:aws:apigateway:us-east-1:kinesis:action/ListStreams",
          "responses": {
            "default": {
              "statusCode": "200"
            }
          },
          "requestParameters": {
            "integration.request.header.Content-Type": "'application/x-amz-
json-1.1'"
          },
          "requestTemplates": {
            "application/json": "{\n}"
          },
          "passthroughBehavior": "when_no_match",
          "httpMethod": "POST"
        }
      }
    }
  },
  "components": {
    "schemas": {
      "Empty": {
        "type": "object"
      },
      "PutRecordsMethodRequestPayload": {
        "type": "object",
        "properties": {
          "records": {
            "type": "array",
```

```
            "items": {
              "type": "object",
              "properties": {
                "data": {
                  "type": "string"
                },
                "partition-key": {
                  "type": "string"
                }
              }
            }
          }
        }
      }
    }
  }
}
```

## OpenAPI 2.0

```
{
  "swagger": "2.0",
  "info": {
    "version": "2016-03-31T18:25:32Z",
    "title": "KinesisProxy"
  },
  "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": {
          "type": "aws",
          "credentials": "arn:aws:iam::123456789012:role/apigAwsProxyRole",
          "uri": "arn:aws:apigateway:us-east-1:kinesis:action/ListStreams",
          "responses": {
            "default": {
              "statusCode": "200"
            }
          },
          "requestParameters": {
            "integration.request.header.Content-Type": "'application/x-amz-
json-1.1'"
          },
          "requestTemplates": {
            "application/json": "{\n}"
          },
          "passthroughBehavior": "when_no_match",
          "httpMethod": "POST"
        }
      }
    },
    "/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": {
              "$ref": "#/definitions/Empty"
            }
          }
        },
        "x-amazon-apigateway-integration": {
          "type": "aws",
          "credentials": "arn:aws:iam::123456789012:role/apigAwsProxyRole",
          "uri": "arn:aws:apigateway:us-east-1:kinesis:action/DescribeStream",
          "responses": {
            "default": {
              "statusCode": "200"
            }
          },
          "requestTemplates": {
            "application/json": "{\n    \"StreamName\": \"$input.params('stream-
name')\"\n}"
          },
          "passthroughBehavior": "when_no_match",
          "httpMethod": "POST"
        }
      },
      "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": {
          "type": "aws",
          "credentials": "arn:aws:iam::123456789012:role/apigAwsProxyRole",
          "uri": "arn:aws:apigateway:us-east-1:kinesis:action/CreateStream",
          "responses": {
            "default": {
              "statusCode": "200"
            }
          },
          "requestParameters": {
            "integration.request.header.Content-Type": "'application/x-amz-
json-1.1'"
          },
          "requestTemplates": {
            "application/json": "{\n    \"ShardCount\": 5,\n    \"StreamName\":
 \"$input.params('stream-name')\"\n}"
          },
          "passthroughBehavior": "when_no_match",
          "httpMethod": "POST"
        }
      },
      "delete": {
        "consumes": [
          "application/json"
        ],
        "produces": [
          "application/json"
        ],
        "parameters": [
          {
            "name": "stream-name",
            "in": "path",
            "required": true,
            "type": "string"
          }
        ],
        "responses": {
          "200": {
            "description": "200 response",
            "schema": {
              "$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": {
          "type": "aws",
          "credentials": "arn:aws:iam::123456789012:role/apigAwsProxyRole",
          "uri": "arn:aws:apigateway:us-east-1:kinesis:action/DeleteStream",
          "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"
              }
            }
          },
          "requestParameters": {
            "integration.request.header.Content-Type": "'application/x-amz-
json-1.1'"
          },
          "requestTemplates": {
            "application/json": "{\n    \"StreamName\": \"$input.params('stream-
name')\"\n}"
          },
          "passthroughBehavior": "when_no_match",
          "httpMethod": "POST"
        }
      }
    },
    "/streams/{stream-name}/record": {
      "put": {
        "consumes": [
          "application/json"
        ],
        "produces": [
          "application/json"
        ],
        "parameters": [
          {
            "name": "stream-name",
            "in": "path",
            "required": true,
            "type": "string"
          }
        ],
        "responses": {
          "200": {
            "description": "200 response",
            "schema": {
              "$ref": "#/definitions/Empty"
            }
          }
        },
        "x-amazon-apigateway-integration": {
          "type": "aws",
```

```
          "credentials": "arn:aws:iam::123456789012:role/apigAwsProxyRole",
          "uri": "arn:aws:apigateway:us-east-1:kinesis:action/PutRecord",
          "responses": {
            "default": {
              "statusCode": "200"
            }
          },
          "requestParameters": {
            "integration.request.header.Content-Type": "'application/x-amz-
json-1.1'"
          },
          "requestTemplates": {
            "application/json": "{\n    \"StreamName\": \"$input.params('stream-
name')\",\n    \"Data\": \"$util.base64Encode($input.json('$.Data'))\",\n
 \"PartitionKey\": \"$input.path('$.PartitionKey')\"\n}"
          },
          "passthroughBehavior": "when_no_match",
          "httpMethod": "POST"
        }
      }
    },
    "/streams/{stream-name}/records": {
      "get": {
        "consumes": [
          "application/json"
        ],
        "produces": [
          "application/json"
        ],
        "parameters": [
          {
            "name": "stream-name",
            "in": "path",
            "required": true,
            "type": "string"
          },
          {
            "name": "Shard-Iterator",
            "in": "header",
            "required": false,
            "type": "string"
          }
        ],
        "responses": {
```

```
          "200": {
            "description": "200 response",
            "schema": {
              "$ref": "#/definitions/Empty"
            }
          }
        },
        "x-amazon-apigateway-integration": {
          "type": "aws",
          "credentials": "arn:aws:iam::123456789012:role/apigAwsProxyRole",
          "uri": "arn:aws:apigateway:us-east-1:kinesis:action/GetRecords",
          "responses": {
            "default": {
              "statusCode": "200"
            }
          },
          "requestParameters": {
            "integration.request.header.Content-Type": "'application/x-amz-
json-1.1'"
          },
          "requestTemplates": {
            "application/json": "{\n    \"ShardIterator\": \"$input.params('Shard-
Iterator')\"\n}"
          },
          "passthroughBehavior": "when_no_match",
          "httpMethod": "POST"
        }
      },
      "put": {
        "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": {
              "$ref": "#/definitions/PutRecordsMethodRequestPayload"
            }
          },
          {
            "in": "body",
            "name": "PutRecordsMethodRequestPayload",
            "required": true,
            "schema": {
              "$ref": "#/definitions/PutRecordsMethodRequestPayload"
            }
          }
        ],
        "responses": {
          "200": {
            "description": "200 response",
            "schema": {
              "$ref": "#/definitions/Empty"
            }
          }
        },
        "x-amazon-apigateway-integration": {
          "type": "aws",
          "credentials": "arn:aws:iam::123456789012:role/apigAwsProxyRole",
          "uri": "arn:aws:apigateway:us-east-1:kinesis:action/PutRecords",
          "responses": {
            "default": {
              "statusCode": "200"
            }
          },
          "requestParameters": {
            "integration.request.header.Content-Type": "'application/x-amz-
json-1.1'"
          },
          "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($foreach.hasNext),#end\n    ]\n}",
            "application/x-amz-json-1.1": "{\n  \"StreamName\":
 \"$input.params('stream-name')\",\n  \"records\" : [\n    {\n        \"Data
\" : \"$elem.data\",\n        \"PartitionKey\" : \"$elem.partition-key\"\n
  }#if($foreach.hasNext),#end\n  ]\n}"
          },
          "passthroughBehavior": "when_no_match",
          "httpMethod": "POST"
        }
      }
    },
    "/streams/{stream-name}/sharditerator": {
      "get": {
        "consumes": [
          "application/json"
        ],
        "produces": [
          "application/json"
        ],
        "parameters": [
          {
            "name": "stream-name",
            "in": "path",
            "required": true,
            "type": "string"
          },
          {
            "name": "shard-id",
            "in": "query",
            "required": false,
            "type": "string"
          }
        ],
        "responses": {
          "200": {
            "description": "200 response",
            "schema": {
              "$ref": "#/definitions/Empty"
            }
          }
        },
```

```
        "x-amazon-apigateway-integration": {
          "type": "aws",
          "credentials": "arn:aws:iam::123456789012:role/apigAwsProxyRole",
          "uri": "arn:aws:apigateway:us-east-1:kinesis:action/GetShardIterator",
          "responses": {
            "default": {
              "statusCode": "200"
            }
          },
          "requestParameters": {
            "integration.request.header.Content-Type": "'application/x-amz-
json-1.1'"
          },
          "requestTemplates": {
            "application/json": "{\n    \"ShardId\": \"$input.params('shard-
id')\",\n    \"ShardIteratorType\": \"TRIM_HORIZON\",\n    \"StreamName\":
 \"$input.params('stream-name')\"\n}"
          },
          "passthroughBehavior": "when_no_match",
          "httpMethod": "POST"
        }
      }
    }
  },
  "definitions": {
    "Empty": {
      "type": "object"
    },
    "PutRecordsMethodRequestPayload": {
      "type": "object",
      "properties": {
        "records": {
          "type": "array",
          "items": {
            "type": "object",
            "properties": {
              "data": {
                "type": "string"
              },
              "partition-key": {
                "type": "string"
              }
            }
          }
```

```
        }
      }
    }
  }
}
```

## 教學課程：使用 AWS SDKs或 建立 REST API AWS CLI

使用下列教學課程建立支援 `GET /pets` 和 `GET /pets/{petId}` 方法的 PetStore API。該方法會與 HTTP 端點整合。您可以使用適用於 JavaScript 的 AWS SDK、 適用於 Python (Boto3) 的 AWS SDK 或 來遵循本教學課程 AWS CLI。您可以使用下列函數或命令來設定 API：

JavaScript v3

- CreateRestApiCommand
- CreateResourceCommand
- PutMethodCommand
- PutMethodResponseCommand
- PutIntegrationCommand
- PutIntegrationResponseCommand
- CreateDeploymentCommand

Python

- create_rest_api
- create_resource
- put_method
- put_method_response
- put_integration
- put_integration_response
- create_deployment

AWS CLI

- create-rest-api

- create-resource
- put-method
- put-method-response
- put-integration
- put-integration-response
- create-deployment

如需適用於 JavaScript 的 AWS SDK v3 的詳細資訊，請參閱什麼是適用於 JavaScript 的 AWS SDK？。如需 SDK for Python (Boto3) 的詳細資訊，請參閱 適用於 Python (Boto3) 的 AWS SDK。如需 的詳細資訊 AWS CLI，請參閱什麼是 AWS CLI？

## 設定邊緣最佳化的 PetStore API

在本教學課程中，範例命令會使用值 ID 的預留位置值，例如 API ID 和資源 ID。完成教學課程時，請將這些值取代為您自己的值。

使用 AWS SDKs 設定邊緣最佳化 PetStore API

1. 使用以下程式碼建立 `RestApi` 實體：

   JavaScript v3

```
import {APIGatewayClient, CreateRestApiCommand} from "@aws-sdk/client-api-
gateway";
(async function (){
const apig = new APIGatewayClient({region:"us-east-1"});
const command = new CreateRestApiCommand({
    name: "Simple PetStore (JavaScript v3 SDK)",
    description: "Demo API created using the AWS SDK for JavaScript v3",
    version: "0.00.001",
    binaryMediaTypes: [
    '*']
});
try {
    const results = await apig.send(command)
    console.log(results)
} catch (err) {
    console.error(Couldn't create API:\n", err)
}
```

```
})();
```

成功的呼叫會在如下所示的輸出中，傳回您的 API ID 和 API 的根資源 ID：

```
{
  id: 'abc1234',
  name: 'PetStore (JavaScript v3 SDK)',
  description: 'Demo API created using the AWS SDK for node.js',
  createdDate: 2017-09-05T19:32:35.000Z,
  version: '0.00.001',
  rootResourceId: 'efg567'
  binaryMediaTypes: [ '*' ]
}
```

Python

```
import botocore
import boto3
import logging

logger = logging.getLogger()
apig = boto3.client('apigateway')

try:
    result = apig.create_rest_api(
        name='Simple PetStore (Python SDK)',
        description='Demo API created using the AWS SDK for Python',
        version='0.00.001',
        binaryMediaTypes=[
            '*'
        ]
    )
except botocore.exceptions.ClientError as error:
    logger.exception("Couldn't create REST API %s.", error)
    raise
attribute=["id","name","description","createdDate","version","binaryMediaTypes","apiKeyS
filtered_result ={key:result[key] for key in attribute}
print(filtered_result)
```

成功的呼叫會在如下所示的輸出中，傳回您的 API ID 和 API 的根資源 ID：

```
{'id': 'abc1234', 'name': 'Simple PetStore (Python SDK)', 'description':
 'Demo API created using the AWS SDK for Python', 'createdDate':
 datetime.datetime(2024, 4, 3, 14, 31, 39, tzinfo=tzlocal()), 'version':
 '0.00.001', 'binaryMediaTypes': ['*'], 'apiKeySource': 'HEADER',
 'endpointConfiguration': {'types': ['EDGE']}, 'disableExecuteApiEndpoint':
 False, 'rootResourceId': 'efg567'}
```

AWS CLI

```
aws apigateway create-rest-api --name 'Simple PetStore (AWS CLI)' --region us-
west-2
```

以下是此命令的輸出：

```
{
    "id": "abcd1234",
    "name": "Simple PetStore (AWS CLI)",
    "createdDate": "2022-12-15T08:07:04-08:00",
    "apiKeySource": "HEADER",
    "endpointConfiguration": {
        "types": [
            "EDGE"
        ]
    },
    "disableExecuteApiEndpoint": false,
    "rootResourceId": "efg567"
}
```

您建立的 API 具有 abcd1234 的 API ID 和 efg567 的根資源 ID。您可以在 API 的設定中使用這些值。

2. 若要在根的下方附加子資源，請將 RootResourceId 指定為 parentId 屬性值。使用下列範例為您的 API 建立 /pets 資源：

JavaScript v3

```
import {APIGatewayClient,  CreateResourceCommand } from "@aws-sdk/client-api-
gateway";
(async function (){
const apig = new APIGatewayClient({region:"us-east-1"});
```

```
const command = new CreateResourceCommand({
    restApiId: 'abcd1234',
    parentId: 'efg567',
    pathPart: 'pets'
});
try {
    const results = await apig.send(command)
    console.log(results)
} catch (err) {
    console.log("The '/pets' resource setup failed:\n", err)
}
})();
```

成功的呼叫會傳回如下之輸出中的資源相關資訊：

```
{
    "path": "/pets",
    "pathPart": "pets",
    "id": "aaa111",
    "parentId": "efg567'"
}
```

Python

```
import botocore
import boto3
import logging

logger = logging.getLogger()
apig = boto3.client('apigateway')

try:
    result = apig.create_resource(
        restApiId='abcd1234',
        parentId='efg567',
        pathPart='pets'
    )
except botocore.exceptions.ClientError as error:
    logger.exception("The '/pets' resource setup failed: %s.", error)
    raise
attribute=["id","parentId", "pathPart", "path",]
filtered_result ={key:result[key] for key in attribute}
```

```
print(filtered_result)
```

成功的呼叫會傳回如下之輸出中的資源相關資訊：

```
{'id': 'aaa111', 'parentId': 'efg567', 'pathPart': 'pets', 'path': '/pets'}
```

AWS CLI

```
aws apigateway create-resource --rest-api-id abcd1234 \
  --region us-west-2 \
  --parent-id efg567 \
  --path-part pets
```

以下是此命令的輸出：

```
{
    "id": "aaa111",
    "parentId": "efg567",
    "pathPart": "pets",
    "path": "/pets"
}
```

您建立的 /pets 資源具有 aaa111 的資源 ID。您可以在 API 的設定中使用此值。

3. 接著，您會在 /pets 資源下附加子資源。此資源 /{petId} 具有 {petId} 的路徑參數。若要將路徑部分設為路徑參數，請將它括在一對大括號中 ({ })。使用下列範例為您的 API 建立 /pets/{petId} 資源：

   JavaScript v3

```
import {APIGatewayClient,  CreateResourceCommand } from "@aws-sdk/client-api-
gateway";
(async function (){
const apig = new APIGatewayClient({region:"us-east-1"});
const command = new CreateResourceCommand({
    restApiId: 'abcd1234',
    parentId: 'aaa111',
    pathPart: '{petId}'
});
try {
```

```
    const results = await apig.send(command)
    console.log(results)
} catch (err) {
    console.log("The '/pets/{petId}' resource setup failed:\n", err)
}
})();
```

成功的呼叫會傳回如下之輸出中的資源相關資訊：

```
{
    "path": "/pets/{petId}",
    "pathPart": "{petId}",
    "id": "bbb222",
    "parentId": "aaa111'"
}
```

Python

```
import botocore
import boto3
import logging

logger = logging.getLogger()
apig = boto3.client('apigateway')

try:
    result = apig.create_resource(
        restApiId='abcd1234',
        parentId='aaa111',
        pathPart='{petId}'
    )
except botocore.exceptions.ClientError as error:
    logger.exception("The '/pets/{petId}' resource setup failed: %s.", error)
    raise
attribute=["id","parentId", "pathPart", "path",]
filtered_result ={key:result[key] for key in attribute}
print(filtered_result)
```

成功的呼叫會傳回如下之輸出中的資源相關資訊：

```
{'id': 'bbb222', 'parentId': 'aaa111', 'pathPart': '{petId}', 'path': '/pets/
{petId}'}
```

AWS CLI

```
aws apigateway create-resource --rest-api-id abcd1234 \
  --region us-west-2 \
  --parent-id aaa111 \
  --path-part '{petId}'
```

以下是此命令的輸出：

```
{
    "id": "bbb222",
    "parentId": "aaa111",
    "path": "/pets/{petId}",
    "pathPart": "{petId}"
}
```

您建立的 /pets/{petId} 資源具有 bbb222 的資源 ID。您可以在 API 的設定中使用此值。

4. 在下列兩個步驟中，您將 HTTP 方法新增至您的資源。在本教學課程中，您將方法設定為具有開放存取權，方法是將 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 “存取控制”。

   使用下列範例在 /pets 資源上新增 GET HTTP 方法：

   JavaScript v3

```
import {APIGatewayClient,  PutMethodCommand } from "@aws-sdk/client-api-
gateway";
(async function (){
const apig = new APIGatewayClient({region:"us-east-1"});
const command = new PutMethodCommand({
    restApiId: 'abcd1234',
    resourceId: 'aaa111',
    httpMethod: 'GET',
    authorizationType: 'NONE'
});
```

```
try {
    const results = await apig.send(command)
    console.log(results)
} catch (err) {
    console.log("The 'GET /pets' method setup failed:\n", err)
}
})();
```

成功的呼叫會傳回如下的輸出：

```
{
    "apiKeyRequired": false,
    "httpMethod": "GET",
    "authorizationType": "NONE"
}
```

Python

```
import botocore
import boto3
import logging

logger = logging.getLogger()
apig = boto3.client('apigateway')

try:
    result = apig.put_method(
        restApiId='abcd1234',
        resourceId='aaa111',
        httpMethod='GET',
        authorizationType='NONE'
    )
except botocore.exceptions.ClientError as error:
    logger.exception("The 'GET /pets' method setup failed: %s", error)
    raise
attribute=["httpMethod","authorizationType","apiKeyRequired"]
filtered_result ={key:result[key] for key in attribute}
print(filtered_result)
```

成功的呼叫會傳回如下的輸出：

```
{'httpMethod': 'GET', 'authorizationType': 'NONE', 'apiKeyRequired': False}
```

AWS CLI

```
aws apigateway put-method --rest-api-id abcd1234 \
  --resource-id aaa111 \
  --http-method GET \
  --authorization-type "NONE" \
  --region us-west-2
```

以下是此命令的輸出：

```
{
    "httpMethod": "GET",
    "authorizationType": "NONE",
    "apiKeyRequired": false
}
```

5. 使用下列範例在 /pets/{petId} 資源上新增 GET HTTP 方法，並設定 requestParameters 屬性將用戶端提供的 petId 值傳遞至後端：

JavaScript v3

```
import {APIGatewayClient,  PutMethodCommand } from "@aws-sdk/client-api-
gateway";
(async function (){
const apig = new APIGatewayClient({region:"us-east-1"});
const command = new PutMethodCommand({
    restApiId: 'abcd1234',
    resourceId: 'bbb222',
    httpMethod: 'GET',
    authorizationType: 'NONE'
    requestParameters: {
        "method.request.path.petId" : true
    }
});
try {
    const results = await apig.send(command)
    console.log(results)
} catch (err) {
    console.log("The 'GET /pets/{petId}' method setup failed:\n", err)
```

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

成功的呼叫會傳回如下的輸出：

```
{
    "apiKeyRequired": false,
    "httpMethod": "GET",
    "authorizationType": "NONE",
    "requestParameters": {
       "method.request.path.petId": true
    }
}
```

Python

```
import botocore
import boto3
import logging

logger = logging.getLogger()
apig = boto3.client('apigateway')

try:
    result = apig.put_method(
        restApiId='abcd1234',
        resourceId='bbb222',
        httpMethod='GET',
        authorizationType='NONE',
        requestParameters={
            "method.request.path.petId": True
        }
    )
except botocore.exceptions.ClientError as error:
    logger.exception("The 'GET /pets/{petId}' method setup failed: %s", error)
    raise
attribute=["httpMethod","authorizationType","apiKeyRequired",
 "requestParameters" ]
filtered_result ={key:result[key] for key in attribute}
print(filtered_result)
```

成功的呼叫會傳回如下的輸出：

```
{'httpMethod': 'GET', 'authorizationType': 'NONE', 'apiKeyRequired': False,
 'requestParameters': {'method.request.path.petId': True}}
```

AWS CLI

```
aws apigateway put-method --rest-api-id abcd1234 \
  --resource-id bbb222 --http-method GET \
  --authorization-type "NONE" \
  --region us-west-2 \
  --request-parameters method.request.path.petId=true
```

以下是此命令的輸出：

```
{
    "httpMethod": "GET",
    "authorizationType": "NONE",
    "apiKeyRequired": false,
    "requestParameters": {
        "method.request.path.petId": true
    }
}
```

6. 針對 GET /pets 方法回應，使用下列範例新增 200 OK 方法：

JavaScript v3

```
import {APIGatewayClient,  PutMethodResponseCommand } from "@aws-sdk/client-api-
gateway";
(async function (){
const apig = new APIGatewayClient({region:"us-east-1"});
const command = new PutMethodResponseCommand({
    restApiId: 'abcd1234',
    resourceId: 'aaa111',
    httpMethod: 'GET',
    statusCode: '200'
});
try {
    const results = await apig.send(command)
    console.log(results)
} catch (err) {
```

```
    console.log("Set up the 200 OK response for the 'GET /pets' method failed:
\n", err)
}
})();
```

成功的呼叫會傳回如下的輸出：

```
{
    "statusCode": "200"
}
```

Python

```
import botocore
import boto3
import logging

logger = logging.getLogger()
apig = boto3.client('apigateway')

try:
    result = apig.put_method_response(
        restApiId='abcd1234',
        resourceId='aaa111',
        httpMethod='GET',
        statusCode='200'
    )
except botocore.exceptions.ClientError as error:
    logger.exception("Set up the 200 OK response for the 'GET /pets' method
 failed %s.", error)
    raise
attribute=["statusCode"]
filtered_result ={key:result[key] for key in attribute}
logger.info(filtered_result)
```

成功的呼叫會傳回如下的輸出：

```
{'statusCode': '200'}
```

AWS CLI

```
aws apigateway put-method-response --rest-api-id abcd1234 \
  --resource-id aaa111 --http-method GET \
  --status-code 200  --region us-west-2
```

以下是此命令的輸出：

```
{
    "statusCode": "200"
}
```

7. 針對 GET /pets/{petId} 方法回應，使用下列範例新增 200 OK 方法：

JavaScript v3

```
import {APIGatewayClient,  PutMethodResponseCommand } from "@aws-sdk/client-api-
gateway";
(async function (){
const apig = new APIGatewayClient({region:"us-east-1"});
const command = new PutMethodResponseCommand({
    restApiId: 'abcd1234',
    resourceId: 'bbb222',
    httpMethod: 'GET',
    statusCode: '200'
});
try {
    const results = await apig.send(command)
    console.log(results)
} catch (err) {
    console.log("Set up the 200 OK response for the 'GET /pets/{petId}' method
 failed:\n", err)
}
})();
```

成功的呼叫會傳回如下的輸出：

```
{
    "statusCode": "200"
}
```

Python

```
import botocore
import boto3
import logging

logger = logging.getLogger()
apig = boto3.client('apigateway')

try:
    result = apig.put_method_response(
        restApiId='abcd1234',
        resourceId='bbb222',
        httpMethod='GET',
        statusCode='200'
    )
except botocore.exceptions.ClientError as error:
    logger.exception("Set up the 200 OK response for the 'GET /pets/{petId}'
 method failed %s.", error)
    raise
attribute=["statusCode"]
filtered_result ={key:result[key] for key in attribute}
logger.info(filtered_result)
```

成功的呼叫會傳回如下的輸出：

```
{'statusCode': '200'}
```

AWS CLI

```
aws apigateway put-method-response --rest-api-id abcd1234 \
  --resource-id bbb222 --http-method GET \
  --status-code 200  --region us-west-2
```

以下是此命令的輸出：

```
{
    "statusCode": "200"
}
```

8. 使用下列範例設定 GET /pets 方法與 HTTP 端點的整合。HTTP 端點為 http://petstore-demo-endpoint.execute-api.com/petstore/pets。

JavaScript v3

```
import {APIGatewayClient,  PutIntegrationCommand } from "@aws-sdk/client-api-
gateway";
(async function (){
const apig = new APIGatewayClient({region:"us-east-1"});
const command = new PutIntegrationCommand({
    restApiId: 'abcd1234',
    resourceId: 'aaa111',
    httpMethod: 'GET',
    type: 'HTTP',
    integrationHttpMethod: 'GET',
    uri: 'http://petstore-demo-endpoint.execute-api.com/petstore/pets'
});
try {
    const results = await apig.send(command)
    console.log(results)
} catch (err) {
    console.log("Set up the integration of the 'GET /pets' method of the API
 failed:\n", err)
}
})();
```

成功的呼叫會傳回如下的輸出：

```
{
    "httpMethod": "GET",
    "passthroughBehavior": "WHEN_NO_MATCH",
    "cacheKeyParameters": [],
    "type": "HTTP",
    "uri": "http://petstore-demo-endpoint.execute-api.com/petstore/pets",
    "cacheNamespace": "ccc333"
}
```

Python

```
import botocore
import boto3
import logging
```

```
logger = logging.getLogger()
apig = boto3.client('apigateway')

try:
    result = apig.put_integration(
        restApiId='abcd1234',
        resourceId='aaa111',
        httpMethod='GET',
        type='HTTP',
        integrationHttpMethod='GET',
        uri='http://petstore-demo-endpoint.execute-api.com/petstore/pets'
    )
except botocore.exceptions.ClientError as error:
    logger.exception("Set up the integration of the 'GET /' method of the API
 failed %s.", error)
    raise
attribute=["httpMethod","passthroughBehavior","cacheKeyParameters", "type",
 "uri", "cacheNamespace"]
filtered_result ={key:result[key] for key in attribute}
print(filtered_result)
```

成功的呼叫會傳回如下的輸出：

```
{'httpMethod': 'GET', 'passthroughBehavior': 'WHEN_NO_MATCH',
 'cacheKeyParameters': [], 'type': 'HTTP', 'uri': 'http://petstore-demo-
endpoint.execute-api.com/petstore/pets', 'cacheNamespace': 'ccc333'}
```

AWS CLI

```
aws apigateway put-integration --rest-api-id abcd1234 \
  --resource-id aaa111 --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：

```
{
    "type": "HTTP",
    "httpMethod": "GET",
    "uri": "http://petstore-demo-endpoint.execute-api.com/petstore/pets",
```

```
    "connectionType": "INTERNET",
    "passthroughBehavior": "WHEN_NO_MATCH",
    "timeoutInMillis": 29000,
    "cacheNamespace": "6sxz2j",
    "cacheKeyParameters": []
}
```

9. 使用下列範例設定 GET /pets/{petId} 方法與 HTTP 端點的整合。HTTP 端點為 http://petstore-demo-endpoint.execute-api.com/petstore/pets/{id}。在此步驟中，您將 petId 路徑參數映射至 id 的端點路徑參數。

   JavaScript v3

```
import {APIGatewayClient,  PutIntegrationCommand } from "@aws-sdk/client-api-
gateway";
(async function (){
const apig = new APIGatewayClient({region:"us-east-1"});
const command = new PutIntegrationCommand({
    restApiId: 'abcd1234',
    resourceId: 'bbb222',
    httpMethod: 'GET',
    type: 'HTTP',
    integrationHttpMethod: 'GET',
    uri: 'http://petstore-demo-endpoint.execute-api.com/petstore/pets/{id}'
    requestParameters: {
        "integration.request.path.id": "method.request.path.petId"
     }
});
try {
    const results = await apig.send(command)
    console.log(results)
} catch (err) {
    console.log("Set up the integration of the 'GET /pets/{petId}' method of the
 API failed:\n", err)
}
})();
```

   成功的呼叫會傳回如下的輸出：

```
{
    "httpMethod": "GET",
    "passthroughBehavior": "WHEN_NO_MATCH",
```

```
    "cacheKeyParameters": [],
    "type": "HTTP",
    "uri": "http://petstore-demo-endpoint.execute-api.com/petstore/pets/{id}",
    "cacheNamespace": "ddd444",
    "requestParameters": {
       "integration.request.path.id": "method.request.path.petId"
    }
}
```

Python

```
import botocore
import boto3
import logging

logger = logging.getLogger()
apig = boto3.client('apigateway')

try:
    result = apig.put_integration(
        restApiId='ieps9b05sf',
        resourceId='t8zeb4',
        httpMethod='GET',
        type='HTTP',
        integrationHttpMethod='GET',
        uri='http://petstore-demo-endpoint.execute-api.com/petstore/pets/{id}',
        requestParameters={
            "integration.request.path.id": "method.request.path.petId"
        }
    )
except botocore.exceptions.ClientError as error:
    logger.exception("Set up the integration of the 'GET /pets/{petId}' method
 of the API failed %s.", error)
    raise
attribute=["httpMethod","passthroughBehavior","cacheKeyParameters", "type",
 "uri", "cacheNamespace", "requestParameters"]
filtered_result ={key:result[key] for key in attribute}
print(filtered_result)
```

成功的呼叫會傳回如下的輸出：

```
{'httpMethod': 'GET', 'passthroughBehavior': 'WHEN_NO_MATCH',
 'cacheKeyParameters': [], 'type': 'HTTP', 'uri': 'http://petstore-
demo-endpoint.execute-api.com/petstore/pets/{id}', 'cacheNamespace':
 'ddd444', 'requestParameters': {'integration.request.path.id':
 'method.request.path.petId'}}}
```

AWS CLI

```
aws apigateway put-integration --rest-api-id abcd1234 \
  --resource-id bbb222 --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
```

以下是此命令的輸出：

```
{
    "type": "HTTP",
    "httpMethod": "GET",
    "uri": "http://petstore-demo-endpoint.execute-api.com/petstore/pets/{id}",
    "connectionType": "INTERNET",
    "requestParameters": {
        "integration.request.path.id": "method.request.path.petId"
    },
    "passthroughBehavior": "WHEN_NO_MATCH",
    "timeoutInMillis": 29000,
    "cacheNamespace": "rjkmth",
    "cacheKeyParameters": []
}
```

10. 使用下列範例新增對於 GET /pets 整合的整合回應：

JavaScript v3

```
import {APIGatewayClient,  PutIntegrationResponseCommand } from "@aws-sdk/
client-api-gateway";
(async function (){
const apig = new APIGatewayClient({region:"us-east-1"});
const command = new PutIntegrationResponseCommand({
```

```
    restApiId: 'abcd1234',
    resourceId: 'aaa111',
    httpMethod: 'GET',
    statusCode: '200',
    selectionPattern: ''
});
try {
    const results = await apig.send(command)
    console.log(results)
} catch (err) {
    console.log("The 'GET /pets' method integration response setup failed:\n",
 err)
}
})();
```

成功的呼叫會傳回如下的輸出：

```
{
    "selectionPattern": "",
    "statusCode": "200"
}
```

Python

```
import botocore
import boto3
import logging

logger = logging.getLogger()
apig = boto3.client('apigateway')

try:
    result = apig.put_integration_response(
        restApiId='abcd1234',
        resourceId='aaa111',
        httpMethod='GET',
        statusCode='200',
        selectionPattern='',
    )
except botocore.exceptions.ClientError as error:
    logger.exception("Set up the integration response of the 'GET /pets' method
 of the API failed: %s", error)
```

```
    raise
attribute=["selectionPattern","statusCode"]
filtered_result ={key:result[key] for key in attribute}
print(filtered_result)
```

成功的呼叫會傳回如下的輸出：

```
{'selectionPattern': "", 'statusCode': '200'}
```

AWS CLI

```
aws apigateway put-integration-response --rest-api-id abcd1234 \
  --resource-id aaa111 --http-method GET \
  --status-code 200 --selection-pattern ""  \
  --region us-west-2
```

以下是此命令的輸出：

```
{
    "statusCode": "200",
    "selectionPattern": ""
}
```

11. 使用下列範例新增對於 GET /pets/{petId} 整合的整合回應：

JavaScript v3

```
import {APIGatewayClient,  PutIntegrationResponseCommand } from "@aws-sdk/
client-api-gateway";
(async function (){
const apig = new APIGatewayClient({region:"us-east-1"});
const command = new PutIntegrationResponseCommand({
    restApiId: 'abcd1234',
    resourceId: 'bbb222',
    httpMethod: 'GET',
    statusCode: '200',
    selectionPattern: ''
});
try {
    const results = await apig.send(command)
    console.log(results)
```

```
} catch (err) {
    console.log("The 'GET /pets/{petId}' method integration response setup
 failed:\n", err)
}
})();
```

成功的呼叫會傳回如下的輸出：

```
{
    "selectionPattern": "",
    "statusCode": "200"
}
```

Python

```
import botocore
import boto3
import logging

logger = logging.getLogger()
apig = boto3.client('apigateway')

try:
    result = apig.put_integration_response(
        restApiId='abcd1234',
        resourceId='bbb222',
        httpMethod='GET',
        statusCode='200',
        selectionPattern='',
    )
except botocore.exceptions.ClientError as error:
    logger.exception("Set up the integration response of the 'GET /pets/{petId}'
 method of the API failed: %s", error)
    raise
attribute=["selectionPattern","statusCode"]
filtered_result ={key:result[key] for key in attribute}
print(filtered_result)
```

成功的呼叫會傳回如下的輸出：

```
{'selectionPattern': "", 'statusCode': '200'}
```

AWS CLI

```
aws apigateway put-integration-response --rest-api-id abcd1234 \
  --resource-id bbb222 --http-method GET
  --status-code 200 --selection-pattern ""
  --region us-west-2
```

以下是此命令的輸出：

```
{
    "statusCode": "200",
    "selectionPattern": ""
}
```

建立整合回應後，您的 API 可以在 PetStore 網站上查詢可用的寵物，並檢視指定識別碼的個別寵物。在客戶呼叫您的 API 之前，您必須先部署 API。建議您在部署 API 之前先進行測試。

12. 使用下列範例測試 `GET /pets` 方法：

JavaScript v3

```
import {APIGatewayClient,  TestInvokeMethodCommand } from "@aws-sdk/client-api-
gateway";
(async function (){
const apig = new APIGatewayClient({region:"us-east-1"});
const command = new TestInvokeMethodCommand({
    restApiId: 'abcd1234',
    resourceId: 'aaa111',
    httpMethod: 'GET',
    pathWithQueryString: '/',
});
try {
    const results = await apig.send(command)
    console.log(results)
} catch (err) {
    console.log("The test on 'GET /pets' method failed:\n", err)
}
})();
```

Python

```
import botocore
import boto3
import logging

logger = logging.getLogger()
apig = boto3.client('apigateway')

try:
    result = apig.test_invoke_method(
        restApiId='abcd1234',
        resourceId='aaa111',
        httpMethod='GET',
        pathWithQueryString='/',
    )
except botocore.exceptions.ClientError as error:
    logger.exception("Test invoke method on 'GET /pets' failed: %s", error)
    raise
print(result)
```

AWS CLI

```
aws apigateway test-invoke-method --rest-api-id abcd1234 /
  --resource-id aaa111 /
  --http-method GET /
  --path-with-query-string '/'
```

13. 使用下列範例且 `petId` 為 3 來測試 `GET /pets/{petId}` 方法：

JavaScript v3

```
import {APIGatewayClient,  TestInvokeMethodCommand } from "@aws-sdk/client-api-
gateway";
(async function (){
const apig = new APIGatewayClient({region:"us-east-1"});
const command = new TestInvokeMethodCommand({
    restApiId: 'abcd1234',
    resourceId: 'bbb222',
    httpMethod: 'GET',
    pathWithQueryString: '/pets/3',
});
```

```
try {
    const results = await apig.send(command)
    console.log(results)
} catch (err) {
    console.log("The test on 'GET /pets/{petId}' method failed:\n", err)
}
})();
```

Python

```
import botocore
import boto3
import logging

logger = logging.getLogger()
apig = boto3.client('apigateway')

try:
    result = apig.test_invoke_method(
        restApiId='abcd1234',
        resourceId='bbb222',
        httpMethod='GET',
        pathWithQueryString='/pets/3',
    )
except botocore.exceptions.ClientError as error:
    logger.exception("Test invoke method on 'GET /pets/{petId}' failed: %s",
 error)
    raise
print(result)
```

AWS CLI

```
aws apigateway test-invoke-method --rest-api-id abcd1234 /
  --resource-id bbb222 /
  --http-method GET /
  --path-with-query-string '/pets/3'
```

成功測試 API 後，您可以將其部署至階段。

14. 使用下列範例將 API 部署至名為 `test` 的階段。當您將 API 部署至階段時，API 呼叫者就能調用您的 API。

JavaScript v3

```
import {APIGatewayClient,  CreateDeploymentCommand } from "@aws-sdk/client-api-
gateway";
(async function (){
const apig = new APIGatewayClient({region:"us-east-1"});
const command = new CreateDeploymentCommand({
    restApiId: 'abcd1234',
    stageName: 'test',
    stageDescription: 'test deployment'
});
try {
    const results = await apig.send(command)
    console.log("Deploying API succeeded\n", results)
} catch (err) {
    console.log("Deploying API failed:\n", err)
}
})();
```

Python

```
import botocore
import boto3
import logging

logger = logging.getLogger()
apig = boto3.client('apigateway')

try:
    result = apig.create_deployment(
        restApiId='ieps9b05sf',
        stageName='test',
        stageDescription='my test stage',
    )
except botocore.exceptions.ClientError as error:
    logger.exception("Error deploying stage  %s.", error)
    raise
print('Deploying API succeeded')
print(result)
```

AWS CLI

```
aws apigateway create-deployment --rest-api-id abcd1234 \
  --region us-west-2 \
  --stage-name test \
  --stage-description 'Test stage' \
  --description 'First deployment'
```

以下是此命令的輸出：

```
{
    "id": "ab1c1d",
    "description": "First deployment",
    "createdDate": "2022-12-15T08:44:13-08:00"
}
```

現在客戶可以呼叫您的 API。您可以在瀏覽器中輸入 `https://abcd1234.execute-api.us-west-2.amazonaws.com/test/pets` URL 來測試此 API，並以您 API 的識別碼替代 `abcd1234`。

如需如何使用 AWS SDKs或 建立或更新 API 的更多範例 AWS CLI，請參閱使用 AWS SDKs 的 API Gateway 動作。

## 自動化 API 的設定作業

您可以使用 OpenAPI 或 Terraform 來建立 API，以自動建立和清除 AWS 資源 CloudFormation，而不是step-by-step建立 API。

OpenAPI 3.0 定義

您可以將 OpenAPI 定義匯入 API Gateway。如需詳細資訊，請參閱the section called “OpenAPI”。

```
{
  "openapi" : "3.0.1",
  "info" : {
    "title" : "Simple PetStore (OpenAPI)",
    "description" : "Demo API created using OpenAPI",
    "version" : "2024-05-24T20:39:34Z"
  },
```

```
  "servers" : [ {
    "url" : "{basePath}",
    "variables" : {
      "basePath" : {
        "default" : "Prod"
      }
    }
  } ],
  "paths" : {
    "/pets" : {
      "get" : {
        "responses" : {
          "200" : {
            "description" : "200 response",
            "content" : { }
          }
        },
        "x-amazon-apigateway-integration" : {
          "type" : "http",
          "httpMethod" : "GET",
          "uri" : "http://petstore-demo-endpoint.execute-api.com/petstore/pets",
          "responses" : {
            "default" : {
              "statusCode" : "200"
            }
          },
          "passthroughBehavior" : "when_no_match",
          "timeoutInMillis" : 29000
        }
      }
    },
    "/pets/{petId}" : {
      "get" : {
        "parameters" : [ {
          "name" : "petId",
          "in" : "path",
          "required" : true,
          "schema" : {
            "type" : "string"
          }
        } ],
        "responses" : {
          "200" : {
            "description" : "200 response",
```

```
            "content" : { }
          }
        },
        "x-amazon-apigateway-integration" : {
          "type" : "http",
          "httpMethod" : "GET",
          "uri" : "http://petstore-demo-endpoint.execute-api.com/petstore/pets/{id}",
          "responses" : {
            "default" : {
              "statusCode" : "200"
            }
          },
          "requestParameters" : {
            "integration.request.path.id" : "method.request.path.petId"
          },
          "passthroughBehavior" : "when_no_match",
          "timeoutInMillis" : 29000
        }
      }
    }
  },
  "components" : { }
}
```

AWS CloudFormation 範本

若要部署 CloudFormation 範本，請參閱在 AWS CloudFormation 主控台上建立堆疊。

```
AWSTemplateFormatVersion: 2010-09-09
Resources:
  Api:
    Type: 'AWS::ApiGateway::RestApi'
    Properties:
      Name: Simple PetStore (AWS CloudFormation)
  PetsResource:
    Type: 'AWS::ApiGateway::Resource'
    Properties:
      RestApiId: !Ref Api
      ParentId: !GetAtt Api.RootResourceId
      PathPart: 'pets'
  PetIdResource:
    Type: 'AWS::ApiGateway::Resource'
    Properties:
      RestApiId: !Ref Api
```

```
      ParentId: !Ref PetsResource
      PathPart: '{petId}'
  PetsMethodGet:
    Type: 'AWS::ApiGateway::Method'
    Properties:
      RestApiId: !Ref Api
      ResourceId: !Ref PetsResource
      HttpMethod: GET
      AuthorizationType: NONE
      Integration:
        Type: HTTP
        IntegrationHttpMethod: GET
        Uri: http://petstore-demo-endpoint.execute-api.com/petstore/pets/
        IntegrationResponses:
          - StatusCode: '200'
      MethodResponses:
        - StatusCode: '200'
  PetIdMethodGet:
    Type: 'AWS::ApiGateway::Method'
    Properties:
      RestApiId: !Ref Api
      ResourceId: !Ref PetIdResource
      HttpMethod: GET
      AuthorizationType: NONE
      RequestParameters:
        method.request.path.petId: true
      Integration:
        Type: HTTP
        IntegrationHttpMethod: GET
        Uri: http://petstore-demo-endpoint.execute-api.com/petstore/pets/{id}
        RequestParameters:
          integration.request.path.id: method.request.path.petId
        IntegrationResponses:
          - StatusCode: '200'
      MethodResponses:
        - StatusCode: '200'
  ApiDeployment:
    Type: 'AWS::ApiGateway::Deployment'
    DependsOn:
      - PetsMethodGet
    Properties:
      RestApiId: !Ref Api
      StageName: Prod
Outputs:
```

```
  ApiRootUrl:
    Description: Root Url of the API
    Value: !Sub 'https://${Api}.execute-api.${AWS::Region}.amazonaws.com/Prod'
```

### Terraform 組態

如需 Terraform 的詳細資訊，請參閱 Terraform。

```
provider "aws" {
  region = "us-east-1" # Update with your desired region
}
resource "aws_api_gateway_rest_api" "Api" {
  name        = "Simple PetStore (Terraform)"
  description = "Demo API created using Terraform"
}
resource "aws_api_gateway_resource" "petsResource"{
    rest_api_id = aws_api_gateway_rest_api.Api.id
    parent_id = aws_api_gateway_rest_api.Api.root_resource_id
    path_part = "pets"
}
resource "aws_api_gateway_resource" "petIdResource"{
    rest_api_id = aws_api_gateway_rest_api.Api.id
    parent_id = aws_api_gateway_resource.petsResource.id
    path_part = "{petId}"
}
resource "aws_api_gateway_method" "petsMethodGet" {
  rest_api_id   = aws_api_gateway_rest_api.Api.id
  resource_id   = aws_api_gateway_resource.petsResource.id
  http_method   = "GET"
  authorization = "NONE"
}


resource "aws_api_gateway_method_response" "petsMethodResponseGet" {
    rest_api_id = aws_api_gateway_rest_api.Api.id
    resource_id = aws_api_gateway_resource.petsResource.id
    http_method = aws_api_gateway_method.petsMethodGet.http_method
    status_code ="200"
}

resource "aws_api_gateway_integration" "petsIntegration" {
  rest_api_id = aws_api_gateway_rest_api.Api.id
  resource_id = aws_api_gateway_resource.petsResource.id
  http_method = aws_api_gateway_method.petsMethodGet.http_method
```

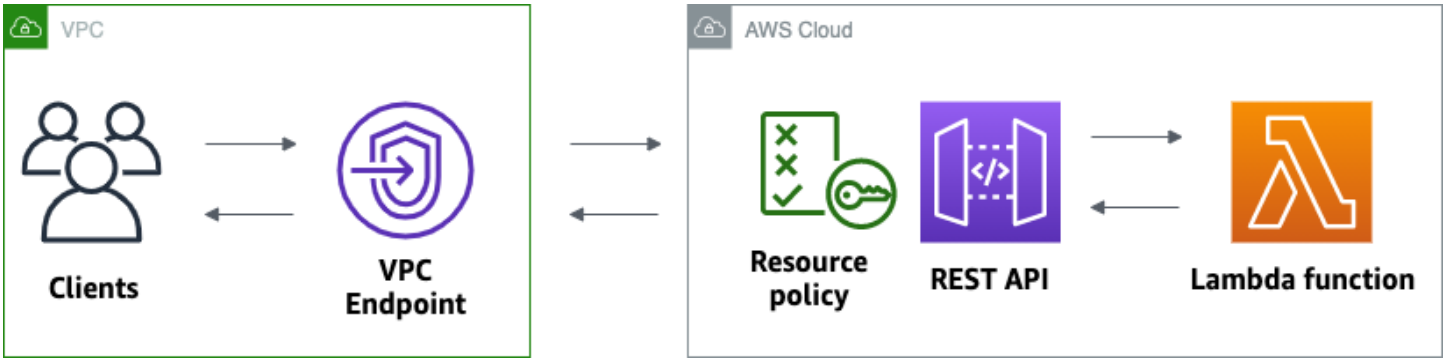
```
  type        = "HTTP"

  uri                     = "http://petstore-demo-endpoint.execute-api.com/petstore/
pets"
  integration_http_method = "GET"
  depends_on              = [aws_api_gateway_method.petsMethodGet]
}

resource "aws_api_gateway_integration_response" "petsIntegrationResponse" {
    rest_api_id = aws_api_gateway_rest_api.Api.id
    resource_id = aws_api_gateway_resource.petsResource.id
    http_method = aws_api_gateway_method.petsMethodGet.http_method
    status_code = aws_api_gateway_method_response.petsMethodResponseGet.status_code
}

resource "aws_api_gateway_method" "petIdMethodGet" {
    rest_api_id   = aws_api_gateway_rest_api.Api.id
    resource_id   = aws_api_gateway_resource.petIdResource.id
    http_method   = "GET"
    authorization = "NONE"
    request_parameters = {"method.request.path.petId" = true}
}

resource "aws_api_gateway_method_response" "petIdMethodResponseGet" {
    rest_api_id = aws_api_gateway_rest_api.Api.id
    resource_id = aws_api_gateway_resource.petIdResource.id
    http_method = aws_api_gateway_method.petIdMethodGet.http_method
    status_code ="200"
}


resource "aws_api_gateway_integration" "petIdIntegration" {
    rest_api_id = aws_api_gateway_rest_api.Api.id
    resource_id = aws_api_gateway_resource.petIdResource.id
    http_method = aws_api_gateway_method.petIdMethodGet.http_method
    type        = "HTTP"
    uri                     = "http://petstore-demo-endpoint.execute-api.com/petstore/
pets/{id}"
    integration_http_method = "GET"
    request_parameters = {"integration.request.path.id" = "method.request.path.petId"}
    depends_on              = [aws_api_gateway_method.petIdMethodGet]
}

resource "aws_api_gateway_integration_response" "petIdIntegrationResponse" {
```

```
    rest_api_id = aws_api_gateway_rest_api.Api.id
    resource_id = aws_api_gateway_resource.petIdResource.id
    http_method = aws_api_gateway_method.petIdMethodGet.http_method
    status_code = aws_api_gateway_method_response.petIdMethodResponseGet.status_code
}


resource "aws_api_gateway_deployment" "Deployment" {
  rest_api_id = aws_api_gateway_rest_api.Api.id
  depends_on  =
 [aws_api_gateway_integration.petsIntegration,aws_api_gateway_integration.petIdIntegration ]
}
resource "aws_api_gateway_stage" "Stage" {
  stage_name    = "Prod"
  rest_api_id   = aws_api_gateway_rest_api.Api.id
  deployment_id = aws_api_gateway_deployment.Deployment.id
}
```

## 教學：建立私有 REST API

在本教學課程中，您需建立私有 REST API。用戶端只能從您的 Amazon VPC 中存取 API。該 API 與公有網際網路隔離，這是一個常見安全要求。

本教學課程需要約 30 分鐘的時間完成。首先，您可以使用 CloudFormation 範本來建立 Amazon VPC、VPC 端點、 AWS Lambda 函數，並啟動您用來測試 API 的 Amazon EC2 執行個體。接著，您可以使用 AWS 管理主控台 建立私有 API 並連接資源政策，僅允許從您的 VPC 端點存取。最後，測試您的 API。




若要完成本教學課程，您需要 AWS 帳戶和具有主控台存取權 AWS Identity and Access Management 的使用者。如需詳細資訊，請參閱先決條件。

在本教學課程中，您需使用 AWS 管理主控台。如需建立此 API 和所有相關資源的 CloudFormation 範本，請參閱 template.yaml。

主題


- 步驟 1：建立相依性
- 步驟 2：建立私有 API
- 步驟 3：建立方法與整合
- 步驟 4：連接資源政策
- 步驟 5：部署 API
- 步驟 6：確認您的 API 無法公開存取
- 步驟 7：連接到 VPC 中的執行個體，並叫用您的 API
- 步驟 8：清理
- 後續步驟：使用 自動化 CloudFormation


## 步驟 1：建立相依性

下載並解壓縮此 CloudFormation 範本。您可以使用範本為私有 API 建立所有相依性，包括 Amazon VPC、VPC 端點，以及做為 API 後端的 Lambda 函數。您可以稍後建立私有 API。

建立 CloudFormation 堆疊

1. 在 https：//https://console.aws.amazon.com/cloudformation 開啟 CloudFormation 主控台。
2. 選擇 Create stack (建立堆疊)，然後選擇 With new resources (standard) (使用新資源 (標準))。
3. 對於 Specify template (指定範本)，選擇 Upload a template file (上傳範本檔案)。
4. 選取您下載的範本。
5. 選擇 Next (下一步)。
6. 針對 Stack name (堆疊名稱)，輸入 **`private-api-tutorial`**，然後選擇 Next (下一步)。
7. 針對 Configure stack options (設定堆疊選項)，選擇 Next (下一步)。
8. 針對 功能，確認 CloudFormation 可以在您的帳戶中建立 IAM 資源。
9. 選擇下一步，然後選擇提交。

CloudFormation 會為您的 API 佈建相依性，這可能需要幾分鐘的時間。當您 CloudFormation 堆疊的狀態為 CREATE_COMPLETE 時，請選擇輸出。請記下您的 VPC 端點 ID。您在本教程課程中的後續步驟需要它。

## 步驟 2：建立私有 API

您可以建立私有 API，並僅允許 VPC 中的用戶端存取它。

建立私有 API

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 選擇 Create API (建立 API)，然後針對 REST API，選擇 Build (建置)。
3. 針對 API name (API 名稱)，請輸入 **`private-api-tutorial`**。
4. 針對 API 端點類型，選取私有。
5. 針對 VPC IDs，輸入 CloudFormation 堆疊輸出的 VPC 端點 ID。
6. 對於 IP 位址類型，請選擇 Dualstack。
7. 選擇建立 API。

## 步驟 3：建立方法與整合

您可以建立 GET 方法和 Lambda 整合來處理對 API 的 GET 請求。當用戶端叫用您的 API 時，API Gateway 會將請求傳送至您在步驟 1 中建立的 Lambda 函數，然後傳回回應給用戶端。

建立方法與整合

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 選擇您的 API。
3. 選擇建立方法。
4. 針對方法類型，選取 GET。
5. 針對整合類型，選取 Lambda 函數。
6. 開啟 Lambda 代理整合。透過 Lambda 代理整合，API Gateway 會將事件傳送至具有定義結構的 Lambda，並將回應從您的 Lambda 函數轉換為 HTTP 回應。
7. 針對 Lambda 函數，選擇您在步驟 1 中使用 CloudFormation 範本建立的函數。函數的名稱以 **`private-api-tutorial`** 開頭。
8. 選擇建立方法。

## 步驟 4：連接資源政策

您可以將 resource policy (資源政策) 連接至 API，該 API 允許用戶端僅透過 VPC 端點叫用您的 API。若要進一步限制存取 API，您也可以為 VPC 端點設定 VPC 端點政策，但這對於本教學課程而言並不需要。

連接資源政策

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 選擇您的 API。
3. 選擇資源政策，然後選擇建立政策。
4. 輸入下列政策。將 *vpceID* 取代為 CloudFormation 堆疊輸出中的 VPC 端點 ID。

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Principal": "*",
            "Action": "execute-api:Invoke",
            "Resource": "execute-api:/*",
            "Condition": {
                "StringNotEquals": {
                    "aws:sourceVpce": "vpce-abcd1234"
                }
            }
        },
        {
            "Effect": "Allow",
            "Principal": "*",
            "Action": "execute-api:Invoke",
            "Resource": "execute-api:/*"
        }
    ]
}
```

5. 選擇儲存變更。

## 步驟 5：部署 API

接下來，您需部署 API，使其可供 Amazon VPC 中的用戶端使用。

部署 API

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 選擇您的 API。
3. 選擇部署 API。
4. 針對階段，選取新階段。
5. 針對階段名稱，輸入 **test**。
6. 在描述，請輸入描述。
7. 選擇部署。

現在您已準備好測試 API。

## 步驟 6：確認您的 API 無法公開存取

使用 `curl` 驗證您無法從 Amazon VPC 外部叫用 API。

測試您的 API

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 選擇您的 API。
3. 在主導覽窗格中，選擇階段，然後選擇測試階段。
4. 在階段詳細資訊下，選擇複製圖示以複製 API 的調用 URL。URL 看起來像 `https://abcdef123.execute-api.us-west-2.amazonaws.com/test`。您在步驟 1 中建立的 VPC 端點已啟用私有 DNS，因此您可以使用提供的 URL 來叫用 API。
5. 使用 curl 嘗試從 VPC 外部叫用您的 API。

   ```
   curl https://abcdef123.execute-api.us-west-2.amazonaws.com/test
   ```

   curl 表示無法解析您的 API 端點。如果您收到不同的回應，請返回步驟 2，並確定您為 API 的端點類型選擇 Private (私有)。

```
curl: (6) Could not resolve host: abcdef123.execute-api.us-west-2.amazonaws.com/
test
```

接下來，連接到 VPC 中的 Amazon EC2 執行個體以叫用您的 API。

## 步驟 7：連接到 VPC 中的執行個體，並叫用您的 API

接下來，從 Amazon VPC 中測試您的 API。若要存取您的私有 API，請連接到 VPC 中的 Amazon EC2 執行個體，然後使用 curl 來叫用您的 API。您可以使用 Systems Manager 工作階段管理員，在瀏覽器中連接到您的執行個體。

測試您的 API

1. 在 https://console.aws.amazon.com/ec2/ 開啟 Amazon EC2 主控台。
2. 選擇 Instances (執行個體)。
3. 選擇您在步驟 1 中使用 CloudFormation 範本建立的名為 private-api-tutorial 的執行個體。
4. 選擇 Connect (連線)，然後選擇 Session Manager (工作階段管理員)。
5. 選擇 Connect (連線) 以對您的執行個體啟動瀏覽器型工作階段。
6. 在您的工作階段管理員工作階段中，使用 curl 來叫用您的 API。您可以叫用 API，因為您正在 Amazon VPC 中使用執行個體。

   ```
   curl https://abcdef123.execute-api.us-west-2.amazonaws.com/test
   ```

   驗證您收到回應 `Hello from Lambda!`。

您成功建立了一個只能從 Amazon VPC 中存取的 API，然後驗證它可以運作。

### 步驟 8：清理

若要避免不必要的成本，請刪除您在此教學課程中建立的資源。下列步驟會刪除您的 REST API 和 CloudFormation 堆疊。

刪除 REST API

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 在 API 頁面上，選取 API。選擇 API 動作、選擇刪除 API，然後確認您的選擇。

刪除 CloudFormation 堆疊

1. 在 https：//https://console.aws.amazon.com/cloudformation 開啟 CloudFormation 主控台。
2. 選取您的 CloudFormation 堆疊。
3. 選擇刪除，然後確認您的選擇。

### 後續步驟：使用 自動化 CloudFormation

您可以自動化建立和清除本教學課程中涉及的所有 AWS 資源。如需完整的 CloudFormation 範本範例，請參閱 template.yaml。

# Amazon API Gateway HTTP API 教學課程

下列教學課程提供實際操作練習，協助您了解 API Gateway HTTP API。

主題

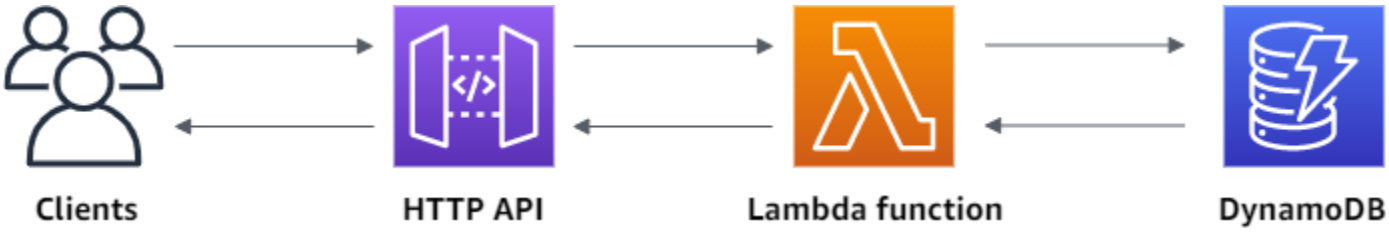
- 教學：使用 Lambda 和 DynamoDB 建立 CRUD HTTP API
- 教學：透過私有整合至 Amazon ECS 服務來建立 HTTP API

## 教學：使用 Lambda 和 DynamoDB 建立 CRUD HTTP API

在本教學課程中，您會建立無伺服器 API，以建立、讀取、更新和刪除 DynamoDB 資料表中的項目。DynamoDB 是全受管 NoSQL 資料庫服務，提供快速且可預期的效能，以及無縫的可擴展性。本教學課程大約需要 30 分鐘完成，您可以在 AWS 免費方案中完成。

首先，您可以使用 DynamoDB 主控台建立 DynamoDB 資料表。然後使用 AWS Lambda 主控台建立 Lambda 函數。接著，您可以使用 API Gateway 主控台建立 HTTP API。最後，測試您的 API。

在您叫用 HTTP API 時，API Gateway 會將請求路由至您的 Lambda 函數。Lambda 函數會與 DynamoDB 互動，並傳回 API Gateway 的回應。API Gateway 隨後將回應傳回給您。




若要完成此練習，您需要 AWS 帳戶和具有主控台存取權 AWS Identity and Access Management 的使用者。如需詳細資訊，請參閱先決條件。

在本教學課程中，您需使用 AWS 管理主控台。如需建立此 API 和所有相關資源的 AWS SAM 範本，請參閱 `template.yaml`。

主題

- 步驟 1：建立 DynamoDB 資料表
- 步驟 2：建立 Lambda 函數
- 步驟 3：建立 HTTP API
- 步驟 4：建立路由
- 步驟 5：建立整合
- 步驟 6：將整合連接至路由
- 步驟 7：測試您的 API
- 步驟 8：清理
- 後續步驟：使用 AWS SAM 或 自動化 CloudFormation

## 步驟 1：建立 DynamoDB 資料表

您可以使用 DynamoDB 資料表來儲存 API 的資料。

每個項目都有一個唯一的 ID，我們將其作為資料表的分割區索引鍵。

建立 DynamoDB 資料表

1. 請在 https://console.aws.amazon.com/dynamodb/ 開啟 DynamoDB 主控台。
2. 選擇 Create Table (建立資料表)。
3. 對於 Table name (資料表名稱)，請輸入 **`http-crud-tutorial-items`**。

4. 對於 Partition key (分割區索引鍵)，請輸入 **id**。
5. 選擇 Create Table (建立資料表)。

## 步驟 2：建立 Lambda 函數

您可以為您的 API 的後端建立 Lambda 函數。此 Lambda 函數會從 DynamoDB 建立、讀取、更新和刪除項目。此函數會使用來自 API Gateway 的事件來決定如何與 DynamoDB 互動。為了簡單起見，本教學課程使用了單一 Lambda 函數。最佳實務是，您應該為每個路由建立不同的函數。如需 Lambda 的詳細資訊，請參閱 Lambda Monolith。

建立 Lambda 函數

1. 在以下網址登入 Lambda 主控台：https://console.aws.amazon.com/lambda。
2. 選擇 Create function (建立函數)。
3. 針對 Function name (函數名稱)，請輸入 **http-crud-tutorial-function**。
4. 針對執行期，請選擇最新支援的 Node.js 或 Python 執行期。
5. 在 Permissions (許可) 下選擇 Change default execution role (變更預設執行角色)。
6. 選取從 AWS 政策範本建立新角色。
7. 針對 Role name (角色名稱)，請輸入 **http-crud-tutorial-role**。
8. 對於 Policy templates (政策範本)，請選擇 **Simple microservice permissions**。此政策會授予 Lambda 函數許可，以與 DynamoDB 互動。

> **Note**
>
> 本教學課程為了簡單起見，使用受管理政策。最佳實務是，您應建立自己的 IAM 政策以授予所需的最低許可。

9. 選擇 Create function (建立函數)。
10. 在主控台的程式碼編輯器中開啟 Lambda 函數，並用下列程式碼取代其內容。選擇 Deploy (部署) 以更新您的功能。

Node.js

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

```
  ScanCommand,
  PutCommand,
  GetCommand,
  DeleteCommand,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});

const dynamo = DynamoDBDocumentClient.from(client);

const tableName = "http-crud-tutorial-items";

export const handler = async (event, context) => {
  let body;
  let statusCode = 200;
  const headers = {
    "Content-Type": "application/json",
  };

  try {
    switch (event.routeKey) {
      case "DELETE /items/{id}":
        await dynamo.send(
          new DeleteCommand({
            TableName: tableName,
            Key: {
              id: event.pathParameters.id,
            },
          })
        );
        body = `Deleted item ${event.pathParameters.id}`;
        break;
      case "GET /items/{id}":
        body = await dynamo.send(
          new GetCommand({
            TableName: tableName,
            Key: {
              id: event.pathParameters.id,
            },
          })
        );
        body = body.Item;
        break;
      case "GET /items":
```

```
        body = await dynamo.send(
          new ScanCommand({ TableName: tableName })
        );
        body = body.Items;
        break;
      case "PUT /items":
        let requestJSON = JSON.parse(event.body);
        await dynamo.send(
          new PutCommand({
            TableName: tableName,
            Item: {
              id: requestJSON.id,
              price: requestJSON.price,
              name: requestJSON.name,
            },
          })
        );
        body = `Put item ${requestJSON.id}`;
        break;
      default:
        throw new Error(`Unsupported route: "${event.routeKey}"`);
    }
  } catch (err) {
    statusCode = 400;
    body = err.message;
  } finally {
    body = JSON.stringify(body);
  }

  return {
    statusCode,
    body,
    headers,
  };
};
```

## Python

```
import json
import boto3
from decimal import Decimal

client = boto3.client('dynamodb')
```

```
dynamodb = boto3.resource("dynamodb")
table = dynamodb.Table('http-crud-tutorial-items')
tableName = 'http-crud-tutorial-items'


def lambda_handler(event, context):
    print(event)
    body = {}
    statusCode = 200
    headers = {
        "Content-Type": "application/json"
    }

    try:
        if event['routeKey'] == "DELETE /items/{id}":
            table.delete_item(
                Key={'id': event['pathParameters']['id']})
            body = 'Deleted item ' + event['pathParameters']['id']
        elif event['routeKey'] == "GET /items/{id}":
            body = table.get_item(
                Key={'id': event['pathParameters']['id']})
            body = body["Item"]
            responseBody = [
                {'price': float(body['price']), 'id': body['id'], 'name':
 body['name']}]
            body = responseBody
        elif event['routeKey'] == "GET /items":
            body = table.scan()
            body = body["Items"]
            print("ITEMS----")
            print(body)
            responseBody = []
            for items in body:
                responseItems = [
                    {'price': float(items['price']), 'id': items['id'], 'name':
 items['name']}]
                responseBody.append(responseItems)
            body = responseBody
        elif event['routeKey'] == "PUT /items":
            requestJSON = json.loads(event['body'])
            table.put_item(
                Item={
                    'id': requestJSON['id'],
                    'price': Decimal(str(requestJSON['price'])),
```

```
                    'name': requestJSON['name']
                })
            body = 'Put item ' + requestJSON['id']
    except KeyError:
        statusCode = 400
        body = 'Unsupported route: ' + event['routeKey']
    body = json.dumps(body)
    res = {
        "statusCode": statusCode,
        "headers": {
            "Content-Type": "application/json"
        },
        "body": body
    }
    return res
```

## 步驟 3：建立 HTTP API

HTTP API 將針對您的 Lambda 函數提供 HTTP 端點。在此步驟中，您將建立空白 API。在下列步驟中，您可以設定路由和整合，以連接您的 API 和 Lambda 函數。

建立 HTTP API

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 選擇 Create API (建立 API)，然後針對 HTTP API，選擇 Build (建置)。
3. 針對 API name (API 名稱)，請輸入 **`http-crud-tutorial-api`**。
4. 針對 IP 位址類型，選擇 IPv4。
5. 選擇下一步。
6. 對於 Configure routes (設定路由)，請選擇 Next (下一步) 以略過建立路由。您可以稍後建立路由。
7. 檢閱 API Gateway 為您建立的階段，然後選擇 Next (下一步)。
8. 選擇 Create (建立)。

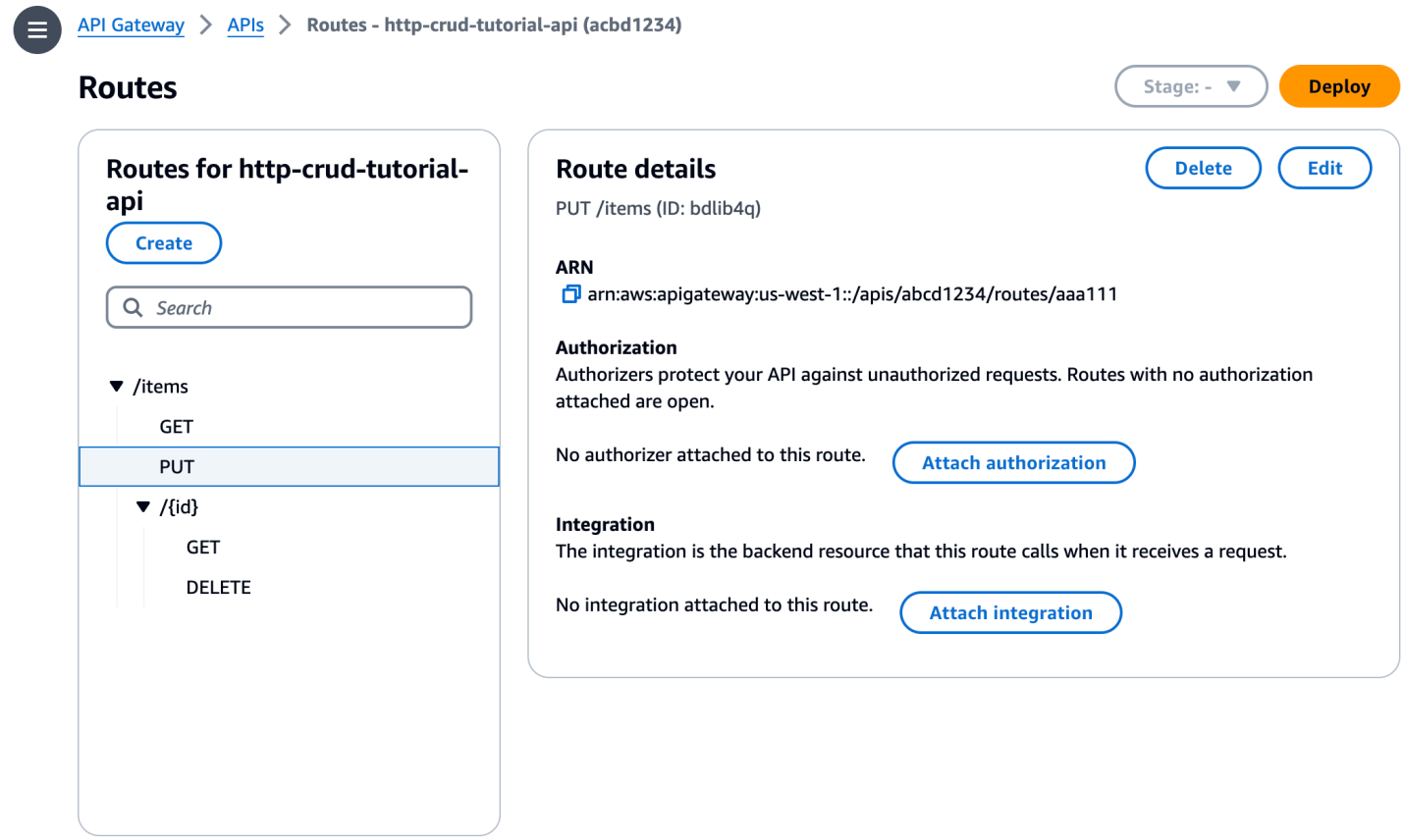
## 步驟 4：建立路由

路由是將傳入的 API 請求傳送到後端資源的一種方式。路由由兩部分組成：HTTP 方法和資源路徑，例如 `GET /items`。對於此示例 API，我們建立了四個路由：

- `GET /items/{id}`
- `GET /items`
- `PUT /items`
- `DELETE /items/{id}`

若要建立路由

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 選擇您的 API。
3. 選擇 Routes (路由)。
4. 選擇 Create (建立)。
5. 對於 Method (方法)，請選擇 **GET**。
6. 對於路徑，請輸入 **/items/{id}**。路徑結尾處 {id} 是一個路徑參數，是當用戶端發出請求時，API Gateway 從請求路徑中擷取的參數。
7. 選擇 Create (建立)。
8. 對 `GET /items`、`DELETE /items/{id}` 和 `PUT /items` 重複步驟 4-7。




## 步驟 5：建立整合

您可以建立整合以連接到後端資源的路由。在此範例 API 中，您可以建立一個用於所有路由的 Lambda 整合。

若要建立整合

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 選擇您的 API。
3. 選擇 Integrations (整合)。
4. 選擇 Manage integrations (管理整合)，然後選擇 Create (建立)。
5. 略過 Attach this integration to a route (將此整合連接到路由)。您可以在稍後的步驟中完成該操作。
6. 對於 Integration type (整合類型)，請選擇 Lambda function (Lambda 函數)。
7. 對於 Lambda function (Lambda 函數)，請輸入 **`http-crud-tutorial-function`**。
8. 選擇 Create (建立)。

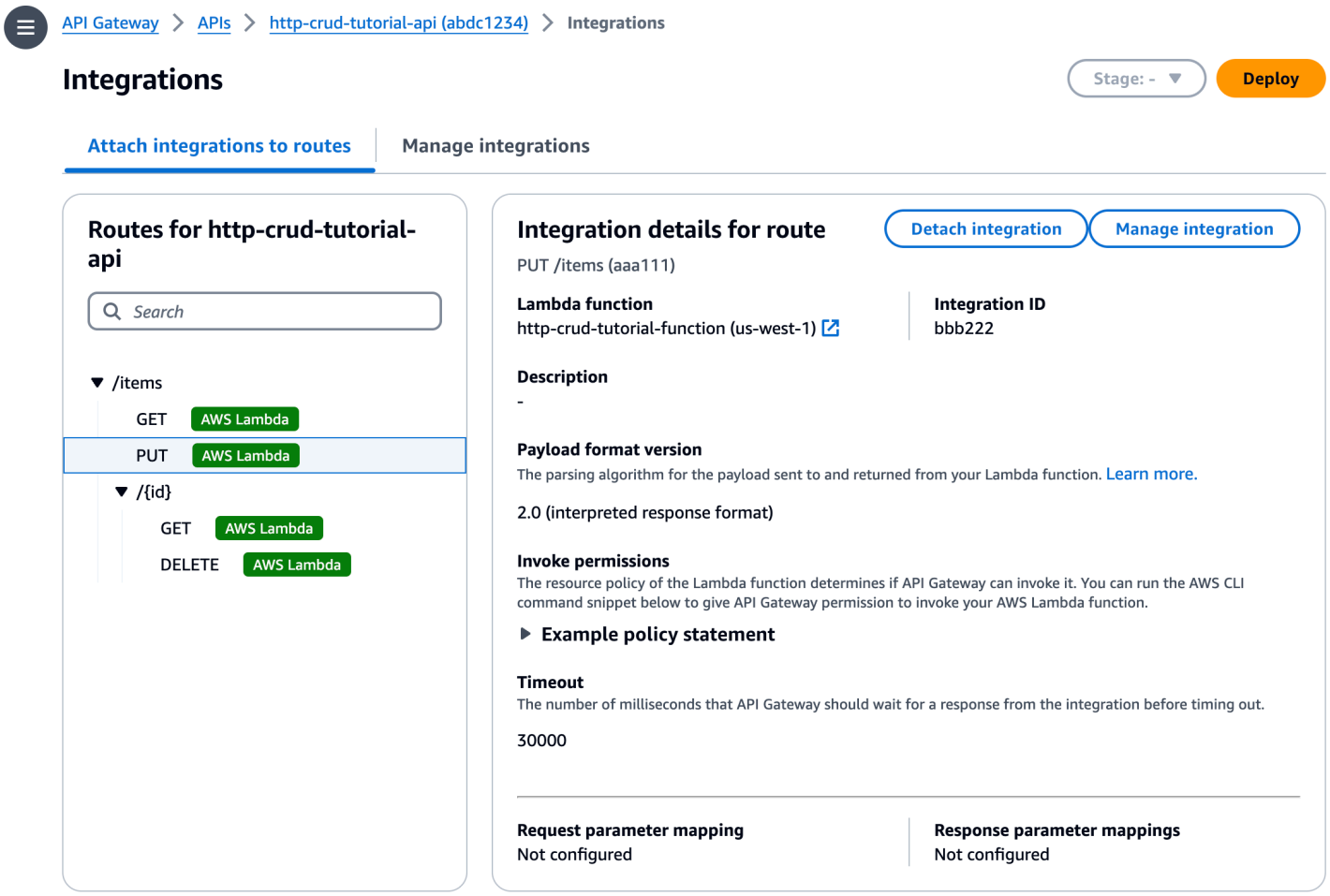
## 步驟 6：將整合連接至路由

在此範例 API 中，您可以對所有路由使用相同的 Lambda 整合。將整合連接至所有 API 的路由之後，當用戶端呼叫您的任何路由時，會叫用 Lambda 函數。

若要將整合連接至路由

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 選擇您的 API。
3. 選擇 Integrations (整合)。
4. 選擇路由。
5. 在 Choose an existing integration (選擇現有的整合) 下，請選擇 **`http-crud-tutorial-function`**。
6. 選擇 Attach integration (連接整合)。
7. 對所有路由重複步驟 4-6。

所有路由都顯示已連接 AWS Lambda 整合。



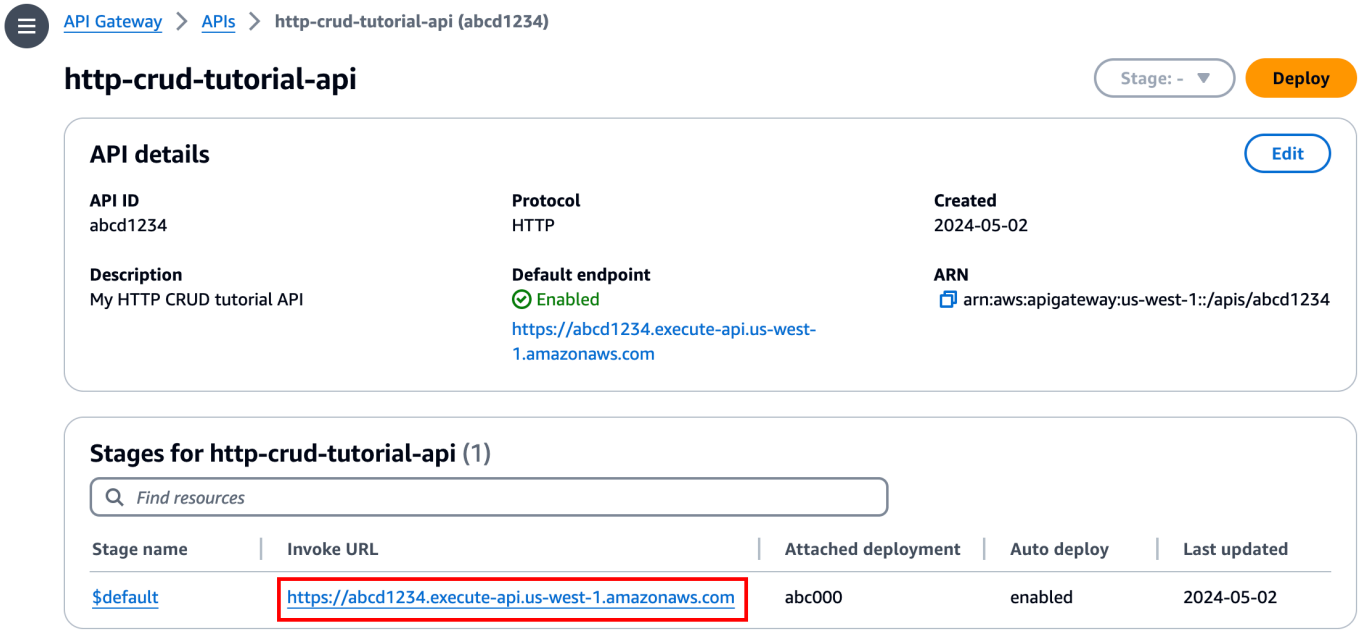

現在您擁有包含路由和整合的 HTTP API，您可以測試您的 API 了。

## 步驟 7：測試您的 API

為了確保您的 API 正常工作，您可以使用 curl。

要取得 URL 來叫用您的 API

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 選擇您的 API。
3. 請注意 API 的叫用 URL。它會出現在 Details (詳細資訊) 頁面上的 Invoke URL (叫用 URL) 下。




4. 複製您的 API 叫用 URL。

   完整的 URL 如 `https://abcdef123.execute-api.us-west-2.amazonaws.com`。

若要建立或更新項目

- 請使用下列命令來建立或更新項目。該命令包括具有項目 ID、價格和名稱的請求主體。

```
curl -X "PUT" -H "Content-Type: application/json" -d "{\"id\": \"123\",
 \"price\": 12345, \"name\": \"myitem\"}" https://abcdef123.execute-api.us-
west-2.amazonaws.com/items
```

若要取得所有項目

- 使用下列命令列出所有項目。

```
curl https://abcdef123.execute-api.us-west-2.amazonaws.com/items
```

若要取得項目

- 使用下列命令，透過其 ID 取得項目。

```
curl https://abcdef123.execute-api.us-west-2.amazonaws.com/items/123
```

刪除項目

1. 使用下列命令刪除項目。

```
curl -X "DELETE" https://abcdef123.execute-api.us-west-2.amazonaws.com/items/123
```

2. 取得所有項目，以確認該項目已刪除。

```
curl https://abcdef123.execute-api.us-west-2.amazonaws.com/items
```

## 步驟 8：清理

若要避免不必要的成本，請刪除您在此入門練習中建立的資源。採用下列步驟刪除您的 HTTP API、您的 Lambda 函式和相關資源。

刪除 DynamoDB 資料表

1. 請在 https://console.aws.amazon.com/dynamodb/ 開啟 DynamoDB 主控台。
2. 選取您的資料表。
3. 選擇 Delete table (刪除資料表)。
4. 確認您的選擇，然後選擇 Delete (刪除)。

刪除 HTTP API

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 在 API 頁面上，選取 API。選擇 Actions (動作)，然後選擇 Delete (刪除)。
3. 選擇 Delete (刪除)。

刪除 Lambda 函數

1. 在以下網址登入 Lambda 主控台：https://console.aws.amazon.com/lambda。
2. 在 Functions (函數) 頁面上，選取函數。選擇 Actions (動作)，然後選擇 Delete (刪除)。
3. 選擇 Delete (刪除)。

刪除 Lambda 函數的日誌群組

1. 在 Amazon CloudWatch 主控台中，開啟 Log groups (日誌群組) 頁面。
2. 在 Log groups (日誌群組) 頁面上，選取函數的日誌群組 (`/aws/lambda/http-crud-tutorial-function`)。選擇 Actions (動作)，然後選擇 Delete log group (刪除日誌群組)。
3. 選擇 Delete (刪除)。

刪除 Lambda 函數的執行角色

1. 在 AWS Identity and Access Management 主控台中，開啟角色頁面。
2. 選取函數的角色，例如，`http-crud-tutorial-role`。
3. 選擇 Delete role (刪除角色)。
4. 選擇 Yes, delete (是，刪除)。

### 後續步驟：使用 AWS SAM 或 自動化 CloudFormation

您可以使用 CloudFormation 或 自動化 AWS 資源的建立和清除 AWS SAM。如需本教學課程的 AWS SAM 範本範例，請參閱 `template.yaml`。

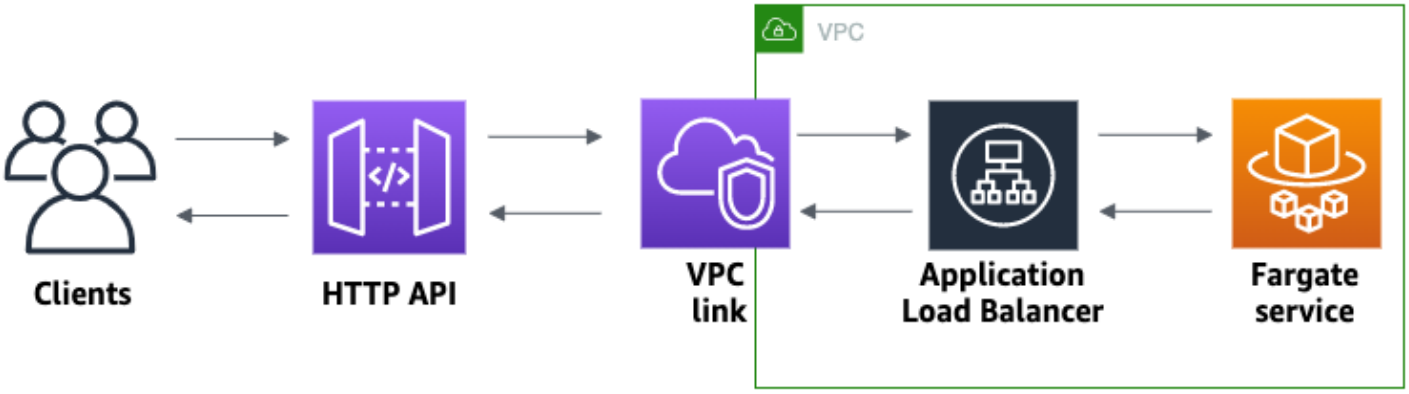
如需範例 CloudFormation 範本，請參閱範例 CloudFormation 範本。

## 教學：透過私有整合至 Amazon ECS 服務來建立 HTTP API

在本教學課程中，您會建立一個無伺服器 API，它會連接到在 Amazon VPC 中執行的 Amazon ECS 服務。不在您 Amazon VPC 上的客戶可以使用該 API 來存取您的 Amazon ECS 服務。

此教學課程需約 1 小時方能完成。首先，您使用 CloudFormation 範本來建立 Amazon VPC 和 Amazon ECS 服務。然後，您可以使用 API Gateway 主控台建立 VPC 連結。VPC 連結允許 API Gateway 存取在您 Amazon VPC 中執行的 Amazon ECS 服務。接下來，您需建立使用 VPC 連結連接到您 Amazon ECS 服務的 HTTP API。最後，測試您的 API。

當您叫用您的 HTTP API 時，API Gateway 會透過 VPC 連結將請求導向到您的 Amazon ECS 服務，然後傳回來自服務的回應。




若要完成本教學課程，您需要 AWS 帳戶和具有主控台存取權 AWS Identity and Access Management 的使用者。如需詳細資訊，請參閱先決條件。

在本教學課程中，您需使用 AWS 管理主控台。如需建立此 API 和所有相關資源的 CloudFormation 範本，請參閱 template.yaml。

主題

- 步驟 1：建立 Amazon ECS 服務
- 步驟 2：建立 VPC 連結
- 步驟 3：建立 HTTP API
- 步驟 4：建立路由
- 步驟 5：建立整合
- 步驟 6：測試您的 API
- 步驟 7：清理
- 後續步驟：使用 自動化 CloudFormation

## 步驟 1：建立 Amazon ECS 服務

Amazon ECS 是一種容器管理服務，可以在叢集上輕鬆執行、停止及管理 Docker 容器。在本教學課程中，您可以在 Amazon ECS 管理的無伺服器基礎設施上執行叢集。

下載並解壓縮此 CloudFormation 範本，這會建立服務的所有相依性，包括 Amazon VPC。您可以使用範本來建立使用 Application Load Balancer 的 Amazon ECS 服務。

建立 CloudFormation 堆疊

1. 在 https：//https://console.aws.amazon.com/cloudformation 開啟 CloudFormation 主控台。

2. 選擇 Create stack (建立堆疊)，然後選擇 With new resources (standard) (使用新資源 (標準))。
3. 對於 Specify template (指定範本)，選擇 Upload a template file (上傳範本檔案)。
4. 選取您下載的範本。
5. 選擇 Next (下一步)。
6. 針對 Stack name (堆疊名稱)，輸入 **`http-api-private-integrations-tutorial`**，然後選擇 Next (下一步)。
7. 針對 Configure stack options (設定堆疊選項)，選擇 Next (下一步)。
8. 對於 功能，確認 CloudFormation 可以在您的帳戶中建立 IAM 資源。
9. 選擇下一步，然後選擇提交。

CloudFormation 會佈建 ECS 服務，這可能需要幾分鐘的時間。當您的 CloudFormation 堆疊狀態為 CREATE_COMPLETE 時，您就可以繼續進行下一個步驟。

## 步驟 2：建立 VPC 連結

VPC 連結允許 API Gateway 存取 Amazon VPC 中的私有資源。您可以使用 VPC 連結允許用戶端透過 HTTP API 存取您的 Amazon ECS 服務。

建立 VPC 連結

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 在主要導覽窗格中，選擇 VPC 連結，然後選擇建立。

   您可能需要選擇選單圖示，才能開啟主要導覽窗格。
3. 針對  Choose a VPC link version (選擇 VPC 連結版本)，請選取 VPC link for HTTP APIs (HTTP API 的 VPC 連結)。
4. 針對 Name (名稱)，輸入 **`private-integrations-tutorial`**。
5. 在 VPC 欄位選擇您在步驟 1 建立的 VPC。名稱應該以 PrivateIntegrationsStack 開始。
6. 對於 Subnets (子網路)，請在您的 VPC 中選取兩個私有子網路。它們的名稱要以 `PrivateSubnet` 結尾。
7. 對於安全群組，選取以 `private-integrations-tutorial` 開頭且描述為 `PrivateIntegrationsStack/PrivateIntegrationsTutorialService/Service/SecurityGroup` 的群組 ID。
8. 選擇建立。

建立 VPC 連結後，API Gateway 會佈建彈性網路界面以存取您的 VPC。該程序需要幾分鐘的時間。同時，您可以創建您的 API。

## 步驟 3：建立 HTTP API

HTTP API 將針對您的 Amazon ECS 服務提供 HTTP 端點。在此步驟中，您將建立空白 API。在步驟 4 和 5 中，您可以設定路由和整合，以連接 API 和 Amazon ECS 服務。

建立 HTTP API

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 選擇 Create API (建立 API)，然後針對 HTTP API，選擇 Build (建置)。
3. 針對 API name (API 名稱)，請輸入 **`http-private-integrations-tutorial`**。
4. 針對 IP 位址類型，選擇 IPv4。
5. 選擇下一步。
6. 對於 Configure routes (設定路由)，請選擇 Next (下一步) 以略過建立路由。您可以稍後建立路由。
7. 檢閱 API Gateway 為您建立的階段。API Gateway 會建立啟用自動部署的 `$default` 階段，就本教學課程來說，這是最佳選擇。選擇 Next (下一步)。
8. 選擇 Create (建立)。

## 步驟 4：建立路由

路由是將傳入的 API 請求傳送到後端資源的一種方式。路由由兩部分組成：HTTP 方法和資源路徑，例如，`GET /items`。對於此示例 API，我們建立了 1 個路由。

建立路由

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 選擇您的 API。
3. 選擇 Routes (路由)。
4. 選擇 Create (建立)。
5. 對於 Method (方法)，請選擇 **`ANY`**。
6. 對於路徑，請輸入 **`/{proxy+}`**。路徑末尾的 `{proxy+}` 是一個 Greedy 路徑變數。API Gateway 會傳送所有對 API 的請求到此路由。

7. 選擇 Create (建立)。

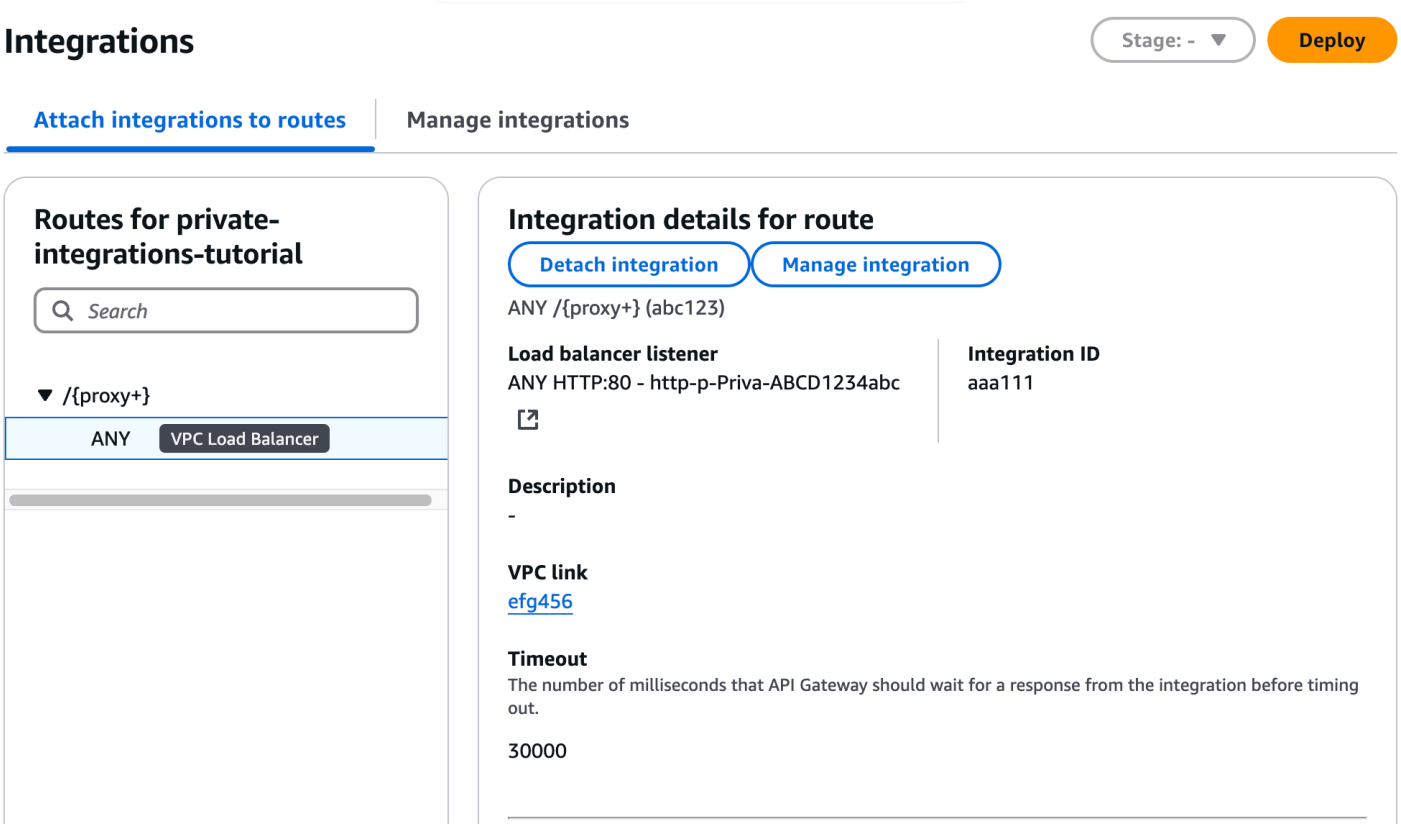
## 步驟 5：建立整合

您可以建立整合以連接到後端資源的路由。

若要建立整合

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 選擇您的 API。
3. 選擇 Integrations (整合)。
4. 選擇 Manage integrations (管理整合)，然後選擇 Create (建立)。
5. 對於 Attach this integration to a route (將此整合連接到路由)，請選取您先前建立的 ANY/{proxy+} 路由。
6. 對於 Integration type (整合類型)，請選擇 Private resoure (私有資源)。
7. 對於 Integration details (整合詳細資訊)，請選擇手動選取 (Select manually)。
8. 對於 Target service (目標服務)，請選擇 ALB/NLB。
9. 對於 Load balancer (負載平衡器)，請選擇您在步驟 1 中使用 CloudFormation 範本建立的負載平衡器。它的名字應該以 http-Priva 開始。
10. 對於 Listener (接聽程式)，請選擇 **`HTTP 80`**。
11. 對於 VPC link (VPC 連結)，請選擇您在步驟 2 中建立的 VPC 連結。它的名字應該為 `private-integrations-tutorial`。
12. 選擇 Create (建立)。

若要確認您的路由和整合已正確設定，請選取 Attach integration to routes (將整合連接至路由)。主控台顯示您擁有與 VPC 負載平衡器整合的 `ANY /{proxy+}` 路由。



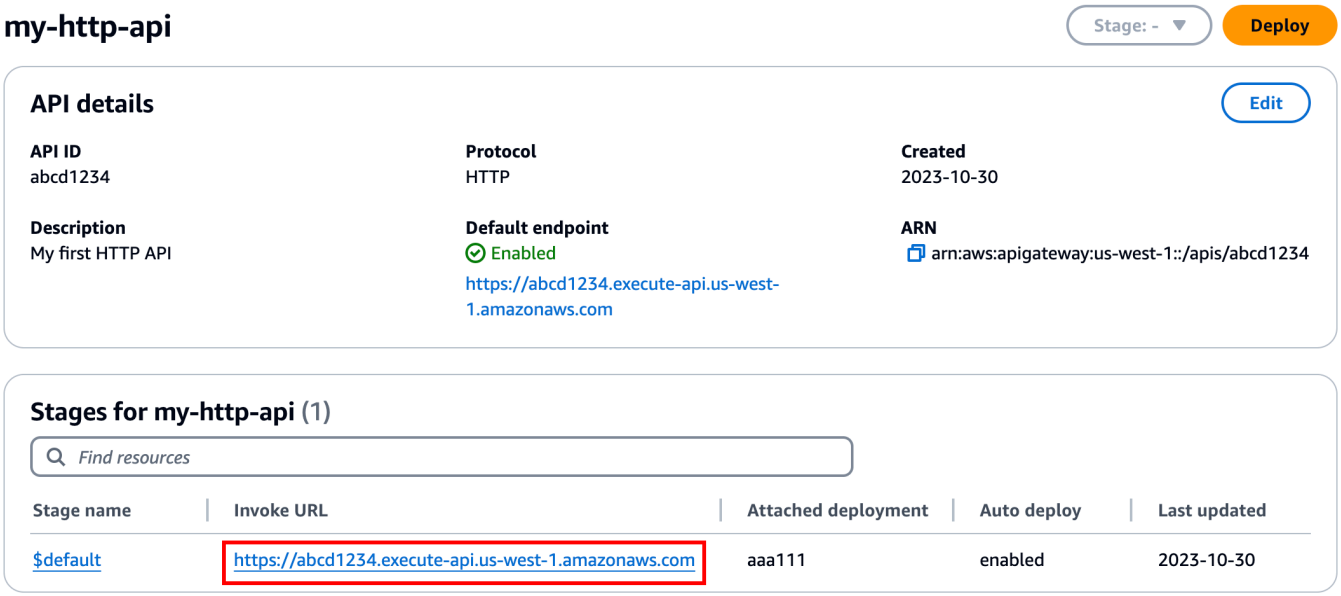

現在您已準備好測試 API。

## 步驟 6：測試您的 API

接著，您將測試您的 API 以確保其正常工作。為簡單起見，請使用 Web 瀏覽器來叫用您的 API。

測試您的 API

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 選擇您的 API。
3. 請注意 API 的叫用 URL。




4. 在 Web 瀏覽器中，請前往您的 API 叫用的 URL。

   完整 URL 看起來應該會像這樣：`https://abcdef123.execute-api.us-east-2.amazonaws.com`。

   您的瀏覽器將向 API 傳送 `GET` 請求。

5. 確認 API 的回應是一則歡迎訊息，告知您應用程式正在 Amazon ECS 上執行。

   如果您看到歡迎訊息，表示您已成功建立在 Amazon VPC 中執行的 Amazon ECS 服務，並使用具有 VPC 連結的 API Gateway HTTP API 來存取 Amazon ECS 服務。

## 步驟 7：清理

若要避免不必要的成本，請刪除您在此教學課程中建立的資源。下列步驟會刪除您的 VPC 連結、CloudFormation 堆疊和 HTTP API。

### 刪除 HTTP API

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 在 API 頁面上，選取 API。依次選擇 Actions (動作)、Delete (刪除)，然後確認您的選擇。

### 刪除 VPC 連結

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。

2. 選擇 VPC link (VPC 連結)。
3. 選取您的 VPC 連結、選擇 Delete (刪除)，然後確認您的選擇。

刪除 CloudFormation 堆疊

1. 在 https：//https://console.aws.amazon.com/cloudformation 開啟 CloudFormation 主控台。
2. 選取您的 CloudFormation 堆疊。
3. 選擇刪除，然後確認您的選擇。

### 後續步驟：使用 自動化 CloudFormation

您可以自動化建立和清除本教學課程中涉及的所有 AWS 資源。如需完整的 CloudFormation 範本範例，請參閱 template.yaml。

## Amazon API Gateway WebSocket API 教學課程

下列教學課程提供實作練習以協助您了解 API Gateway WebSocket API。

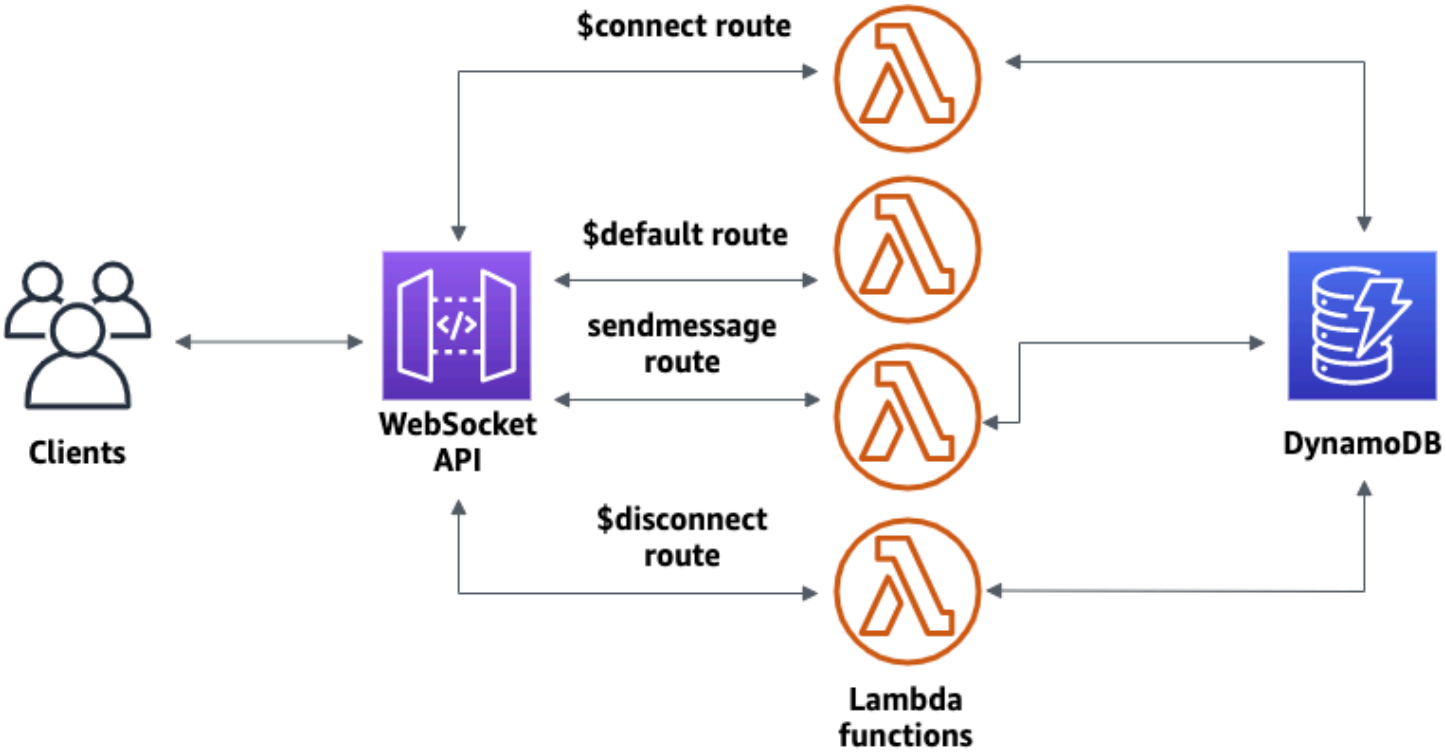
主題

- 教學：使用 WebSocket API、Lambda 和 DynamoDB 建立 WebSocket 聊天應用程式
- 教學課程：使用 AWS 整合建立 WebSocket API

## 教學：使用 WebSocket API、Lambda 和 DynamoDB 建立 WebSocket 聊天應用程式

在本教學課程中，您將建立 WebSocket API 的無伺服器聊天應用程式。使用 WebSocket API，您可以支持客户端之間的雙向通訊。客户端無需輪詢來進行更新即可以接收訊息。

本教學課程需要約 30 分鐘的時間完成。首先，您將使用 CloudFormation 範本來建立處理 API 請求的 Lambda 函數，以及存放用戶端 IDs DynamoDB 資料表。然後，您將使用 API Gateway 主控台來建立與 Lambda 函數整合的 WebSocket API。最後，您將測試 API 以確認是否能傳送和接收訊息。




若要完成本教學課程，您需要 AWS 帳戶和具有主控台存取權 AWS Identity and Access Management 的使用者。如需詳細資訊，請參閱先決條件。

您還需要 wscat 來連線到 API。如需詳細資訊，請參閱the section called “使用 wscat 以連接到 WebSocket API 和將訊息傳送到其中”。

主題

- 步驟 1：建立 Lambda 函數和 DynamoDB 資料表
- 步驟 2：建立 WebSocket API
- 步驟 3：測試您的 API
- 步驟 4：清理
- 後續步驟：使用 自動化 CloudFormation

## 步驟 1：建立 Lambda 函數和 DynamoDB 資料表

下載並解壓縮 的應用程式建立範本 CloudFormation。您將使用此範本建立 Amazon DynamoDB 資料表，以存放您應用程式的用户端 ID。每個連線的用户端都有一個唯一 ID，我們將使用它作為資料表的分區索引鍵。此範本也建立 Lambda 函數，其用於更新 DynamoDB 中的客户端連線並處理傳送給連線客户端的訊息。

建立 CloudFormation 堆疊

1. 在 https：//https://console.aws.amazon.com/cloudformation 開啟 CloudFormation 主控台。
2. 選擇 Create stack (建立堆疊)，然後選擇 With new resources (standard) (使用新資源 (標準))。
3. 對於 Specify template (指定範本)，選擇 Upload a template file (上傳範本檔案)。
4. 選取您下載的範本。
5. 選擇 Next (下一步)。
6. 針對 Stack name (堆疊名稱)，輸入 **`websocket-api-chat-app-tutorial`**，然後選擇 Next (下一步)。
7. 針對 Configure stack options (設定堆疊選項)，選擇 Next (下一步)。
8. 針對 功能，確認 CloudFormation 可以在您的帳戶中建立 IAM 資源。
9. 選擇下一步，然後選擇提交。

CloudFormation 會佈建範本中指定的資源。完成資源佈建可能需要幾分鐘的時間。當您的 CloudFormation 堆疊狀態為 CREATE_COMPLETE 時，您就可以繼續進行下一個步驟。

## 步驟 2：建立 WebSocket API

您將建立 WebSocket API 來處理客戶端連線，並將請求路由到您在步驟 1 中建立的 Lambda 函數。

若要建立 WebSocket API

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 選擇 Create API (建立 API)。然後對於 WebSocket API，選擇 Build (建置)。
3. 對於 API name (API 名稱)，輸入 **`websocket-chat-app-tutorial`**。
4. 針對 IP 位址類型，選擇 IPv4。
5. 對於 Route selection expression (路由選擇表達式)，輸入 **`request.body.action`**。路由選取表達式用於決定當用戶端傳送訊息時 API Gateway 要叫用的路由。
6. 選擇下一步。
7. 對於 Predefined routes (預先定義路由)，選擇 Add $connect (新增 $connect)、Add $disconnect (新增 $disconnect)，以及 Add $default (新增 $default)。$connect 和 $disconnect 路由是 API Gateway 在用户端連線到 API 或中斷連線時自動叫用的特殊路由。當沒有其他路由與請求相符時，API Gateway 會叫用 `$default` 路由。

8. 對於 Custom routes (自訂路由)，選擇 Add custom route (新增自訂路由)。對於 Route key (路由金鑰)，輸入 **sendmessage**。此自定義路由會處理傳送到已連線用户端的訊息。
9. 選擇下一步。
10. 在 Attach integrations (連接整合) 下，為每個路由和 Integration type (整合類型) 選擇 Lambda。

    針對 Lambda，選擇您在步驟 1 CloudFormation 中使用 建立的對應 Lambda 函數。每個函數的名稱會與一個路由相符。例如，對於 $connect 路由，選擇名為 **websocket-chat-app-tutorial-ConnectHandler** 的函數。
11. 檢閱 API Gateway 為您建立的階段。依預設，API Gateway 會建立階段名稱 `production`，並自動將您的 API 部署到該階段。選擇下一步。
12. 選擇 Create and deploy (建立和部署)。

## 步驟 3：測試您的 API

接著，您將測試您的 API 以確保其正常工作。若要連接至 API，請使用 `wscat` 命令。

取得 URL 來叫用您的 API

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 選擇您的 API。
3. 選擇 Stages (階段)，然後選擇 production (產品)。
4. 請注意 API 的 WebSocket URL。URL 看起來應該會像這樣：`wss://`*`abcdef123`*`.execute-api.`*`us-east-2`*`.amazonaws.com/production`。

連線到您的 API

1. 使用下列命令來連線到 API。當您連線到 API 時，API Gateway 會叫用 `$connect` 路由。當叫用此路由時，該路由會叫用可將連線 ID 存放在 DynamoDB 中的 Lambda 函數。

```
wscat -c wss://abcdef123.execute-api.us-west-2.amazonaws.com/production
```

```
Connected (press CTRL+C to quit)
```

2. 開啟新的終端機並使用下列參數再次執行 wscat 命令。

```
wscat -c wss://abcdef123.execute-api.us-west-2.amazonaws.com/production
```

```
Connected (press CTRL+C to quit)
```

這將為您提供兩個可交換訊息的已連線用户端。

傳送訊息

- API Gateway 根據 API 的路由選擇表達式確定要叫用的路由。您的 API 的路由選擇表達式是 `$request.body.action`。因此，當您傳送以下訊息時 API Gateway 會叫用 `sendmessage` 路由：

  ```
  {"action": "sendmessage", "message": "hello, everyone!"}
  ```

  與叫用路由相關聯的 Lambda 函數會從 DynamoDB 收集用户端 ID。然後，該函數會叫用 API Gateway Management API 並將訊息傳送到這些用户端。所有連線的用户端都會收到下列訊息：

  ```
  < hello, everyone!
  ```

叫用您 API 的 $default 路由

- 當用户端傳送與您定義路由不相符的訊息時，API Gateway 會叫用 API 的預設路由。與 `$default` 路由相關聯的 Lambda 函數會使用 API Gateway Management API 傳送其連線相關的用户端資訊。

  ```
  test
  ```

  ```
  Use the sendmessage route to send a message. Your info:
   {"ConnectedAt":"2022-01-25T18:50:04.673Z","Identity":
  {"SourceIp":"192.0.2.1","UserAgent":null},"LastActiveAt":"2022-01-25T18:50:07.642Z","connec
  ```

中斷與 API 的連線

- 若要中斷與 API 的連線，請按 **CTRL+C**。當用户端從 API 中斷連線時，API Gateway 會調用 API 的 `$disconnect` 路由。API 的 `$disconnect` 路由之 Lambda 整合會從 DynamoDB 移除連線 ID。

### 步驟 4：清理

若要避免不必要的成本，請刪除您在此教學課程中建立的資源。下列步驟會刪除您的 CloudFormation 堆疊和 WebSocket API。

刪除 WebSocket API

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 在 APIs (API) 頁面上，選取 `websocket-chat-app-tutorial` API。依次選擇 Actions (動作)、Delete (刪除)，然後確認您的選擇。

刪除 CloudFormation 堆疊

1. 在 https：//https://console.aws.amazon.com/cloudformation 開啟 CloudFormation 主控台。
2. 選取您的 CloudFormation 堆疊。
3. 選擇刪除，然後確認您的選擇。

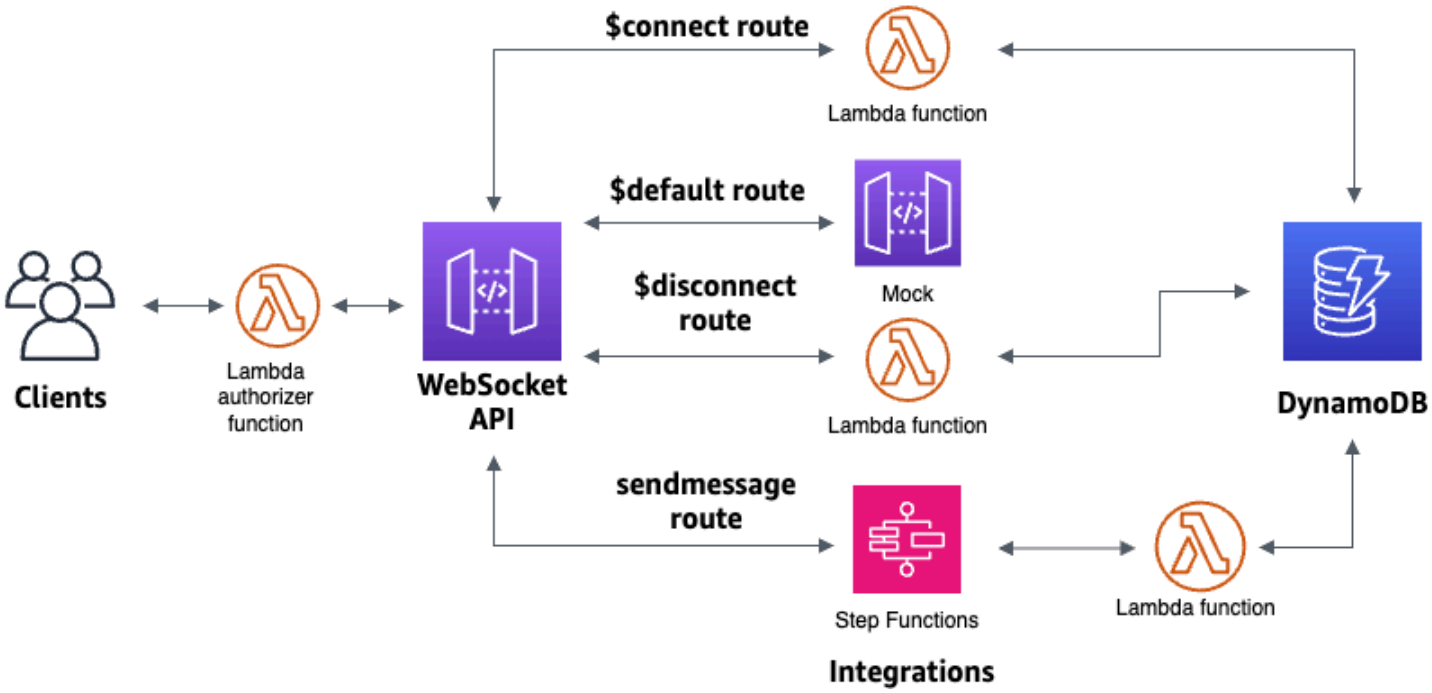
### 後續步驟：使用 自動化 CloudFormation

您可以自動化建立和清除本教學課程中涉及的所有 AWS 資源。如需建立此 API 和所有相關資源的 CloudFormation 範本，請參閱 ws-chat-app.yaml。

## 教學課程：使用 AWS 整合建立 WebSocket API

在本教學課程中，您將建立 WebSocket API 的無伺服器廣播應用程式。客戶端無需輪詢來進行更新即可以接收訊息。

本教學課程顯示如何將訊息廣播至已連線的用戶端，並包含 Lambda 授權方、模擬整合和非代理整合到 Step Functions 的範例。




使用 CloudFormation 範本建立資源後，您將使用 API Gateway 主控台來建立與 AWS 資源整合的 WebSocket API。您將連接 Lambda 授權方到您的 API，並與 Step Functions 建立 AWS 服務整合，以啟動狀態機器執行。Step Functions 狀態機器會調用 Lambda 函數，將訊息傳送至所有連線的用戶端。

建置 API 後，您會測試與 API 的連線，並確認已傳送和接收訊息。本教學課程需要約 45 分鐘的時間完成。

主題


- 先決條件
- 步驟 1：建立資源
- 步驟 2：建立 WebSocket API
- 步驟 3：建立 Lambda 授權方
- 步驟 4：建立模擬雙向整合
- 步驟 5：建立與 Step Functions 的非代理整合
- 步驟 6：測試您的 API
- 步驟 7：清除
- 後續步驟

## 先決條件

您需要以下的事前準備：

- 具有主控台存取權 AWS 的帳戶和 AWS Identity and Access Management 使用者。如需詳細資訊，請參閱先決條件。
- `wscat` 連線到您的 API。如需詳細資訊，請參閱the section called “使用 `wscat` 以連接到 WebSocket API 和將訊息傳送到其中”。

建議您在開始本教學課程之前，先完成 WebSocket 聊天應用程式教學課程。若要完成 WebSocket 聊天應用程式教學課程，請參閱 the section called “教學：建立 WebSocket 聊天應用程式”。

## 步驟 1：建立資源

下載並解壓縮 的應用程式建立範本 CloudFormation。您會使用此範本建立下列項目：

- 處理 API 請求並授權存取 API 的 Lambda 函數。
- DynamoDB 資料表，用於存放用戶端 ID 和 Lambda 授權方傳回的主要使用者身分。
- Step Functions 狀態機器，可傳送訊息至已連線的用戶端。

建立 CloudFormation 堆疊

1. 在 https：//https://console.aws.amazon.com/cloudformation 開啟 CloudFormation 主控台。
2. 選擇 Create stack (建立堆疊)，然後選擇 With new resources (standard) (使用新資源 (標準))。
3. 對於 Specify template (指定範本)，選擇 Upload a template file (上傳範本檔案)。
4. 選取您下載的範本。
5. 選擇 Next (下一步)。
6. 針對 Stack name (堆疊名稱)，輸入 **`websocket-step-functions-tutorial`**，然後選擇 Next (下一步)。
7. 針對 Configure stack options (設定堆疊選項)，選擇 Next (下一步)。
8. 針對 功能，確認 CloudFormation 可以在您的帳戶中建立 IAM 資源。
9. 選擇下一步，然後選擇提交。

CloudFormation 會佈建範本中指定的資源。完成資源佈建可能需要幾分鐘的時間。選擇輸出標籤，查看您建立的資源及其 ARNs。當您的 CloudFormation 堆疊狀態為 CREATE_COMPLETE 時，您就可以繼續進行下一個步驟。

## 步驟 2：建立 WebSocket API

您會建立 WebSocket API，用以處理用户端連線，並將請求路由到您在步驟 1 中建立的資源。

若要建立 WebSocket API

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 選擇 Create API (建立 API)。然後對於 WebSocket API，選擇 Build (建置)。
3. 對於 API name (API 名稱)，輸入 **`websocket-step-functions-tutorial`**。
4. 針對 IP 位址類型，選擇 IPv4。
5. 對於 Route selection expression (路由選擇表達式)，輸入 **`request.body.action`**。

   路由選取表達式用於決定當用戶端傳送訊息時 API Gateway 要叫用的路由。
6. 選擇下一步。
7. 對於預先定義路由，選擇新增 $connect、新增 $disconnect，以及新增 $default。

   $connect 和 $disconnect 路由是 API Gateway 在用户端連線到 API 或中斷連線時自動調用的特殊路由。當沒有其他路由與請求相符時，API Gateway 會調用 $default 路由。您在建立 API 後，會建立自訂路由以連線至 Step Functions。
8. 選擇下一步。
9. 對於 $connect 的整合，請執行下列操作：

   a. 對於 整合類型，選擇 Lambda。

   b. 針對 Lambda 函數，選擇您在步驟 1 CloudFormation 中使用 建立的對應 $connect Lambda 函數。Lambda 函數名稱應以 **`websocket-step`** 開頭。
10. 對於 $disconnect 的整合，請執行下列操作：

    a. 對於 整合類型，選擇 Lambda。

    b. 針對 Lambda 函數，選擇您在步驟 1 CloudFormation 中使用 建立的對應 $disconnect Lambda 函數。Lambda 函數名稱應以 **`websocket-step`** 開頭。
11. 對於 $default 的整合，選擇模擬。

    在模擬整合中，API Gateway 會在沒有整合後端的情況下管理路由回應。

12. 選擇下一步。
13. 檢閱 API Gateway 為您建立的階段。依預設，API Gateway 會建立名為 production 的階段，並自動將您的 API 部署到該階段。選擇下一步。
14. 選擇 Create and deploy (建立和部署)。

## 步驟 3：建立 Lambda 授權方

若要控制對 WebSocket API 的存取，則可建立 Lambda 授權方。 CloudFormation 範本會為您建立 Lambda 授權方函數。您會在 Lambda 主控台看到 Lambda 函數。名稱應以 **`websocket-step-functions-tutorial-AuthorizerHandler`** 開頭。除非 `Authorization` 標頭為 `Allow`，否則此 Lambda 函數會拒絕所有對 WebSocket API 的呼叫。Lambda 函數也會將 `$context.authorizer.principalId` 變數傳遞至您的 API，這稍後會在 DynamoDB 資料表中用於識別 API 呼叫者。

在此步驟中，您將 $connect 路由設定為使用 Lambda 授權方。

建立 Lambda 授權方

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 在主導覽窗格中，選擇授權方。
3. 選擇建立授權方。
4. 在授權方名稱中，輸入 **`LambdaAuthorizer`**。
5. 對於授權方 ARN，輸入 CloudFormation 範本建立的授權方名稱。名稱應以 **`websocket-step-functions-tutorial-AuthorizerHandler`** 開頭。

   > **Note**
   >
   > 建議您不要將此範例授權方用於生產 API。

6. 對於身分來源類型，選擇標頭。在 Key (索引鍵) 欄位，輸入 **`Authorization`**。
7. 選擇建立授權方。

建立授權方之後，將它連接到 API 的 $connect 路由。

將授權方連接至 $connect 路由

1. 在主要導覽窗格中，選擇路由。

2. 選擇 $connect 路由。
3. 在路由請求設定區段中，選擇編輯。
4. 對於授權，選擇下拉式選單，再選取您請求授權方。
5. 選擇儲存變更。

## 步驟 4：建立模擬雙向整合

接著，您可以為 $default 路由建立雙向模擬整合。模擬整合可讓您將回應傳送至用戶端，而不需使用後端。當您為 $default 路由建立整合時，您可以向用戶端顯示如何與 API 互動。

您將 $default 路由設定為通知用戶端使用 sendmessage 路由。

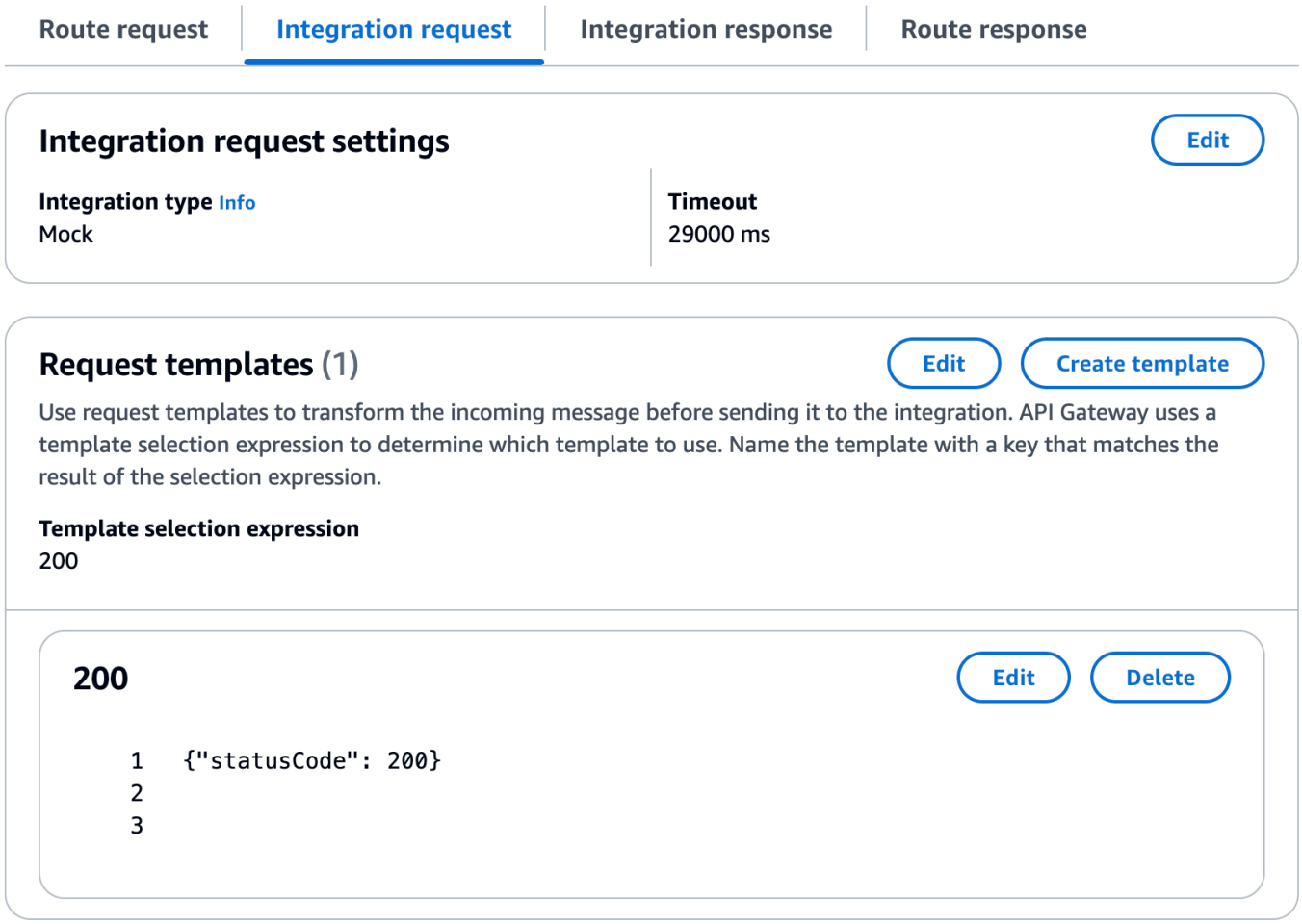
建立模擬整合

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 選擇 $default 路由，然後選擇整合請求標籤。
3. 對於請求範本，選擇編輯。
4. 對於範本選擇表達式，輸入 **200**，然後選擇編輯。
5. 在整合請求標籤上，針對請求範本選擇建立範本。
6. 對於範本金鑰，輸入 **200**。
7. 對於產生範本，輸入下列映射範本：

```
{"statusCode": 200}
```

選擇建立範本。

結果應如下列範例所示：

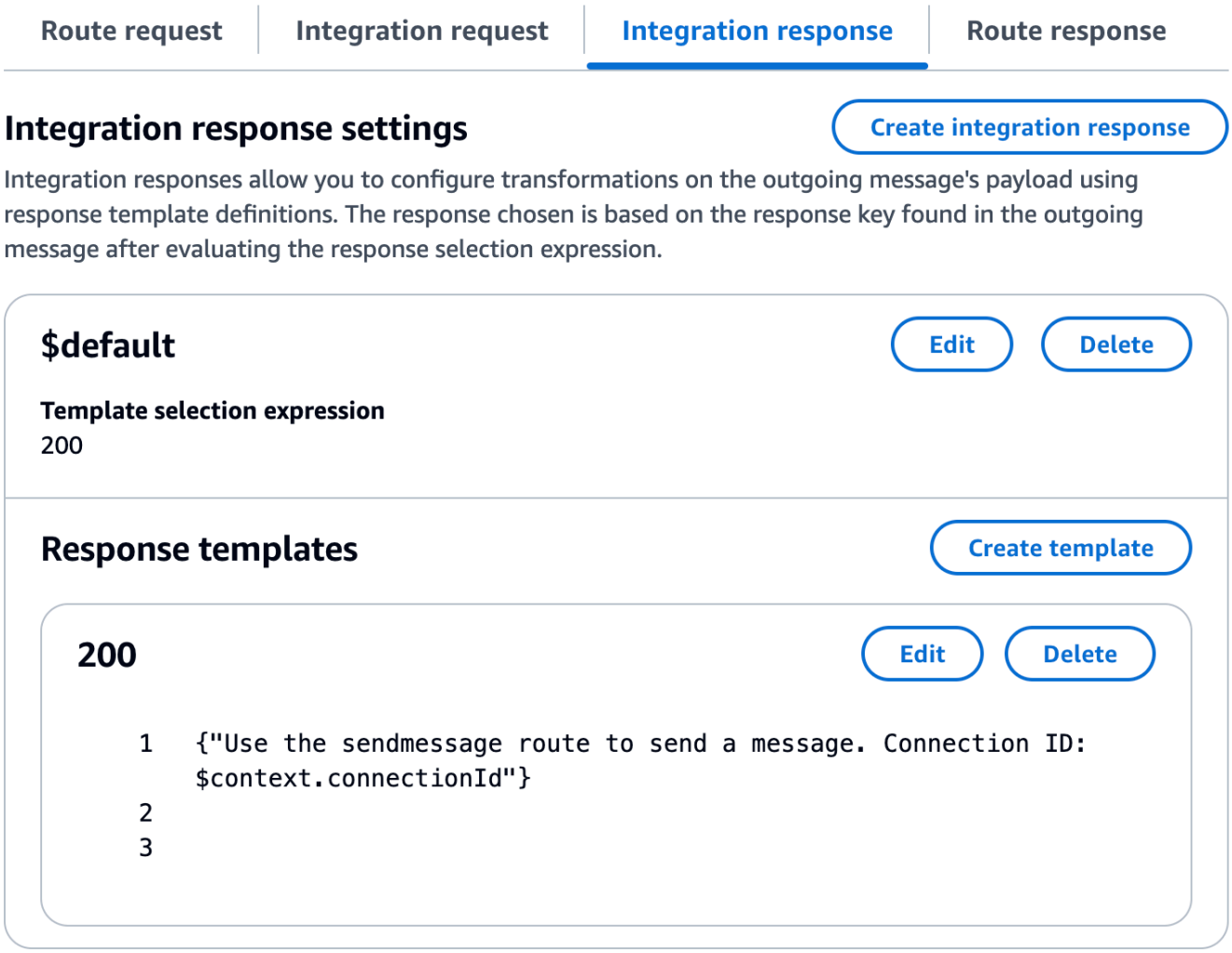



8. 對於 $default 路由窗格，選擇啟用雙向通訊。
9. 選擇整合回應標籤，然後選擇建立整合回應。
10. 在回應金鑰中，輸入 **`$default`**。
11. 對於範本選擇表達式，輸入 **`200`**。
12. 選擇建立回應。
13. 在回應範本下，選擇建立範本。
14. 對於範本金鑰，輸入 **`200`**。
15. 對於回應範本，輸入下列映射範本：

```
{"Use the sendmessage route to send a message. Connection ID:
 $context.connectionId"}
```

16. 選擇建立範本。

結果應如下列範例所示：




## 步驟 5：建立與 Step Functions 的非代理整合

接著，您會建立 sendmessage 路由。用戶端可以調用 sendmessage 路由，將訊息廣播到所有連線的用戶端。傳送訊息路由與 有非代理 AWS 服務整合 AWS Step Functions。整合會針對 CloudFormation 範本為您建立的 Step Functions 狀態機器叫用 StartExecution 命令。

建立非代理整合

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 選擇 Create route (建立路由)。
3. 對於 Route key (路由金鑰)，輸入 **`sendmessage`**。

4. 對於整合類型，選擇 AWS 服務。
5. 針對AWS 區域，輸入您部署 CloudFormation 範本的 區域。
6. 對於 AWS 服務，選擇 Step Functions。
7. 針對 HTTP 方法，請選擇 POST。
8. 針對動作名稱，輸入 **StartExecution**。
9. 針對執行角色，輸入 CloudFormation 範本建立的執行角色。名稱應該是 WebsocketTutorialApiRole。
10. 選擇 Create route (建立路由)。

接著，建立映射範本以將請求參數傳送至 Step Functions 狀態機器。

建立映射範本

1. 選擇 sendmessage 路由，然後選擇整合請求標籤。
2. 在請求範本區段中，選擇編輯。
3. 對於範本選擇表達式，輸入 **\$default**。
4. 選擇編輯。
5. 在請求範本區段中，選擇建立範本。
6. 對於範本金鑰，輸入 **\$default**。
7. 對於產生範本，輸入下列映射範本：

```
#set($domain = "$context.domainName")
#set($stage = "$context.stage")
#set($body = $input.json('$'))
#set($getMessage = $util.parseJson($body))
#set($mymessage = $getMessage.message)
{
"input": "{\"domain\": \"$domain\", \"stage\": \"$stage\", \"message\":
 \"$mymessage\"}",
"stateMachineArn": "arn:aws:states:us-east-2:123456789012:stateMachine:WebSocket-
Tutorial-StateMachine"
}
```

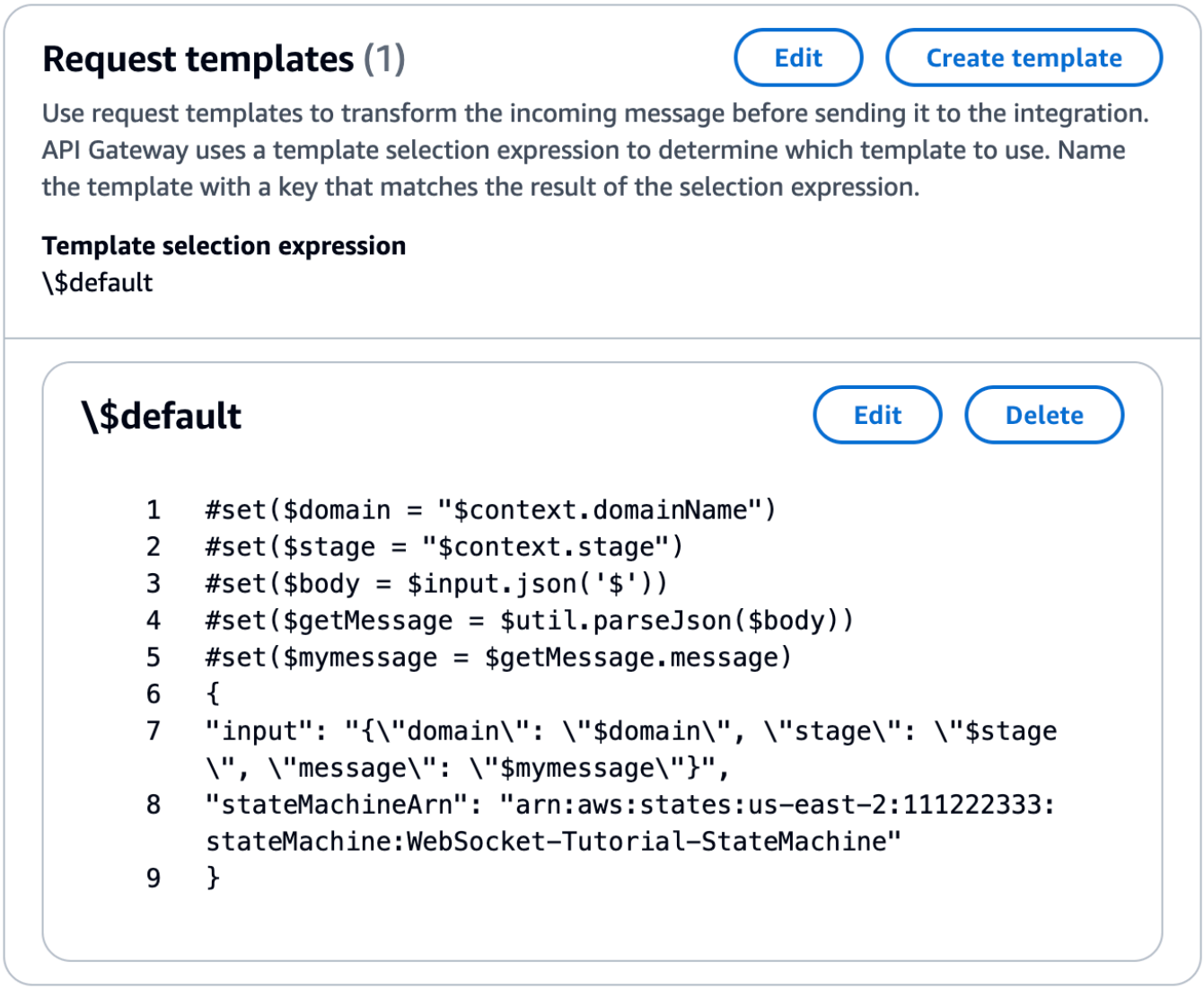
將 *stateMachineArn* 取代為 所建立狀態機器的 ARN CloudFormation。

映射範本可執行下列操作：

- 使用內容變數 domainName 建立變數 $domain。
- 使用內容變數 stage 建立變數 $stage。

  建置回電 URL 需要 $domain 和 $stage 變數。
- 接收傳入的 sendmessage JSON 訊息，然後擷取 message 屬性。
- 建立狀態機器的輸入。輸入是 WebSocket API 的網域和階段，以及來自 sendmessage 路由的訊息。

8. 選擇建立範本。




您可以在 $connect 或 $disconnect 路由上建立非代理整合，以直接從 DynamoDB 資料表新增或移除連線 ID，而無需調用 Lambda 函數。

## 步驟 6：測試您的 API

接著，您會部署和測試您的 API 以確保正常運作。您會使用 `wscat` 命令連線至 API，然後使用斜線命令傳送 ping 訊框來檢查與 WebSocket API 的連線。

部署 API

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 在主要導覽窗格中，選擇路由。
3. 選擇部署 API。
4. 針對階段，選擇生產。
5. (選用) 在部署說明中輸入說明。
6. 選擇部署。

部署 API 之後即可調用。使用調用 URL 呼叫您的 API。

針對您的 API 取得調用 URL

1. 選擇您的 API。
2. 選擇 Stages (階段)，然後選擇 production (產品)。
3. 請注意 API 的 WebSocket URL。URL 看起來應該會像這樣：`wss://abcdef123.execute-api.us-east-2.amazonaws.com/production`。

現在您已擁有調用 URL，即可測試與 WebSocket API 的連線。

測試與 API 的連線

1. 使用下列命令來連線到 API。首先，您可以調用 `/ping` 路徑以測試連線。

```
wscat -c wss://abcdef123.execute-api.us-east-2.amazonaws.com/production -H
 "Authorization: Allow" --slash -P
```

```
Connected (press CTRL+C to quit)
```

2. 輸入下列命令以 ping 控制器。您可以使用控制訊框以保持與用戶端的連線。

```
/ping
```

結果應如下列範例所示：

```
< Received pong (data: "")
```

現在您已測試連線，您可以測試 API 是否正常運作。在此步驟中，您會開啟新的終端機視窗，讓 WebSocket API 可以傳送訊息給所有連線的用戶端。

測試您的 API

1. 開啟新的終端機並使用下列參數再次執行 `wscat` 命令。

```
wscat -c wss://abcdef123.execute-api.us-east-2.amazonaws.com/production -H
 "Authorization: Allow"
```

```
Connected (press CTRL+C to quit)
```

2. API Gateway 會根據 API 的路由請求選擇表達式來確定要調用的路由。您的 API 路由選擇表達式為 `$request.body.action`。因此，當您傳送以下訊息時 API Gateway 會調用 `sendmessage` 路由：

```
{"action": "sendmessage", "message": "hello, from Step Functions!"}
```

與路由相關聯的 Step Functions 狀態機器，會使用訊息和回電 URL 調用 Lambda 函數。然後，Lambda 函數會呼叫 API Gateway Management API，並將訊息傳送到所有連線的用戶端。所有的用戶端都會收到下列訊息：

```
< hello, from Step Functions!
```

現在您已測試 WebSocket API，您可以中斷與 API 的連線。

中斷與 API 的連線

- 若要中斷與 API 的連線，請按 `CTRL+C`。

當用戶端從 API 中斷連線時，API Gateway 會調用 API 的 $disconnect 路由。API 的 $disconnect 路由之 Lambda 整合會從 DynamoDB 移除連線 ID。

## 步驟 7：清除

若要避免不必要的成本，請刪除您在此教學課程中建立的資源。下列步驟會刪除您的 CloudFormation 堆疊和 WebSocket API。

刪除 WebSocket API

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 在 API 頁面上，選取 websocket-api。
3. 依次選擇 Actions (動作)、Delete (刪除)，然後確認您的選擇。

刪除 CloudFormation 堆疊

1. 在 https：//https://console.aws.amazon.com/cloudformation 開啟 CloudFormation 主控台。
2. 選取您的 CloudFormation 堆疊。
3. 選擇刪除，然後確認您的選擇。

## 後續步驟

您可以自動化建立和清除本教學課程中涉及的所有 AWS 資源。如需本教學課程自動化這些動作的 CloudFormation 範本範例，請參閱 ws-sfn.zip。

# API Gateway REST API

API Gateway 中的 REST API 是與後端 HTTP 端點、Lambda 函數或其他 AWS 服務整合的資源和方法集合。您可以使用 API Gateway 功能來協助您處理 API 生命週期的各個層面，從建立到監控生產 API。

API Gateway REST API 會使用請求/回應模型，其中用戶端會傳送請求至服務，而服務會同步回應。這種模型適用於依賴同步通訊進行的許多不同種類的應用。

主題

- 在 APIs Gateway 中開發 REST API
- 發布 REST API 以供客戶調用
- 最佳化 REST API 的效能
- 將您的 REST API 分發至 API Gateway 中的用戶端
- 在 API Gateway 中保護您的 REST API
- 在 API Gateway 中監控 REST API

## 在 APIs Gateway 中開發 REST API

在 Amazon API Gateway 中，您可以將 REST API 建置為可程式化實體的集合，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 實體可以擁有一或多個 Method 資源。Method 是用戶端提交的傳入請求，可包含下列請求參數：路徑參數、標頭或查詢字串參數。此外，根據 HTTP 方法而定，請求可以包含內文。您的方法會定義用戶端存取公開 Resource 的方式。若要將 Method 與後端端點整合，也稱為整合端點，您可以建立整合資源。這會將傳入請求轉送到指定的整合端點 URI。如有必要，您可以轉換請求參數或請求內文以符合後端需求，也可以建立代理整合，其中 API Gateway 會以標準化格式將整個請求傳送至整合端點 URI，然後直接將回應傳送至用戶端。

針對回應，您可以建立 MethodResponse 資源來代表用戶端接收的回應，並建立 IntegrationResponse 資源來代表後端傳回的回應。請先使用整合回應來轉換後端回應資料，再將資料傳回至用戶端，或將後端回應依現狀傳遞到用戶端。

# REST API 的範例資源

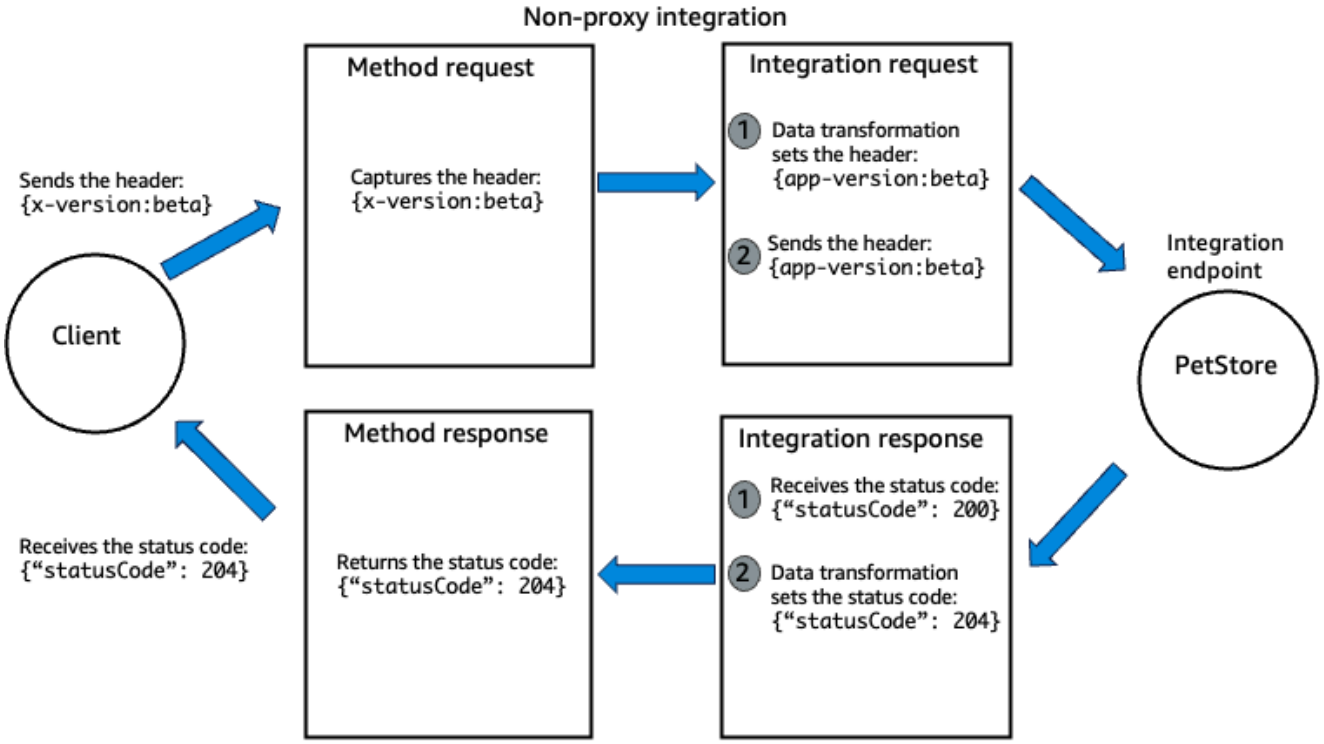
下圖顯示 API Gateway 如何針對 `GET /pets` 資源的 HTTP 代理和 HTTP 非代理整合，實作此請求/回應模型。用戶端會將 `x-version:beta` 標頭傳送至 API Gateway，而 API Gateway 會將 `204` 狀態碼傳送至用戶端。

在非代理整合中，API Gateway 會修改整合請求和整合回應來執行資料轉換，以滿足後端需求。在非代理整合中，您可以在方法請求中存取內文，但會在整合請求中進行轉換。當整合端點傳回具有內文的回應時，您可以在整合回應中存取和轉換內文。您無法在方法回應中修改內文。

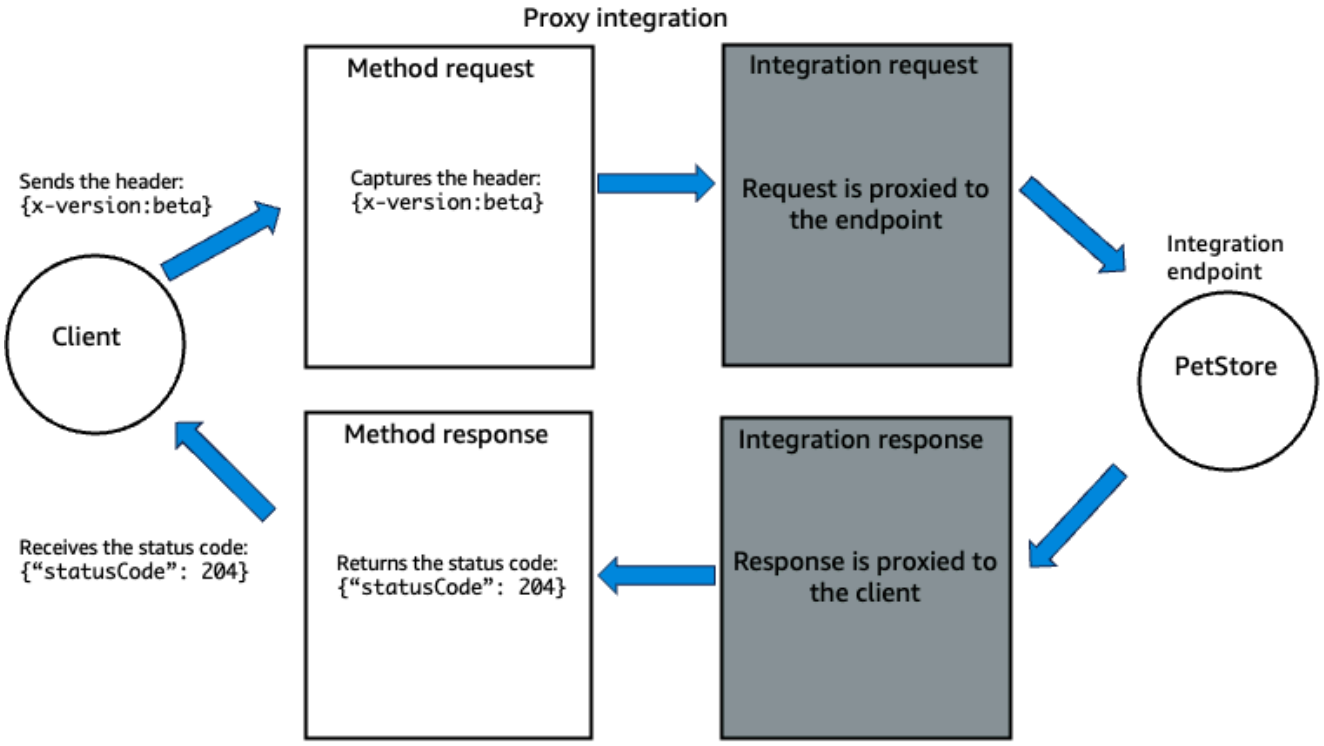
在代理整合中，整合端點會修改請求和回應。API Gateway 不會修改整合請求或整合回應，且會將傳入的請求依原樣傳送至後端。

無論整合類型為何，用戶端都會將請求傳送至 API Gateway，而 API Gateway 會同步回應。

Non-proxy integration



Proxy integration




下列範例執行日誌顯示 API Gateway 在上一個範例中記錄的內容。部分值和初始日誌已移除，以便清楚呈現內容：

Non-proxy integration

```
Wed Feb 12 23:56:44 UTC 2025 : Starting execution for request: abcd-1234-5678
Wed Feb 12 23:56:44 UTC 2025 : HTTP Method: GET, Resource Path: /pets
Wed Feb 12 23:56:44 UTC 2025 : Method request path: {}
Wed Feb 12 23:56:44 UTC 2025 : Method request query string: {}
Wed Feb 12 23:56:44 UTC 2025 : Method request headers: {x-version=beta}
Wed Feb 12 23:56:44 UTC 2025 : Method request body before transformations:
Wed Feb 12 23:56:44 UTC 2025 : Endpoint request URI: http://petstore-demo-
endpoint.execute-api.com/petstore/pets
Wed Feb 12 23:56:44 UTC 2025 : Endpoint request headers: {app-version=beta}
Wed Feb 12 23:56:44 UTC 2025 : Endpoint request body after transformations:
Wed Feb 12 23:56:44 UTC 2025 : Sending request to http://petstore-demo-
endpoint.execute-api.com/petstore/pets
Wed Feb 12 23:56:45 UTC 2025 : Received response. Status: 200, Integration latency:
 123 ms
```

```
Wed Feb 12 23:56:45 UTC 2025 : Endpoint response headers: {Date=Wed, 12 Feb 2025
 23:56:45 GMT}
Wed Feb 12 23:56:45 UTC 2025 : Endpoint response body before transformations:
Wed Feb 12 23:56:45 UTC 2025 : Method response body after transformations: (null)
Wed Feb 12 23:56:45 UTC 2025 : Method response headers: {X-Amzn-Trace-Id=Root=1-
abcd-12345}
Wed Feb 12 23:56:45 UTC 2025 : Successfully completed execution
Wed Feb 12 23:56:45 UTC 2025 : Method completed with status: 204
```

Proxy integration

```
Wed Feb 12 23:59:42 UTC 2025 : Starting execution for request: abcd-1234-5678
Wed Feb 12 23:59:42 UTC 2025 : HTTP Method: GET, Resource Path: /pets
Wed Feb 12 23:59:42 UTC 2025 : Method request path: {}
Wed Feb 12 23:59:42 UTC 2025 : Method request query string: {}
Wed Feb 12 23:59:42 UTC 2025 : Method request headers: {x-version=beta}
Wed Feb 12 23:59:42 UTC 2025 : Method request body before transformations:
Wed Feb 12 23:59:42 UTC 2025 : Endpoint request URI: http://petstore-demo-
endpoint.execute-api.com/petstore/pets
Wed Feb 12 23:59:42 UTC 2025 : Endpoint request headers: { x-version=beta}
Wed Feb 12 23:59:42 UTC 2025 : Endpoint request body after transformations:
Wed Feb 12 23:59:42 UTC 2025 : Sending request to http://petstore-demo-
endpoint.execute-api.com/petstore/pets
Wed Feb 12 23:59:43 UTC 2025 : Received response. Status: 204, Integration latency:
 123 ms
Wed Feb 12 23:59:43 UTC 2025 : Endpoint response headers: {Date=Wed, 12 Feb 2025
 23:59:43 GMT}
Wed Feb 12 23:59:43 UTC 2025 : Endpoint response body before transformations:
Wed Feb 12 23:59:43 UTC 2025 : Method response body after transformations:
Wed Feb 12 23:59:43 UTC 2025 : Method response headers: {Date=Wed, 12 Feb 2025
 23:59:43 GMT}
Wed Feb 12 23:59:43 UTC 2025 : Successfully completed execution
Wed Feb 12 23:59:43 UTC 2025 : Method completed with status: 204
```

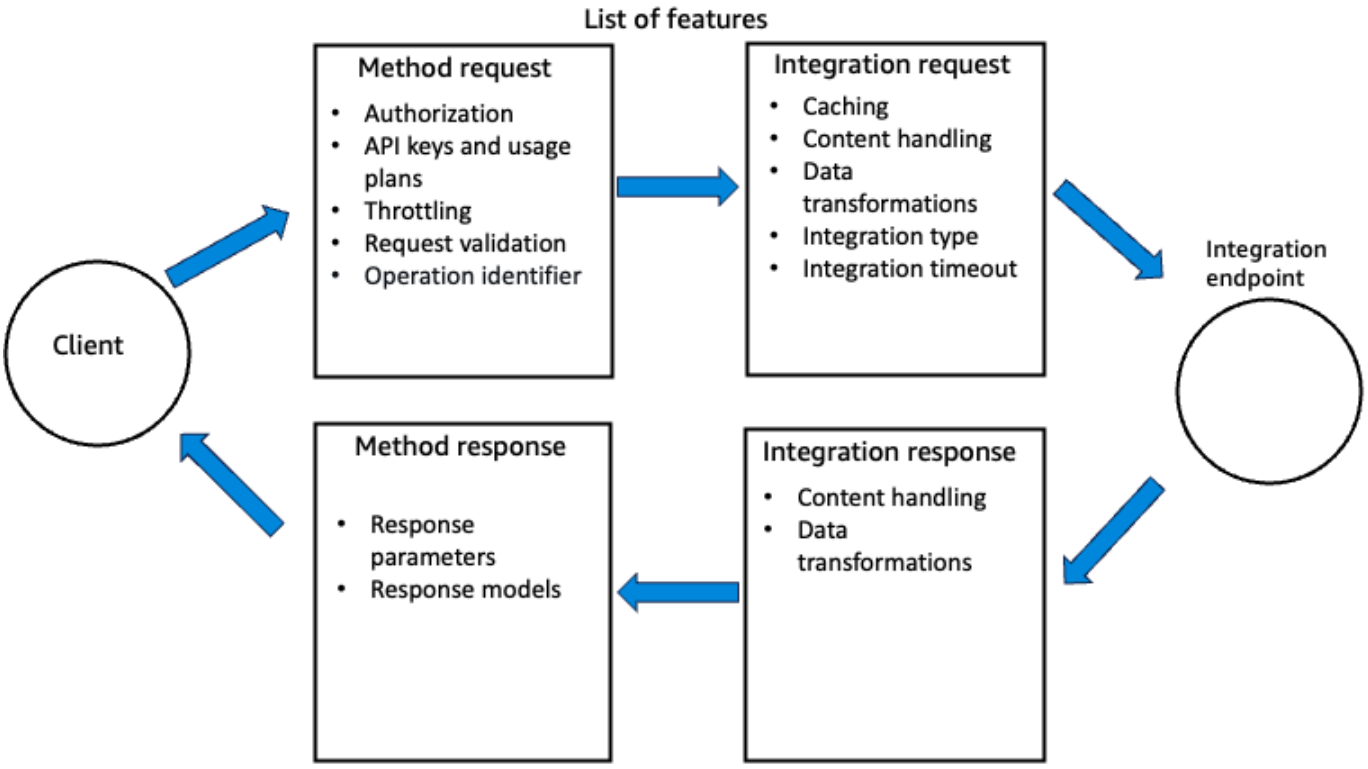
若要匯入類似的 API 並在 中測試它 AWS 管理主控台，請參閱範例 API。

## 用於開發的其他 REST API 功能

API Gateway 支援其他用於開發 REST API 的功能。例如，為了協助客戶了解您的 API，您可以提供 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、Lambda 授權方或 Amazon Cognito 使用者集區。若要測量 API 的用量，請設定 usage plans (用量方案) 來調節 API 請求。您可以在建立或更新 API 時啟用這些功能。

下圖顯示可用於 REST API 開發的功能，以及請求/回應模型中可設定這些功能的位置。




如需有關如何建立 API 的簡介，請參閱 the section called “教學：建立具 Lambda 代理整合的 REST API”。若要進一步了解開發 REST API 時可能會使用的 API Gateway 功能，請參閱下列主題。這些主題包含您可以使用 API Gateway 主控台、API Gateway REST API AWS CLI、 或其中一個 AWS SDKs 執行的概念資訊和程序。

主題

- API Gateway 中 REST API 的 API 端點類型
- APIs中 REST API 的安全政策
- API Gateway 中 REST API 的 IP 位址類型
- API Gateway 中 REST API 的方法
- 在 API Gateway 中控制和管理對 REST API 的存取
- API Gateway 中 REST API 的整合
- API Gateway 中的 REST API 的請求驗證

- API Gateway 中用於 REST API 的資料轉換
- API Gateway 中 REST API 的閘道回應
- API Gateway 中 REST API 的 CORS
- API Gateway 中 REST API 的二進位媒體類型
- 在 API Gateway 中調用 REST API
- 在 API Gateway 中使用 OpenAPI 開發 REST API

## API Gateway 中 REST API 的 API 端點類型

API 端點 類型是指 API 的主機名稱。根據您大部分 API 流量的來源而定，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 端點通常會將請求路由至最靠近的 CloudFront 連接點 (POP)，該連接點可在您的用戶端分散各地的情況下提供協助。這是 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`)。

CloudFront 依據 Cookie 名稱以自然順序排序 HTTP Cookie，然後將請求轉送到原始伺服器。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。

針對區域 API，您使用的任何自訂網域名稱都是專屬於 API 部署所在的區域。如果您在多個區域中部署區域 API，其可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。如需詳細資訊，請參閱the section called “設定區域性自訂網域名稱”及the section called “設定 API Gateway 中的邊緣最佳化的自訂網域名稱”。

所有區域和 API 端點會依原狀傳遞所有標頭名稱。

> **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 分佈搭配，以確保 API Gateway 不會將 API 與服務控制的 CloudFront 分佈建立關聯，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 分佈搭配？。

## 私有 API 端點

私有 API 端點是僅能從 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)。如需更多詳細資訊，請參閱 the section called “私有 REST API”。

所有私有 API 端點會依原狀傳遞所有標頭名稱。

## 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 SDK 變更現有的 API 類型。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：

- 從邊緣最佳化變更為區域性或私有
- 從區域性變更為邊緣最佳化或私有
- 從私有改為區域性

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 從邊緣最佳化變更為區域性 (反之亦然)，邊緣最佳化的 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 API 仍會傳送此標頭，雖然此標頭名稱可能會重新映射到其他一些名稱。了解其中差異有助於決定如何將邊緣最佳化的 API 更新為區域 API，或將區域 API 更新為邊緣最佳化的 API。

主題

- 使用 API Gateway 主控台變更 API 端點類型
- 使用 AWS CLI 變更 API 端點類型

### 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 (以及反向轉換)

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 選擇 REST API。
3. 選擇 API 設定。
4. 在 API 詳細資訊區段中，選擇編輯。
5. 對於 API 端點類型，選取邊緣最佳化或區域。
6. 選擇儲存變更。
7. 重新部署您的 API，變更才會生效。

### 將私有端點轉換為區域端點

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 選擇 REST API。
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. 選擇 API 設定。
5. 在 API 詳細資訊區段中，選擇編輯。
6. 對於 API 端點類型，選取區域。
7. 選擇儲存變更。
8. 從您的 API 移除資源政策。
9. 重新部署您的 API，變更才會生效。

   由於您要將端點類型從私有移轉至區域，因此 API Gateway 會將 IP 位址類型變更為 IPv4。如需詳細資訊，請參閱the section called “API Gateway 中 REST API 的 IP 位址類型”。

### 將區域端點轉換為私有端點

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 選擇 REST API。
3. 建立授予 VPC 或 VPC 端點存取權的資源政策。如需詳細資訊，請參閱???。
4. 選擇 API 設定。
5. 在 API 詳細資訊區段中，選擇編輯。

6. 針對 API 端點類型，選取私有。
7. (選用) 針對 VPC 端點 ID，選取您要與私有 API 建立關聯的 VPC 端點 ID。
8. 選擇儲存變更。
9. 重新部署您的 API，變更才會生效。

   由於您要將端點類型從區域移轉至私有，因此 API Gateway 會將 IP 位址類型變更為雙堆疊。如需詳細資訊，請參閱the section called “API Gateway 中 REST API 的 IP 位址類型”。

使用 AWS CLI 變更 API 端點類型

以下 update-rest-api 命令會將邊緣最佳化 API 更新為區域 API：

```
aws apigateway update-rest-api \
    --rest-api-id a1b2c3 \
    --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": "a1b2c3",
    "name": "PetStore imported as edge-optimized"
}
```

以下 update-rest-api 命令會將區域 API 更新為邊緣最佳化 API：

```
aws apigateway update-rest-api \
    --rest-api-id a1b2c3 \
    --patch-operations op=replace,path=/endpointConfiguration/types/REGIONAL,value=EDGE
```

由於 put-rest-api 適用於更新 API 定義，因此不適用於更新 API 端點類型。

# APIs中 REST API 的安全政策

安全政策是 API Gateway 所提供的最低 TLS 版本與密碼套件的預先定義組合。當用戶端建立 API 的 TLS 交握或自訂網域名稱時，安全政策會強制執行 API Gateway 接受的 TLS 版本和密碼套件。安全政策可保護您的 API 和自訂網域名稱，使其免於發生網路安全問題，例如在用戶端和伺服器之間竄改和竊聽。

API Gateway 支援舊版安全政策和增強型安全政策。 TLS_1_0和 TLS_1_2 是舊版安全政策。使用這些安全政策來實現回溯相容性。任何以 開頭的政策SecurityPolicy_都是增強型安全政策。將這些政策用於受規範的工作負載、進階治理，或使用後量子密碼學。使用增強型安全政策時，您也必須設定端點存取模式，以進行額外的治理。如需詳細資訊，請參閱the section called “端點存取模式”。

## API Gateway 如何套用安全政策

下列範例顯示 API Gateway 如何使用安全政策作為範例來套用SecurityPolicy_TLS13_1_3_2025_09安全政策。

SecurityPolicy_TLS13_1_3_2025_09 安全政策接受 TLS 1.3 流量，並拒絕 TLS 1.2 和 TLS 1.0 流量。對於 TLS 1.3 流量，安全政策接受下列密碼套件：

- TLS_AES_128_GCM_SHA256
- TLS_AES_256_GCM_SHA384
- TLS_CHACHA20_POLY1305_SHA256

API Gateway 不接受任何其他密碼套件。例如，安全政策會拒絕任何使用密碼套件的 AES128-SHA TLS 1.3 流量。如需支援的 TLS 版本和密碼的詳細資訊，請參閱 the section called “支援的安全政策”。

若要監控用於存取 API Gateway 的 TLS 通訊協定和加密用戶端，您可以在存取日誌中使用 $context.tlsVersion和 $context.cipherSuite內容變數。如需詳細資訊，請參閱the section called “監控”。

## 端點存取模式

端點存取模式是您必須為任何使用以 SecurityPolicy_ 開頭的增強型安全政策的 REST API 或自訂網域名稱所指定的額外參數。您可以在建立資源時執行此操作，或將安全政策從舊版政策變更為增強型政策時執行此操作。

當端點存取模式設定為 時STRICT，對 REST API 或自訂網域名稱的任何請求都必須通過下列檢查：

- 請求必須來自與資源相同的 API Gateway 端點類型。這可能來自區域、邊緣最佳化或私有端點。
- 如果您使用區域或私有端點，API Gateway 會使用 SNI 主機比對。如果您使用邊緣最佳化端點，API Gateway 會符合 CloudFront 的網域前端保護。如需詳細資訊，請參閱網域前端。

如果不符合上述任一條件，API Gateway 會拒絕請求。我們建議您盡可能使用STRICT端點存取模式。

若要遷移現有的 API 或網域名稱以使用嚴格的端點存取模式，請先將安全政策更新為增強型安全政策，並將端點存取模式設定為 BASIC。驗證流量和存取日誌之後，請將端點存取模式設定為 STRICT。當您將端點存取模式從 遷移STRICT至 時BASIC，隨著變更傳播，您的端點將無法使用約 15 分鐘。

您不應STRICT針對特定應用程式架構將端點存取模式設定為 ，而是將端點存取模式設定為 BASIC。下表顯示一些應用程式架構和建議，讓您的 REST API 或自訂網域名稱可以使用STRICT端點存取模式。

| Architecture | 建議的遷移 |
| --- | --- |
| 使用 VPC 端點存取公有自訂網域名稱。 | 此架構使用跨端點類型流量。我們建議您遷移至 the section called “私有 API 的自訂網域名稱”。 |
| 使用任何方法叫用不使用自訂網域名稱或私有 DNS 名稱的私有 API。 | 此架構會在主機標頭與 TLS 交握中使用的 SNI 之間建立不相符，並且不會通過 CloudFront 的網域前端限制。我們建議您遷移 VPC 以使用私有 DNS。 |
| 使用網域碎片將內容分散到多個網域或子網域。 | 此架構會在主機標頭與 TLS 交握中使用的 SNI 之間建立不相符，並且不會通過 CloudFront 的網域前端限制。我們建議您使用 HTTP/2並從此反模式遷移。 |

以下是使用端點存取模式的考量：

- 如果 API 或網域名稱的端點存取模式為 STRICT，則您無法變更端點類型。若要變更端點類型，請先將端點存取模式變更為 BASIC。
- 將端點存取模式從 變更為 BASIC 後STRICT，API Gateway 會延遲 15 分鐘，以強制執行嚴格的端點存取模式。

- 當您將安全政策從開頭為 SecurityPolicy_ 的政策變更為舊版政策時，您必須將端點存取模式取消設定為 ""。

## 考量事項

以下是 API Gateway 中 REST APIs的安全政策考量：

- 您可以在 OpenAPI 定義檔案中匯入安全政策。如需詳細資訊，請參閱x-amazon-apigateway-security-policy。
- 您的 API 可以映射到與 API 具有不同安全政策的自訂網域名稱。當您叫用該自訂網域名稱時，API Gateway 會使用自訂網域的安全政策來交涉 TLS 交握。如果您停用預設 API 端點，則可能會影響發起人調用 API 的方式。
- 如果您變更安全政策，更新大約需要 15 分鐘才能完成。您可以監控 API apiStatus的 。隨著 API 更新， apiStatus 是 UPDATING ，當它完成時，它將是 AVAILABLE。當您的 API 狀態為時UPDATING，您仍然可以叫用它。
- API Gateway 支援所有 APIs的安全政策。不過，您只能選擇 REST APIs的安全政策。API Gateway 僅支援 HTTP 或 WebSocket APIsTLS_1_2的安全政策。
- 您無法將 API 的安全政策從 更新TLS_1_0為 TLS_1_2。
- 有些安全政策同時支援 ECDSA 和 RSA 密碼套件。如果您將這種類型的政策與自訂網域名稱搭配使用，加密套件會符合客戶提供的憑證金鑰類型，可以是 RSA 或 ECDSA。如果您將這種類型的政策與 REST API 搭配使用，加密套件會符合與 RSA 憑證類型相容的加密套件。

## 支援的安全政策

下表說明可為每個 REST API 端點類型和自訂網域名稱類型指定的安全政策。這些政策可讓您控制傳入連線。API Gateway 僅支援輸出的 TLS 1.2。您可以隨時更新 API 或自訂網域名稱的安全政策。

標題FIPS中包含 的政策與聯邦資訊處理標準 (FIPS) 相容，這是美國和加拿大政府標準，可指定保護敏感資訊之密碼編譯模組的安全要求。若要進一步了解，請參閱AWS 雲端安全合規頁面上的聯邦資訊處理標準 (FIPS) 140。

所有 FIPS 政策都會利用 AWS-LC FIPS 驗證的密碼編譯模組。若要進一步了解，請參閱 NIST 密碼編譯模組驗證計劃網站上的 AWS-LC 密碼編譯模組頁面。

標題PQ中包含 的政策使用後量子密碼編譯 (PQC) 實作 TLS 的混合金鑰交換演算法，以確保流量對未來量子運算威脅的機密性。

標題PFS中包含 的政策會使用 Perfect Forward Secrecy (PFS)，以確保工作階段金鑰不會洩露。

在其標題PQ中同時包含 FIPS和 的政策支援這兩種功能。

預設安全政策

當您建立新的 REST API 或自訂網域時，資源會獲指派預設的安全政策。下表顯示這些資源的預設安全政策。

| Resource | 預設安全政策名稱 | | |
|---|---|---|---|
| 區域 API | TLS_1_0 | | |
| 邊緣最佳化的 API | TLS_1_0 | | |
| 私有 API | TLS_1_2 | | |
| 區域網域 | TLS_1_2 | | |
| Edge 最佳化網域 | TLS_1_2 | | |
| 私有網域 | TLS_1_2 | | |

區域和私有 APIs以及自訂網域名稱支援的安全政策

下表說明可為區域和私有 APIs 以及自訂網域名稱指定的安全政策：

| 安全政策 | 支援的 TLS 版本 | 支援的加密 |
|---|---|---|
| SecurityPolicy_TLS13_1_3_2025_09 | TLS1.3 | TLS1.3<br>• TLS_AES_128_GCM_SHA256<br>• TLS_AES_256_GCM_SHA384<br>• TLS_CHACHA20_POLY1305_SHA256 |
| SecurityPolicy_TLS13_1_3_FIPS_2025_09 | TLS1.3 | TLS1.3<br>• TLS_AES_128_GCM_SHA256 |

| 安全政策 | 支援的 TLS 版本 | 支援的加密 |
| --- | --- | --- |
| | | • TLS_AES_256_GCM_SHA384 |
| SecurityPolicy_TLS13_1_2_FIPS_PFS_PQ_2025_09 | TLS1.3<br><br>TLS1.2 | TLS1.3<br>• TLS_AES_128_GCM_SHA256<br>• TLS_AES_256_GCM_SHA384<br><br>TLS1.2<br>• ECDHE-ECDSA-AES128-GCM-SHA256<br>• ECDHE-RSA-AES128-GCM-SHA256<br>• ECDHE-ECDSA-AES256-GCM-SHA384<br>• ECDHE-RSA-AES256-GCM-SHA384 |

| 安全政策 | 支援的 TLS 版本 | 支援的加密 |
| --- | --- | --- |
| SecurityPolicy_TLS13_1_2_PFS_PQ_2025_09 | TLS1.3<br><br>TLS1.2 | TLS1.3<br><br>• TLS_AES_128_GCM_SHA256<br>• TLS_AES_256_GCM_SHA384<br>• TLS_CHACHA20_POLY1305_SHA256<br><br>TLS1.2<br><br>• ECDHE-ECDSA-AES128-GCM-SHA256<br>• ECDHE-RSA-AES128-GCM-SHA256<br>• ECDHE-ECDSA-AES256-GCM-SHA384<br>• ECDHE-RSA-AES256-GCM-SHA384 |

| 安全政策 | 支援的 TLS 版本 | 支援的加密 |
| --- | --- | --- |
| SecurityPolicy_TLS13_1_2_PQ_2025_09 | TLS1.3<br>TLS1.2 | TLS1.3<br>• TLS_AES_128_GCM_SHA256<br>• TLS_AES_256_GCM_SHA384<br>• TLS_CHACHA20_POLY1305_SHA256<br>TLS1.2<br>• ECDHE-ECDSA-AES128-GCM-SHA256<br>• ECDHE-RSA-AES128-GCM-SHA256<br>• ECDHE-ECDSA-AES128-SHA256<br>• ECDHE-RSA-AES128-SHA256<br>• ECDHE-ECDSA-AES256-GCM-SHA384<br>• ECDHE-RSA-AES256-GCM-SHA384<br>• ECDHE-ECDSA-AES256-SHA384<br>• ECDHE-RSA-AES256-SHA384<br>• AES128-GCM-SHA256<br>• AES128-SHA256<br>• AES256-GCM-SHA384<br>• AES256-SHA256 |

| 安全政策 | 支援的 TLS 版本 | 支援的加密 |
| --- | --- | --- |
| SecurityPolicy_TLS13_1_2_2021_06 | TLS1.3<br><br>TLS1.2 | TLS1.3<br><br>• TLS_AES_128_GCM_SHA256<br>• TLS_AES_256_GCM_SHA384<br>• TLS_CHACHA20_POLY1305_SHA256<br><br>TLS1.2<br><br>• ECDHE-ECDSA-AES128-GCM-SHA256<br>• ECDHE-RSA-AES128-GCM-SHA256<br>• ECDHE-ECDSA-AES128-SHA256<br>• ECDHE-RSA-AES128-SHA256<br>• ECDHE-ECDSA-AES256-GCM-SHA384<br>• ECDHE-RSA-AES256-GCM-SHA384<br>• ECDHE-ECDSA-AES256-SHA384<br>• ECDHE-RSA-AES256-SHA384<br>• AES128-GCM-SHA256<br>• AES128-SHA256<br>• AES256-GCM-SHA384<br>• AES256-SHA256 |

| 安全政策 | 支援的 TLS 版本 | 支援的加密 |
| --- | --- | --- |
| TLS_1_2 | TLS1.3<br><br>TLS1.2 | TLS1.3<br>• TLS_AES_128_GCM_SHA256<br>• TLS_AES_256_GCM_SHA384<br>• TLS_CHACHA20_POLY1305_SHA256<br><br>TLS1.2<br>• ECDHE-ECDSA-AES128-GCM-SHA256<br>• ECDHE-RSA-AES128-GCM-SHA256<br>• ECDHE-ECDSA-AES128-SHA256<br>• ECDHE-RSA-AES128-SHA256<br>• ECDHE-ECDSA-AES256-GCM-SHA384<br>• ECDHE-RSA-AES256-GCM-SHA384<br>• ECDHE-ECDSA-AES256-SHA384<br>• ECDHE-RSA-AES256-SHA384<br>• AES128-GCM-SHA256<br>• AES128-SHA256<br>• AES256-GCM-SHA384<br>• AES256-SHA256 |

| 安全政策 | 支援的 TLS 版本 | 支援的加密 |
| --- | --- | --- |
| TLS_1_0 | TLS1.3<br>TLS1.2<br>TLS1.1<br>TLS1.0 | TLS1.3<br>• TLS_AES_128_GCM_SHA256<br>• TLS_AES_256_GCM_SHA384<br>• TLS_CHACHA20_POLY1305_SHA256<br><br>TLS1.0-TLS1.2<br>• ECDHE-ECDSA-AES128-GCM-SHA256<br>• ECDHE-RSA-AES128-GCM-SHA256<br>• ECDHE-ECDSA-AES128-SHA256<br>• ECDHE-RSA-AES128-SHA256<br>• ECDHE-ECDSA-AES128-SHA<br>• ECDHE-RSA-AES128-SHA<br>• ECDHE-ECDSA-AES256-GCM-SHA384<br>• ECDHE-RSA-AES256-GCM-SHA384<br>• ECDHE-ECDSA-AES256-SHA384<br>• ECDHE-RSA-AES256-SHA384<br>• ECDHE-ECDSA-AES256-SHA |

| 安全政策 | 支援的 TLS 版本 | 支援的加密 |
| --- | --- | --- |
| | | • ECDHE-RSA-AES256-SHA<br>• AES128-GCM-SHA256<br>• AES128-SHA256<br>• AES128-SHA<br>• AES256-GCM-SHA384<br>• AES256-SHA256<br>• AES256-SHA |

### 邊緣最佳化 APIs 和自訂網域名稱支援的安全政策

下表說明可針對邊緣最佳化 APIs 和邊緣最佳化自訂網域名稱指定的安全政策：

| 安全政策名稱 | 支援的 TLS 版本 | 支援的加密 |
| --- | --- | --- |
| SecurityPolicy_TLS13_2025_EDGE | TLS1.3 | TLS1.3<br>• TLS_AES_128_GCM_SHA256<br>• TLS_AES_256_GCM_SHA384<br>• TLS_CHACHA20_POLY1305_SHA256 |
| SecurityPolicy_TLS12_PFS_2025_EDGE | TLS1.3<br>TLS1.2 | TLS1.3<br>• TLS_AES_128_GCM_SHA256<br>• TLS_AES_256_GCM_SHA384<br>• TLS_CHACHA20_POLY1305_SHA256 |

| 安全政策名稱 | 支援的 TLS 版本 | 支援的加密 |
| --- | --- | --- |
| | | TLS1.2<br>• ECDHE-ECDSA-AES128-GCM-SHA256<br>• ECDHE-ECDSA-AES256-GCM-SHA384<br>• ECDHE-ECDSA-CHACHA20-POLY1305<br>• ECDHE-RSA-AES128-GCM-SHA256<br>• ECDHE-RSA-AES256-GCM-SHA384<br>• ECDHE-RSA-CHACHA20-POLY1305 |

| 安全政策名稱 | 支援的 TLS 版本 | 支援的加密 |
| --- | --- | --- |
| SecurityPolicy_TLS12_2018_EDGE | TLS1.3<br><br>TLS1.2 | TLS1.3<br><br>• TLS_AES_128_GCM_SHA256<br>• TLS_AES_256_GCM_SHA384<br>• TLS_CHACHA20_POLY1305_SHA256<br><br>TLS1.2<br><br>• ECDHE-ECDSA-AES128-GCM-SHA256<br>• ECDHE-ECDSA-AES128-SHA256<br>• ECDHE-ECDSA-AES256-GCM-SHA384<br>• ECDHE-ECDSA-CHACHA20-POLY1305<br>• ECDHE-ECDSA-AES256-SHA384<br>• ECDHE-RSA-AES128-GCM-SHA256<br>• ECDHE-RSA-AES128-SHA256<br>• ECDHE-RSA-AES256-GCM-SHA384<br>• ECDHE-RSA-CHACHA20-POLY1305<br>• ECDHE-RSA-AES256-SHA384 |

| 安全政策名稱 | 支援的 TLS 版本 | 支援的加密 |
| --- | --- | --- |
| TLS_1_2（僅限邊緣最佳化自訂網域名稱） | TLS1.3<br><br>TLS1.2 | TLS1.3<br><br>• TLS_AES_128_GCM_SHA256<br>• TLS_AES_256_GCM_SHA384<br>• TLS_CHACHA20_POLY1305_SHA256<br><br>TLS1.2<br><br>• ECDHE-ECDSA-AES128-GCM-SHA256<br>• ECDHE-ECDSA-AES128-SHA256<br>• ECDHE-ECDSA-AES256-GCM-SHA384<br>• ECDHE-ECDSA-CHACHA20-POLY1305<br>• ECDHE-ECDSA-AES256-SHA384<br>• ECDHE-RSA-AES128-GCM-SHA256<br>• ECDHE-RSA-AES128-SHA256<br>• ECDHE-RSA-AES256-GCM-SHA384<br>• ECDHE-RSA-CHACHA20-POLY1305<br>• ECDHE-RSA-AES256-SHA384<br>• AES128-GCM-SHA256 |

| 安全政策名稱 | 支援的 TLS 版本 | 支援的加密 |
| --- | --- | --- |
| | | • AES256-GCM-SHA384<br>• AES128-SHA256 |

| 安全政策名稱 | 支援的 TLS 版本 | 支援的加密 |
| --- | --- | --- |
| TLS_1_0 | TLS1.3<br>TLS1.2<br>TLS1.1<br>TLS1.0 | TLS1.3<br>• TLS_AES_128_GCM_SHA256<br>• TLS_AES_256_GCM_SHA384<br>• TLS_CHACHA20_POLY1305_SHA256<br><br>TLS1.0-TLS1.2<br>• ECDHE-ECDSA-AES128-GCM-SHA256<br>• ECDHE-RSA-AES128-GCM-SHA256<br>• ECDHE-ECDSA-AES128-SHA256<br>• ECDHE-RSA-AES128-SHA256<br>• ECDHE-ECDSA-AES128-SHA<br>• ECDHE-RSA-AES128-SHA<br>• ECDHE-ECDSA-AES256-GCM-SHA384<br>• ECDHE-RSA-AES256-GCM-SHA384<br>• ECDHE-ECDSA-AES256-SHA384<br>• ECDHE-RSA-AES256-SHA384<br>• ECDHE-ECDSA-AES256-SHA |

| 安全政策名稱 | 支援的 TLS 版本 | 支援的加密 |
| --- | --- | --- |
| | | • ECDHE-RSA-AES256-SHA<br>• AES256-SHA256<br>• AES256-SHA |

OpenSSL 和 RFC 密碼名稱

OpenSSL 和 IETF RFC 5246 使用相同密碼的不同名稱。下表將 OpenSSL 名稱對應到每個密碼的 RFC 名稱。如需詳細資訊，請參閱 OpenSSL 文件中的加密。

| OpenSSL 和密碼名稱 | RFC 密碼名稱 |
| --- | --- |
| TLS_AES_128_GCM_SHA256 | TLS_AES_128_GCM_SHA256 |
| TLS_AES_256_GCM_SHA384 | TLS_AES_256_GCM_SHA384 |
| TLS_CHACHA20_POLY1305_SHA256 | TLS_CHACHA20_POLY1305_SHA256 |
| ECDHE-RSA-AES128-GCM-SHA256 | TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 |
| ECDHE-RSA-AES128-SHA256 | TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 |
| ECDHE-RSA-AES128-SHA | TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA |

| OpenSSL 和密碼名稱 | RFC 密碼名稱 |
| --- | --- |
| ECDHE-RSA-AES256-GCM-SHA384 | TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 |
| ECDHE-RSA-AES256-SHA384 | TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384 |
| ECDHE-RSA-AES256-SHA | TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA |
| AES128-GCM-SHA256 | TLS_RSA_WITH_AES_128_GCM_SHA256 |
| AES256-GCM-SHA384 | TLS_RSA_WITH_AES_256_GCM_SHA384 |
| AES128-SHA256 | TLS_RSA_WITH_AES_128_CBC_SHA256 |
| AES256-SHA | TLS_RSA_WITH_AES_256_CBC_SHA |
| AES128-SHA | TLS_RSA_WITH_AES_128_CBC_SHA |
| DES-CBC3-SHA | TLS_RSA_WITH_3DES_EDE_CBC_SHA |

## 如何變更安全政策

您可以變更 API 的安全政策。如果您要透過自訂網域名稱將流量傳送至 APIs，則 API 和自訂網域名稱不需要具有相同的安全政策。當您叫用該自訂網域名稱時，API Gateway 會使用自訂網域的安全政策來交涉 TLS 交握。不過，為了保持一致性，我們建議您針對自訂網域名稱和 API 使用相同的安全政策。

如果您變更安全政策，更新大約需要 15 分鐘才能完成。您可以監控 API `apiStatus`的 。隨著 API 更新， `apiStatus` 是 `UPDATING` ，當它完成時，它將是 `AVAILABLE`。更新 API 時，您仍然可以叫用它。

AWS 管理主控台

變更 API 的安全政策

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 選擇 REST API。
3. 選擇 API 設定，然後選擇編輯。
4. 針對安全政策，選取以 開頭的新政策`SecurityPolicy_`。
5. 針對端點存取模式，選擇嚴格。
6. 選擇儲存變更。

   重新部署 API 以使變更生效。由於您將端點存取模式變更為嚴格，因此變更完全傳播大約需要 15 分鐘。

AWS CLI

下列 update-rest-api 命令會更新 API 以使用`SecurityPolicy_TLS13_1_3_2025_09`安全政策：

```
aws apigateway update-rest-api \
    --rest-api-id abcd1234 \
    --patch-operations '[
        {
            "op": "replace",
            "path": "/securityPolicy",
            "value": "SecurityPolicy_TLS13_1_3_2025_09"
        },
        {
            "op": "replace",
            "path": "/endpointAccessMode",
            "value": "STRICT"
        }
    ]'
```

輸出將如下所示：

```
{
    "id": "abcd1234",
    "name": "MyAPI",
```

```
    "description": "My API with a new security policy",
    "createdDate": "2025-02-04T11:47:06-08:00",
    "apiKeySource": "HEADER",
    "endpointConfiguration": {
        "types": [
            "REGIONAL"
        ],
        "ipAddressType": "dualstack"
    },
    "tags": {},
    "disableExecuteApiEndpoint": false,
    "securityPolicy": "SecurityPolicy_TLS13_1_3_2025_09",
    "endpointAccessMode": "STRICT"
    "rootResourceId": "efg456"
}
```

下列 update-rest-api 命令會更新使用增強型安全政策來使用TLS_1_0安全政策的 API。

```
aws apigateway update-rest-api \
    --rest-api-id abcd1234 \
    --patch-operations '[
        {
            "op": "replace",
            "path": "/securityPolicy",
            "value": "TLS_1_0"
        },
        {
            "op": "replace",
            "path": "/endpointAccessMode",
            "value": ""
        }
    ]'
```

輸出將如下所示：

```
{
    "id": "abcd1234",
    "name": "MyAPI",
    "description": "My API with a new security policy",
    "createdDate": "2025-02-04T11:47:06-08:00",
    "apiKeySource": "HEADER",
    "endpointConfiguration": {
        "types": [
```

```
            "REGIONAL"
        ],
        "ipAddressType": "dualstack"
    },
    "tags": {},
    "disableExecuteApiEndpoint": false,
    "securityPolicy": "TLS_1_0",
    "rootResourceId": "efg456"
}
```

# API Gateway 中 REST API 的 IP 位址類型

建立 API 時，您可以指定可調用 API 的 IP 位址類型。您可以選擇 IPv4 來解析 IPv4 位址以調用 API，也可以選擇雙堆疊，以同時允許 IPv4 和 IPv6 位址調用您的 API。我們建議您將 IP 位址類型設定為雙堆疊，以緩解 IP 空間耗盡或用於安全狀態。如需雙堆疊 IP 位址類型優點的詳細資訊，請參閱 AWS 上的 IPv6。

若要將您的 API 僅限用於 IPv6 流量，您可以建立資源政策，並將來源 IP 位址僅限於 IPv6 範圍。您可以藉由更新 API 的組態來變更 IP 位址類型。此變更會立即生效，而且您不需要重新部署 API。如需詳細資訊，請參閱the section called “範例：拒絕根據來源 IP 地址或範圍的 API 流量”

## IP 位址類型的考量事項

下列考量事項可能會影響您使用 IP 位址類型：

- 所有區域和邊緣最佳化 API 的預設 IP 位址類型都是 IPv4。
- 私有 API 只能具有雙堆疊 IP 位址類型。
- 如果您將現有 API 的 IP 位址類型從 IPv4 變更為雙堆疊，請確認控制存取 API 的任何政策都已更新，以便進行 IPv6 呼叫。當您變更 IP 位址類型時，變更會立即生效。
- 如果您將 API 的端點類型從區域或邊緣最佳化移轉至私有，則 API Gateway 會將 IP 位址類型變更為雙堆疊。如需更多詳細資訊，請參閱 the section called “變更公有或私有 API 端點類型”。
- 如果您將 API 的端點類型從私有移轉至區域，則您必須將 IP 位址類型設為雙堆疊。端點移轉完成後，您可以將 IP 位址類型變更為 IPv4。如需更多詳細資訊，請參閱 the section called “變更公有或私有 API 端點類型”。
- 您的 API 可以對應至與您的 API 具有不同 IP 位址類型的自訂網域名稱。如果您停用預設 API 端點，則可能會影響呼叫方調用 API 的方式。
- 您無法使用外部定義檔案來設定 API 的 IP 位址類型。

## 變更 REST API 的 IP 位址類型

您可以藉由更新 API 的組態來變更 IP 位址類型。您可以使用 AWS 管理主控台、AWS CLI、CloudFormation 或 AWS SDK 來更新 API 的組態。如果您變更 API 的 IP 位址類型，您不需重新部署 API 讓變更生效。在您變更 IP 位址類型之前，請先確認控制存取 API 的任何政策都已更新，以便進行 IPv6 呼叫。

AWS 管理主控台

若要變更 REST API 的 IP 位址類型

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 選擇 REST API。
3. 選擇 API 設定，然後選擇編輯。
4. 針對 IP 位址類型，選取 IPv4 或雙堆疊。
5. 選擇儲存變更。

   對 API 組態的變更會立即生效。

AWS CLI

以下 update-rest-api 命令會將 API 更新為雙堆疊 IP 位址類型：

```
aws apigateway update-rest-api \
    --rest-api-id abcd1234 \
    --patch-operations "op='replace',path='/endpointConfiguration/
ipAddressType',value='dualstack'"
```

輸出將如下所示：

```
{
    "id": "abcd1234",
    "name": "MyAPI",
    "description": "My API with a dualstack IP address type",
    "createdDate": "2025-02-04T11:47:06-08:00",
    "apiKeySource": "HEADER",
    "endpointConfiguration": {
        "types": [
            "REGIONAL"
```

```
        ],
        "ipAddressType": "dualstack"
    },
    "tags": {},
    "disableExecuteApiEndpoint": false,
    "rootResourceId": "efg456"
}
```

## API Gateway 中 REST API 的方法

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 並提供有關無效的請求的適當錯誤訊息。

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 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。

## 先決條件

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。請遵循中的說明進行教學：建立具有 HTTP 非代理整合的 REST API
- 如果您想要讓方法與 Lambda 函數通訊，您必須已在 IAM 中建立 Lambda 呼叫角色與 Lambda 執行角色。您也必須已建立 Lambda 函數與您要在 AWS Lambda中用來通訊的方法。若要建立角色與函數，請使用選擇 AWS Lambda 整合教學課程 之為 Lambda 非代理整合建立 Lambda 函數中的說明。
- 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 在 API Gateway 中，API Gateway 支援的 HTTP 與 HTTP 代理整合憑證授權單位。

主題

- 在 API Gateway 中設定方法請求
- 在 API Gateway 中設定方法回應
- 使用 API Gateway 主控台設定方法

## 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
- AWS CLI 命令 (create-resource 和 put-method)
- AWS SDK 函數 （ 例如，在 Node.js 中，createResource 和 putMethod)
- API Gateway REST API (resource:create 與 method:put)。

主題

- 設定 API 資源
- 設定 HTTP 方法
- 設定方法請求參數
- 設定方法請求模型
- 設定方法請求授權
- 設定方法請求驗證

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。

以 PetStore 範例 API 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。

以下 get-resources 命令會擷取 API 的所有資源：

```
aws apigateway get-resources --rest-api-id apiId
```

在 PetStore 範例 API 中，輸出如下所示：

```
{
    "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資源之後，使用下列命令來新增子資源：

```
aws apigateway create-resource --rest-api-id apiId \
    --parent-id parentId \
    --path-part resourceName
```

例如，若要在 PetStore 網站上新增販售的寵物食品，請使用下列命令：

```
aws apigateway create-resource --rest-api-id a1b2c3 \
    --parent-id svzr2028x8 \
    --path-part food
```

輸出將如下所示：

```
{
    "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。您可以使用任何字串作為 greedy 路徑參數名稱。

以下 create-resource 命令會在根目錄 (/{proxy+}) 下建立代理資源：

```
aws apigateway create-resource --rest-api-id apiId \
    --parent-id rootResourceId \
    --path-part {proxy+}
```

輸出將如下所示：

```
{
    "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，假設 /parent/{proxy+} 和 /parent/{child}/{proxy+} 不在相同父系層級。

```
/{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。

下表顯示 API Gateway 如何將請求路由至 API prod 階段的下列資源。

```
ANY /{proxy+}
GET /pets/{proxy+}
GET /pets/dog
```

| 請求 | 選取的路由 | 說明 |
| --- | --- | --- |
| GET https://*api-id*.execute-api.*region*.amazonaws.com/prod/pets/dog | GET /pets/dog | 請求與此資源完全相符。 |
| GET https://*api-id*.execute-api.*region*.amazonaws.com/prod/pets/cats | GET /pets/{proxy+} | /pets/{proxy+} greedy 路徑變數會擷取此請求。 |
| GET https://*api-id*.execute-api.*region*.amazonaws.com/prod/animals | GET /{proxy+} | /{proxy+} greedy 路徑變數會擷取此請求。 |

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 方法資源所封裝。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。

以下 put-method 命令會使用 IAM 許可建立 ANY 動詞的方法請求，以控制其存取。

```
aws apigateway put-method --rest-api-id vaz7da96z6 \
    --resource-id 6sxz2j \
    --http-method ANY \
    --authorization-type AWS_IAM
```

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”」。

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，以下 put-method 命令會建立方法與必要的路徑參數：

```
aws apigateway put-method --rest-api-id vaz7da96z6 \
    --resource-id rjkmth \
    --http-method GET \
    --authorization-type "NONE" \
    --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 方法使用選用的查詢字串參數 type 與選用的標頭參數 age，則您可以使用以下 put-method 命令來宣告這些參數：

```
aws apigateway put-method --rest-api-id vaz7da96z6 \
    --resource-id 6sxz2j \
    --http-method GET \
    --authorization-type "NONE" \
    --request-parameters
 method.request.querystring.type=false,method.request.header.age=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.age":false}'
```

有了這項設定，用戶端就可依類型查詢寵物：

```
GET /pets?type=dog
```

而且用戶端可以查詢哪些狗是幼犬，如下所示：

```
GET /pets?type=dog
age:puppy
```

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 “整合”」。

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、產生軟體開發套件，並初始化映射範本以在 API Gateway 主控台中設定整合。如需如何建立模型的相關資訊，請參閱了解資料模型。

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。API Gateway 使用 `Content-Type` 請求標頭來確定內容類型。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`映射。

若要使用相同的模型，而不論內容類型為何，請指定 `$default` 為索引鍵。

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，您可以使用以下 put-method 命令：

```
aws apigateway put-method \
    --rest-api-id vaz7da96z6 \
    --resource-id 6sxz2j \
    --http-method POST \
    --authorization-type "NONE" \
    --request-models '{"application/json":"petModel"}'
```

在此範例中，`petModel` 是描述寵物之 `name` 資源的 `Model` 屬性值。實際結構描述定義會以 `schema` 資源之 `Model` 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`**。在您設定此選項時，API Gateway 會根據發起人的憑證驗證發起人對要求的簽章。如果經驗證的使用者擁有呼叫方法的許可，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。除非發起人擁有呼叫 API 方法的許可，否則對方法的呼叫不會成功。以下 IAM 政策會向發起人授予呼叫在同一 AWS 帳戶中建立之任何 API 方法的許可：

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "execute-api:Invoke"
            ],
            "Resource": "arn:aws:execute-api:*:*:*"
        }
    ]
}
```

如需詳細資訊，請參閱the section called “使用 IAM 許可”。

目前，您只能將此政策授予 API 擁有者的 AWS 帳戶中的使用者、群組和角色。來自不同 的使用者只有在允許在 API 擁有者的 中擔任角色，並 AWS 帳戶 具有呼叫`execute-api:Invoke`動作的必要許可時， AWS 帳戶 才能呼叫 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 AWS CLI開發套件或 REST API 用戶端，例如 Postman，其會實作 Signature 第 4 版 (SigV4) 簽署。

若要使用 Lambda 授權方來授權對 API 方法的存取，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”。

若要使用 Amazon Cognito 使用者集區來授權對 API 方法的存取，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。如需有關建立 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 Amazon Cognito 使用者集區做為授權方”。

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。以下 create-request-validator 命令會建立僅限內文的請求驗證程式。

```
aws apigateway create-request-validator \
    --rest-api-id 7zw9uyk9kl \
    --name bodyOnlyValidator \
    --validate-request-body  \
    --no-validate-request-parameters
```

輸出將如下所示：

```
{
    "validateRequestParameters": false,
    "validateRequestBody": true,
    "id": "jgpyy6",
    "name": "bodyOnlyValidator"
}
```

只要使用此請求驗證程式，您就可以在設定方法請求時使用請求驗證。以下 put-method 命令會建立方法請求，以要求傳入的請求內文符合 PetModel，並且具有兩個非必要的請求參數：

```
aws apigateway put-method \
    --rest-api-id 7zw9uyk9kl \
    --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 jgpyy6
```

若要將請求參數納入請求驗證中，您必須將請求驗證程式的 `validateRequestParameters` 設定為 `true`，並將 `put-method` 命令中的特定請求參數設定為 `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的結果，API Gateway 才能成功將整合回應映射至方法回應。

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` 回應作為預設值。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。以下 put-method-response 命令會建立 `200` 方法回應。

```
aws apigateway put-method-response \
    --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 `responseParameters` 格式之 `MethodResponse` 索引鍵/值組的 `"{parameter-name}":"{boolean}"` 映射。以下 put-method-response 命令會設定 `my-header` 標頭。

```
aws apigateway put-method-response \
        --rest-api-id vaz7da96z6 \
        --resource-id 6sxz2j \
        --http-method GET \
        --status-code 200  \
        --response-parameters method.response.header.my-header=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。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。

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。以下 create-model 命令會建立 `PetStorePet` 模型，以描述 `GET /pets/{petId}` 方法請求的回應內文。

```
aws apigateway create-model \
    --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 API Gateway `Model` 資源。

若要設定方法回應模型來定義酬載格式，請將 "application/json":"PetStorePet" 鍵/值對新增至 `MethodResponse` 資源的 `requestModels` 映射。以下 put-method-response 命令會建立使用回應模型來定義承載格式的方法回應：

```
aws apigateway put-method-response \
    --rest-api-id vaz7da96z6 \
    --resource-id 6sxz2j \
    --http-method GET \
    --status-code 200  \
    --response-parameters method.response.header.my-header=false \
    --response-models '{"application/json":"PetStorePet"}'
```

## 使用 API Gateway 主控台設定方法

當您使用 REST API 主控台建立方法時，可以同時設定整合請求和方法請求。根據預設，API Gateway 會為您的方法建立 `200` 方法回應。

下列指示顯示如何編輯方法請求設定，以及如何為您的方法建立其他方法回應。

主題

- 在 API Gateway 主控台編輯 API Gateway 方法請求
- 使用 API Gateway 主控台設定 API Gateway 方法回應

### 在 API Gateway 主控台編輯 API Gateway 方法請求

這些說明假設您已建立方法請求。如需有關如何建立方法的詳細資訊，請參閱 the section called “ 使用主控台設定整合請求”。

1. 在資源窗格中，選擇您的方法，然後選擇方法請求標籤。
2. 在方法請求設定區段中，選擇編輯。
3. 針對授權，選取可用的授權方。

    a. 若要對任何使用者開放方法的存取權，請選取無。如果尚未變更預設設定，則可以略過此步驟。

    b. 若要使用 IAM 許可來控制用戶端對方法的存取權，請選取 `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：

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "execute-api:Invoke"
      ],
      "Resource": [
        "arn:aws:execute-api:us-east-1:111111111111:aaabbb/*/GET/"
      ]
    }
  ]
}
```

在此存取政策中，*arn:aws:execute-api:us-east-1:111111111111:aaabbb/*/GET/* 是方法的 ARN。您可以在資源頁面上選取方法，以尋找方法的 ARN。如需設定 IAM 許可的詳細資訊，請參閱使用 IAM 許可權控制 REST API 的存取。

若要建立 IAM 角色，您可以調整下列教學課程 ??? 中的說明。

c. 若要使用 Lambda 授權方，請選取權杖或請求授權方。請先建立 Lambda 授權方，下拉式選單中才會顯示此選項。如需如何建立 Lambda 授權方的資訊，請參閱使用 API Gateway Lambda 授權方。

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。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，下拉式選單中才會顯示此選項。如需有關如何建立 Amazon Cognito 使用者集區授權方的資訊，請參閱使用 Amazon Cognito 使用者集區做為授權方，藉以控制對 REST API 的存取。

4. 若要指定請求驗證，請從請求驗證程式下拉式選單中選取值。若要關閉請求驗證，請選取無。如需每個選項的詳細資訊，請參閱「API Gateway 中的 REST API 的請求驗證」。

5. 選取需要 API 金鑰，以要求提供 API 金鑰。啟用時，可使用用量方案中的 API 金鑰來調節用戶端流量。

6. (選用) 若要在此 API 的 Java SDK 中指派 API Gateway 所產生的操作名稱，請在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。

7. 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 查詢字串參數，然後選擇新增查詢字串。

   b. 針對名稱，輸入查詢字串參數的名稱。

   c. 若要使用新建立的查詢字串參數進行請求驗證，請選取必要。如需請求驗證的詳細資訊，請參閱「API Gateway 中的 REST API 的請求驗證」。

   d. 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，請選取快取。如需快取的詳細資訊，請參閱「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」。

   若要移除查詢字串參數，請選擇移除。

8. 若要將標頭參數新增至方法，請執行下列操作：

   a. 選擇 HTTP 請求標頭，然後選擇新增標頭。

   b. 在名稱中，輸入標頭的名稱。

   c. 若要使用新建立的標頭進行請求驗證，請選取必要。如需請求驗證的詳細資訊，請參閱「API Gateway 中的 REST API 的請求驗證」。

   d. 若要將新建立的標頭當作快取金鑰的一部分來使用，請選取快取。如需快取的詳細資訊，請參閱「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」。

   若要移除標頭，請選擇移除。

9. 若要使用 POST、PUT 或 PATCH HTTP 動詞來宣告方法請求的承載格式，請選擇請求內文，然後執行下列操作：

   a. 選擇 Add model (新增模型)。

   b. 針對內容類型，輸入 MIME 類型 (例如 application/json)。

   c. 對於模型，從下拉式選單中選取模型。目前 API 的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「REST API 的資料模型」。

> **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。

10. 選擇儲存。

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. 在資源窗格中，選擇您的方法，然後選擇方法回應標籤。您可能需要選擇向右箭頭按鈕才能顯示此索引標籤。
2. 在方法回應設定區段中，選擇建立回應。
3. 針對 HPPT 狀態碼，輸入 HTTP 狀態碼，例如 200、400 或 500。

   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。
4. 選擇新增標頭。
5. 在標頭名稱中輸入名稱。

   若要從後端傳回標頭至用戶端，請在方法回應中新增標頭。
6. 選擇新增模型，以定義方法回應內文的格式。

   針對內容類型輸入回應承載的媒體類型，然後從模型下拉式選單中選擇模型。
7. 選擇儲存。

若要修改現有回應，請瀏覽至您的方法回應，然後選擇編輯。若要變更 HTTP 狀態碼，請選擇刪除並建立新的方法回應。

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 中控制和管理對 REST API 的存取

API Gateway 支援多種機制來控制和管理 API 的存取。

您可以使用下列機制進行身分驗證和授權：

- 資源政策可讓您建立以資源為基礎的政策，以從指定的來源 IP 地址或 VPC 端點允許或拒絕存取您的 API 和方法。如需詳細資訊，請參閱the section called “使用 API Gateway 資源政策”。
- 標準 AWS IAM 角色和政策提供靈活且強大的存取控制，可套用至整個 API 或個別方法。您可以使用 IAM 角色和政策來控制誰可以建立和管理您的 API，以及誰可以呼叫它們。如需詳細資訊，請參閱the section called “使用 IAM 許可”。
- 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 “屬性型存取控制”。
- 界面 VPC 端點的端點政策可讓您將 IAM 資源政策連接至界面 VPC 端點，以提高私有 API 的安全性。如需詳細資訊，請參閱the section called “使用私有 API 的 VPC 端點政策”。
- Lambda 授權方是 Lambda 函數，其可使用承載字符身分驗證以及標頭、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”。
- Amazon Cognito 使用者集區可讓您為 REST API 建立可自訂的身分驗證和授權解決方案。Amazon Cognito 使用者集區可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 Amazon Cognito 使用者集區做為授權方”。

您可以使用以下機制來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 “CORS”。
- 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 “API Gateway 產生的憑證”。
- AWS WAF 可用來保護您的 API Gateway API 免受常見的網路攻擊。如需詳細資訊，請參閱the section called “AWS WAF”。

您可以使用以下機制來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 “用量計劃”。

## 使用 API Gateway 資源政策來控制對 REST API 的存取

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 角色或群組) 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：

- 來自指定 AWS 帳戶的使用者。
- 指定來源 IP 地址範圍或 CIDR 區塊。
- 指定 Virtual Private Cloud (VPC) 或 VPC 端點 (在任何帳戶)。

您可以使用 、 AWS CLI 或 AWS SDKs AWS 管理主控台，在 API Gateway 中連接任何 API 端點類型的資源政策。處理私有 API 時，您可以將資源政策與 VPC 端點政策搭配使用，藉此控制能具備存取權限的委託人，以及其可存取的資源和動作。如需詳細資訊，請參閱the section called “使用私有 API 的 VPC 端點政策”。

API Gateway 資源政策與 IAM 身分政策不同。IAM 身分政策會連接到 IAM 使用者、群組或角色，並定義這些身分能對哪些資源執行什麼動作。API Gateway 資源政策連接至資源。您可以使用 API Gateway 資源政策搭配 IAM 政策。如需更多詳細資訊，請參閱 以身份為基礎和以資源為基礎的政策。

主題

- Amazon API Gateway 的存取原則語言概觀
- API Gateway 資源政策如何影響授權工作流程
- API Gateway 資源政策範例
- 建立 API Gateway 資源政策並連接至 API
- AWS 可用於 API Gateway 資源政策的條件索引鍵

### Amazon API Gateway 的存取原則語言概觀

本頁說明 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。如需完整的政策語言資訊，請參閱 IAM 使用者指南中的 IAM 政策概觀和 AWS Identity and Access Management 政策參考。

如需有關 AWS 服務如何決定是否應允許或拒絕特定請求的資訊，請參閱決定是否允許或拒絕請求。

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。您也可以使用縮寫語法，當您儲存資源政策時，API Gateway 語法會自動將它展開為完整 ARN。如需詳細資訊，請參閱 API Gateway 資源政策範例。

  如需完整 `Resource` 元素的格式，請參閱在 API Gateway 中執行 API 之許可的資源格式。
- 動作 - 針對每個資源，Amazon API Gateway 皆支援一組操作。您可使用動作關鍵字，來識別允許 (或拒絕) 資源操作。

  例如，`execute-api: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。
- 效果 - 當使用者請求特定動作時會有什麼效果，它可以是 `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 - 允許存取陳述式中動作與資源的帳戶或使用者。在資源政策中，委託人是接收此許可的使用者或帳戶。

下列範例資源政策會顯示先前的常用政策元素。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。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": "*",
            "Action": "execute-api:Invoke",
```

```
            "Resource": "arn:aws:execute-api:us-east-1:111111111111:*"
        },
        {
            "Effect": "Deny",
            "Principal": "*",
            "Action": "execute-api:Invoke",
            "Resource": "arn:aws:execute-api:us-east-1:111111111111:*",
            "Condition": {
                "NotIpAddress": {
                    "aws:SourceIp": "123.4.5.6/24"
                }
            }
        }
    ]
}
```

API Gateway 資源政策如何影響授權工作流程

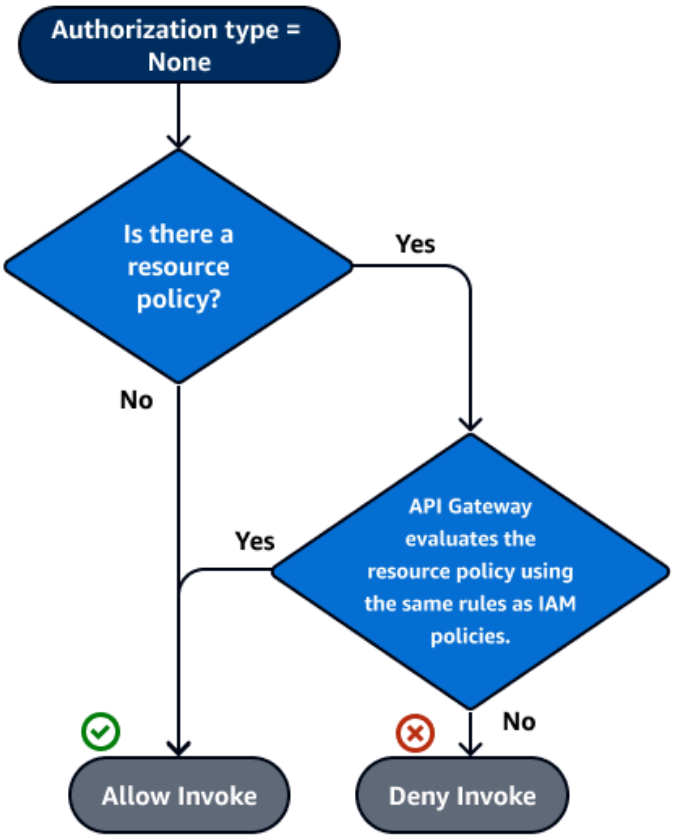
API Gateway 評估連接到您 API 的資源政策時，其結果會受到您為 API 定義的身分驗證類型所影響，如下節流程圖所示。

主題

- 僅限 API Gateway 資源政策
- Lambda 授權方和資源政策
- IAM 身分驗證和資源政策
- Amazon Cognito 身分驗證和資源政策
- 政策評估結果表

僅限 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 定義身分驗證類型。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。

JSON

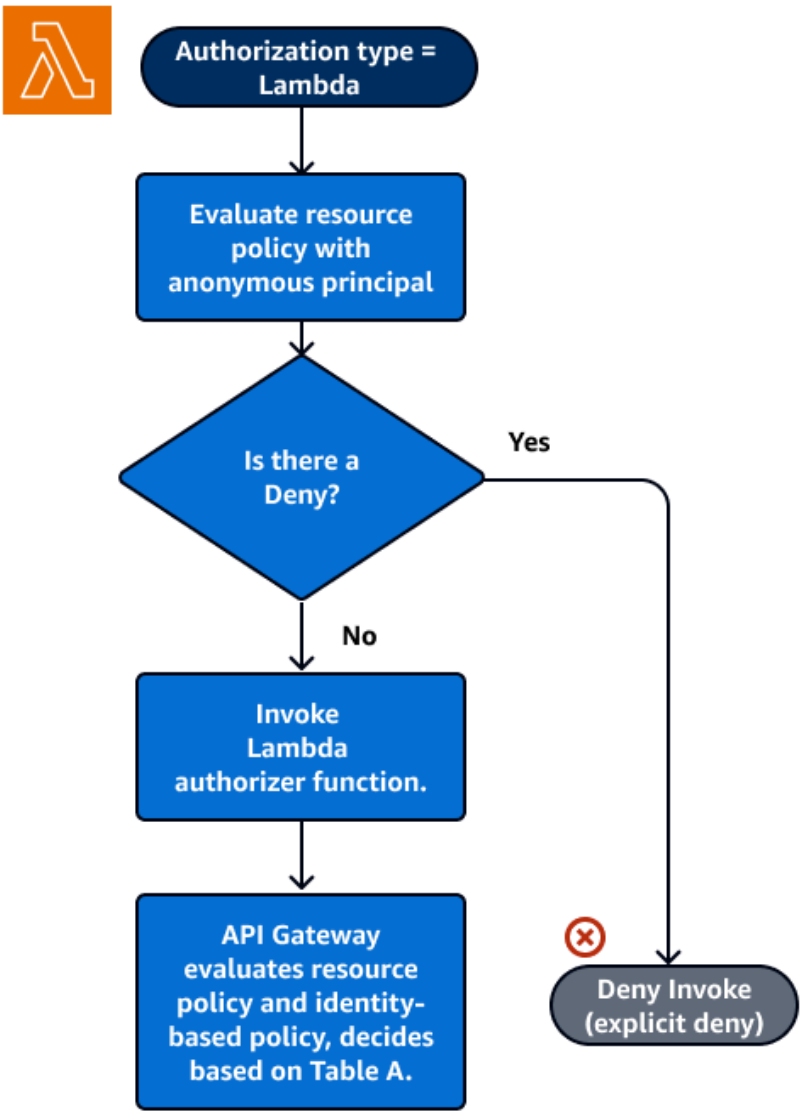
```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": "*",
            "Action": "execute-api:Invoke",
            "Resource": "arn:aws:execute-api:us-east-1:111111111111:api-id/",
            "Condition": {
                "IpAddress": {
                    "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。在呼叫 Lambda 授權方之前，API Gateway 會先評估政策並檢查是否有任何明確的拒絕。若有找到，會立即拒絕呼叫者存取。否則，會呼叫 Lambda 授權方，而它將會傳回政策文件，並與資源政策一起評估。如果您的授權方使用快取，API Gateway 可能會傳回快取的政策文件。結果會根據表 A 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，只會允許範例中來自以下指定之 VPC 端點的呼叫前進並評估 Lambda 授權方。所有剩餘的呼叫都會遭到封鎖。如果您針對私有 API 使用自訂網域名稱，則此授權工作流程相同。



JSON

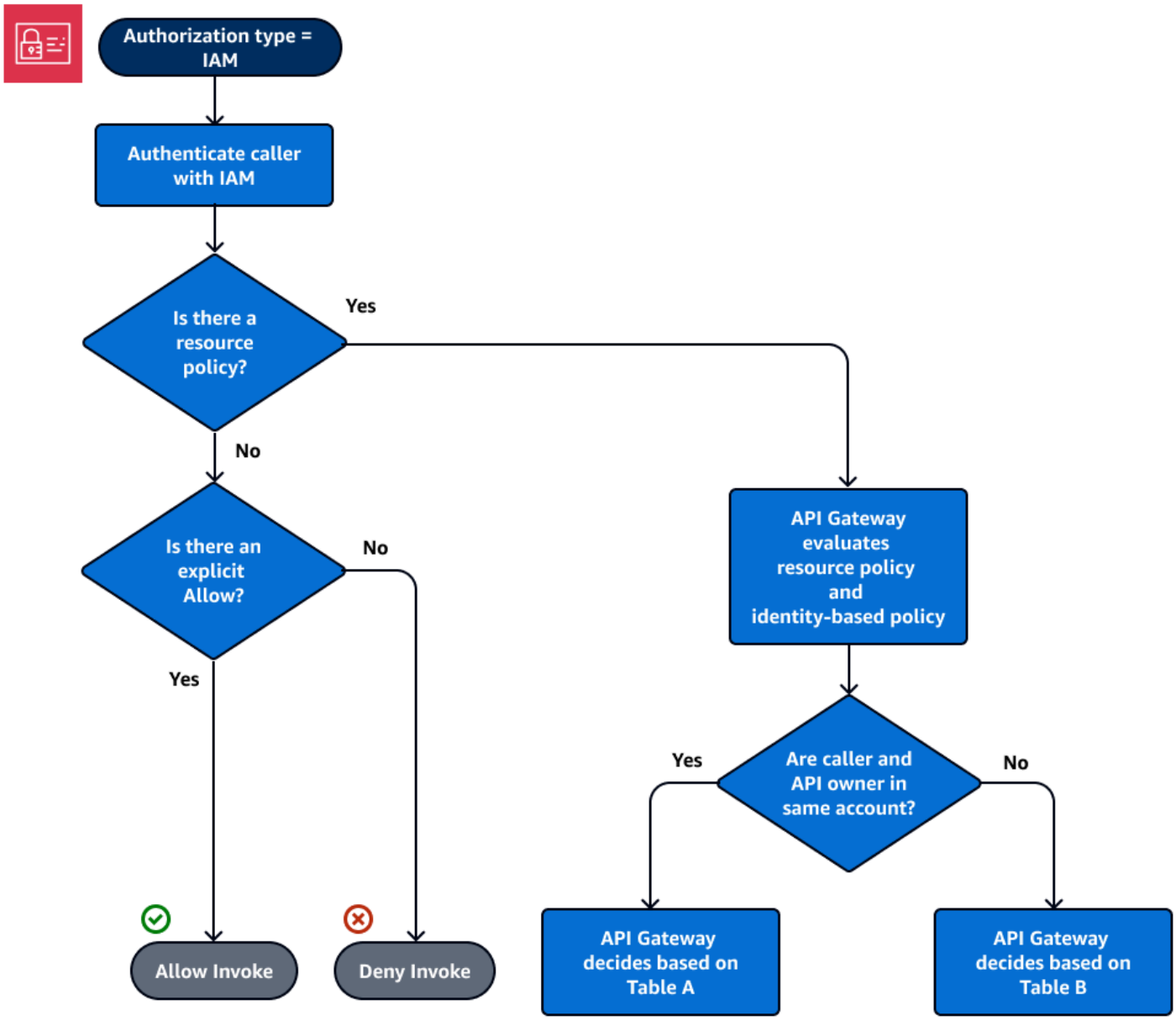
```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Principal": "*",
            "Action": "execute-api:Invoke",
            "Resource": [
                "arn:aws:execute-api:us-east-1:111111111111:api-id/"
            ],
            "Condition" : {
                "StringNotEquals": {
                    "aws:SourceVpce": "vpce-1a2b3c4d"
                }
            }
        }
    ]
}
```

IAM 身分驗證和資源政策

在此工作流程中，除了資源政策外，您也會為 API 設定 IAM 身分驗證。在您使用 IAM 服務對使用者進行身份驗證後，API 會評估連接到使用者的政策和資源政策。結果會根據發起人是位於 AWS 帳戶與 API 擁有者相同的 AWS 帳戶 還是單獨的 而有所不同。

如果發起人和 API 擁有者來自不同的帳戶，IAM 政策和資源政策都需明確允許發起人繼續。如需更多詳細資訊，請參閱表 B。

然而，如果發起人和 API 擁有者位於相同的 AWS 帳戶，則 IAM 使用者政策或資源政策必須明確允許發起人繼續。如需更多詳細資訊，請參閱表 A。




以下是跨帳戶資源政策的範例。假設 IAM 政策包含允許效果，則該資源政策僅允許來自 VPC ID 為 *vpc-2f09a348* 的 VPC 的呼叫。如需更多詳細資訊，請參閱表 B。

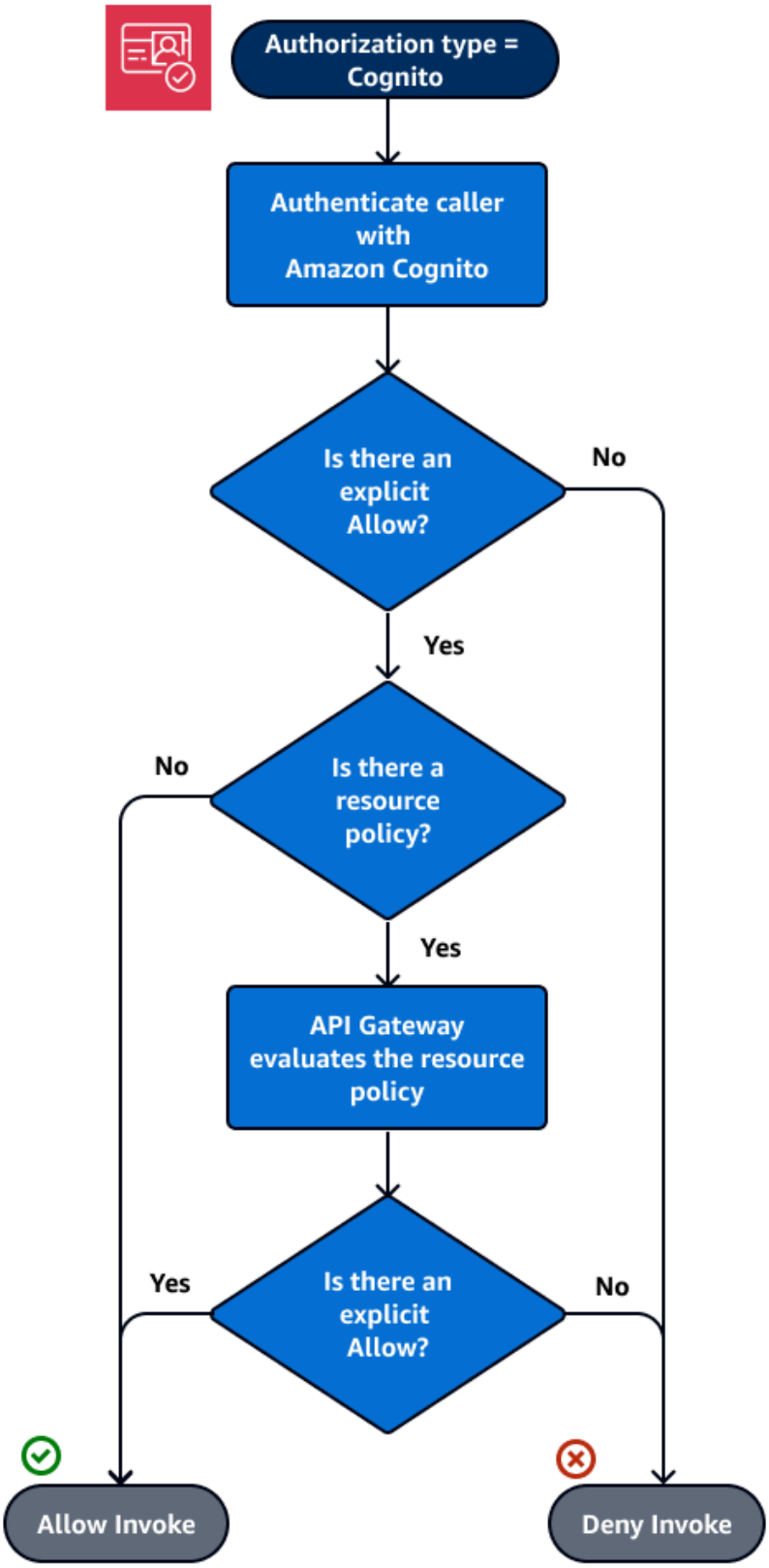
JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
```

```
            "Principal": "*",
            "Action": "execute-api:Invoke",
            "Resource": [
                "arn:aws:execute-api:us-east-1:111111111111:api-id/"
            ],
            "Condition" : {
                "StringEquals": {
                    "aws:SourceVpc": "vpc-2f09a348"
                    }
            }
        }
    ]
}
```

### Amazon Cognito 身分驗證和資源政策

在此工作流程中，除了資源政策之外，還會為 API 設定 Amazon Cognito 使用者集區。API Gateway 會先嘗試透過 Amazon Cognito 驗證發起人。這通常會透過發起人提供的 JWT 字符來執行。如果身分驗證成功，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 使用者集區使用的資源政策範例。


Authorization type = Cognito
Authenticate caller with Amazon Cognito
Is there an explicit Allow?
No
Yes
Is there a resource policy?
No
Yes
API Gateway evaluates the resource policy
Is there an explicit Allow?
Yes
No
Allow Invoke
Deny Invoke

以下是僅允許來自指定來源 IP 之呼叫的資源政策範例，其中假設 Amazon Cognito 身分驗證字符包含 Allow (允許)。如需更多詳細資訊，請參閱表 B。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": "*",
            "Action": "execute-api:Invoke",
            "Resource": "arn:aws:execute-api:us-east-1:111111111111: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 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) 時，所產生的行為。

| IAM 政策 (或 Lambda 授權方) | API Gateway 資源政策 | 產生行為 |
| --- | --- | --- |
| Allow | Allow | Allow |
| Allow | 不允許也不拒絕 | Allow |
| Allow | 拒絕 | 明確拒絕 |
| 不允許也不拒絕 | Allow | Allow |
| 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 |

| IAM 政策 (或 Lambda 授權方) | API Gateway 資源政策 | 產生行為 |
| --- | --- | --- |
| 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 |

表 B 列出 API Gateway API 的存取由 IAM 政策或 Amazon Cognito 使用者集區授權方和 API Gateway 資源政策控制時所產生的行為，這些政策位於不同位置 AWS 帳戶。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權需要資源政策和 IAM 政策或 Amazon Cognito 使用者集區授權方明確授與存取權。

| IAM 政策 (或 Amazon Cognito 使用者集區授權方) | API Gateway 資源政策 | 產生行為 |
| --- | --- | --- |
| Allow | Allow | Allow |
| Allow | 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。

下列範例政策使用簡化的語法來指定 API 資源。這種簡化的語法是您可以參考 API 資源的縮寫方式，而不是指定完整的 Amazon Resource Name (ARN)。當您儲存政策時，API Gateway 會將縮寫的語法轉換為完整的 ARN。例如，您可以在資源政策中指定資源 `execute-api:/stage-name/GET/pets`。當您儲存資源政策時，API Gateway 會將資源轉換為 `arn:aws:execute-api:us-east-2:123456789012:aabbccddee/stage-name/GET/pets`。API Gateway 會使用目前的區域、 AWS 您的帳戶 ID 和與資源政策相關聯的 REST API ID 來建置完整的 ARN。您可以使用 `execute-api:/*` 來表示目前 API 中的所有階段、方法和路徑。如需存取原則語言的詳細資訊，請參閱 Amazon API Gateway 的存取原則語言概觀。

主題

- 範例：允許另一個 AWS 帳戶中的角色使用 API
- 範例：拒絕根據來源 IP 地址或範圍的 API 流量
- 範例：使用私有 API 時，根據來源 IP 位址或範圍拒絕 API 流量
- 範例：允許以來源 VPC 或 VPC 端點為依據的私有 API 流量

### 範例：允許另一個 AWS 帳戶中的角色使用 API

下列範例資源政策會透過 Signature 第 4 版 (SigV4) 或 Signature 第 4a 版 (SigV4a) 通訊協定，將一個 AWS 帳戶中的 API 存取權授予不同 AWS 帳戶中的兩個角色。具體而言， 所識別 AWS 帳戶的開發人員和管理員角色*account-id-2*會獲授予 `execute-api:Invoke`動作，以對您 AWS 帳戶中`pets`的資源 (API) 執行 `GET`動作。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": [
                    "arn:aws:iam::111122223333:role/developer",
                    "arn:aws:iam::111122223333:role/Admin"
                ]
            },
            "Action": "execute-api:Invoke",
            "Resource": [
                "execute-api:/stage/GET/pets"
```

```
            ]
        }
    ]
}
```

範例：拒絕根據來源 IP 地址或範圍的 API 流量

以下範例資源政策會拒絕 (封鎖) 從兩個指定的來源 IP 位址區塊傳入 API 流量。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": "*",
            "Action": "execute-api:Invoke",
            "Resource": [
                "execute-api:/*"
            ]
        },
        {
            "Effect": "Deny",
            "Principal": "*",
            "Action": "execute-api:Invoke",
            "Resource": [
               "execute-api:/*"
            ],
            "Condition" : {
                "IpAddress": {
                    "aws:SourceIp": ["192.0.2.0/24", "198.51.100.0/24" ]
                }
            }
        }
    ]
}
```

如果您使用任何 IAM 使用者政策或 API Gateway 資源政策來控制 API Gateway 或任何 API Gateway API 的存取權，請確認您的政策已更新並包含 IPv6 位址範圍。若政策未更新，因而無法處理 IPv6 位

址，則可能在用戶端開始使用雙堆疊端點時，影響用戶端對 API Gateway 的存取。如需詳細資訊，請參閱the section called “在 IAM 原則中使用 IPv6 地址”。

範例：使用私有 API 時，根據來源 IP 位址或範圍拒絕 API 流量

以下範例資源政策會拒絕 (封鎖) 從兩個指定的來源 IP 位址區塊傳入 API 流量。使用私有 API 時，`execute-api` 的 VPC 端點會重新寫入原始來源 IP 位址。`aws:VpcSourceIp` 條件會根據原始請求者 IP 位址篩選請求。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": "*",
            "Action": "execute-api:Invoke",
            "Resource": [
                "execute-api:/*"
            ]
        },
        {
            "Effect": "Deny",
            "Principal": "*",
            "Action": "execute-api:Invoke",
            "Resource": [
               "execute-api:/*"
            ],
            "Condition" : {
                "IpAddress": {
                    "aws:VpcSourceIp": ["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 Virtual Private Cloud (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：

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": "*",
            "Action": "execute-api:Invoke",
            "Resource": [
                "execute-api:/*"
            ]
        },
        {
            "Effect": "Deny",
            "Principal": "*",
            "Action": "execute-api:Invoke",
            "Resource": [
                "execute-api:/*"
            ],
            "Condition" : {
                "StringNotEquals": {
                   "aws:SourceVpc": "vpc-1a2b3c4d"
                }
            }
        }
    ]
}
```

此範例資源政策會指定來源 VPC 端點：

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": "*",
```

```
            "Action": "execute-api:Invoke",
            "Resource": [
                "execute-api:/*"
            ]
        },
        {
            "Effect": "Deny",
            "Principal": "*",
            "Action": "execute-api:Invoke",
            "Resource": [
                "execute-api:/*"
            ],
            "Condition" : {
                "StringNotEquals": {
                    "aws:SourceVpce": "vpce-1a2b3c4d"
                }
            }
        }
    ]
}
```

建立 API Gateway 資源政策並連接至 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，並將政策連接到 API。當您將政策附加至 API，會將政策中的許可權套用至 API 中的方法。如果更新資源政策，您需要部署 API。

主題

- 先決條件
- 將資源政策連接至 API Gateway API
- 為資源政策進行疑難排解

先決條件

若要更新 API Gateway 資源政策，您需要 `apigateway:UpdateRestApiPolicy` 許可和 `apigateway:PATCH` 許可。

對於邊緣最佳化的或區域 API，您可以在建立資源政策時或部署之後，將資源政策連接至 API。對於私有 API，您無法在沒有資源政策的情況下部署 API。如需詳細資訊，請參閱the section called “私有 REST API”。

將資源政策連接至 API Gateway API

下列程序顯示如何將資源政策連接至 API Gateway API。

AWS 管理主控台

將資源政策連接至 API Gateway API

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 選擇 REST API。
3. 在主導覽窗格中，選擇資源政策。
4. 選擇建立政策。
5. (選用) 選擇選取範本以產生範例政策。

   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。
6. 如果您不使用其中一個範本範例，請輸入您的資源政策。
7. 選擇儲存變更。

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。

AWS CLI

若要使用 AWS CLI 建立新的 API 並連接資源政策，請使用下列 create-rest-api 命令：

```
aws apigateway create-rest-api \
    --name "api-name" \
    --policy "{\"jsonEscapedPolicyDocument\"}"
```

若要使用 AWS CLI 將資源政策連接至現有的 API，請使用下列 update-rest-api 命令：

```
aws apigateway update-rest-api \
    --rest-api-id api-id \
    --patch-operations op=replace,path=/
policy,value='"{\"jsonEscapedPolicyDocument\"}"'
```

您也可以將您的資源政策附加為不同的 `policy.json` 檔案，並將其包含在 create-rest-api 命令中。以下 create-rest-api 命令會建立具有資源政策的新 API：

```
aws apigateway create-rest-api \
```

```
    --name "api-name" \
    --policy file://policy.json
```

policy.json 是 API Gateway 資源政策，例如 the section called “範例：拒絕根據來源 IP 地址或範圍的 API 流量”。

AWS CloudFormation

您可以使用 CloudFormation 建立具有資源政策的 API。下列範例會使用範例資源政策 the section called “範例：拒絕根據來源 IP 地址或範圍的 API 流量” 建立 REST API。

```
AWSTemplateFormatVersion: 2010-09-09
Resources:
  Api:
    Type: 'AWS::ApiGateway::RestApi'
    Properties:
      Name: testapi
      Policy:
        Statement:
          - Action: 'execute-api:Invoke'
            Effect: Allow
            Principal: '*'
            Resource: 'execute-api:/*'
          - Action: 'execute-api:Invoke'
            Effect: Deny
            Principal: '*'
            Resource: 'execute-api:/*'
            Condition:
              IpAddress:
                'aws:SourceIp': ["192.0.2.0/24", "198.51.100.0/24" ]
        Version: 2012-10-17
  Resource:
    Type: 'AWS::ApiGateway::Resource'
    Properties:
      RestApiId: !Ref Api
      ParentId: !GetAtt Api.RootResourceId
      PathPart: 'helloworld'
  MethodGet:
    Type: 'AWS::ApiGateway::Method'
    Properties:
      RestApiId: !Ref Api
      ResourceId: !Ref Resource
      HttpMethod: GET
      ApiKeyRequired: false
```

```
      AuthorizationType: NONE
      Integration:
        Type: MOCK
        RequestTemplates:
          application/json: '{"statusCode": 200}'
        IntegrationResponses:
          - StatusCode: 200
            ResponseTemplates:
              application/json: '{}'
      MethodResponses:
        - StatusCode: 200
          ResponseModels:
            application/json: 'Empty'
  ApiDeployment:
    Type: 'AWS::ApiGateway::Deployment'
    DependsOn:
      - MethodGet
    Properties:
      RestApiId: !Ref Api
      StageName: test
```

### 為資源政策進行疑難排解

下列疑難排解指引可能有助於解決您的資源政策問題。

**我的 API 傳回 {"Message":"User: anonymous is not authorized to perform: execute-api:Invoke on resource: arn:aws:execute-api:us-east-1:********/****/****/"}**

在您的資源政策中，如果您將主體設定為 AWS 委託人，如下所示：

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": [
                    "arn:aws:iam::111111111111:role/developer",
                    "arn:aws:iam::111111111111:role/Admin"
                ]
```

```
            },
            "Action": "execute-api:Invoke",
            "Resource": [
                "execute-api:/stage/GET/pets"
            ]
        }
    ]
}
```

您必須對 API 中的每個方法使用 AWS_IAM 授權，否則您的 API 會傳回先前的錯誤訊息。如需有關如何為方法開啟 AWS_IAM 授權的詳細資訊，請參閱 the section called “方法”。

我的資源政策未更新

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 the section called “部署 REST API”。

我的資源政策傳回下列錯誤：政策文件無效。請檢查政策語法，並確保主體有效。

若要針對此錯誤進行疑難排解，建議您先檢查政策語法。如需詳細資訊，請參閱the section called “Amazon API Gateway 的存取原則語言概觀”。我們也建議您確認所有指定的主體是否有效，且尚未刪除。

此外，如果您的 API 位於選擇加入區域，請確認資源政策中的所有帳戶都已啟用區域。

AWS 可用於 API Gateway 資源政策的條件索引鍵

下表包含 AWS 條件索引鍵，可用於 APIs Gateway 中每個授權類型的 API 資源政策。

如需 AWS 條件金鑰的詳細資訊，請參閱AWS 全域條件內容金鑰。

| 條件索引鍵 | 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 |

| 條件索引鍵 | 條件 | 需要 **AuthN**？ | 授權類型 |
| --- | --- | --- | --- |
| aws:MultiFactorAuthPresent | 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:MultiFactorAuthAge | 金鑰僅在 MFA 存在於請求時才會出現。 | 是 | IAM |
| aws:PrincipalAccount | 無 | 是 | IAM |
| aws:PrincipalArn | 無 | 是 | IAM |
| aws:PrincipalOrgID | 只有在委託人是組織的成員時，請求內容中才會包含此金鑰。 | 是 | IAM |
| aws:PrincipalOrgPaths | 只有在委託人是組織的成員時，請求內容中才會包含此金鑰。 | 是 | IAM |
| aws:PrincipalTag | 若委託人搭配連接標籤使用 IAM 使用者，此鍵會包含在請求內容中。其會針對委託人，使用具備連接標籤或工作階段標籤的 IAM 角色包含在其中。 | 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 |

| 條件索引鍵 | 條件 | 需要 **AuthN**？ | 授權類型 |
| --- | --- | --- | --- |
| `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:VpcSourceIp` | 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 許可權控制 REST API 的存取

透過控制對下列兩個 API Gateway 元件程序的存取，您可以使用 IAM 許可控制對 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 之 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 的 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。您會將許可政策附加到使用者、角色或群組。

若要提供存取權，請新增權限至您的使用者、群組或角色：

- 中的使用者和群組 AWS IAM Identity Center：

  建立權限合集。請按照《AWS IAM Identity Center 使用者指南》中的建立權限合集說明進行操作。
- 透過身分提供者在 IAM 中管理的使用者：

  建立聯合身分的角色。遵循《IAM 使用者指南》的為第三方身分提供者 (聯合) 建立角色中的指示。
- IAM 使用者：
  - 建立您的使用者可擔任的角色。請按照《IAM 使用者指南》的為 IAM 使用者建立角色中的指示。
  - (不建議) 將政策直接附加至使用者，或將使用者新增至使用者群組。請遵循《IAM 使用者指南》的新增許可到使用者 (主控台) 中的指示。

如需如何使用此許可模型的詳細資訊，請參閱「the section called “API Gateway 身分類型政策”」。

用於呼叫 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`，以要求發起人提交使用者的憑證以進行身份驗證。API Gateway 支援第 4a 版簽署程序 (SigV4a) 和第 4 版簽署程序 (SigV4) 來驗證使用者的憑證。如需詳細資訊，請參閱 AWS 第 4 版簽署程序。然後您可以將政策連接至使用者、角色或群組。

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「控制對 API 的呼叫存取權」。

當 API 在後端與 AWS 服務 （ 例如， AWS Lambda) 整合時，API Gateway 也必須具有代表 API 呼叫者存取整合 AWS 資源 （ 例如，叫用 Lambda 函數 ） 的許可。若要授予這些許可，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時，所產生的角色會包含下列 IAM 信任政策，其中宣告 API Gateway 是允許擔任該角色的受信任實體：

JSON

```
{
  "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軟體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。請勿嘗試直接在 IAM 管理主控台或 calling AWS CLI create-policy 命令或對應的 SDK 方法中建立此類政策。

若要讓 API Gateway 呼叫整合 AWS 服務，您還必須將 連接到此角色，以呼叫整合 AWS 服務的合適 IAM 許可政策。例如，若要呼叫 Lambda 函數，您必須在 IAM 角色中納入下列 IAM 許可政策：

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "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，不會要求您明確設定此 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**
>
> 若要對 AWS 服務制定存取控制，您可以使用以發起人為基礎的許可模型，其中許可政策直接連接到發起人的使用者或群組，或以角色為基礎的許可模型，其中許可政策連接到 API Gateway 可以擔任的 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。您可以利用此功能，要求使用者只能透過 API Gateway API 存取 AWS 服務。

### 控制對 API 的呼叫存取權

在本節中，您會了解使用 IAM 許可控制 API 存取權的許可模型。啟用 IAM 授權時，用戶端必須使用 Signature 第 4a 版 (SigV4a) 和 Signature 第 4 版 (SigV4) 以 AWS 登入資料簽署其請求。如需詳細資訊，請參閱 AWS 第 4 版簽署程序。

在此區段中，我們會展示範本 IAM 政策聲明和政策聲明參考。政策聲明參考包括與 API 執行服務相關之 `Action` 與 `Resource` 欄位的格式。使用這些參考來建立您的 IAM 政策聲明。建立 IAM 政策聲明時，您可能需要考慮 API Gateway 資源政策影響授權工作流程的方式。如需詳細資訊，請參閱the section called “資源政策如何影響授權工作流程”。

處理私有 API 時，您應結合使用 API Gateway 資源政策與 VPC 端點政策。如需詳細資訊，請參閱下列主題：

- the section called “使用 API Gateway 資源政策”
- the section called “使用私有 API 的 VPC 端點政策”

### 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。

JSON

```
{
```

```
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Permission",
      "Action": [
        "execute-api:Execution-operation"
      ],
      "Resource": [
        "arn:aws:execute-api:region:123456789012: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。

> **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 身分驗證。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：

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "execute-api:Invoke"
      ],
      "Resource": [
```

```
        "arn:aws:execute-api:us-east-1:111111111111:api-id/*/GET/pets"
      ]
    },
    {
      "Effect": "Deny",
      "Action": [
        "execute-api:Invoke"
      ],
      "Resource": [
        "arn:aws:execute-api:us-east-1:111111111111: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：

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "execute-api:Invoke"
            ],
            "Resource": [
                "arn:aws:execute-api:us-east-1:111122223333:api-id/*/
GET/pets/a1b2"
            ]
        }
    ]
}
```

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`* 是對應至 方法之已部署 API AWS 的區域 （ 例如 **`us-east-1`**或**`*`**所有 AWS 區域的 )。
- *`account-id`* 是 REST API 擁有者的 12 位數 AWS 帳戶 ID。
- *`api-id`* 是 API Gateway 分配給該方法之 API 的標識符。
- *`stage-name`* 是與方法相關聯的階段名稱。
- *`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

> **Note**
>
> 如果您指定萬用字元 (*)，則 `Resource` 表達式會將萬用字元套用至表達式的其餘部分。

一些範例資源表達式包括：

- **arn:aws:execute-api:*:*:*** 用於任何階段中的任何資源路徑， 用於任何 AWS 區域中的任何 API。
- **arn:aws:execute-api:us-east-1:*:*** 對於任何階段中的任何資源路徑，對於 AWS 區域中的任何 APIus-east-1。
- **arn:aws:execute-api:us-east-1:*:*api-id*/*** 針對任何階段中的任何資源路徑，針對 us-east-1 AWS 區域中識別符為 *api-id* 的 API。
- **arn:aws:execute-api:us-east-1:*:*api-id*/test/***，表示 us-east-1 的 AWS 區域中識別符為 *api-id* 之 API `test` 階段的任何資源路徑。

如需詳細資訊，請參閱 API Gateway Amazon Resource Name (ARN) 參考資料。

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「控制對 API 的呼叫存取權」。

下列政策聲明會授予使用者許可，以呼叫識別符為 a123456789 之 API 的 `test` 階段中沿著 `mydemoresource` 路徑的任何 POST 方法 (假設對應的 API 已部署至 us-east-1 的 AWS 區域)：

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "execute-api:Invoke"
      ],
      "Resource": [
        "arn:aws:execute-api:us-east-1:*:a123456789/test/POST/my-demo-resource-
path/*"
      ]
    }
  ]
}
```

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 `petstorewalkthrough/pets` 之 API 的 a123456789 區域中，呼叫對應 API 之任何階段中資源路徑 AWS 上的任何方法：

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "execute-api:Invoke"
      ],
      "Resource": [
        "arn:aws:execute-api:*:*:a123456789/*/*/petstorewalkthrough/pets"
      ]
    }
  ]
}
```

## 在 API Gateway 中使用私有 API 的 VPC 端點政策

若要提高私有 API 的安全性，您可以建立 VPC 端點政策。VPC 端點政策為 IAM 資源政策，您可將其連接至 VPC 端點。如需詳細資訊，請參閱使用 VPC 端點控制服務的存取。

您可能會想要建立 VPC 端點政策來執行下列任務。

- 僅允許特定組織或資源存取您的 VPC 端點，並調用您的 API。
- 使用單一政策，避免使用工作階段型或角色型政策控制 API 的流量。
- 從現場部署遷移到 時，請緊固應用程式的安全周邊 AWS。

### VPC 端點政策考量事項

以下是 VPC 端點政策的考量事項：

- 呼叫者的身分是根據 `Authorization` 標頭值進行評估。首先會評估 VPC 端點政策，然後 API Gateway 會根據方法請求上設定的授權類型評估請求。下表顯示如何根據 `Authorization` 標頭值的內容評估 VPC 端點政策。

| **Authorization** 標頭值的內容 | 如何評估 VPC 端點政策 |
| --- | --- |
| 無內容 | 調用方會評估為匿名使用者 |
| 有效的 SigV4 或 SigV4a 簽章 | 調用方會評估為來自簽章的已驗證 IAM 身分 |
| 無效的 SigV4 或 SigV4a 簽章 | API Gateway 拒絕存取 |
| 非 SigV4 授權資訊，例如持有人權杖 | 調用方會評估為匿名使用者 |

- 如果您的存取控制取決於使用持有人權杖，例如 Lambda 或 Amazon Cognito 授權方，則您可以使用資源的屬性來控制您的安全邊界。
- 如果您的授權控制項使用 IAM 授權，則您可以使用資源的屬性和主體的屬性來控制您的安全邊界。
- VPC 端點政策可與 API Gateway 資源政策搭配使用。API Gateway 資源政策指定由哪個主體存取 API。端點政策指定誰可以存取 VPC，以及可以從 VPC 端點呼叫哪些 API。您的私有 API 需要資源政策，但您不需要建立自訂 VPC 端點政策。

VPC 端點政策範例

您可以為 Amazon API Gateway 建立 Amazon Virtual Private Cloud 端點的政策，您可以在其中指定如下：

- 可執行動作的主體。
- 可執行的動作。
- 可對其執行動作的資源。

這可能取決於授權標頭的內容。如需詳細資訊，請參閱the section called “VPC 端點政策考量事項”。如需其他範例政策，請參閱 GitHub 網站上的資料周界政策範例。

您需要使用 VPC 主控台，才能將政策連接至 VPC 端點。如需詳細資訊，請參閱使用 VPC 端點控制服務的存取。

範例 1：授予兩個 API 存取權限的 VPC 端點政策

以下範例政策會透過連接政策的 VPC 端點，僅授予使用者存取兩個特定 API 的權限。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Principal": "*",
            "Action": [
                "execute-api:Invoke"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:execute-api:us-east-1:123412341234:a1b2c3d4e5/*",
                "arn:aws:execute-api:us-east-1:123412341234:aaaaa11111/*"
            ]
        }
    ]
}
```

範例 2：授予 GET 方法存取權限的 VPC 端點政策

以下範例政策會透過連接政策的 VPC 端點，授予使用者存取特定 API GET 方法的權限。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Principal": "*",
            "Action": [
                "execute-api:Invoke"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:execute-api:us-east-1:123412341234:a1b2c3d4e5/stageName/
GET/*"
            ]
        }
    ]
```

```
}
```

範例 3：授予某個使用者特定 API 存取權限的 VPC 端點政策

以下範例政策會透過連接政策的 VPC 端點，授予某個使用者存取特定 API 的權限。

在此情況下，由於政策限制對特定 IAM 主體的存取，您必須將方法 `authorizationType` 設定為 `AWS_IAM` 或 `NONE`。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Principal": {
                "AWS": [
                    "arn:aws:iam::123412341234:user/MyUser"
                ]
            },
            "Action": [
                "execute-api:Invoke"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:execute-api:us-east-1:123412341234:a1b2c3d4e5/*"
            ]
        }
    ]
}
```

範例 4：VPC 端點政策授予使用者對於特定自訂網域名稱，和每個映射到網域的 API 存取權

以下範例政策會透過連接政策的 VPC 端點，授予使用者存取私有 API 之特定自訂網域名稱的權限。若使用此政策，只要使用者已在 VPC 端點與自訂網域名稱之間建立網域名稱存取關聯，並被授予調用自訂網域名稱和映射至自訂網域名稱的任何私有 API 的存取權，則使用者即可調用映射至此自訂網域名稱的任何 API。如需詳細資訊，請參閱the section called “私有 API 的自訂網域名稱”。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "*"
      },
      "Action": "execute-api:Invoke",
      "Resource": [
        "*"
      ],
       "Condition": {
        "ArnEquals": {
          "execute-api:viaDomainArn": "arn:aws:execute-api:us-
west-2:111122223333:/domainnames/private.test.com+f4g5h6"
        }
      }
    }
  ]
}
```

範例 5：授予或拒絕存取特定 API 和網域資源的 VPC 端點政策

下列範例政策會授予使用者對特定 API 和網域資源的存取權。若使用此政策，只要使用者已在 VPC 端點與自訂網域名稱之間建立網域名稱存取關聯，並被授予調用自訂網域名稱和映射至自訂網域名稱的任何私有 API 的存取權，使用者即可調用允許的私有 API 和網域資源。如需詳細資訊，請參閱the section called “私有 API 的自訂網域名稱”。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "*"
```

```
      },
      "Action": "execute-api:Invoke",
      "Resource": [
        "arn:aws:execute-api:us-west-2:111122223333:/domainnames/private.test.com
+f4g5h6",
        "arn:aws:execute-api:us-west-2:111122223333:a1b2c3d4e5/*"
      ]
    },
    {
      "Effect": "Deny",
      "Principal": {
        "AWS": "*"
      },
      "Action": "execute-api:Invoke",
      "Resource": [
        "arn:aws:execute-api:us-west-2:111122223333:a1b2c3d4e5/admin/*",
        "arn:aws:execute-api:us-west-2:111122223333:bcd123455/*"
      ]
    }
  ]
}
```

範例 6：授予或拒絕屬於組織的主體和資源進行存取的 VPC 端點政策

下列範例政策會將存取權授予屬於組織的主體和資源。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Condition": {
                "StringEquals": {
                    "aws:ResourceOrgID": "o-abcd1234",
                    "aws:PrincipalOrgID": "o-abcd1234"
                }
            },
            "Action": "*",
            "Resource": "*",
            "Effect": "Allow",
            "Principal": {
```

```
                "AWS": "*"
            },
            "Sid": "AllowRequestsByOrgsIdentitiesToOrgsResources"
        }
    ]
}
```

## 使用標籤控制 API Gateway 中的 REST API 存取權

您可以在 IAM 政策中使用屬性型存取控制，以微調對 REST API 的存取許可。

如需詳細資訊，請參閱the section called “屬性型存取控制”。

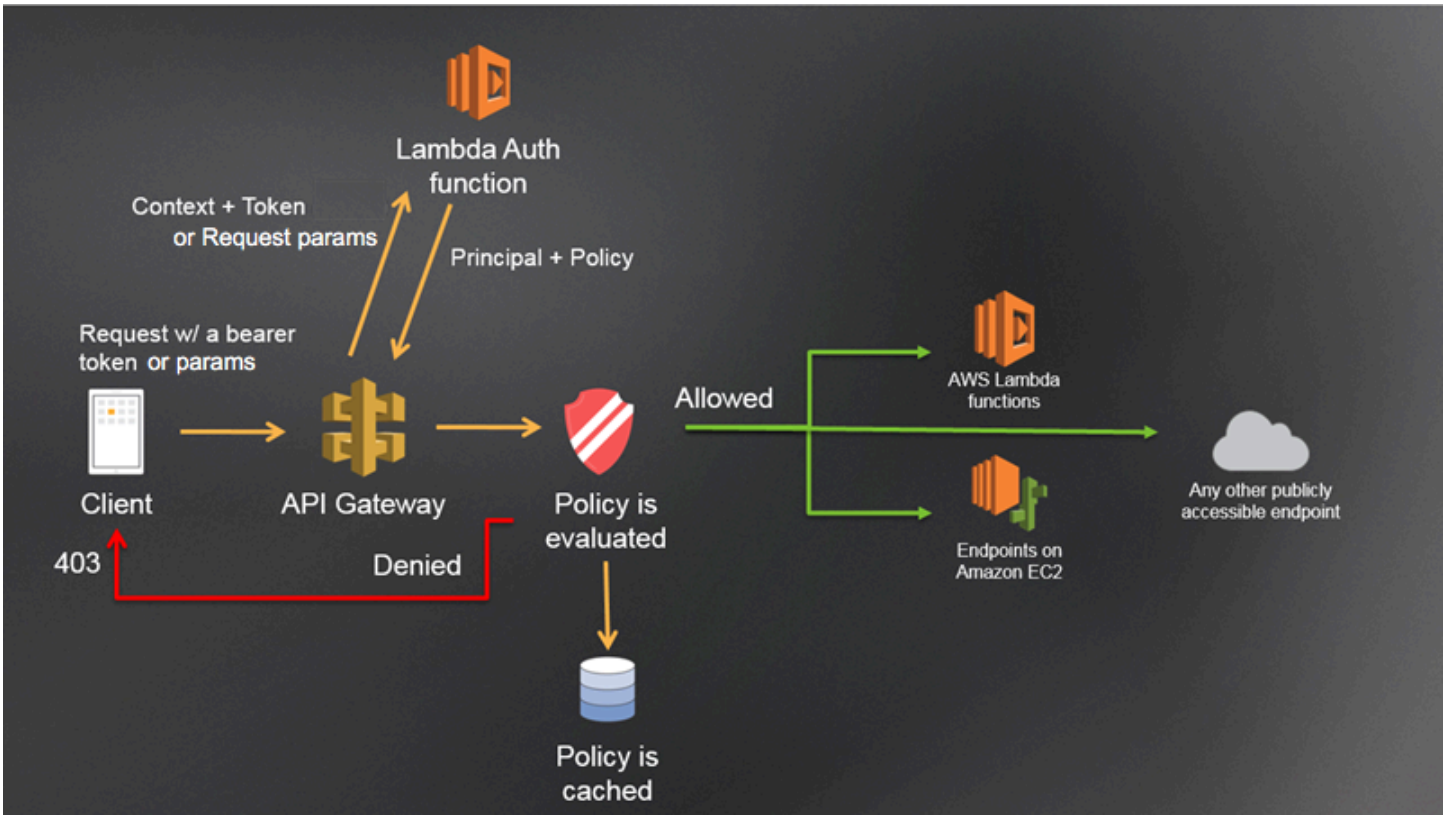
## 使用 API Gateway Lambda 授權方

使用 Lambda 授權方 (先前稱為自訂授權方) 控制存取 API。當用戶端請求 API 的方法時，API Gateway 會呼叫您的 Lambda 授權方。Lambda 授權方會將呼叫者的身分當作輸入項，並傳回 IAM 政策做為輸出。

使用 Lambda 授權方實作自訂授權機制。您的方案可以使用請求參數來判斷呼叫者的身分，或使用承載字符驗證策略，例如 OAuth 或 SAML。在 API Gateway REST API 主控台中使用 AWS CLI或 AWS SDK 建立 Lambda 授權方。

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. 用戶端在 API Gateway API 上呼叫方法，傳遞承載字符或請求參數。
2. API Gateway 會確認是否有使用 Lambda 授權方設定方法請求。如果已設定，API Gateway 將會呼叫 Lambda 函式。
3. Lambda 函數會驗證呼叫者。函數可以利用以下方式進行驗證：
   - 呼叫 OAuth 提供者以取得 OAuth 存取字符。
   - 呼叫 SAML 提供者以取得 SAML 聲明。
   - 根據請求參數值產生 IAM 政策。
   - 從資料庫擷取憑證。
4. Lambda 函數會傳回 IAM 政策和主體識別碼。如果 Lambda 函數未傳回該資訊，呼叫便失敗。
5. API Gateway 會評估 IAM 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 會調用該方法。

如果啟用授權快取，API Gateway 會快取該政策，如此就不需要再次調用 Lambda 授權方函式。請確認您的政策適用於 API 中的所有資源和方法。

您可以自訂 `403 ACCESS_DENIED` 或 `401 UNAUTHORIZED` 閘道回應。如需詳細資訊，請參閱 the section called “閘道回應”。

選擇 Lambda 授權方的類型

Lambda 授權方有兩種類型：

請求參數型 Lambda 授權方 (**REQUEST** 授權方)

REQUEST 授權方利用標頭、查詢字串參數、`stageVariables` 和 `$context` 變數的組合方式接呼叫者的身分。您可以使用 REQUEST 授權方，以根據來自多個身分來源的資訊建立精細政策，例如 `$context.path` 和 `$context.httpMethod` 內容變數。

如果您開啟 REQUEST 授權方的授權快取，API Gateway 會驗證請求中是否存在所有指定的身分來源。如果指定的身分來源遺漏、為 Null 或空白，API Gateway 會傳回 `401 Unauthorized` HTTP 回應，而無需呼叫 Lambda 授權方函數。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，並保留順序。您可以使用多個身分來源來定義精細快取金鑰。

如果您變更任何快取金鑰部分並重新部署您的 API，則授權方會捨棄快取的政策文件，並產生新的政策文件。

如果關閉 REQUEST 授權方的授權快取，則 API Gateway 會直接將請求傳遞給 Lambda 函數。

以字符為基礎的 Lambda 授權方 (**TOKEN** 授權方)

TOKEN 授權方會以承載字符接收呼叫者的身分，例如 JSON Web Token (JWT) 或 OAuth 字符。

如果您開啟 TOKEN 授權方的授權快取，則字符來源中指定的標頭名稱會成為快取金鑰。

此外，您可以使用字符驗證來輸入 RegEx 陳述式。API Gateway 會對此表達式執行輸入字符的初始驗證，並在成功驗證時調用 Lambda 授權方函數。這樣做有助於減少對 API 的呼叫。

`IdentityValidationExpression` 屬性僅支援 TOKEN 授權方。如需詳細資訊，請參閱the section called “`x-amazon-apigateway-authorizer`”。

> **Note**
>
> 建議您使用 REQUEST 授權方來控制對 API 的存取。與使用 TOKEN 授權方時的單一身分來源相比，您可以在使用 REQUEST 授權方時，根據多個身分來源控制對 API 的存取。此外，您可以使用 REQUEST 授權方的多個身分來源來分隔快取金鑰。

### **REQUEST** 授權方 Lambda 函數範例

下列範例程式碼會建立 Lambda 授權方函數，當用戶端提供的 HeaderAuth1 標頭、QueryString1 查詢參數和階段變數 StageVar1 分別符合 headerValue1、queryValue1 和 stageValue1 指定值時，允許請求。

Node.js

```
// 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.

export const handler = function(event, context, callback) {
    console.log('Received event:', JSON.stringify(event, null, 2));

    // 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.


    if (headers.HeaderAuth1 === "headerValue1"
        && queryStringParameters.QueryString1 === "queryValue1"
        && stageVariables.StageVar1 === "stageValue1") {
        callback(null, generateAllow('me', event.methodArn));
    }  else {
        callback(null, generateDeny('me', event.methodArn));
    }
}

// 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);
}
```

### Python

```
# 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.

def lambda_handler(event, context):
    print(event)

    # Retrieve request parameters from the Lambda function input:
    headers = event['headers']
    queryStringParameters = event['queryStringParameters']
    pathParameters = event['pathParameters']
    stageVariables = event['stageVariables']

    # Parse the input for the parameter values
    tmp = event['methodArn'].split(':')
    apiGatewayArnTmp = tmp[5].split('/')
    awsAccountId = tmp[4]
    region = tmp[3]
    restApiId = apiGatewayArnTmp[0]
    stage = apiGatewayArnTmp[1]
    method = apiGatewayArnTmp[2]
    resource = '/'

    if (apiGatewayArnTmp[3]):
        resource += apiGatewayArnTmp[3]

    # Perform authorization to return the Allow policy for correct parameters
    # and the 'Unauthorized' error, otherwise.

   if (headers['HeaderAuth1'] == "headerValue1" and
 queryStringParameters['QueryString1'] == "queryValue1" and
 stageVariables['StageVar1'] == "stageValue1"):
        response = generateAllow('me', event['methodArn'])
        print('authorized')
```

```
        return response
    else:
        print('unauthorized')
        response = generateDeny('me', event['methodArn'])
        return response
    # Help function to generate IAM policy


def generatePolicy(principalId, effect, resource):
    authResponse = {}
    authResponse['principalId'] = principalId
    if (effect and resource):
        policyDocument = {}
        policyDocument['Version'] = '2012-10-17'
        policyDocument['Statement'] = []
        statementOne = {}
        statementOne['Action'] = 'execute-api:Invoke'
        statementOne['Effect'] = effect
        statementOne['Resource'] = resource
        policyDocument['Statement'] = [statementOne]
        authResponse['policyDocument'] = policyDocument

    authResponse['context'] = {
        "stringKey": "stringval",
        "numberKey": 123,
        "booleanKey": True
    }

    return authResponse


def generateAllow(principalId, resource):
    return generatePolicy(principalId, 'Allow', resource)


def generateDeny(principalId, resource):
    return generatePolicy(principalId, 'Deny', resource)
```

在此範例中，Lambda 授權方函數會檢查輸入參數，並運作如下：

- 如果所有的必要參數值皆符合預期值，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：

JSON

```
{
  "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，而且方法請求失敗。

除了傳回 IAM 政策，Lambda 授權方函數還必須傳回發起人的主要識別符。也可以選擇傳回 `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，請參閱來自 API Gateway Lambda 授權方的輸出。

在生產程式碼中，您可能需要驗證使用者，再授予授權。您可以在 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驗證提供者。

**TOKEN** 授權方 Lambda 函數範例

下列範例程式碼會建立 `TOKEN` Lambda 授權方函數，允許呼叫者在用戶端提供的字符值為 `allow` 時，調用方法。如果字符值為 `deny`，則不允許呼叫者調用請求。如果字符值為 `unauthorized` 或空字串，則授權方函數會傳回 `401 UNAUTHORIZED` 回應。

Node.js

```
// 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.

export const 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;
```

```
}
```

### Python

```
# 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.

import json


def lambda_handler(event, context):
    token = event['authorizationToken']
    if token == 'allow':
        print('authorized')
        response = generatePolicy('user', 'Allow', event['methodArn'])
    elif token == 'deny':
        print('unauthorized')
        response = generatePolicy('user', 'Deny', event['methodArn'])
    elif token == 'unauthorized':
        print('unauthorized')
        raise Exception('Unauthorized')  # Return a 401 Unauthorized response
        return 'unauthorized'
    try:
        return json.loads(response)
    except BaseException:
        print('unauthorized')
        return 'unauthorized'  # Return a 500 error


def generatePolicy(principalId, effect, resource):
    authResponse = {}
    authResponse['principalId'] = principalId
    if (effect and resource):
```

```
        policyDocument = {}
        policyDocument['Version'] = '2012-10-17'
        policyDocument['Statement'] = []
        statementOne = {}
        statementOne['Action'] = 'execute-api:Invoke'
        statementOne['Effect'] = effect
        statementOne['Resource'] = resource
        policyDocument['Statement'] = [statementOne]
        authResponse['policyDocument'] = policyDocument
    authResponse['context'] = {
        "stringKey": "stringval",
        "numberKey": 123,
        "booleanKey": True
    }
    authResponse_JSON = json.dumps(authResponse)
    return authResponse_JSON
```

在此範例中，當 API 收到方法請求時，API Gateway 會在 `event.authorizationToken` 屬性中將來源字符傳遞至此 Lambda 授權方函數。Lambda 授權方函數會讀取字符，並運作如下：

- 如果字符值為 `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：

  JSON

  ```
  {
    "Version":"2012-10-17",
    "Statement": [
      {
        "Action": "execute-api:Invoke",
        "Effect": "Allow",
        "Resource": "arn:aws:execute-api:us-east-1:123456789012:ivdtdhp7b5/
  ESTestInvoke-stage/GET/"
      }
    ]
  }
  ```

- 如果字符值為 `deny`，授權方函數會傳回一個 `200 OK` HTTP 回應和一個如下的 Deny IAM 政策，而且方法請求會失敗：

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Action": "execute-api:Invoke",
      "Effect": "Deny",
      "Resource": "arn:aws:execute-api:us-east-1:123456789012:ivdtdhp7b5/
ESTestInvoke-stage/GET/"
    }
  ]
}
```

> **Note**
>
> 在測試環境之外，則 API Gateway 會傳回 `403 Forbidden` HTTP 回應，而且方法請求會失敗。

- 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。

除了傳回 IAM 政策，Lambda 授權方函數還必須傳回發起人的主要識別符。也可以選擇傳回 `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，請參閱來自 API Gateway Lambda 授權方的輸出。

在生產程式碼中，您可能需要驗證使用者，再授予授權。您可以在 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驗證提供者。

Lambda 授權方函數的其他範例

下列清單顯示 Lambda 授權方函數的其他範例。您可以在與您建立 API 的相同帳戶或不同帳戶中建立 Lambda 函數。

對於先前的範例 Lambda 函數，您可以使用內建的 AWSLambdaBasicExecutionRole，因為這些函數不會呼叫其他 AWS 服務。如果您的 Lambda 函數呼叫其他 AWS 服務，您將需要將 IAM 執行角色指派給 Lambda 函數。若要建立角色，請依照 AWS Lambda 執行角色中的指示。

Lambda 授權方函數的其他範例

- 如需範例應用程式，請參閱 GitHub 上的 Open Banking Brazil - Authorization Samples。
- 如需更多的範例 Lambda 函數，請參閱 GitHub 上的 aws-apigateway-lambda-authorizer-blueprints。
- 您可以建立 Lambda 授權方，以使用 Amazon Cognito 使用者集區對使用者進行驗證，並使用 Verified Permissions 根據政策存放區授權給呼叫者。如需詳細資訊，請參閱the section called “利用 Verified Permissions 根據身分的屬性控制存取權”。
- Lambda 主控台會提供 Python 藍圖，您可以選擇 Use a blueprint (使用藍圖)，並選擇 api-gateway-authorizer-python 藍圖來使用此藍圖。

設定 API Gateway Lambda 授權方

建立 Lambda 函數後，可以將 Lambda 函數設定為 API 的授權方。然後，可以設定方法來調用 Lambda 授權方，以判斷呼叫者是否可以調用您的方法。您可以在與您建立 API 的相同帳戶或不同帳戶中建立 Lambda 函數。

您可以使用 API Gateway 主控台的內建工具，或使用 Postman 來測試 Lambda 授權方。如需有關如何使用 Postman 測試 Lambda 授權方函數的指示，請參閱 the section called “使用 Lambda 授權方呼叫 API”。

設定 Lambda 授權方 (主控台)

下列程序顯示如何在 API Gateway REST API 主控台建立 Lambda 授權方。若要進一步了解不同類型的 Lambda 授權方，請參閱 the section called “選擇 Lambda 授權方的類型”。

REQUEST authorizer

設定 **REQUEST** Lambda 授權方

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 選取 API，然後選擇授權方。
3. 選擇建立授權方。
4. 針對授權方名稱，輸入授權方的名稱。
5. 針對授權方類型，選取 Lambda。
6. 針對 Lambda 函數，選取您建立 Lambda 授權方函數 AWS 區域 的 ，然後輸入函數名稱。
7. 將 Lambda 調用角色保留空白，以便讓 API Gateway REST API 主控台設定資源型政策。此政策會授與 API Gateway 許可，以調用 Lambda 授權方函數。您也可以選擇輸入 IAM 角色的

名稱，以允許 API Gateway 調用 Lambda 授權方函數。如需範例角色，請參閱 建立可擔任的 IAM 角色。

8. 針對 Lambda 事件承載，選取請求。
9. 針對身分來源類型，選取參數類型。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，請選擇新增參數。
10. 若要快取授權方產生的授權政策，請將授權快取保持開啟狀態。啟用政策快取時，您可以修改 TTL 值。將 TTL 設定為零會停用政策快取。

    如果啟用快取，您的授權方必須傳回適用於 API 之所有方法的政策。若要強制執行方法特定的政策，請使用內容變數 `$context.path` 和 `$context.httpMethod`。
11. 選擇建立授權方。

TOKEN authorizer

設定 **`TOKEN`** Lambda 授權方

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 選取 API，然後選擇授權方。
3. 選擇建立授權方。
4. 針對授權方名稱，輸入授權方的名稱。
5. 針對授權方類型，選取 Lambda。
6. 針對 Lambda 函數，選取您建立 Lambda 授權方函數 AWS 區域 的 ，然後輸入函數名稱。
7. 將 Lambda 調用角色保留空白，以便讓 API Gateway REST API 主控台設定資源型政策。此政策會授與 API Gateway 許可，以調用 Lambda 授權方函數。您也可以選擇輸入 IAM 角色的名稱，以允許 API Gateway 調用 Lambda 授權方函數。如需範例角色，請參閱 建立可擔任的 IAM 角色。
8. 針對 Lambda 事件承載，選取權杖。
9. 針對權杖來源，輸入包含授權權杖的標頭名稱。呼叫者必須包含此名稱的標頭，才能將授權字符傳送到 Lambda 授權方。
10. 或者，針對符記驗證，輸入 RegEx 陳述式。API Gateway 會對此表達式執行輸入字符的初始驗證，並在成功驗證時調用授權方。
11. 若要快取授權方產生的授權政策，請將授權快取保持開啟狀態。啟用政策快取時，權杖來源中指定的標頭名稱會成為快取金鑰。啟用政策快取時，您可以修改 TTL 值。將 TTL 設定為零會停用政策快取。

如果啟用快取，您的授權方必須傳回適用於 API 之所有方法的政策。若要強制執行方法特定政策，您可以關閉授權快取。

12. 選擇建立授權方。

建立 Lambda 授權方之後，您可以測試該授權方。下列程序顯示如何測試您的 Lambda 授權方。

REQUEST authorizer

測試 **REQUEST** Lambda 授權方

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 選取授權方的名稱。
3. 在測試授權方下，輸入身分來源的值。

   如果您使用的是 the section called “REQUEST 授權方 Lambda 函數範例”，請執行以下操作：

   a. 選取標頭並輸入 **headerValue1**，然後選擇新增參數。
   b. 在身分來源類型底下，選取查詢字串並輸入 **queryValue1**，然後選擇新增參數。
   c. 在身分來源類型底下，選取階段變數並輸入 **stageValue1**。

   您無法修改測試調用的內容變數，但您可以修改 Lambda 函數的 API Gateway Authorizer 測試事件範本。然後，您可以使用修訂內容變數來測試 Lambda 授權方函數。如需詳細資訊，請參閱《AWS Lambda 開發人員指南》中的在主控台測試 Lambda 函數。
4. 選擇測試授權方。

TOKEN authorizer

測試 **TOKEN** Lambda 授權方

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 選取授權方的名稱。
3. 在測試授權方下，輸入字符的值。

   如果您使用的是 the section called “TOKEN 授權方 Lambda 函數範例”，請執行以下操作：

   - 對於 authorizationToken 值，輸入 **allow**。

4. 選擇測試授權方。

   如果您的 Lambda 授權方成功拒絕測試環境中的請求，測試會以 `200 OK` HTTP 回應來回應。然而，在測試環境之外，API Gateway 會傳回 `403 Forbidden` HTTP 回應，方法請求失敗。

### 設定 Lambda 授權方 (AWS CLI)

以下 create-authorizer 命令指示使用 AWS CLI建立 Lambda 授權方。

#### REQUEST authorizer

以下 create-authorizer 命令會建立 `REQUEST` 授權方，並使用 `Authorizer` 標頭和 `accountId` 內容變數作為身分來源：

```
aws apigateway create-authorizer \
    --rest-api-id 1234123412 \
    --name 'First_Request_Custom_Authorizer' \
    --type REQUEST \
    --authorizer-uri 'arn:aws:apigateway:us-west-2:lambda:path/2015-03-31/functions/
arn:aws:lambda:us-west-2:123412341234:function:customAuthFunction/invocations' \
    --identity-source 'method.request.header.Authorization,context.accountId' \
    --authorizer-result-ttl-in-seconds 300
```

#### TOKEN authorizer

以下 create-authorizer 命令會建立 `TOKEN` 授權方，並使用 `Authorization` 標頭作為身分來源：

```
aws apigateway create-authorizer \
    --rest-api-id 1234123412 \
    --name 'First_Token_Custom_Authorizer' \
    --type TOKEN \
    --authorizer-uri 'arn:aws:apigateway:us-west-2:lambda:path/2015-03-31/functions/
arn:aws:lambda:us-west-2:123412341234:function:customAuthFunction/invocations' \
    --identity-source 'method.request.header.Authorization' \
    --authorizer-result-ttl-in-seconds 300
```

建立 Lambda 授權方之後，您可以測試該授權方。以下 test-invoke-authorizer 命令會測試 Lambda 授權方：

```
aws apigateway test-invoke-authorizer --rest-api-id 1234123412 \
```

```
    --authorizer-id efg1234 \
    --headers Authorization='Value'
```

設定使用 Lambda 授權方 (主控台) 的方法

設定 Lambda 授權方之後，您必須將其連接至 API 的方法。如果您的授權方使用授權快取，請務必更新政策以控制其他方法的存取。

設定 API 方法使用 Lambda 授權方

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 選取 API。
3. 選擇資源，然後選擇新方法，或選擇現有方法。
4. 在方法請求索引標籤的方法請求設定下，選擇編輯。
5. 針對授權方，從下拉式選單選取您剛才建立的 Lambda 授權方。
6. (選用) 如果您要將授權權杖傳遞至後端，請選擇 HTTP 請求標頭。選擇新增標頭，然後新增授權標頭的名稱。在名稱中，輸入標頭名稱，此名稱須符合您在建立 API 的 Lambda 授權方時指定的符記來源名稱。此步驟不適用於 REQUEST 授權方。
7. 選擇儲存。
8. 選擇部署 API 將 API 部署到階段。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，並在階段編輯器頁面上指定這些變數的值。

設定使用 Lambda 授權方的方法 (AWS CLI)

設定 Lambda 授權方之後，您必須將其連接至 API 的方法。您可以建立新的方法，或使用修補程式操作，將授權方連接到現有方法。如果您的授權方使用授權快取，請務必更新政策以控制其他方法的存取。

以下 put-method 命令會建立使用 Lambda 授權方的新方法：

```
aws apigateway put-method --rest-api-id 1234123412 \
  --resource-id a1b2c3 \
  --http-method PUT \
  --authorization-type CUSTOM \
  --authorizer-id efg1234
```

以下 update-method 命令會更新現有方法以使用 Lambda 授權方：

```
aws apigateway update-method \
    --rest-api-id 1234123412 \
    --resource-id a1b2c3 \
    --http-method PUT \
    --patch-operations op="replace",path="/authorizationType",value="CUSTOM"
 op="replace",path="/authorizerId",value="efg1234"
```

輸入至 API Gateway Lambda 授權方

下一節說明從 API Gateway 到 Lambda 授權方的輸入格式。

**TOKEN** 輸入格式

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，它會傳遞字符作為 Lambda 函數之 `event` 物件的 `authorizationToken` 屬性，並傳遞方法 ARN 作為 `methodArn` 屬性：

```
{
    "type":"TOKEN",
    "authorizationToken":"{caller-supplied-token}",
    "methodArn":"arn:aws:execute-api:{regionId}:{accountId}:{apiId}/{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}`* 源自用戶端要求的授權標頭，可以是任何字串值。`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。

**REQUEST** 輸入格式

針對 REQUEST 類型的 Lambda 授權方，API Gateway 會將請求參數傳遞到授權方 Lambda 函數，做為 `event` 物件的一部分。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**
>
> 路徑參數可作為請求參數傳遞到 Lambda 授權方函數，但無法作為身分來源使用。

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:abcdef123/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": "..."
  },
  "queryStringParameters": {
    "QueryString1": "queryValue1"
  },
  "pathParameters": {},
  "stageVariables": {
    "StageVar1": "stageValue1"
  },
  "requestContext": {
    "path": "/request",
    "accountId": "123456789012",
    "resourceId": "05c7jb",
    "stage": "test",
    "requestId": "...",
    "identity": {
      "apiKey": "...",
      "sourceIp": "...",
      "clientCert": {
        "clientCertPem": "CERT_CONTENT",
        "subjectDN": "www.example.com",
        "issuerDN": "Example issuer",
        "serialNumber": "a1:a1:a1:a1:a1:a1:a1:a1:a1:a1:a1:a1:a1:a1:a1:a1",
        "validity": {
          "notBefore": "May 28 12:30:02 2019 GMT",
          "notAfter": "Aug  5 09:36:04 2021 GMT"
        }
      }
```

```
    },
    "resourcePath": "/request",
    "httpMethod": "GET",
    "apiId": "abcdef123"
  }
}
```

requestContext 是鍵值對的對應，並對應到 $context 變數。它的結果是與 API 相關。

API Gateway 可能會將新的金鑰加入至地圖。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。

來自 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，其中必須包含主要識別符 (principalId) 以及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。

JSON

```
{
  "principalId": "yyyyyyyy",
  "policyDocument": {
    "Version":"2012-10-17",
    "Statement": [
      {
        "Action": "execute-api:Invoke",
        "Effect": "Allow|Deny",
        "Resource": "arn:aws:execute-
api:{regionId}:{accountId}:{apiId}/{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)。您可能需要根據您的授權方來控制對多個資源的存取。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」。

針對啟用授權的方法 ARN (例如 arn:aws:execute-api:*{regionId}*:*{accountId}*:*{apiId}*/*{stage}*/*{httpVerb}*/[*{resource}*/[*{child-resources}*]])，長度上限為 1600 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。如需詳細資訊，請參閱 資料轉換的內容變數。

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。

對於 Lambda 代理整合，API Gateway 會將 context 物件從 Lambda 授權方直接傳遞至後端 Lambda 函式，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 索引鍵/值組。

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 “用量計劃”。

以下顯示範例 Lambda 授權方的範例輸出。範例輸出包含政策陳述式，可針對 AWS 帳戶 (Deny) 的 API () dev階段封鎖 (ymy8tbxw7b) 對 GET 方法的呼叫123456789012。

JSON

```
{
  "principalId": "user",
  "policyDocument": {
    "Version":"2012-10-17",
    "Statement": [
      {
        "Action": "execute-api:Invoke",
        "Effect": "Deny",
        "Resource": "arn:aws:execute-api:us-west-2:123456789012:ymy8tbxw7b/dev/
GET/"
      }
    ]
  }
}
```

使用 API Gateway Lambda 授權方呼叫 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。

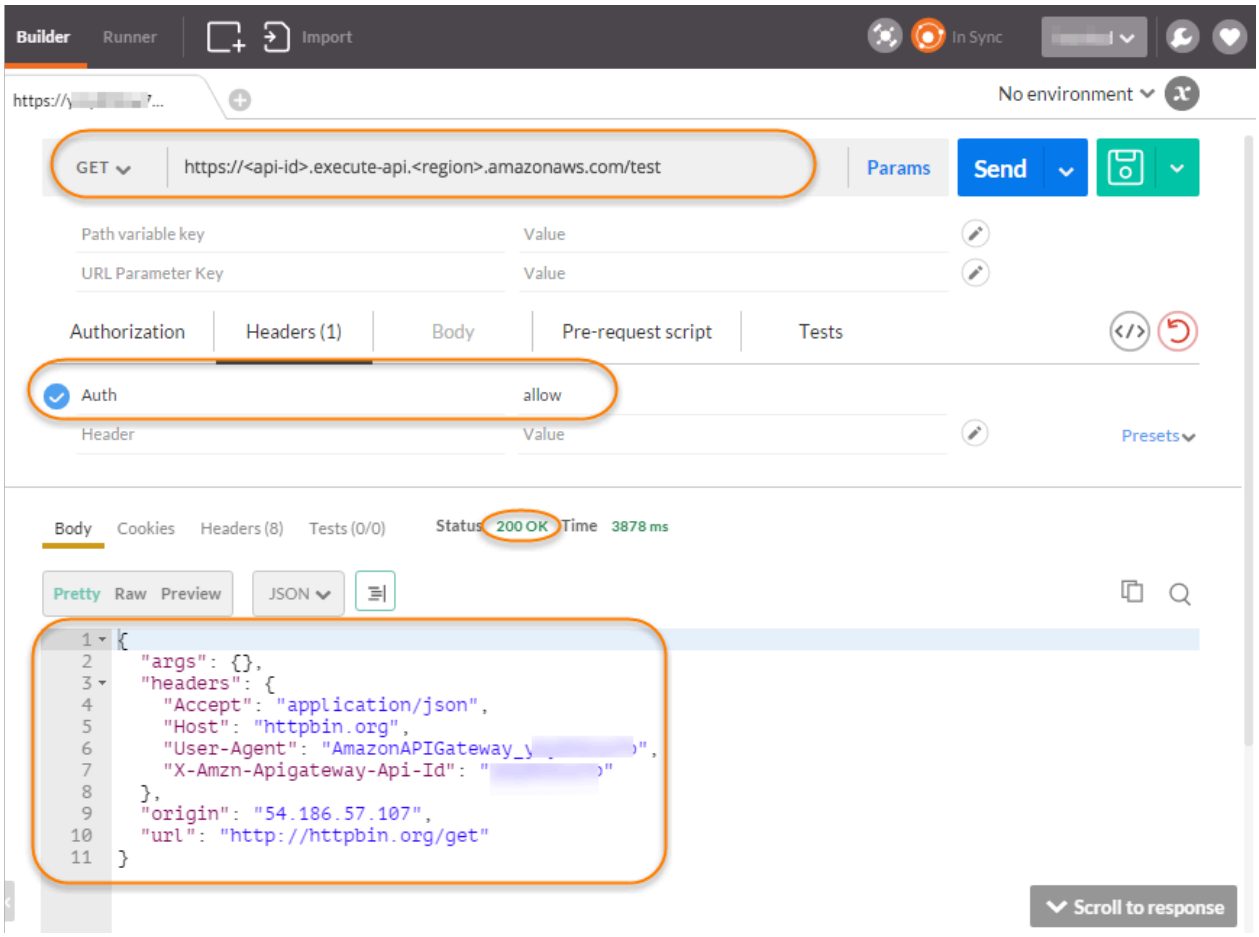
> **Note**
>
> 呼叫啟用授權方的方法時，如果 TOKEN 授權方的必要字符未設定、為 Null 或因指定的 Token validation expression (字符驗證表達式) 而失效，則 API Gateway 不會將呼叫記錄到 CloudWatch。同樣地，如果 REQUEST 授權方的任何必要身分來源未設定、為 Null 或空白，API Gateway 不會將呼叫記錄到 CloudWatch。

以下示範如何使用 Postman 來呼叫或測試具有 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

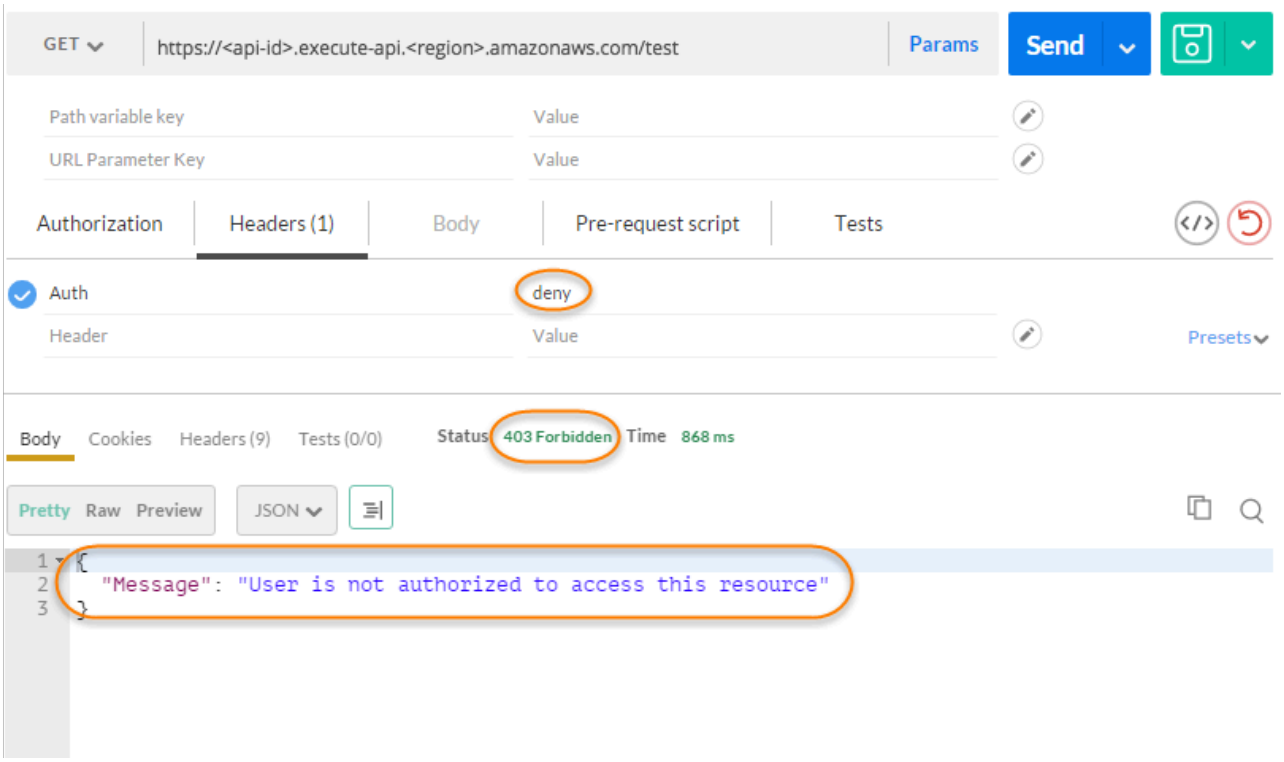
1. 開啟 Postman，選擇 GET 方法，然後將 API 的 Invoke URL (呼叫 URL) 貼到相鄰的 URL 欄位中。

新增 Lambda 授權字符標頭，並將值設定為 `allow`。選擇 Send (傳送)。



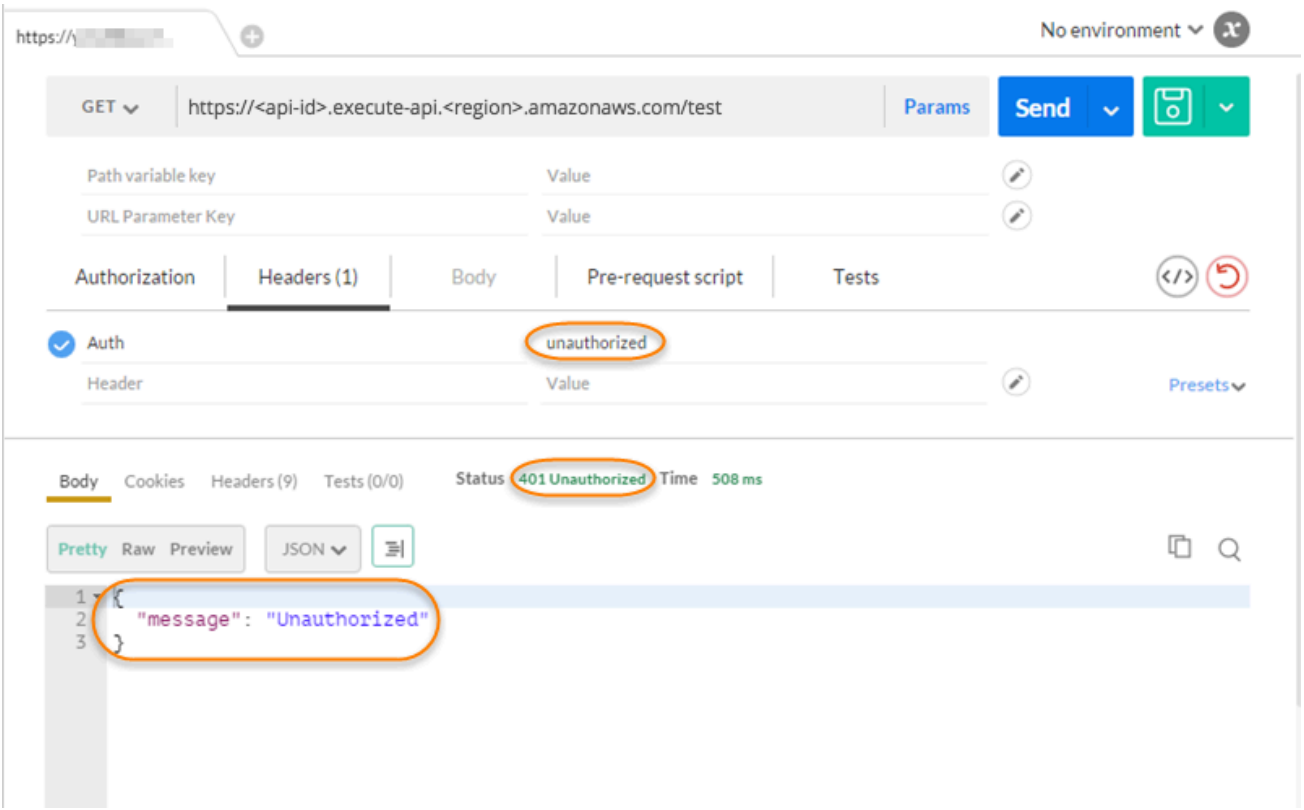

回應顯示 API Gateway Lambda 授權方傳回 200 OK (200 OK) 回應，並成功授權呼叫存取與方法整合的 HTTP 端點 (http://httpbin.org/get)。

2. 同樣在 Postman 中，將 Lambda 授權字符標頭值變更為 `deny`。選擇 Send (傳送)。



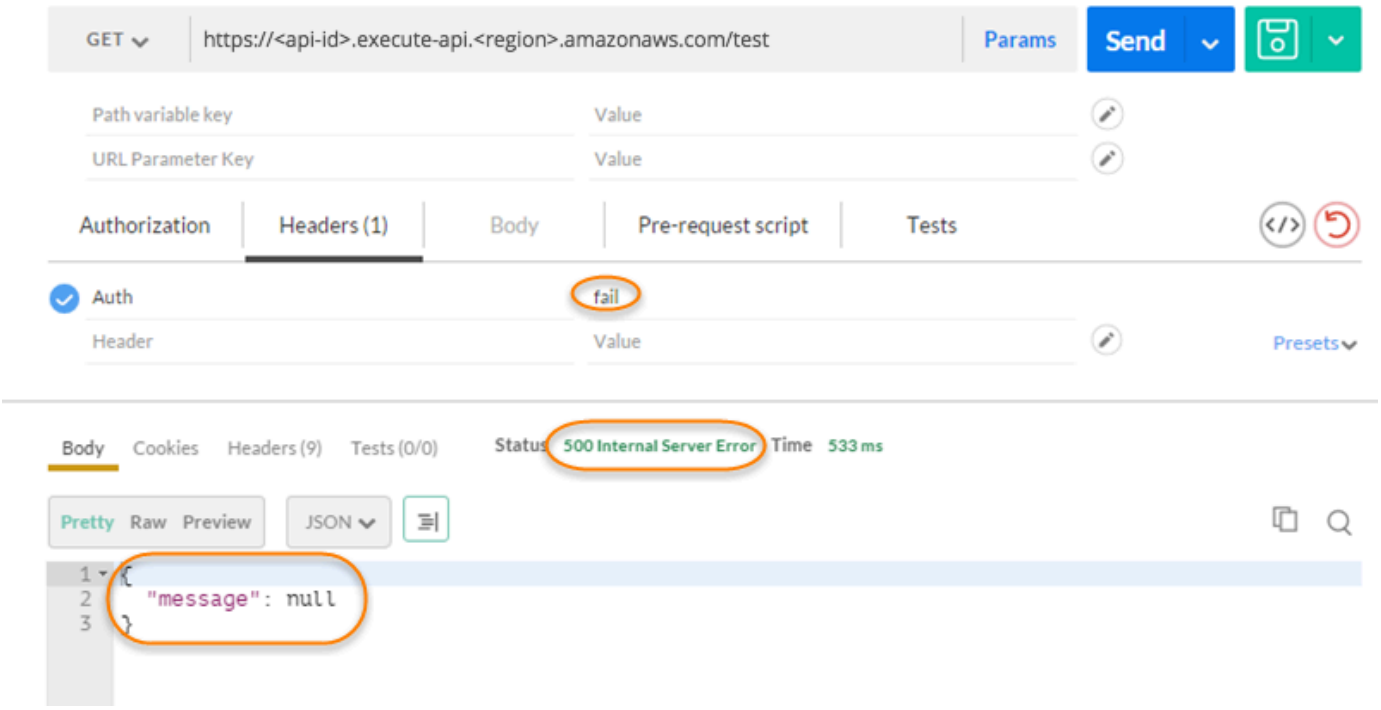

回應顯示 API Gateway Lambda 授權方傳回 403 Forbidden (403 禁止) 回應，而不授權呼叫存取 HTTP 端點。

3. 在 Postman 中，將 Lambda 授權字符標頭值變更為 `unauthorized`，然後選擇 Send (傳送)。




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 API Gateway Lambda 授權方

您現在可以使用來自不同 AWS 帳戶的 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符身分驗證策略，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，我們將示範如何使用 Amazon API Gateway 主控台設定跨帳戶 Lambda 授權方功能。

這些指示假設您在一個 AWS 帳戶中已有 API Gateway API，而另一個帳戶中已有 Lambda 授權方函數。

#### 使用 API Gateway 主控台設定跨帳戶 Lambda 授權方

在您 API 所在的帳戶中，登入 Amazon API Gateway 主控台，然後執行下列操作：

1. 選擇您的 API，然後在主導覽窗格中選擇授權方。
2. 選擇建立授權方。
3. 針對授權方名稱，輸入授權方的名稱。
4. 針對授權方類型，選取 Lambda。
5. 對於 Lambda 函數，輸入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，您可以在主控台視窗的右上角找到適用於您函數的 ARN。

6. 此時會出現包含 `aws lambda add-permission` 命令的警告。此政策會授與 API Gateway 許可，以調用授權方 Lambda 函數。複製指令並儲存，以供日後使用。建立授權方之後，您可以執行命令。
7. 針對 Lambda 事件承載，選擇權杖代表 `TOKEN` 授權方，或選擇請求代表 `REQUEST` 授權方。
8. 根據上一個步驟的選擇，執行下列其中一項操作：

   a. 針對權杖選項，執行下列操作：

- 針對權杖來源，輸入包含授權權杖的標頭名稱。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。
- 或者，針對權杖驗證，輸入 RegEx 陳述式。API Gateway 會對此表達式執行輸入字符的初始驗證，並在成功驗證時調用授權方。這樣做有助於減少對 API 的呼叫。
- 若要快取授權方產生的授權政策，請將授權快取保持開啟狀態。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，權杖來源中指定的標頭名稱會成為快取金鑰。如果在請求中將多個值傳遞給此標頭，則所有值都將成為快取金鑰，並會保留順序。

> **Note**
>
> 預設 TTL 值為 300 秒。最大值為 3600 秒，而且無法增加此限制。

b. 針對 請求 選項，執行下列操作：

- 針對身分來源類型，選取參數類型。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，請選擇新增參數。
- 若要快取授權方產生的授權政策，請將授權快取保持開啟狀態。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。

  API Gateway 使用指定的身分來源作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 或是空白，則 API Gateway 會傳回 `401 Unauthorized` 回應，而不會呼叫授權方 Lambda 函式。

  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。如果在請求中傳遞具有多個值的標頭，則所有值將成為快取金鑰的一部分，並會保留順序。
- 快取關閉時，不需指定身分來源。

> **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。若要強制執行方法特定政策，您可以關閉授權快取。

9. 選擇建立授權方。

10. 將您在上一個步驟中複製的`aws lambda add-permission`命令字串貼到為第二個帳戶設定的 AWS CLI 視窗中。將授權方 ID 取代為 `AUTHORIZER_ID` 。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。

利用 Verified Permissions 根據身分的屬性控制存取權

使用 Amazon Verified Permissions 控制對 API Gateway API 的存取。當您搭配 Verified Permissions 使用 API Gateway 時，Verified Permissions 會建立 Lambda 授權方，該授權方會使用精細的授權決策來控制對 API 的存取。Verified Permissions 會根據政策存放區結構描述和政策，使用 Cedar 政策語言來授權呼叫者，以定義應用程式使用者的精細許可權。如需詳細資訊，請參閱《Amazon Verified Permissions 使用者指南》中的使用連線 API 和身分提供者建立政策存放區。

Verified Permissions 支援 Amazon Cognito 使用者集區或 OpenID Connect (OIDC) 身分提供者做為身分來源。Verified Permissions 假設主體先前已識別，並已驗證身分。Verified Permissions 僅支援區域和邊緣最佳化的 REST API。

使用 Verified Permissions 建立 Lambda 授權方

Verified Permissions 會建立 Lambda 授權方，用以判斷主體是否可在 API 上採取行動。您可以建立 Verified Permissions 用來執行其授權任務的 Cedar 政策。

以下是一個 Cedar 策略範例，允許 `developer` 群組基於 Amazon Cognito 使用者集區 `us-east-1_ABC1234` 對 API 的 `GET /users` 資源調用 API。Verified Permissions 會剖析呼叫者身分的承載字符來判定群組成員資格。

```
permit(
  principal in MyAPI::UserGroup::"us-east-1_ABC1234|developer",
  action in [ MyAPI::Action::"get /users" ],
  resource
  );
```

或者，Verified Permissions 可以將授權方附加至 API 的方法。在 API 的生產階段，建議您不要允許 Verified Permissions 為您附加授權方。

下列清單顯示如何設定 Verified Permissions，以附加或不附加 Lambda 授權方至 API 方法的方法請求。

為您附加授權方 (AWS 管理主控台)

當您在 Verified Permissions 主控台選擇建立政策存放區時，請在部署應用程式整合頁面上選擇現在。

請勿為您附加授權方 (AWS 管理主控台)

當您在 Verified Permissions 主控台選擇建立政策存放區時，請在部署應用程式整合頁面上選擇稍後。

Verified Permissions 仍會為您建立 Lambda 授權方。Lambda 授權方以 `AVPAuthorizerLambda-` 開頭。如需有關如何在方法上附加授權方的詳細資訊，請參閱 the section called “設定使用 Lambda 授權方 (主控台) 的方法”。

為您附加授權方 (CloudFormation)

在 Verified Permissions 產生的 CloudFormation 範本中，在 `Conditions`區段中，將 `"Ref": "shouldAttachAuthorizer"`設定為 `true`。

請勿為您附加授權方 (CloudFormation)

在 Verified Permissions 產生的 CloudFormation 範本中，在 `Conditions`區段中，將 `"Ref": "shouldAttachAuthorizer"`設定為 `false`。

Verified Permissions 仍會為您建立 Lambda 授權方。Lambda 授權方以 `AVPAuthorizerLambda-` 開頭。如需有關如何在方法上附加授權方的詳細資訊，請參閱 the section called “設定使用 Lambda 授權方的方法 (AWS CLI)”。

使用 Verified Permissions 呼叫 Lambda 授權方

您可以在 `Authorization` 標頭中提供身分或存取字符，以呼叫 Lambda 授權方。如需詳細資訊，請參閱the section called “使用 Lambda 授權方呼叫 API”。

API Gateway 會花 120 秒時間快取 Lambda 授權方傳回的政策。您可以在 API Gateway 主控台或使用 AWS CLI修改 TTL。

## 使用 Amazon Cognito 使用者集區做為授權方，藉以控制對 REST API 的存取

除了使用 IAM 角色和政策或 Lambda 授權方 (先前稱為自訂授權方) 之外，您還可以使用 Amazon Cognito 使用者集區來控制誰可以在 Amazon API Gateway 中存取您的 API。

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/SDK 或 API 來建立使用者集區，或使用另一個 AWS 帳戶擁有的使用者集區。
- 使用 API Gateway 主控台、CLI/軟體開發套件或 API 建立 API Gateway authorizer 與所選的使用者集區。
- 使用 API Gateway 主控台、CLI/軟體開發套件或 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/軟體開發套件或 API 將使用者登入至所選擇的使用者集區，並取得身分字符或存取字符。若要進一步了解如何使用 SDKs，請參閱 Amazon Cognito AWS SDKs程式碼範例。
- 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。

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，同時取得暫時性登入資料以使用 IAM 角色的許可，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

- 為 REST API 建立 Amazon Cognito 使用者集區
- 整合 REST API 與 Amazon Cognito 使用者集區
- 呼叫與 Amazon Cognito 使用者集區整合的 REST API

- 使用 API Gateway 主控台為 REST API 設定跨帳戶的 Amazon Cognito 授權方
- 使用 為 REST API 建立 Amazon Cognito 授權方 CloudFormation

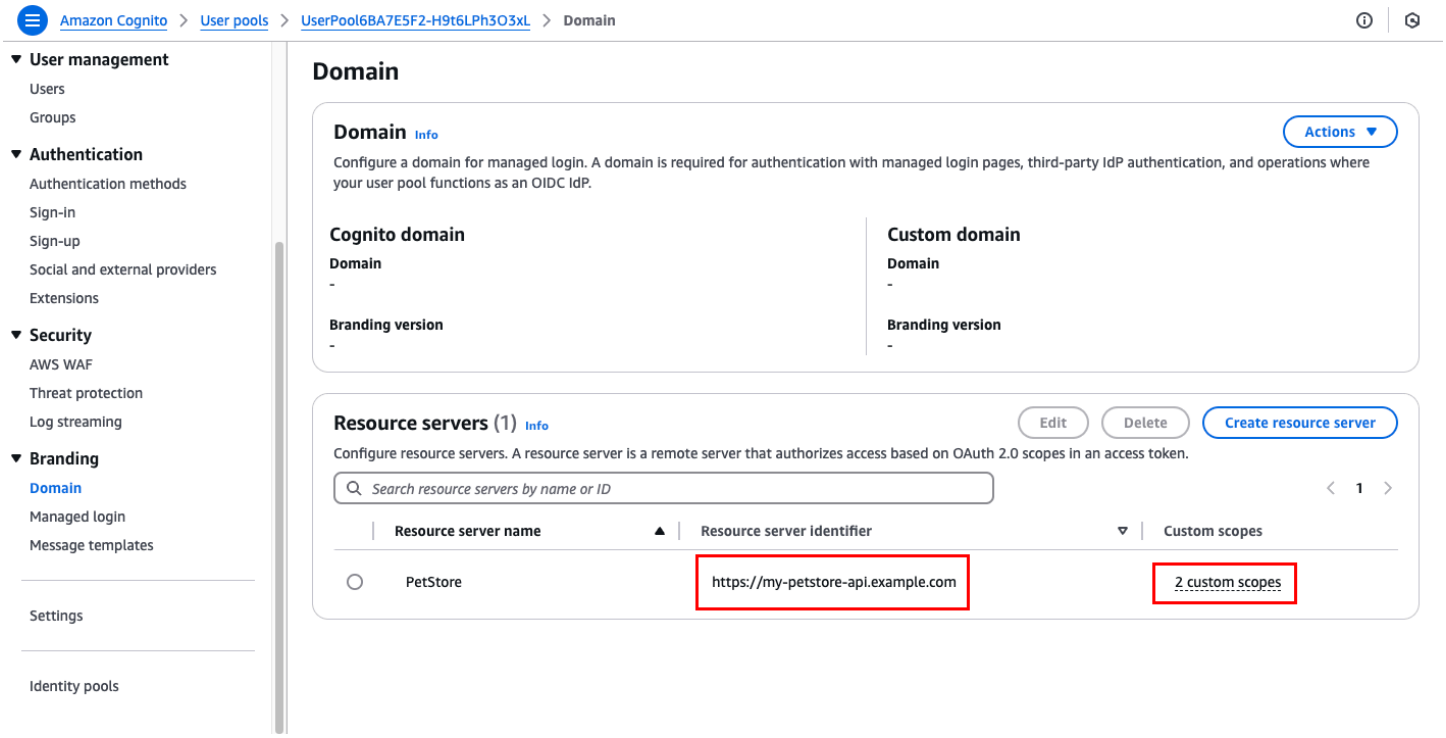
為 REST API 建立 Amazon Cognito 使用者集區

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。您的使用者集區組態必須遵循所有 Amazon Cognito 的資源配額。所有使用者定義的 Amazon Cognito 變數 (例如群組、使用者和角色) 僅可使用英數字元。如需如何建立使用者集區的說明，請參閱《Amazon Cognito 開發人員指南》中的教學課程：建立使用者集區。

請記下使用者集區 ID、用戶端 ID 和任何用戶端密碼。用戶端必須將這些資訊提供給 Amazon Cognito 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符，來呼叫以使用者集區設定的 API 方法。此外，當您在 API Gateway 中將使用者集區設定為授權方時，必須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。如需將字符用於 Amazon Cognito 使用者集區的詳細資訊，請參閱將字符用於使用者集區。如需資源伺服器的詳細資訊，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。



整合 REST API 與 Amazon Cognito 使用者集區

建立 Amazon Cognito 使用者集區後，您必須在 API Gateway 中建立使用該使用者集區的 COGNITO_USER_POOLS 授權方。下列程序示範如何使用 API Gateway 主控台來執行此操作。

> **Note**
>
> 您可以使用 CreateAuthorizer 動作來建立使用多個使用者集區的 COGNITO_USER_POOLS 授權者。一個 COGNITO_USER_POOLS 授權者最多可以使用 1,000 個使用者集區。此限制無法提高。

> **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 API Gateway 中部署 REST API。

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. 在主導覽窗格中，選擇授權方。
3. 選擇建立授權方。
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. 針對授權方名稱，輸入名稱。

   b. 針對授權方類型，選取 Cognito。

   c. 針對 Cognito 使用者集區，選擇您建立 Amazon Cognito AWS 區域 的 ，然後選取可用的使用者集區。

      您可以使用階段變數來定義使用者集區。使用者集區可使用下列格式：arn:aws:cognito-idp:*us-east-2*:*111122223333*:userpool/${stageVariables.*MyUserPool*}。

   d. 針對權杖來源，輸入 **Authorization** 作為標頭名稱，以在使用者成功登入時，傳遞 Amazon Cognito 傳回的身分或存取權杖。

   e. (選用) 在權杖驗證欄位中輸入規則表達式，以驗證身分權杖的 aud (對象) 欄位，再透過 Amazon Cognito 授權請求。請注意，當使用存取字符時，由於存取字符不包含該 aud 字段，所以此驗證拒絕該請求。

f. 選擇建立授權方。

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來測試其調用情形。您無法使用存取字符來測試調用授權方。

   您可以呼叫 Amazon Cognito 身分軟體開發套件來取得此身分字符，藉此執行使用者登入。您也可以使用 `InitiateAuth` 動作。如果您未設定任何授權範圍，API Gateway 會將提供的權杖視為身分權杖。

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。

在方法中設定 **COGNITO_USER_POOLS** 授權方

1. 選擇資源。選擇新方法或選擇現有方法。如有需要，請建立資源。
2. 在方法請求索引標籤的方法請求設定下，選擇編輯。
3. 針對授權方，從下拉式選單選取您剛才建立的 Amazon Cognito 使用者集區授權方。
4. 若要使用身分字符，請執行下列操作：

   a. 將授權範圍保留空白。

   b. 如有需要，在整合請求中，於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遞到後端。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']"
    }
}
```

5. 若要使用存取字符，請執行下列操作：

   a. 針對授權範圍，輸入在建立 Amazon Cognito 使用者集區時所設定之範圍的一或多個完整名稱。例如，在 為 REST API 建立 Amazon Cognito 使用者集區 提供的範例之後，其中一個範圍是 `https://my-petstore-api.example.com/cats.read`。

      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。

   b. 選擇儲存。

6. 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"
              }
            }
          }
        },
        "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/scope, ...] }`：

```
  "paths": {
    "/": {
      "get": {
        "consumes": [
```

```
          "application/json"
        ],
        "produces": [
          "text/html"
        ],
        "responses": {
          "200": {
            "description": "200 response",
            "headers": {
              "Content-Type": {
                "type": "string"
              }
            }
          }
        },
        "security": [
          {
            "MyUserPool": ["https://my-petstore-api.example.com/cats.read",
"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[API Gateway 的 OpenAPI 擴充功能]。

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或存取字符。
- 在 `Authorization` 標頭 (或您在建立授權方時指定的另一個標頭) 中包含字符。

您可以使用 AWS Amplify 來執行這些任務。如需詳細資訊，請參閱將 Amazon Cognito 與 Web 和行動應用程式整合。

- 如果是 Android 版，請參閱 Android 版 Amplify 入門。
- 若要使用 iOS，請參閱 iOS 版 Amplify 入門。
- 若要使用 JavaScript，請參閱 JavaScript 版 Amplify 入門。

使用 API Gateway 主控台為 REST API 設定跨帳戶的 Amazon Cognito 授權方

您現在可以使用來自不同 AWS 帳戶的 Amazon Cognito 使用者集區做為您的 API 授權方。Amazon Cognito 使用者集區可以使用承載字符身分驗證政策，例如 OAuth 或 SAML。這可讓您輕鬆集中管理並跨多個 API Gateway API 共用一個中央 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 主控台設定跨帳戶的 Amazon Cognito 使用者集區。

這些指示假設您在一個 AWS 帳戶中已有 API Gateway API，而另一個帳戶中已有 Amazon Cognito 使用者集區。

為 REST API 設定跨帳戶 Amazon Cognito 授權方

在您 API 所在的帳戶中，登入 Amazon API Gateway 主控台，然後執行下列操作：

1. 在 API Gateway 中建立新的 API 或選取現有的 API。
2. 在主導覽窗格中，選擇授權方。
3. 選擇建立授權方。
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. 針對授權方名稱，輸入名稱。

b. 針對授權方類型，選取 Cognito。

c. 針對 Cognito 使用者集區，輸入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，您可以在 General Settings (一般設定) 窗格的 Pool ARN (集區 ARN) 欄位中找到適用於您使用者集區的 ARN。

d. 針對權杖來源，輸入 **`Authorization`** 作為標頭名稱，以在使用者成功登入時，傳遞 Amazon Cognito 傳回的身分或存取權杖。

e. (選用) 在權杖驗證欄位中輸入規則表達式，以驗證身分權杖的 aud (對象) 欄位，再透過 Amazon Cognito 授權請求。請注意，當使用存取字符時，由於存取字符不包含該 aud 字段，所以此驗證拒絕該請求。

f. 選擇建立授權方。

使用 為 REST API 建立 Amazon Cognito 授權方 CloudFormation

您可以使用 CloudFormation 建立 Amazon Cognito 使用者集區和 Amazon Cognito 授權方。範本範例 CloudFormation 會執行下列動作：

- 建立 Amazon Cognito 使用者集區。用戶端必須先將使用者登入使用者集區，並取得身分或存取字符。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。
- 使用 `GET` 方法建立 API Gateway API。
- 建立使用 `Authorization` 標頭作為字符來源的 Amazon Cognito 授權方。

```
AWSTemplateFormatVersion: 2010-09-09
Resources:
  UserPool:
    Type: AWS::Cognito::UserPool
    Properties:
      AccountRecoverySetting:
        RecoveryMechanisms:
          - Name: verified_phone_number
            Priority: 1
          - Name: verified_email
            Priority: 2
```

```
      AdminCreateUserConfig:
        AllowAdminCreateUserOnly: true
      EmailVerificationMessage: The verification code to your new account is {####}
      EmailVerificationSubject: Verify your new account
      SmsVerificationMessage: The verification code to your new account is {####}
      VerificationMessageTemplate:
        DefaultEmailOption: CONFIRM_WITH_CODE
        EmailMessage: The verification code to your new account is {####}
        EmailSubject: Verify your new account
        SmsMessage: The verification code to your new account is {####}
    UpdateReplacePolicy: Retain
    DeletionPolicy: Retain
  CogAuthorizer:
    Type: AWS::ApiGateway::Authorizer
    Properties:
      Name: CognitoAuthorizer
      RestApiId:
        Ref: Api
      Type: COGNITO_USER_POOLS
      IdentitySource: method.request.header.Authorization
      ProviderARNs:
        - Fn::GetAtt:
            - UserPool
            - Arn
  Api:
    Type: AWS::ApiGateway::RestApi
    Properties:
      Name: MyCogAuthApi
  ApiDeployment:
    Type: AWS::ApiGateway::Deployment
    Properties:
      RestApiId:
        Ref: Api
    DependsOn:
      - CogAuthorizer
      - ApiGET
  ApiDeploymentStageprod:
    Type: AWS::ApiGateway::Stage
    Properties:
      RestApiId:
        Ref: Api
      DeploymentId:
        Ref: ApiDeployment
      StageName: prod
```

```
  ApiGET:
    Type: AWS::ApiGateway::Method
    Properties:
      HttpMethod: GET
      ResourceId:
        Fn::GetAtt:
          - Api
          - RootResourceId
      RestApiId:
        Ref: Api
      AuthorizationType: COGNITO_USER_POOLS
      AuthorizerId:
        Ref: CogAuthorizer
      Integration:
        IntegrationHttpMethod: GET
        Type: HTTP_PROXY
        Uri: http://petstore-demo-endpoint.execute-api.com/petstore/pets
Outputs:
  ApiEndpoint:
    Value:
      Fn::Join:
        - ""
        - - https://
          - Ref: Api
          - .execute-api.
          - Ref: AWS::Region
          - "."
          - Ref: AWS::URLSuffix
          - /
          - Ref: ApiDeploymentStageprod
          - /
```

# API Gateway 中 REST API 的整合

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 網頁或服務 AWS 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；指定呼叫哪個 Lambda 函數、指定傳入請求要轉送到哪個 HTTP 伺服器，或指定要叫用的 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` 資源回應。

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 許可以存取後端，並指定映射以轉換傳入的請求資料，然後將它傳送到後端。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
- 在 API Gateway 中設定整合回應
- API Gateway 中 REST API 的 Lambda 整合
- API Gateway 中 REST API 的 HTTP 整合
- 在 API Gateway 中串流代理整合的整合回應
- API Gateway 中 REST API 的私有整合
- API Gateway 中 REST API 的模擬整合


## 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. 對於與 Lambda 函數和其他 AWS 服務動作的整合，請設定具有必要許可的 IAM 角色，讓 API Gateway 代表您呼叫後端。
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) 宣告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 的整合資源，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 SDK 或 API Gateway REST API 來執行此操作。

主題

- API 整合請求的基本任務
- 選擇 API Gateway API 整合類型
- 設定代理整合與代理資源
- 使用 API Gateway 主控台設定 API 整合請求

## API 整合請求的基本任務

整合請求是一項 HTTP 請求，由 API Gateway 提交到後端，隨用戶端提交的請求資料一併傳送，並在必要時轉換資料。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。「the section called "參數映射"」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。

使用另一個 AWS 服務動作設定整合請求時，整合請求 URI 也是 ARN，類似於與 Lambda Invoke動作的整合。例如，與 Amazon S3 GetBucket 動作的整合，其整合請求 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，可以透過其名稱來參考 AWS 服務動作。使用動作名稱，Amazon S3 GetBucket 動作的整合請求 URI 會變成如下：

```
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}*) 中指定儲存貯體名稱 ({ Bucket: "*{bucket}*" })，遵循 GetBucket 動作的輸入格式。

對於 AWS 整合，您還必須設定登入資料，以允許 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 宣告 (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。如需詳細資訊，請參閱the section called “資料轉換”。

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。對於 Lambda 函數，您可以擁有 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。對於 AWS 服務動作，您只能 AWS 整合非代理類型。API Gateway 也支援以 API Gateway 作為整合端點回應方法請求的模擬整合。

Lambda 自訂整合是 AWS 整合的特殊案例，其中整合端點對應至 Lambda 服務的函數叫用動作。

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，其值為 `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`：這類整合可將 API 方法與 Lambda 函數叫用動作，以靈活、多樣化和簡化的整合設定進行整合。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作輸入傳送到後端 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傳回結果。

這是透過 API Gateway 呼叫 Lambda 函數的偏好整合類型，不適用於任何其他 AWS 服務動作，包括函數叫用動作以外的 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` 方法以模擬整合支援 CORS。閘道回應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”。

> **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 會傳回預設的 `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，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/a0/a1/.../aN`、`/site/b0/b1/.../bM` 等等。如果您將 `ANY` 之代理資源上的 `/api/{proxy+}` 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 `[a0, a1, ..., aN, b0, b1, ...bM, ...]` 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 可套用預設映射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 主控台來建立整合請求。

主題


- 設定 Lambda 整合
- 設定 HTTP 整合
- 設定 AWS 服務整合
- 設定模擬整合


## 設定 Lambda 整合

使用 Lambda 函數整合，將您的 API 與 Lambda 函數整合。在 API 層級，如果您建立非代理整合，則這會是 AWS 整合類型，如果您建立代理整合，則這會是 AWS_PROXY 整合類型。

設定 Lambda 整合

1. 在資源窗格中，選擇建立方法。
2. 針對方法類型，選取 HTTP 方法。
3. 對於 Integration type (整合類型)，請選擇 Lambda function (Lambda 函數)。
4. 若要使用 Lambda 代理整合，請開啟 Lambda 代理整合。若要深入了解 Lambda 代理整合，請參閱 the section called “ 了解 Lambda 代理整合 ”。
5. 針對 Lambda 函數，輸入 Lambda 函數的名稱。

   如果您在與 API 不同的區域中使用 Lambda 函數，請從下拉式選單中選取區域，並輸入 Lambda 函數的名稱。如果您使用的是跨帳戶 Lambda 函數，請輸入函數 ARN。
6. 若要使用 29 秒的預設逾時值，請將預設逾時保持開啟。若要設定自訂逾時，請選擇預設逾時，然後輸入介於 50 和 29000 毫秒之間的逾時值。
7. (選用) 您可以使用下列下拉式選單來配置方法請求設定值。選擇方法請求設定，並設定您的方法請求。如需詳細資訊，請參閱 the section called “在主控台編輯方法請求” 的步驟 3。

   您也可以在建立方法後配置方法請求設定值。
8. 選擇建立方法。

## 設定 HTTP 整合

使用 HTTP 整合將您的 API 與 HTTP 端點整合。在 API 層級，這是 HTTP 整合類型。

設定 HTTP 整合

1. 在資源窗格中，選擇建立方法。
2. 針對方法類型，選取 HTTP 方法。
3. 對於整合類型，選擇 HTTP。
4. 若要使用 HTTP 代理整合，請開啟 HTTP 代理整合。若要進一步了解 HTTP 代理整合，請參閱 the section called “在 API Gateway 中設定 HTTP 代理整合”。
5. 針對 HTTP method (HTTP 方法)，選擇最符合 HTTP 後端中方法的 HTTP 方法類型。
6. 針對端點 URL，輸入您希望此方法使用之 HTTP 後端的 URL。
7. 針對內容處理，選取內容處理行為。
8. 若要使用 29 秒的預設逾時值，請將預設逾時保持開啟。若要設定自訂逾時，請選擇預設逾時，然後輸入介於 `50` 和 `29000` 毫秒之間的逾時值。
9. (選用) 您可以使用下列下拉式選單來配置方法請求設定值。選擇方法請求設定，並設定您的方法請求。如需詳細資訊，請參閱 the section called “在主控台編輯方法請求” 的步驟 3。

   您也可以在建立方法後配置方法請求設定值。
10. 選擇建立方法。

設定 AWS 服務整合

使用 AWS 服務整合將您的 API 直接與 AWS 服務整合。在 API 層級，這是 `AWS` 整合類型。

若要設定 API Gateway API，請執行下列任一個動作：

- 建立新 Lambda 函數。
- 在 Lambda 函數上設定資源許可。
- 執行任何其他 Lambda 服務動作。

您必須選擇 AWS 服務。

設定 AWS 服務整合

1. 在資源窗格中，選擇建立方法。
2. 針對方法類型，選取 HTTP 方法。
3. 對於整合類型，選擇 AWS 服務。
4. 針對AWS 區域，選擇 AWS 您希望此方法用來呼叫 動作的區域。

5. 針對AWS 服務，選擇您希望此方法呼叫 AWS 的服務。
6. 針對AWS 子網域，輸入 AWS 服務所使用的子網域。這個項目一般會保持空白。有些 AWS 服務可支援子網域做為主機的一部分。請參閱服務文件以了解可用性及詳細資訊 (如有)。
7. 針對 HTTP method (HTTP 方法)，選擇對應動作的 HTTP 方法類型。對於 HTTP 方法類型，請參閱您為 AWS 服務選擇之AWS 服務的 API 參考文件。
8. 針對動作類型，選取使用動作名稱以使用 API 動作，或選取使用路徑覆寫以使用自訂資源路徑。如需可用的動作和自訂資源路徑，請參閱您為 AWS 服務選擇之AWS 服務的 API 參考文件。
9. 輸入動作名稱或路徑覆寫。
10. 針對執行角色，輸入方法將用於呼叫動作之 IAM 角色的 ARN。

    若要建立 IAM 角色，您可以調整 the section called “步驟 1：建立 AWS 服務代理執行角色” 中的指示。指定包含所需動作數和資源陳述式的存取政策。如需詳細資訊，請參閱the section called “Amazon API Gateway 與 IAM 搭配運作的方式”。

    如需動作和資源陳述式語法，請參閱您為 AWS 服務選擇AWS 之服務的文件。

    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：

    JSON

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

11. 若要使用 29 秒的預設逾時值，請將預設逾時保持開啟。若要設定自訂逾時，請選擇預設逾時，然後輸入介於 50 和 29000 毫秒之間的逾時值。
12. (選用) 您可以使用下列下拉式選單來配置方法請求設定值。選擇方法請求設定，並設定您的方法請求。如需詳細資訊，請參閱 the section called “在主控台編輯方法請求” 的步驟 3。

您也可以在建立方法後配置方法請求設定值。

13. 選擇建立方法。

設定模擬整合

如果您希望 API Gateway 充當您的後端傳回靜態回應，請使用模擬整合。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。

設定模擬整合

1. 在資源窗格中，選擇建立方法。
2. 針對方法類型，選取 HTTP 方法。
3. 對於整合類型，選擇模擬。
4. (選用) 您可以使用下列下拉式選單來配置方法請求設定值。選擇方法請求設定，並設定您的方法請求。如需詳細資訊，請參閱 the section called “在主控台編輯方法請求” 的步驟 3。

   您也可以在建立方法後配置方法請求設定值。
5. 選擇建立方法。

## 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。您可以將規則表達式指派給 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，請參閱處理 API Gateway 中的 Lambda 錯誤。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。如需詳細資訊，請參閱覆寫 API Gateway 中 REST API 的 API 請求和回應參數及狀態碼。

## API Gateway 中 REST API 的 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，必要設定非常簡單。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，並且允許 API Gateway 代表您呼叫 Lambda 函數。

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

- API Gateway 中的 Lambda 代理整合
- 在 API Gateway 中設定 Lambda 自訂整合
- 設定後端 Lambda 函數的非同步叫用
- 處理 API Gateway 中的 Lambda 錯誤

## API Gateway 中的 Lambda 代理整合

下一節顯示如何使用 Lambda 代理整合。

主題

- 了解 API Gateway Lambda 代理整合
- 支援多值標頭和查詢字串參數
- 代理整合之 Lambda 函數的輸入格式
- 代理整合之 Lambda 函數的輸出格式
- 使用 設定 API Gateway 的 Lambda 代理整合 AWS CLI
- 使用 Lambda 代理整合與 OpenAPI 定義設定代理資源

### 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。函數會存取其他服務的許多資源或功能 AWS ，包括呼叫其他 Lambda 函數。

在 Lambda 代理整合中，當用戶端提交一個 API 請求時，API Gateway 會對整合的 Lambda 函數傳遞事件物件，但不會保留請求參數的順序。此請求資料包含請求標頭、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 輸出當作 API 回應傳遞至用戶端，Lambda 函數必須以此格式傳回結果。

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 代理整合的執行時間行為：

- 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，以將資料作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，以將資料作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，以將資料作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。

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 現在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。如需詳細資訊，請參閱代理整合之 Lambda 函數的輸入格式及代理整合之 Lambda 函數的輸出格式。

代理整合之 Lambda 函數的輸入格式

在 Lambda 代理整合中，API Gateway 會將整個用戶端請求映射至後端 Lambda 函數的輸入 event 參數。下列範例顯示 API Gateway 傳送至 Lambda 代理整合之事件的結構。

在此範例中，我們假設 API Gateway 的調用如下：

```
curl 'https://a1b2c3.execute-api.us-east-1.amazonaws.com/my/path?
parameter1=value1&parameter2=value1&parameter2=value2&parameter3=value1,value2'
 -H 'header1: value1' -H 'header2: value1' -H 'header2: value2' -H 'header3:
 value1,value2'
```

輸出看起來如下：

```
{
  "resource": "/my/path",
  "path": "/my/path",
  "httpMethod": "GET",
  "headers": {
      "header1": "value1",
      "header2": "value2",
      "header3": "value1,value2"
  },
  "multiValueHeaders": {
    "header1": ["value1"],
    "header2": ["value1","value2"],
    "header3": ["value1,value2"]
```

```
  },
  "queryStringParameters": {
      "parameter1": "value1",
      "parameter2": "value2",
      "parameter3": "value1,value2"
  },
  "multiValueQueryStringParameters": {
    "parameter1": ["value1"],
    "parameter2": ["value1","value2"],
    "parameter3": ["value1,value2"]
  },
  "requestContext": {
    "accountId": "123456789012",
    "apiId": "id",
    "authorizer": {
      "claims": null,
      "scopes": null
    },
    "domainName": "id.execute-api.us-east-1.amazonaws.com",
    "domainPrefix": "id",
    "extendedRequestId": "request-id",
    "httpMethod": "GET",
    "identity": {
      "accessKey": null,
      "accountId": null,
      "caller": null,
      "cognitoAuthenticationProvider": null,
      "cognitoAuthenticationType": null,
      "cognitoIdentityId": null,
      "cognitoIdentityPoolId": null,
      "principalOrgId": null,
      "sourceIp": "IP",
      "user": null,
      "userAgent": "user-agent",
      "userArn": null,
      "clientCert": {
        "clientCertPem": "CERT_CONTENT",
        "subjectDN": "www.example.com",
        "issuerDN": "Example issuer",
        "serialNumber": "a1:a1:a1:a1:a1:a1:a1:a1:a1:a1:a1:a1:a1:a1:a1:a1",
        "validity": {
          "notBefore": "May 28 12:30:02 2019 GMT",
          "notAfter": "Aug  5 09:36:04 2021 GMT"
        }
```

```
      }
    },
    "path": "/my/path",
    "protocol": "HTTP/1.1",
    "requestId": "id=",
    "requestTime": "04/Mar/2020:19:15:17 +0000",
    "requestTimeEpoch": 1583349317135,
    "resourceId": null,
    "resourcePath": "/my/path",
    "stage": "$default"
  },
  "pathParameters": null,
  "stageVariables": null,
  "body": "Hello from Lambda!",
  "isBase64Encoded": false
}
```

> **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。

在後端 Lambda 函數的輸入中，`requestContext` 物件是索引鍵/值組的映射。在各對中，鍵是 $context 變數屬性的名稱，而值是該屬性的值。API Gateway 可能會將新的金鑰加入至地圖。

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。

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，您可以透過將 `isBase64Encoded` 設定為 `true`將其編碼為 Base64 編碼字串，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。如需 Lambda 函數的範例，請參閱在 API Gateway 中從 Lambda 代理整合傳回二進位媒體。

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 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。

使用 設定 API Gateway 的 Lambda 代理整合 AWS CLI

在本節中，我們示範如何使用 AWS CLI設定具有 Lambda 代理整合的 API。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：建立具 Lambda 代理整合的 REST API。

例如，我們使用下列範例 Lambda 函數作為 API 的後端：

```
export const handler = async(event, context) => {
    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;
        }
```

```
    } 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 + "!";
    return res
};
```

在 the section called “ 設定 Lambda 自訂整合 ” 中將此項目與 Lambda 自訂整合設定做比較，可以在請求參數與內文中表達此 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 設定 Lambda 代理整合 AWS CLI

1. 使用以下 create-rest-api 命令建立 API：

   ```
   aws apigateway create-rest-api --name 'HelloWorld (AWS CLI)'
   ```

   輸出將如下所示：

   ```
   {
       "name": "HelloWorldProxy (AWS CLI)",
       "id": "te6si5ach7",
       "rootResourceId" : "krznpq9xpg",
       "createdDate": 1508461860
   }
   ```

   在此範例中，您會使用 API `id` (`te6si5ach7`) 和 `rootResourceId` (`krznpq9xpg`)。

2. 使用以下 create-resource 命令建立 `/greeting` 的 API Gateway 資源：

   ```
   aws apigateway create-resource \
         --rest-api-id te6si5ach7 \
         --parent-id krznpq9xpg \
         --path-part {proxy+}
   ```

輸出將如下所示：

```
{
    "path": "/{proxy+}",
    "pathPart": "{proxy+}",
    "id": "2jf6xt",
    "parentId": "krznpq9xpg"
}
```

在下一個步驟中，您會使用 {proxy+} 資源的 id 值 (2jf6xt) 在 /{proxy+} 資源上建立方法。

3. 使用以下 put-method 建立 ANY /{proxy+} 的方法請求 ANY：

```
aws apigateway put-method --rest-api-id te6si5ach7 \
       --resource-id 2jf6xt \
       --http-method ANY \
       --authorization-type "NONE"
```

輸出將如下所示：

```
{
    "apiKeyRequired": false,
    "httpMethod": "ANY",
    "authorizationType": "NONE"
}
```

此 API 方法可讓用戶端從後端的 Lambda 函數接收或傳送問候語。

4. 使用以下 put-integration 命令來設定 ANY /{proxy+} 方法與名為 HelloWorld 之 Lambda 函式的整合。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 \
      --rest-api-id te6si5ach7 \
      --resource-id 2jf6xt \
      --http-method ANY \
      --type AWS_PROXY \
      --integration-http-method POST \
      --uri arn:aws:apigateway:us-west-2:lambda:path/2015-03-31/functions/
arn:aws:lambda:us-west-2:123456789012:function:HelloWorld/invocations \
```

```
      --credentials arn:aws:iam::123456789012:role/apigAwsProxyRole
```

> **Important**
>
> 針對 Lambda 整合，您必須根據進行函數叫用之 Lambda 服務動作的規格，使用 HTTP 方法 POST 進行整合請求。IAM 角色 apigAwsProxyRole 的政策必須允許 apigateway 服務叫用 Lambda 函數。如需 IAM 許可的相關資訊，請參閱 the section called “用於呼叫 API 的 API Gateway 許可模型”。

輸出將如下所示：

```
{
    "passthroughBehavior": "WHEN_NO_MATCH",
    "cacheKeyParameters": [],
    "uri": "arn:aws:apigateway:us-west-2:lambda:path/2015-03-31/functions/
arn:aws:lambda:us-west-2:1234567890:function:HelloWorld/invocations",
    "httpMethod": "POST",
    "cacheNamespace": "vvom7n",
    "credentials": "arn:aws:iam::1234567890:role/apigAwsProxyRole",
    "type": "AWS_PROXY"
}
```

您可以使用 add-permission 命令來新增資源型許可，而不提供 credentials 的 IAM 角色。這就是 API Gateway 主控台的功能。

5. 使用以下 create-deployment 命令將 API 部署至 test 階段：

```
aws apigateway create-deployment  \
      --rest-api-id te6si5ach7 \
      --stage-name test
```

6. 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'
```

   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 '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/hi \
  -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!
```

使用 Lambda 代理整合與 OpenAPI 定義設定代理資源

若要以 Lambda 代理整合類型設定代理資源，請以 Greedy 路徑參數建立 API 資源 (例如 /parent/{proxy+})，並將此資源與 arn:aws:lambda:us-west-2:123456789012:function:SimpleLambda4ProxyResource 方法上的 Lambda 函數後端 (例如 ANY) 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"
          }
        ],
        "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"
        }
      }
    }
```

```
  }
}
```

在 Lambda 代理整合中，API Gateway 會在執行時間將傳入的請求映射到 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」說明輸入格式。若要讓 API Gateway 成功將 Lambda 輸出映射至 HTTP 回應，Lambda 函數必須以 代理整合之 Lambda 函數的輸出格式 中所述的格式輸出結果。

在透過 ANY 方法之代理資源的 Lambda 代理整合中，單一後端 Lambda 函數透過代理資源作為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。

在 API Gateway 中設定 Lambda 自訂整合

為了示範如何設定 Lambda 自訂或非代理整合，我們會建立 API Gateway API，以公開 `GET /greeting?greeter={name}` 方法來叫用 Lambda 函數。請為您的 API 使用下列其中一個 Lambda 函數範例。

使用下列其中一個 Lambda 函數範例：

Node.js

```
'use strict';
var days = ['Sunday', 'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday',
 'Saturday'];
var times = ['morning', 'afternoon', 'evening', 'night', 'day'];

export const handler = async(event) => {
  console.log(event);
  // Parse the input for the name, city, time and day property values
  let name = event.name === null || event.name === undefined || event.name === "" ?
 '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
  return greeting;
};
```

Python

```
import json


def lambda_handler(event, context):
    print(event)
    res = {
        "statusCode": 200,
        "headers": {
            "Content-Type": "*/*"
        }
    }

    if event['greeter'] == "":
        res['body'] = "Hello, World"
    elif (event['greeter']):
        res['body'] = "Hello, " + event['greeter'] + "!"
    else:
        raise Exception('Missing the required greeter parameter.')

    return res
```

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。

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 設定 Lambda 自訂整合 AWS CLI

1. 使用以下 create-rest-api 命令建立 API：

```
aws apigateway create-rest-api --name 'HelloWorld (AWS CLI)'
```

輸出將如下所示：

```
{
    "name": "HelloWorld (AWS CLI)",
    "id": "te6si5ach7",
    "rootResourceId" : "krznpq9xpg",
    "createdDate": 1508461860
}
```

在此範例中，您會使用 API `id` (`te6si5ach7`) 和 `rootResourceId` (`krznpq9xpg`)。

2. 使用以下 create-resource 命令建立 `/greeting` 的 API Gateway 資源：

```
aws apigateway create-resource \
      --rest-api-id te6si5ach7 \
      --parent-id krznpq9xpg \
      --path-part greeting
```

輸出將如下所示：

```
{
    "path": "/greeting",
    "pathPart": "greeting",
    "id": "2jf6xt",
    "parentId": "krznpq9xpg"
}
```

在下一個步驟中，您會使用 `greeting` 資源的 `id` 值 (`2jf6xt`) 在 `/greeting` 資源上建立方法。

3. 使用以下 put-method 命令建立 `GET /greeting?greeter={name}` 的 API 方法請求：

```
aws apigateway put-method --rest-api-id te6si5ach7 \
       --resource-id 2jf6xt \
```

```
        --http-method GET \
        --authorization-type "NONE" \
        --request-parameters method.request.querystring.greeter=false
```

輸出將如下所示：

```
{
    "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。

4. 使用以下 put-method-response 命令來設定 GET /greeting?greeter={name} 之方法請求的 200 OK 回應：

```
aws apigateway put-method-response \
        --rest-api-id te6si5ach7 \
        --resource-id 2jf6xt \
        --http-method GET \
        --status-code 200
```

5. 使用以下 put-integration 命令來設定 GET /greeting?greeter={name} 方法與名為 HelloWorld 之 Lambda 函式的整合。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 \
        --rest-api-id te6si5ach7 \
        --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 '{"application/json":"{\"greeter\":
\"$input.params('greeter')\"}"}' \
        --credentials arn:aws:iam::123456789012:role/apigAwsProxyRole
```

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 函數的輸入必須在內文中表示。

> **Important**
>
> 針對 Lambda 整合，您必須根據進行函數調用之 Lambda 服務動作的規格，使用 HTTP 方法 POST 進行整合請求。uri 參數是函數呼叫動作的 ARN。
> 輸出將如下所示：

```
{
    "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 服務叫用 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 主控台新增這些許可的方式。

6. 使用以下 put-integration-response 命令來設定整合回應，以將 Lambda 函式輸出傳遞至用戶端作為 200 OK 方法回應：

```
 aws apigateway put-integration-response \
        --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。

輸出將如下所示：

```
 {
    "selectionPattern": "",
    "statusCode": "200"
}
```

7. 使用以下 create-deployment 命令將 API 部署至 test 階段：

```
aws apigateway create-deployment \
        --rest-api-id te6si5ach7 \
        --stage-name test
```

8. 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 函數的非同步叫用

在 Lambda 非代理 (自訂) 整合中，系統依預設會同步叫用後端 Lambda 函數。這是大部分 REST API 操作所需的行為。不過，有些應用程式要求系統以非同步方式 (以批次操作方式或長時間延遲操作方式) 來執行工作，通常會由個別後端元件執行。在此情況下，系統會非同步叫用後端 Lambda 函數，且前端 REST API 方法不會傳回結果。

您可以將 Lambda 非代理整合的 Lambda 函數設定為以非同步方式叫用，方法是將 'Event' 指定為 Lambda 叫用類型。此作法如下所示：

在 API Gateway 主控台中設定 Lambda 非同步叫用

對於所有呼叫都是非同步的：

- 在整合請求中，新增具有 'Event' 靜態值的 X-Amz-Invocation-Type 標頭。

讓客戶決定呼叫是非同步或同步：

1. 在方法請求中，新增 InvocationType 標頭。
2. 在整合請求中，新增包含 method.request.header.InvocationType 對應表達式的 X-Amz-Invocation-Type 標頭。
3. 用戶端可以在 API 請求中包含 InvocationType: Event 標頭以進行非同步呼叫，或包含 InvocationType: RequestResponse 以進行同步呼叫。

使用 OpenAPI 設定 Lambda 非同步調用

對於所有呼叫都是非同步的：

- 將 X-Amz-Invocation-Type 標頭加到 x-amazon-apigateway-integration 區段。

```
"x-amazon-apigateway-integration" : {
          "type" : "aws",
          "httpMethod" : "POST",
          "uri" : "arn:aws:apigateway:us-east-2:lambda:path/2015-03-31/functions/
arn:aws:lambda:us-east-2:123456789012:function:my-function/invocations",
          "responses" : {
            "default" : {
              "statusCode" : "200"
            }
          },
          "requestParameters" : {
            "integration.request.header.X-Amz-Invocation-Type" : "'Event'"
          },
          "passthroughBehavior" : "when_no_match",
          "contentHandling" : "CONVERT_TO_TEXT"
        }
```

讓客戶決定呼叫是非同步或同步：

1. 在任何 OpenAPI 路徑項目物件上新增下列標頭。

```
"parameters" : [ {
"name" : "InvocationType",
"in" : "header",
"schema" : {
  "type" : "string"
```

```
}
} ]
```

2. 將 X-Amz-Invocation-Type 標頭加到 x-amazon-apigateway-integration 區段。

```
"x-amazon-apigateway-integration" : {
          "type" : "aws",
          "httpMethod" : "POST",
          "uri" : "arn:aws:apigateway:us-east-2:lambda:path/2015-03-31/functions/
arn:aws:lambda:us-east-2:123456789012:function:my-function/invocations",
          "responses" : {
            "default" : {
              "statusCode" : "200"
            }
          },
          "requestParameters" : {
            "integration.request.header.X-Amz-Invocation-Type" :
 "method.request.header.InvocationType"
          },
          "passthroughBehavior" : "when_no_match",
          "contentHandling" : "CONVERT_TO_TEXT"
        }
```

3. 用戶端可以在 API 請求中包含 InvocationType: Event 標頭以進行非同步呼叫，或包含 InvocationType: RequestResponse 以進行同步呼叫。

使用 設定 Lambda 非同步調用 CloudFormation

下列 CloudFormation 範本示範如何AWS::ApiGateway::Method為非同步調用設定 。

對於所有呼叫都是非同步的：

```
AsyncMethodGet:
    Type: 'AWS::ApiGateway::Method'
    Properties:
      RestApiId: !Ref Api
      ResourceId: !Ref AsyncResource
      HttpMethod: GET
      ApiKeyRequired: false
      AuthorizationType: NONE
      Integration:
        Type: AWS
        RequestParameters:
```

```
          integration.request.header.X-Amz-Invocation-Type: "'Event'"
        IntegrationResponses:
            - StatusCode: '200'
        IntegrationHttpMethod: POST
        Uri: !Sub arn:aws:apigateway:${AWS::Region}:lambda:path/2015-03-31/functions/
${myfunction.Arn}$/invocations
      MethodResponses:
        - StatusCode: '200'
```

讓客戶決定呼叫是非同步或同步：

```
AsyncMethodGet:
    Type: 'AWS::ApiGateway::Method'
    Properties:
      RestApiId: !Ref Api
      ResourceId: !Ref AsyncResource
      HttpMethod: GET
      ApiKeyRequired: false
      AuthorizationType: NONE
      RequestParameters:
        method.request.header.InvocationType: false
      Integration:
        Type: AWS
        RequestParameters:
          integration.request.header.X-Amz-Invocation-Type:
 method.request.header.InvocationType
        IntegrationResponses:
            - StatusCode: '200'
        IntegrationHttpMethod: POST
        Uri: !Sub arn:aws:apigateway:${AWS::Region}:lambda:path/2015-03-31/functions/
${myfunction.Arn}$/invocations
      MethodResponses:
        - StatusCode: '200'
```

用戶端可以在 API 請求中包含 `InvocationType: Event` 標頭以進行非同步呼叫，或包含 `InvocationType: RequestResponse` 以進行同步呼叫。

### 處理 API Gateway 中的 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，映射到您用戶端的標準 HTTP 錯誤回應。否則，預設 Lambda 錯誤會傳回 `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 錯誤映射到 HTTP 錯誤回應。如需 API Gateway 將錯誤類型 (例如 `InvalidParameterException`) 當作回應的一部分傳送到用戶端，Lambda 函數必須在 `headers` 屬性中包含標頭 (例如 `"X-Amzn-ErrorType":"InvalidParameterException"`)。

主題

- 處理 API Gateway 中的標準 Lambda 錯誤
- 處理 API Gateway 中的自訂 Lambda 錯誤

### 處理 API Gateway 中的標準 Lambda 錯誤

標準 AWS Lambda 錯誤具有下列格式：

```
{
  "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。

```
export const 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 ...` 作為錯誤訊息：

```
{
  "errorMessage": "Malformed input ...",
  "errorType": "Error",
  "stackTrace": [
    "export const 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 錯誤映射到方法回應，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，設定規則表達式模式。在 API Gateway 主控台中，於每個整合回應下的整合回應區段中，這個 `selectionPattern` 會表示為 Lambda 錯誤 Regex。

> **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。

例如，使用以下 put-integration-response 來設定新的 `selectionPattern` 表達式：

```
aws apigateway put-integration-response --rest-api-id z0vprf0mdh --resource-id x3o5ih
 --http-method GET --status-code 400 --selection-pattern "Malformed.*" --region us-
west-2
```

請務必也要在方法回應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` 屬性規則表達式的模式比對 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 會傳回錯誤作為預設回應，或者若未設定預設回應，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。如果您使用 `.+` 作為選取模式來篩選回應，請注意，它可能不會比對包含新行 (\n) 字元的回應。

若要使用 更新現有`selectionPattern`值 AWS CLI，請呼叫 update-integration-response 操作，將`/selectionPattern`路徑值取代為`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 錯誤 Regex 文字方塊中輸入表達式。

處理 API Gateway 中的自訂 Lambda 錯誤

AWS Lambda 可讓您以 JSON 字串傳回自訂錯誤物件，而不是上一節所述的標準錯誤。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：

```
export const 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 會在執行標頭映射時還原序列化 `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映射，不適用於使用 `$input.body` 的內文到內文映射。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。

## API Gateway 中 REST API 的 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
- 在 API Gateway 中設定 HTTP 自訂整合

### 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 `https://petstore-demo-endpoint.execute-api.com/petstore/{proxy}` 方法上的 HTTP 後端端點 (例如 ANY) 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：建立具有 HTTP 代理整合的 REST API。

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**，將 **{proxy+}** 設定為 **pets**

  從前端啟動的方法請求：

  ```
  GET https://4z9giyi2c1.execute-api.us-west-2.amazonaws.com/test/pets HTTP/1.1
  ```

  傳送到後端的整合請求：

  ```
  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**，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
  ```

  傳送到後端的整合請求：

```
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**，將 **{proxy+}** 設定為 **pets/{petId}**

  從前端啟動的方法請求：

  ```
  GET https://4z9giyi2c1.execute-api.us-west-2.amazonaws.com/test/pets/1 HTTP/1.1
  ```

  傳送到後端的整合請求：

  ```
  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**，將 **{proxy+}** 設定為 **pets**

  從前端啟動的方法請求：

  ```
  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
  }
  ```

  傳送到後端的整合請求：

  ```
  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**，將 **{proxy+}** 設定為 **pets/cat**

  從前端啟動的方法請求：

  ```
  GET https://4z9giyi2c1.execute-api.us-west-2.amazonaws.com/test/pets/cat
  ```

  傳送到後端的整合請求：

  ```
  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**，將 **{proxy+}** 設定為 **null**

  從前端啟動的方法請求：

  ```
  GET https://4z9giyi2c1.execute-api.us-west-2.amazonaws.com/test
  ```

  傳送到後端的整合請求：

  ```
  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

若使用 HTTP 自訂整合 (也稱為非代理整合)，您就能更精細地控制要在 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 responseParameters 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，您可以在映射的 Integration 資源上設定 requestParameters 屬性，來指定參數對參數的映射。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 Method Response 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。接著您可以設定 selectionPattern，以根據後端的回應選擇整合回應。對於非代理 HTTP 整合，這是規則表達式。例如，若要將所有 2xx HTTP 回應狀態碼從 HTTP 端點對應至此輸出對應，請使用 2\d{2}。

> **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。

然後，在整合回應設定過程中，您可以在映射的 IntegrationResponse 資源上設定 responseParameters 屬性，來指定參數對參數的映射。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API Gateway 中用於 REST API 的資料轉換。

# 在 API Gateway 中串流代理整合的整合回應

您可以設定代理整合，以控制 API Gateway 傳回整合回應的方式。根據預設，API Gateway 會在開始傳輸之前，等待收到完整回應。不過，如果您將整合的回應傳輸模式設定為 STREAM，API Gateway 不會等待完全計算回應，再將其傳送至用戶端。回應串流適用於所有 REST API 端點類型。

針對下列使用案例使用回應串流：

- 降低聊天機器人等生成式 AI 應用程式time-to-first-byte (TTFB)。
- 不使用 S3 預先簽章的 URL 來串流大型影像、影片或音樂檔案。
- 執行長時間執行的操作，同時報告增量進度，例如伺服器傳送事件 (SSE)。
- 超過 API Gateway 的 10 MB 回應承載限制。
- 超過 API Gateway 的 29 秒逾時限制，而不要求增加整合逾時限制。
- 接收二進位承載而不設定二進位媒體類型。

## 回應承載串流的考量

下列考量可能會影響您對回應承載串流的使用：

- 您只能對 HTTP_PROXY或 AWS_PROXY整合類型使用回應承載串流。這包括 Lambda 代理整合和使用整合的私有HTTP_PROXY整合。
- 預設傳輸模式設定為 BUFFERED。若要使用回應串流，您必須將回應傳輸模式變更為 STREAM。
- 回應串流僅支援 REST APIs。
- 不支援請求串流。
- 您最多可以串流回應 15 分鐘。
- 您的串流會受到閒置逾時的影響。對於區域或私有端點，逾時為 5 分鐘。對於邊緣最佳化端點，逾時為 30 秒。
- 如果您使用區域 REST API 的回應串流搭配您自己的 CloudFront 分佈，您可以透過增加 CloudFront 分佈的回應逾時，達到超過 30 秒的閒置逾時。如需詳細資訊，請參閱回應逾時。
- 當回應傳輸模式設定為 時STREAM，API Gateway 不支援需要緩衝整個整合回應的功能。因此，回應串流不支援下列功能：
  - 端點快取
  - 內容編碼。如果您想要壓縮整合回應，請在整合中執行此操作。
  - 使用 VTL 進行回應轉換

- 在每個串流回應中，前 10MB 的回應承載不受任何頻寬限制。超過 10MB 的回應承載資料限制為 2MB/s。
- 當用戶端和 API Gateway 之間的連線，或 API Gateway 和 Lambda 之間的連線因逾時而關閉時，Lambda 函數可能會繼續執行。如需詳細資訊，請參閱設定 Lambda 函數逾時。
- 回應串流會產生成本。如需詳細資訊，請參閱 API Gateway 定價。

在 API Gateway 中設定 HTTP 代理整合與承載回應串流

當您設定回應承載串流時，您可以在 方法的整合請求中指定回應傳輸模式。您可以在整合請求中設定這些設定，以控制 API Gateway 在整合回應之前和期間的行為。使用回應串流時，您可以設定整合逾時最多 15 分鐘。

當您搭配 HTTP_PROXY整合使用承載回應串流時，API Gateway 不會傳送 HTTP 回應狀態碼或任何 HTTP 回應標頭，直到它完全收到所有標頭為止。

建立 HTTP 代理整合與承載回應串流

下列程序說明如何匯入將 `responseTransferMode` 設為 的新 APISTREAM。如果您有現有的整合 API 並想要修改 `responseTransferMode`，請參閱 the section called “更新 HTTP 代理整合的回應傳輸模式”。

AWS 管理主控台

建立與承載回應串流的 HTTP 代理整合

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 選擇 REST API。
3. 選擇建立資源。
4. 針對資源名稱，輸入 **`streaming`**。
5. 選擇建立資源。
6. 選取 /streaming 資源後，選擇建立方法。
7. 針對方法類型，選擇任何。
8. 對於整合類型，選擇 HTTP。
9. 選擇 HTTP 代理整合。
10. 針對回應傳輸模式，選擇串流。
11. 針對 HTTP 方法，選擇方法。

12. 在端點 URL 中，輸入整合端點。請確定您選擇的端點會產生要串流回給您的大型承載。
13. 選擇建立方法。

建立方法後，請部署您的 API。

部署 API

1. 選擇部署 API。
2. 針對階段，選取新階段。
3. 針對階段名稱，輸入 **prod**。
4. 在描述，請輸入描述。
5. 選擇部署。

AWS CLI

使用承載回應串流建立新的 API

1. 複製下列 Open API 檔案，然後將其儲存為 ResponseStreamDemoSwagger.yaml。在此檔案中， responseTransferMode 設定為 STREAM。整合端點設定為 https://example.com，但建議您將其修改為可產生大型承載的端點，以串流回給您。

```
openapi: "3.0.1"
info:
  title: "ResponseStreamingDemo"
  version: "2025-04-28T17:28:25Z"
servers:
- url: "{basePath}"
  variables:
    basePath:
      default: "prod"
paths:
  /streaming:
    get:
      x-amazon-apigateway-integration:
        httpMethod: "GET"
        uri: "https://example.com"
        type: "http_proxy"
        timeoutInMillis: 900000
        responseTransferMode: "STREAM"
```

2. 使用下列`import-rest-api`命令匯入您的 OpenAPI 定義：

```
aws apigateway import-rest-api \
  --body 'fileb://~/ResponseStreamDemoSwagger.yaml' \
  --parameters endpointConfigurationTypes=REGIONAL \
  --region us-west-1
```

3. 使用下列`create-deployment`命令將您的新 API 部署到階段：

```
aws apigateway create-deployment \
  --rest-api-id a1b2c3 \
  --stage-name prod \
  --region us-west-1
```

更新 HTTP 代理整合的回應傳輸模式

下列程序說明如何更新 HTTP 代理整合的回應傳輸模式。

AWS 管理主控台

更新 HTTP 代理整合的回應傳輸模式

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 選擇 REST API。
3. 選擇一個方法。
4. 在整合請求索引標籤上，於整合請求設定下，選擇編輯。
5. 針對回應傳輸模式，選擇串流。
6. 選擇儲存。

更新方法後，請部署您的 API。

部署 API

1. 選擇部署 API。
2. 針對階段，選取新階段。
3. 針對階段名稱，輸入 **prod**。
4. 在描述，請輸入描述。

5. 選擇部署。

AWS CLI

下列update-integration命令會將 整合的傳輸模式從 更新BUFFERED為 STREAM。對於任何現有的 APIs，所有整合的回應傳輸模式都會設定為 BUFFERED。

```
aws apigateway update-integration \
 --rest-api-id a1b2c3 \
 --resource-id aaa111 \
 --http-method GET \
 --patch-operations "op='replace',path='/responseTransferMode',value=STREAM" \
 --region us-west-1
```

您需要重新部署 API，變更才會生效。如果您自訂整合逾時，此逾時值會移除，因為 API Gateway 會串流您的回應長達 5 分鐘。

下列update-integration命令會將 整合的傳輸模式從 更新STREAM為 BUFFERED：

```
aws apigateway update-integration \
 --rest-api-id a1b2c3 \
 --resource-id aaa111 \
 --http-method GET \
 --patch-operations "op='replace',path='/responseTransferMode',value=BUFFERED" \
 --region us-west-1
```

您需要重新部署 API，變更才會生效。

在 API Gateway 中設定 Lambda 代理整合與承載回應串流

您可以串流 Lambda 函數的回應，以改善達到第一個位元組的時間 (TTFB) 效能，並在用戶端可用時將部分回應傳回給用戶端。API Gateway 要求您使用 InvokeWithResponseStream Lambda API 來叫用 Lambda 函數。API Gateway 會將事件物件傳遞至 Lambda 函數。後端 Lambda 函數會剖析傳入請求資料，以判斷其所傳回的回應。為了讓 API Gateway 串流 Lambda 輸出，Lambda 函數必須輸出 API Gateway 所需的格式。

串流和緩衝回應傳輸模式之間 Lambda 代理整合的差異

下列清單說明 Lambda 代理整合與回應串流的 Lambda 代理整合之間的差異：

- API Gateway 使用 InvokeWithResponseStream API 來叫用 Lambda 代理整合以進行回應串流。這會產生不同的 URI，如下所示：

  ```
  arn:aws:apigateway:us-west-1:lambda:path/2021-11-15/functions/arn:aws:lambda:us-west-1:111122223333:function:my-function-name/response-streaming-invocations
  ```

  相較於 Lambda 代理整合，此 ARN 使用不同的 API 版本日期和服務動作。

  如果您使用 API Gateway 主控台進行回應串流，主控台會為您使用正確的 URI。
- 在 Lambda 代理整合中，API Gateway 只會在收到 Lambda 的完整回應後將回應傳送至用戶端。在回應串流的 Lambda 代理整合中，API Gateway 會在從 Lambda 收到有效的中繼資料和分隔符號後開始承載串流。
- 回應串流的 Lambda 代理整合使用與代理整合相同的輸入格式，但需要不同的輸出格式。

用於回應串流的 Lambda 代理整合格式

當 API Gateway 使用回應串流叫用 Lambda 函數時，輸入格式與 Lambda 函數的輸入格式相同，以進行代理整合。如需詳細資訊，請參閱the section called “代理整合之 Lambda 函數的輸入格式”。

當 Lambda 將回應串流至 API Gateway 時，回應必須遵循下列格式。此格式使用分隔符號來分隔中繼資料 JSON 和原始承載。在此情況下，承載資料會在串流 Lambda 函數傳輸時串流：

```
{
  "headers": {"headerName": "headerValue", ...},
  "multiValueHeaders": { "headerName": ["headerValue", "headerValue2", ...], ... },
  "cookies" : ["cookie1", "cookie2"],
  "statusCode": httpStatusCode
}<DELIMITER>PAYLOAD1 | PAYLOAD2 | PAYLOAD3
```

在輸出中：

- 如果沒有傳回額外的回應標頭headersmultiValueHeaders，則無法指定 、cookies、 和 statusCode金鑰。
- headers 鍵只能包含單一值標頭。
- 輸出預期標頭包含 Transfer-Encoding: chunked或 Content-length: *number*。如果您的函數未傳回其中一個標頭，API Gateway 會附加Transfer-Encoding: chunked到回應標頭。
- 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。
- 中繼資料必須是有效的 JSON。僅支援 `headers`、 `multiValueHeaders`cookies和 `statusCode`金鑰。
- 您必須在中繼資料 JSON 之後提供分隔符號。分隔符號必須是 8 個 null 位元組，且必須出現在串流資料的前 16KB 內。
- API Gateway 不需要方法回應承載的特定格式。

如果您使用函數 URL 串流 Lambda 函數，則必須修改 Lambda 函數的輸入和輸出，以滿足這些要求。

如果您的 Lambda 函數輸出不符合此格式的要求，API Gateway 可能仍會叫用您的 Lambda 函數。下表顯示 API Gateway 支援的 API 整合請求設定和 Lambda 函數程式碼的組合。這包括緩衝的回應傳輸模式支援的組合。

| 回應傳輸模式 | 函數程式碼遵循所需的格式 | Lambda 叫用 API | API Gateway 支援 |
|---|---|---|---|
| 串流 | 是 | InvokeWithResponseStream | 是。API Gateway 會串流您的回應。 |
| 串流 | 否 | InvokeWithResponseStream | 否。API Gateway 會叫用您的 Lambda 函數並傳回 500 錯誤回應。 |
| 串流 | 是 | Invoke | 否。API Gateway 不支援此整合組態。 |
| 串流 | 否 | Invoke | 否。API Gateway 不支援此整合組態。 |
| 緩衝 | 是 | InvokeWithResponseStream | 否。API Gateway 不支援此整合組態。 |
| 緩衝 | 否 | InvokeWithResponseStream | 否。API Gateway 不支援此整合組態。 |

| 回應傳輸模式 | 函數程式碼遵循所需的格式 | Lambda 叫用 API | API Gateway 支援 |
| --- | --- | --- | --- |
| 緩衝 | 是 | Invoke | API Gateway 會傳回 HTTP 標頭和狀態碼，但不會傳回回應內文。 |
| 緩衝 | 否 | Invoke | 是。這是 Lambda 代理整合。如需詳細資訊，請參閱 Lambda 代理整合。 |

在 API Gateway 中設定 Lambda 代理整合與承載回應串流

當您設定回應承載串流時，您可以在資源的整合請求中指定傳輸模式。您可以在整合請求中設定這些設定，以控制 API Gateway 在整合回應之前和期間的行為。

回應串流的 Lambda 函數範例

您的 Lambda 函數必須遵循 the section called “用於回應串流的 Lambda 代理整合格式”。我們建議您使用三個範例 Lambda 函數之一來測試回應串流。當您建立 Lambda 函數時，請務必執行下列動作：

- 為您的函數提供足夠的逾時。我們建議您設定至少 1 分鐘的逾時，以了解回應串流。當您建立生產資源時，請確定 Lambda 函數逾時涵蓋完整的請求週期。如需詳細資訊，請參閱設定 Lambda 函數逾時。
- 使用最新的 Node.js 執行時間。
- 使用可使用 Lambda 回應串流的區域。

Using HttpResponseStream.from

下列程式碼範例使用 `awslambda.HttpResponseStream()`方法將 JSON 中繼資料物件和承載串流回用戶端，而不使用管道方法。您不需要建立分隔符號。如需詳細資訊，請參閱撰寫啟用回應串流功能的 Lambda 函數。

```
export const handler = awslambda.streamifyResponse(
  async (event, responseStream, context) => {
    const httpResponseMetadata = {
```

```
      "statusCode": 200,
      "headers": {
        "x-foo": "bar"
      },
      "multiValueHeaders": {
        "x-mv1": ["hello", "world"],
        "Set-Cookie": ["c1=blue", "c2=red"]
      }
    };

    responseStream = awslambda.HttpResponseStream.from(responseStream,
 httpResponseMetadata);
    await new Promise(r => setTimeout(r, 1000)); // synthetic delay

    responseStream.write("First payload ");
    await new Promise(r => setTimeout(r, 1000)); // synthetic delay

    responseStream.write("Final payload");
    responseStream.end();
});
```

Using the pipeline method

Lambda 建議您在撰寫啟用回應串流的函數時，使用原生 Node.js 執行時間提供的`awslambda.streamifyResponse()`裝飾項目，以及 `pipeline()`方法。當您使用管道方法時，您不需要建立分隔符號，Lambda 會為您執行此操作。如需詳細資訊，請參閱撰寫啟用回應串流功能的 Lambda 函數。

下列程式碼範例會將 JSON 中繼資料物件和三個承載串流回用戶端。

```
import { pipeline } from 'node:stream/promises';
import { Readable } from 'node:stream';

export const handler = awslambda.streamifyResponse(
  async (event, responseStream, context) => {
    const httpResponseMetadata = {
      statusCode: 200,
      headers: {
        "Content-Type": "text/plain",
        "X-Custom-Header": "Example-Custom-Header"
      }
    };
```

```
    responseStream = awslambda.HttpResponseStream.from(responseStream,
 httpResponseMetadata);

    const dataStream = Readable.from(async function* () {
      yield "FIRST payload\n";
      await new Promise(r => setTimeout(r, 1000));
      yield "SECOND payload\n";
      await new Promise(r => setTimeout(r, 1000));
      yield "THIRD payload\n";
      await new Promise(r => setTimeout(r, 1000));
    }());

    await pipeline(dataStream, responseStream);
  }
);
```

Without using the pipeline method

下列程式碼範例會將 JSON 中繼資料物件和三個承載串流回用戶端，而無需使用 awslambda.HttpResponseStream()方法。如果沒有 awslambda.HttpResponseStream()方法，您必須在中繼資料和承載之間包含 8 個 null 位元組的分隔符號。

```
export const handler = awslambda.streamifyResponse(async (event, response, ctx) => {
  response.write('{"statusCode": 200, "headers": {"hdr-x": "val-x"}}');
  response.write("\x00".repeat(8)); // DELIMITER
  await new Promise(r => setTimeout(r, 1000));

  response.write("FIRST payload");
  await new Promise(r => setTimeout(r, 1000));

  response.write("SECOND payload");
  await new Promise(r => setTimeout(r, 1000));

  response.write("FINAL payload");
  response.end();
});
```

### 建立與承載回應串流的 Lambda 代理整合

下列程序說明如何建立與承載回應串流的 Lambda 代理整合。使用範例 Lambda 函數或建立您自己的函數。

AWS 管理主控台

建立與承載回應串流的 Lambda 代理整合

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 選擇 REST API。
3. 選擇建立資源。
4. 針對資源名稱，輸入 **`streaming`**。
5. 選擇建立資源。
6. 選取 /streaming 資源後，選擇建立方法。
7. 針對方法類型，選擇任何。
8. 對於 整合類型，選擇 Lambda。
9. 選擇 Lambda 代理整合。
10. 針對回應傳輸模式，選擇串流。
11. 針對 Lambda 函數，選擇 Lambda 函數的名稱。

    API Gateway 主控台會自動使用 InvokeWithResponseStream API 來叫用 Lambda 函數。您負責撰寫啟用回應串流功能的 Lambda 函數。如需範例，請參閱 the section called “回應串流的 Lambda 函數範例”。
12. 選擇建立方法。

建立方法之後，請部署您的 API。

部署 API

1. 選擇部署 API。
2. 針對階段，選取新階段。
3. 針對階段名稱，輸入 **`prod`**。
4. 在描述，請輸入描述。
5. 選擇部署。

AWS CLI

下列程序說明如何匯入將 responseTransferMode 設為 的新 APISTREAM。如果您有現有的整合 API 並想要修改 responseTransferMode，請參閱 the section called “更新 Lambda 代理整合的回應傳輸模式”。

使用承載回應串流建立新的 API

1. 複製下列 Open API 檔案，然後將其儲存為 ResponseStreamDemoSwagger.yaml。在此檔案中， responseTransferMode 設定為 STREAM，而整合 URI 設定為 arn:aws:apigateway:us-west-1:lambda:path/2021-11-15/functions/arn:aws:lambda:us-west-1:111122223333:function:my-function-name/response-streaming-invocations。

   將 中的函數名稱取代my-function為啟用串流的函數，並將登入資料取代為具有允許apigateway服務呼叫 Lambda 函數之政策的 IAM 角色。

```
openapi: "3.0.1"
info:
  title: "ResponseStreamingDemo"
  version: "2025-04-28T17:28:25Z"
servers:
- url: "{basePath}"
  variables:
    basePath:
      default: "prod"
paths:
  /lambda:
    get:
      x-amazon-apigateway-integration:
        httpMethod: "POST"
        uri: "arn:aws:apigateway:us-west-1:lambda:path/2021-11-15/functions/
arn:aws:lambda:us-west-1:111122223333:function:my-function-name/response-
streaming-invocations"
        type: "aws_proxy"
        timeoutInMillis: 90000
        responseTransferMode: "STREAM"
        credentials: "arn:aws:iam::111122223333:role/apigateway-lambda-role"
```

   您可以使用 Lambda 的 add-permission命令來新增以資源為基礎的許可，而不是為登入資料提供 IAM 角色。

2. 使用下列`import-rest-api`命令匯入您的 OpenAPI 定義：

```
aws apigateway import-rest-api \
  --body 'fileb://~/ResponseStreamDemoSwagger.yaml' \
  --parameters endpointConfigurationTypes=REGIONAL \
  --region us-west-1
```

3. 使用下列`create-deployment`命令將您的新 API 部署到階段：

```
aws apigateway create-deployment \
  --rest-api-id a1b2c2 \
  --stage-name prod \
  --region us-west-1
```

更新 Lambda 代理整合的回應傳輸模式

下列程序說明如何更新 Lambda 代理整合的回應傳輸模式。當您將回應傳輸模式變更為串流時，請更新您的 Lambda 函數，使其遵守回應串流的要求。使用範例 Lambda 函數或建立您自己的函數。

AWS 管理主控台

更新 Lambda 代理整合的回應傳輸模式

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 選擇 REST API。
3. 選擇一個方法。
4. 在整合請求索引標籤上，於整合請求設定下，選擇編輯。
5. 針對回應傳輸模式，選擇串流。
6. 針對 Lambda 函數，選擇 Lambda 函數的名稱。
7. 選擇儲存。

更新方法後，請部署您的 API。

部署 API

1. 選擇部署 API。
2. 針對階段，選取新階段。

3. 針對階段名稱，輸入 **prod**。
4. 在描述，請輸入描述。
5. 選擇部署。

AWS CLI

1. 更新您的 Lambda 函數以啟用串流。
2. 使用下列 AWS CLI 命令來更新整合的整合 URI 和回應傳輸模式：

```
aws apigateway update-integration \
 --rest-api-id a1b2c3 \
 --resource-id aaa111 \
 --http-method ANY \
 --patch-operations "[{\"op\":\"replace\",\"path\":\"/uri\",\"value
\":\"arn:aws:apigateway:us-west-1:lambda:path/2021-11-15/functions/
arn:aws:lambda:us-west-1:111122223333:function:my-function-name/response-
streaming-invocations\"}, {\"op\":\"replace\",\"path\":\"/responseTransferMode
\",\"value\":\"STREAM\"}]" \
 --region us-west-1
```

3. 重新部署 API 以使變更生效。

針對 API Gateway 中的回應串流問題進行故障診斷

下列疑難排解指引可能有助於解決使用回應串流APIs 的問題。

一般性問題的故障診斷

您可以使用 TestInvokeMethod 或主控台的測試索引標籤來測試串流回應。下列考量事項可能會影響您使用測試叫用進行回應串流：

- 當您測試方法時，API Gateway 會緩衝串流的回應承載。一旦符合下列任一條件，API Gateway 便會傳回包含緩衝承載的一次性回應：
  - 請求已完成
  - 已經過 35 秒
  - 已緩衝超過 1 MB 的回應承載
- 如果超過 35 秒才傳回 HTTP 回應狀態和所有標頭，則 TestInvokeMethod 中傳回的回應狀態為 0。
- API Gateway 不會產生任何執行日誌。

部署 API 之後，您可以使用 curl 命令來測試串流回應。建議您使用 `-i`選項，在輸出中包含通訊協定回應標頭。若要查看到達時的回應資料，請使用 curl 選項 `--no-buffer`

針對 cURL 錯誤進行故障診斷

如果您正在測試整合並收到錯誤 `curl: (18) transfer closed with outstanding read data remaining`，請確定整合的逾時時間夠長。如果您使用的是 Lambda 函數，則需要更新 Lambda 函數的回應逾時。如需詳細資訊，請參閱設定 Lambda 函數逾時。

使用存取記錄進行故障診斷

您可以使用 REST API 階段的存取日誌來記錄回應串流並進行疑難排解。除了現有的變數之外，您還可以使用下列存取日誌變數：

`$context.integration.responseTransferMode`

整合的回應傳輸模式。可以是 `BUFFERED` 或 `STREAMED`。

`$context.integration.timeToAllHeaders`

從用戶端接收所有整合回應標頭時，API Gateway 與 建立整合連線之間的時間。

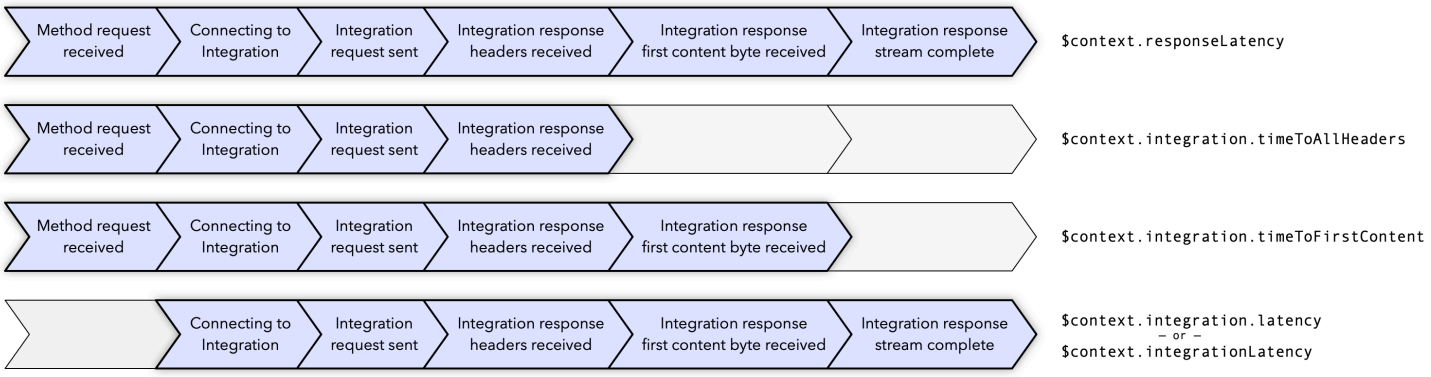
`$context.integration.timeToFirstContent`

API Gateway 在收到第一個內容位元組時建立與 的整合連線之間的時間。

`$context.integration.latency` 或 `$context.integrationLatency`

當整合回應串流完成時，API Gateway 與 建立整合連線的時間。

下圖顯示這些存取日誌變數如何代表回應串流的不同元件。




如需存取日誌的詳細資訊，請參閱the section called “CloudWatch 日誌”。您也可以使用 X-Ray 來監控回應串流。如需詳細資訊，請參閱the section called “X-Ray”。

# API Gateway 中 REST API 的私有整合

使用私有整合公開 Amazon VPC 內的 HTTP/HTTPS 資源，供 VPC 外部的用戶端存取。這會將對私有 VPC 資源的存取延伸到 VPC 邊界之外。您可以使用 API Gateway 支援的任何授權方法來控制對您的 API 的存取。

若要建立私有整合，您必須先建立 VPC 連結。API Gateway 支援適用於 REST APIs VPC 連結 V2。VPC 連結 V2 可讓您建立私有整合，將 REST API 連線至 Application Load Balancer，而無需使用 Network Load Balancer。使用 Application Load Balancer 可讓您連線至 Amazon ECSs容器型應用程式和許多其他後端。VPC 連結 V1 被視為舊版整合類型。雖然 API Gateway 支援這些連結，但建議您不要建立新的 VPC 連結 V1。

考量事項

下列考量事項可能會影響您使用私有整合：

- 所有資源都必須由相同的 擁有 AWS 帳戶。這包括負載平衡器、VPC 連結和 REST API。
- 依預設，私有整合流量會使用 HTTP 通訊協定。若要使用 HTTPS，請指定包含安全伺服器名稱uri的 ，例如 `https://example.com:443/test`。
- 在私有整合中，API Gateway 會在對後端資源的請求中包含 API 端點的階段部分。例如，如果您請求 API 的`test`階段，API Gateway 會在私有整合的請求`test/path`中包含 。若要從對後端資源的請求移除階段名稱，請使用參數映射來建立`$context.requestOverride.path`變數的覆寫。
- AWS Cloud Map 不支援與 的私有整合。

主題

- 在 API Gateway 中設定 VPC 連結 V2
- 設定私有整合
- 使用 VPC 連結 V1 的私有整合 （ 舊版 ）

在 API Gateway 中設定 VPC 連結 V2

VPC 連結可讓您建立私有整合，將 API 路由連線至 VPC 中的私有資源，例如 Application Load Balancer 或 Amazon ECS 容器型應用程式。私有整合使用 VPC 連結 V2 封裝 API Gateway 與目標 VPC 資源之間的連線。您可以重複使用不同資源和 APIs VPC 連結。

當您建立 VPC 連結時，API Gateway 會建立和管理您帳戶中 VPC 連結 V2 的彈性網路介面。此程序需要幾分鐘的時間。當 VPC 連結 V2 可供使用時，其狀態會從 轉換為 `PENDING` `AVAILABLE`。

> **Note**
>
> 如果 60 天內沒有透過 VPC 連結傳送流量，則會變成 INACTIVE。當 VPC 連結處於 INACTIVE 狀態時，API Gateway 會刪除所有 VPC 連結的網路介面。這會導致依賴 VPC 連結的 API 請求失敗。如果 API 請求繼續，API Gateway 會重新佈建網路介面。建立網路介面並重新啟用 VPC 連結可能需要幾分鐘的時間。您可以使用 VPC 連結狀態來監控 VPC 連結的狀態。

使用 建立 VPC 連結 V2 AWS CLI

若要建立 VPC 連結 V2，所有涉及的資源都必須由相同的 AWS 帳戶擁有。以下 create-vpc-link 命令會建立 VPC 連結：

```
aws apigatewayv2 create-vpc-link --name MyVpcLink \
    --subnet-ids subnet-aaaa subnet-bbbb \
    --security-group-ids sg1234 sg5678
```

> **Note**
>
> VPC 連結 V2 是不可變的。建立 VPC 連結 V2 之後，您無法變更其子網路或安全群組。

使用 刪除 VPC 連結 V2 AWS CLI

以下 delete-vpc-link 命令會刪除 VPC 連結。

```
aws apigatewayv2 delete-vpc-link --vpc-link-id abcd123
```

各個區域的供應情形

下列區域和可用區域支援 VPC 連結 V2：

| 區域名稱 | 區域 | 支援的可用區域 |
| --- | --- | --- |
| 美國東部 (俄亥俄) | us-east-2 | use2-az1、use2-az2、use2-az3 |

| 區域名稱 | 區域 | 支援的可用區域 |
| --- | --- | --- |
| 美國東部 (維吉尼亞北部) | us-east-1 | use1-az1、use1-az2、use1-az4、use1-az5、use1-az6 |
| 美國西部 (加利佛尼亞北部) | us-west-1 | usw1-az1、usw1-az3 |
| 美國西部 (奧勒岡) | us-west-2 | usw2-az1、usw2-az2、usw2-az3、usw2-az4 |
| 亞太地區 (香港) | ap-east-1 | ape1-az2、ape1-az3 |
| 亞太區域 (孟買) | ap-south-1 | aps1-az1、aps1-az2、aps1-az3 |
| 亞太地區 (大阪) | ap-northeast-3 | apne3-az1、apne3-az2、apne3-az3 |
| 亞太區域 (首爾) | ap-northeast-2 | apne2-az1、apne2-az2、apne2-az3 |
| 亞太區域 (新加坡) | ap-southeast-1 | apse1-az1、apse1-az2、apse1-az3 |
| 亞太地區 (悉尼) | ap-southeast-2 | apse2-az1、apse2-az2、apse2-az3 |
| 亞太區域 (東京) | ap-northeast-1 | apne1-az1、apne1-az2、apne1-az4 |
| 加拿大 (中部) | ca-central-1 | cac1-az1、cac1-az2 |
| 歐洲 (法蘭克福) | eu-central-1 | euc1-az1、euc1-az2、euc1-az3 |
| 歐洲 (愛爾蘭) | eu-west-1 | euw1-az1、euw1-az2、euw1-az3 |

| 區域名稱 | 區域 | 支援的可用區域 |
| --- | --- | --- |
| 歐洲 (倫敦) | eu-west-2 | euw2-az1、euw2-az2、euw2-az3 |
| Europe (Paris) | eu-west-3 | euw3-az1、euw3-az3 |
| 歐洲 (西班牙) | eu-south-2 | eus2-az1、eus2-az2、eus2-az3 |
| Europe (Stockholm) | eu-north-1 | eun1-az1、eun1-az2、eun1-az3 |
| Middle East (Bahrain) | me-south-1 | mes1-az1、mes1-az2、mes1-az3 |
| 南美洲 (聖保羅) | sa-east-1 | sae1-az1、sae1-az2、sae1-az3 |
| AWS GovCloud （美國西部） | us-gov-west-1 | usgw1-az1、usgw1-az2、usgw1-az3 |

設定私有整合

若要建立與 Application Load Balancer 或 Network Load Balancer 的私有整合，您可以建立 HTTP 代理整合、指定要使用的 VPC 連結 V2，並提供 Network Load Balancer 或 Application Load Balancer 的 ARN。依預設，私有整合流量會使用 HTTP 通訊協定。若要使用 HTTPS，請指定包含安全伺服器名稱`uri`的 ，例如 `https://example.com:443/test`。如需如何使用私有整合建立 REST API 的完整教學課程，請參閱 the section called “教學：建立具私有整合的 REST API”。

建立私有整合

下列程序說明如何使用 VPC 連結 V2 建立連線至負載平衡器的私有整合。

AWS 管理主控台

> 如需如何建立私有整合的教學課程，請參閱 the section called “教學：建立具私有整合的 REST API”。

AWS CLI

下列 put-integration 命令會使用 VPC 連結 V2 建立連線至負載平衡器的私有整合：

```
aws apigateway put-integration \
    --rest-api-id abcdef123 \
    --resource-id aaa000 \
    --integration-target 'arn:aws:elasticloadbalancing:us-
east-2:111122223333:loadbalancer/app/myLoadBalancerName/1234567891011' \
    --uri 'https://example.com:443/path' \
    --http-method GET \
    --type HTTP_PROXY \
    --integration-http-method GET \
    --connection-type VPC_LINK \
    --connection-id bbb111
```

您可以改用階段變數，而不是直接提供連線 ID。當您將 API 部署到階段時，您可以設定 VPC 連結 V2 ID。下列 put-integration 命令會使用 VPC 連結 V2 ID 的階段變數建立私有整合：

```
aws apigateway put-integration \
    --rest-api-id abcdef123 \
    --resource-id aaa000 \
    --integration-target 'arn:aws:elasticloadbalancing:us-
east-2:111122223333:loadbalancer/app/myLoadBalancerName/1234567891011' \
    --uri 'https://example.com:443/path' \
    --http-method GET \
    --type HTTP_PROXY \
    --integration-http-method GET \
    --connection-type VPC_LINK \
    --connection-id "\${stageVariables.vpcLinkV2Id}"
```

請務必對階段變數表達式 (${stageVariables.vpcLinkV2Id}) 進行雙引號，並逸出 $ 字元。

OpenAPI

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 `VpcLinkV2` 的 ID 或參考 `VpcLinkV2` 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 VpcLinkV2 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.vpcLinkIdV2}) 所參考的 VPC 連結：

```
{
  "swagger": "2.0",
  "info": {
    "version": "2017-11-17T04:40:23Z",
    "title": "MyApiWithVpcLinkV2"
  },
  "host": "abcdef123.execute-api.us-west-2.amazonaws.com",
  "basePath": "/test",
  "schemes": [
    "https"
  ],
  "paths": {
    "/": {
      "get": {
        "produces": [
          "application/json"
        ],
        "responses": {
          "200": {
            "description": "200 response",
            "schema": {
              "$ref": "#/definitions/Empty"
            }
          }
        },
        "x-amazon-apigateway-integration": {
          "responses": {
            "default": {
              "statusCode": "200"
            }
          },
          "uri": "https://example.com:443/path",
          "passthroughBehavior": "when_no_match",
          "connectionType": "VPC_LINK",
          "connectionId": "${stageVariables.vpcLinkV2Id}",
```

```
          "integration-target": "arn:aws:elasticloadbalancing:us-
east-2:111122223333:loadbalancer/app/myLoadBalancerName/1234567891011",
          "httpMethod": "GET",
          "type": "http_proxy"
        }
      }
    }
  },
  "definitions": {
    "Empty": {
      "type": "object",
      "title": "Empty Schema"
    }
  }
}
```

更新私有整合

下列範例會更新私有整合的 VPC 連結 V2。

AWS 管理主控台

更新私有整合

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 選擇具有私有整合的 REST API。
3. 選擇使用私有整合的資源和方法。
4. 在整合請求標籤上，於整合請求設定下，選擇編輯。
5. 您可以編輯私有整合的設定。如果您目前正在使用 VPC 連結 V1，您可以將 VPC 連結變更為 VPC 連結 V2。
6. 選擇儲存。
7. 重新部署 API 以使變更生效。

AWS CLI

下列 update-integration 命令會更新私有整合，以使用 VPC 連結 V2：

```
aws apigateway update-integration \
```

```
    --rest-api-id a1b2c3d4e5 \
    --resource-id a1b2c3 \
    --http-method GET \
    --patch-operations "[{\"op\":\"replace\",\"path\":\"/connectionId\",\"value
\":\"pk0000\"}, {\"op\":\"replace\",\"path\":\"/uri\",\"value\":\"http://
example.com\"}, {\"op\":\"replace\",\"path\":\"/integrationTarget\",\"value
\":\"arn:aws:elasticloadbalancing:us-east-2:111122223333:loadbalancer/app/
myLoadBalancerName/1234567891011\"}]"
```

使用 VPC 連結 V1 的私有整合 （ 舊版 ）

> **Note**
>
> 下列私有整合實作會使用 VPC 連結 V1。VPC 連結 V1 是舊版資源。建議您將 VPC 連結 V2 用於 REST APIs。

若要建立私有整合，您必須先建立 Network Load Balancer。您的 Network Load Balancer 必須有一個接聽程式，將請求路由至 VPC 中的資源。為了改善 API 可用性，請確定您的 Network Load Balancer 可將流量路由至 AWS 區域中一個以上可用區域內的資源。然後，您會建立用來連線 API 和 Network Load Balancer 的 VPC 連結。建立 VPC 連結後，您可以建立私有整合，以透過 VPC 連結和 Network Load Balancer 將流量從 API 路由到 VPC 中的資源。Network Load Balancer 和 API 必須擁有相同的 AWS 帳戶。

主題

- 設定 API Gateway 私有整合的 Network Load Balancer （ 舊版 ）
- 授予 API Gateway 建立 VPC 連結的許可 （ 舊版 ）
- 使用 AWS CLI （ 舊版 ）設定具有私有整合的 API Gateway API
- 用於私有整合的 API Gateway 帳戶 （ 舊版 ）

設定 API Gateway 私有整合的 Network Load Balancer （ 舊版 ）

> **Note**
>
> 下列私有整合實作會使用 VPC 連結 V1。VPC 連結 V1 是舊版資源。建議您將 VPC 連結 V2 用於 REST APIs。

下列程序概述使用 Amazon EC2 主控台設定 API Gateway 私有整合之 Network Load Balancer (NLB) 的步驟，並提供每個步驟之詳細說明的參考。

對於每個 VPC 中的資源，您只需要設定一個 NLB 和一個 VPCLink。每個 NLB 皆支援多個接聽程式和目標群組。您可以設定每個服務為 NLB 上特定的接聽程式，使用單一 VPCLink 連接到 NLB。當您在 API Gateway 中建立私有整合時，將使用指定到每個服務的特定連接埠定義每個服務。如需詳細資訊，請參閱the section called “教學：建立具私有整合的 REST API”。Network Load Balancer 和 API 必須擁有相同的 AWS 帳戶。

使用 API Gateway 主控台建立私有整合的 Network Load Balancer

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/ec2/：// 開啟 Amazon EC2 主控台。
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 2 上安裝 LAMP Web Server。
3. 建立 Network Load Balancer、註冊具有目標群組的 EC2 執行個體，並將目標群組新增至 Network Load Balancer 接聽程式。如需詳細資訊，請參閱 Network Load Balancer 入門中的指示。
4. 建立 Network Load Balancer 之後，執行下列操作：

    a. 記下 Network Load Balancer 的 ARN。您需要它才能在 API Gateway 中建立 VPC 連結，以整合 API 與受 Network Load Balancer 保護的 VPC 資源。

    b. 針對 PrivateLink 關閉安全群組評估。

    - 若要使用主控台關閉 PrivateLink 流量的安全群組評估，您可以選擇安全標籤，然後選擇編輯。在安全設定中，清除在 PrivateLink 流量強制執行輸入規則。
    - 使用以下 set-security-groups 命令關閉 PrivateLink 流量的安全群組評估：

```
aws elbv2 set-security-groups --load-balancer-
arn arn:aws:elasticloadbalancing:us-east-2:111122223333:loadbalancer/net/
my-loadbalancer/abc12345 \
  --security-groups sg-123345a --enforce-security-group-inbound-rules-on-
private-link-traffic off
```

> **Note**
>
> 請勿將任何相依性新增至 API Gateway CIDR，因為這些相依性必然會在未通知的情況下變更。

授予 API Gateway 建立 VPC 連結的許可（舊版）

> **Note**
>
> 下列私有整合實作會使用 VPC 連結 V1。VPC 連結 V1 是舊版資源。我們建議您將 VPC 連結 V2 用於 REST APIs。

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、更新和刪除 VPC 連結的許可

1. 建立與下列類似的 IAM 政策：

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "apigateway:POST",
                "apigateway:GET",
                "apigateway:PATCH",
                "apigateway:DELETE"
            ],
            "Resource": [
                "arn:aws:apigateway:us-east-1::/vpclinks",
                "arn:aws:apigateway:us-east-1::/vpclinks/*"
            ]
        },
        {
            "Effect": "Allow",
```

```
            "Action": [
                "elasticloadbalancing:DescribeLoadBalancers"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:CreateVpcEndpointServiceConfiguration",
                "ec2:DeleteVpcEndpointServiceConfigurations",
                "ec2:DescribeVpcEndpointServiceConfigurations",
                "ec2:ModifyVpcEndpointServicePermissions"
            ],
            "Resource": "*"
        }
    ]
}
```

如果您想要啟用 VPC 連結的標記，請務必允許標記操作。如需詳細資訊，請參閱the section called “允許標記操作”。

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 AWS CLI （舊版）設定具有私有整合的 API Gateway API

> **Note**
>
> 下列私有整合實作會使用 VPC 連結 V1。VPC 連結 V1 是舊版資源。建議您將 VPC 連結 V2 用於 REST APIs。

下列教學課程說明如何使用 AWS CLI 來建立 VPC 連結和私有整合。需要下列先決條件：

- 您需要使用 VPC 來源來建立並設定 Network Load Balancer，並將其當成目標。如需詳細資訊，請參閱設定 API Gateway 私有整合的 Network Load Balancer （舊版）。這必須與 API AWS 帳戶 位於相同的 中。您需要 Network Load Balancer ARN 才能建立 VPC 連結。
- 若要建立和管理 `VpcLink`，您需要可在 API 中建立 `VpcLink` 的許可權。使用 `VpcLink` 不需要許可權。如需詳細資訊，請參閱授予 API Gateway 建立 VPC 連結的許可 （舊版）。

使用 設定具有私有整合的 API AWS CLI

1. 使用以下 create-vpc-link 命令會建立以指定的 Network Load Balancer 為目標的 VpcLink：

```
aws apigateway create-vpc-link \
    --name my-test-vpc-link \
    --target-arns arn:aws:elasticloadbalancing:us-east-2:123456789012:loadbalancer/
net/my-vpclink-test-nlb/1234567890abcdef
```

此命令的輸出會確認收到請求，並顯示建立中 VpcLink 的 PENDING 狀態。

```
{
    "status": "PENDING",
    "targetArns": [
        "arn:aws:elasticloadbalancing:us-east-2:123456789012:loadbalancer/net/my-
vpclink-test-nlb/1234567890abcdef"
    ],
    "id": "gim7c3",
    "name": "my-test-vpc-link"
}
```

API Gateway 需要 2-4 分鐘才能完成建立 VpcLink。操作順利完成時，status 是 AVAILABLE。您可以使用下列 get-vpc-link 命令進行驗證：

```
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 Network Load Balancer 的 VpcLink，則會在 statusMessage 屬性上收到下列訊息：

```
"NLB is already associated with another VPC Endpoint Service"
```

順利建立 VpcLink 之後，您可以建立 API 並透過 VpcLink 將其與 VPC 資源整合。

記下新建立之 VpcLink 的 id 值。在此範例輸出中為 gim7c3。您需要它才能設定私有整合。

2. 使用以下 create-rest-api 命令建立 API Gateway RestApi 資源：

```
aws apigateway create-rest-api --name 'My VPC Link Test'
```

請記下所傳回結果中 RestApi 的 id 值，和 RestApi 的 rootResourceId 值。您需要此值才能對 API 執行進一步操作。

接著在根資源 (/) 上建立只有 GET 方法的 API，並整合此方法與 VpcLink。

3. 使用以下 put-method 命令建立 GET / 方法：

```
aws apigateway put-method \
       --rest-api-id  abcdef123 \
       --resource-id skpp60rab7 \
       --http-method GET \
       --authorization-type "NONE"
```

如果您未搭配 VpcLink 使用代理整合，至少也必須設定 200 狀態碼的方法回應。在這裡使用代理整合。

4. 建立 GET / 方法後，您可以設定整合。對於私有整合，您可以使用 connection-id 參數來提供 VpcLink ID。您可以使用階段變數或直接輸入 VpcLink ID。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。

Use the VPC link ID

使用以下 put-integration 命令，以直接在整合中使用 VpcLink ID：

```
aws apigateway put-integration \
    --rest-api-id abcdef123 \
    --resource-id skpp60rab7 \
    --uri 'http://my-vpclink-test-nlb-1234567890abcdef.us-east-2.amazonaws.com'
 \
    --http-method GET \
    --type HTTP_PROXY \
    --integration-http-method GET \
    --connection-type VPC_LINK \
    --connection-id gim7c3
```

Use a stage variable

使用以下 put-integration 命令，以使用階段變數來參考 VPC 連結 ID。當您將 API 部署到階段時，可以設定 VPC 連結 ID。

```
aws apigateway put-integration \
```

```
    --rest-api-id abcdef123 \
    --resource-id skpp60rab7 \
    --uri 'http://my-vpclink-test-nlb-1234567890abcdef.us-east-2.amazonaws.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。

您也可以隨時更新整合，以變更 `connection-id`。使用以下 update-integration 命令來更新整合：

```
 aws apigateway update-integration \
    --rest-api-id abcdef123 \
    --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。

因為您使用了私有代理整合，所以 API 現在已準備好進行部署和執行測試。

5. 如果使用階段變數來定義 `connection-id`，您需要部署 API 來進行測試。使用以下 create-deployment 命令，以部署有階段變數的 API：

```
aws apigateway create-deployment \
    --rest-api-id abcdef123 \
    --stage-name test \
    --variables vpcLinkId=gim7c3
```

若要以不同的 VpcLink ID 更新階段變數 (例如 *asf9d7*)，請使用 update-stage 命令：

```
aws apigateway update-stage \
    --rest-api-id abcdef123 \
    --stage-name test \
```

```
    --patch-operations op=replace,path='/variables/vpcLinkId',value='asf9d7'
```

當使用 VpcLink ID 常值硬式編碼 connection-id 屬性時，您不需要部署 API 來進行測試。使用 test-invoke-method 命令，在部署之前先測試您的 API。

6. 使用下列命令叫用 API：

```
curl -X GET https://abcdef123.execute-api.us-east-2.amazonaws.com/test
```

或者，您可以在 Web 瀏覽器中輸入 API 的調用 URL 來檢視結果。

用於私有整合的 API Gateway 帳戶 （舊版）

當您建立 VpcLink 時，下列特定區域 API Gateway 帳戶 ID 也會作為 AllowedPrincipals 自動新增至 VPC 端點服務。

| 區域 | 帳戶 ID |
| --- | --- |
| us-east-1 | 392220576650 |
| us-east-2 | 718770453195 |
| us-west-1 | 968246515281 |
| us-west-2 | 109351309407 |
| ca-central-1 | 796887884028 |
| eu-west-1 | 631144002099 |
| eu-west-2 | 544388816663 |
| eu-west-3 | 061510835048 |
| eu-central-1 | 474240146802 |
| eu-central-2 | 166639821150 |
| eu-north-1 | 394634713161 |
| eu-south-1 | 753362059629 |

| 區域 | 帳戶 ID |
| --- | --- |
| eu-south-2 | 359345898052 |
| ap-northeast-1 | 969236854626 |
| ap-northeast-2 | 020402002396 |
| ap-northeast-3 | 360671645888 |
| ap-southeast-1 | 195145609632 |
| ap-southeast-2 | 798376113853 |
| ap-southeast-3 | 652364314486 |
| ap-southeast-4 | 849137399833 |
| ap-south-1 | 507069717855 |
| ap-south-2 | 644042651268 |
| ap-east-1 | 174803364771 |
| sa-east-1 | 287228555773 |
| me-south-1 | 855739686837 |
| me-central-1 | 614065512851 |

## API Gateway 中 REST API 的模擬整合

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。如需這類登錄頁面的範例，請參閱教學課程：匯入範例來建立 REST API 中所討論的範例 API 之根資源上的 GET 方法整合請求和回應。

作為 API 開發人員，您可決定 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。

> **Note**
>
> 模擬整合並不支援大型回應範本。如果您的使用案例需要這類範本，應考慮改用 Lambda 整合。

您可使用整合請求映射範本來插入應用程式邏輯，以根據特定條件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。

在本節中，我們將說明如何使用 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

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。請遵循中的說明進行教學：建立具有 HTTP 非代理整合的 REST API

1. 選擇 API 資源，並選擇建立方法。

   若要建立方法，請執行下列動作：

   a. 針對方法類型，選取某個方法。
   b. 針對整合類型，選取模擬。
   c. 選擇建立方法。
   d. 在方法請求索引標籤上，針對方法請求設定，選擇編輯。
   e. 選擇 URL 查詢字串參數。選擇新增查詢字串，然後針對名稱輸入 **scope**。這個查詢參數會判斷發起人是否為內部。
   f. 選擇儲存。
2. 在方法回應索引標籤上，選擇建立回應，然後執行下列動作：

   a. 針對 HTTP 狀態，輸入 **500**。
   b. 選擇儲存。
3. 在整合請求索引標籤上，針對整合請求設定，選擇編輯。
4. 選擇對應範本，然後執行下列動作：

   a. 選擇新增映射範本。

b. 針對內容類型，輸入 **application/json**。

c. 針對範本內文，輸入下列內容：

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

d. 選擇儲存。

5. 在整合回應索引標籤上，針對預設 - 回應選擇編輯。

6. 選擇對應範本，然後執行下列動作：

a. 針對內容類型，輸入 **application/json**。

b. 針對範本內文，輸入下列內容：

```
{
    "statusCode": 200,
    "message": "Go ahead without me"
}
```

c. 選擇儲存。

7. 選擇建立回應。

若要建立 500 回應，請執行下列動作：

a. 對於 HTTP 狀態 regex，輸入 **5\d{2}**。

b. 針對方法回應狀態，選取 **500**。

c. 選擇儲存。

d. 針對 5\d{2} - 回應，選擇編輯。

e. 選擇對應範本，然後選擇新增對應範本。

f. 針對內容類型，輸入 **application/json**。

g. 針對範本內文，輸入下列內容：

```
{
    "statusCode": 500,
```

```
    "message": "The invoked method is not supported on the API resource."
}
```

h. 選擇儲存。

8. 選擇測試標籤。您可能需要選擇向右箭頭按鈕才能顯示此索引標籤。若要測試模擬整合，請執行下列動作：

a. 在查詢字串下，輸入 `scope=internal`。選擇 Test (測試)。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 `Query strings` 下輸入 `scope=public`，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，這就是 API Gateway 主控台透過傳回 CORS 必要標頭來啟用 CORS 支援的作法。

# API Gateway 中的 REST API 的請求驗證

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 中發布驗證結果。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。您可以驗證請求內文，方法為驗證必要的請求參數是否有效且為非 null 值，或指定模型結構描述進行更複雜的資料驗證。

主題

- API Gateway 中的基本請求驗證概觀
- REST API 的資料模型
- 在 API Gateway 中設定基本請求驗證
- AWS CloudFormation 具有基本請求驗證的範例 API 範本

## API Gateway 中的基本請求驗證概觀

API Gateway 可以執行基本請求驗證，以便您可以專注於後端的應用程式特定驗證。針對驗證，API Gateway 會驗證下列任一或兩個條件：

- 是否包含 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。API Gateway 只會檢查參數是否存在，而不會檢查類型或格式。
- 適用的請求承載會遵守方法中，針對所指內容類型設定的 JSON 結構描述請求。如果找不到相符的內容類型，則不會執行請求驗證。若無論內容類型為何都要使用相同的模型，請將資料模型的內容類型設定為 `$default`。

若要開啟驗證，您必須在請求驗證程式中指定驗證規則、將驗證程式新增至 API 的請求驗證程式對應，並將驗證程式指派給個別 API 方法。

> **Note**
>
> 請求內文驗證和 API Gateway 中 REST API 沒有映射範本時，承載的方法請求行為 是兩個獨立的主題。當請求承載沒有相符的模型結構描述時，您可以選擇傳遞或封鎖原始承載。如需詳細資訊，請參閱API Gateway 中 REST API 沒有映射範本時，承載的方法請求行為。

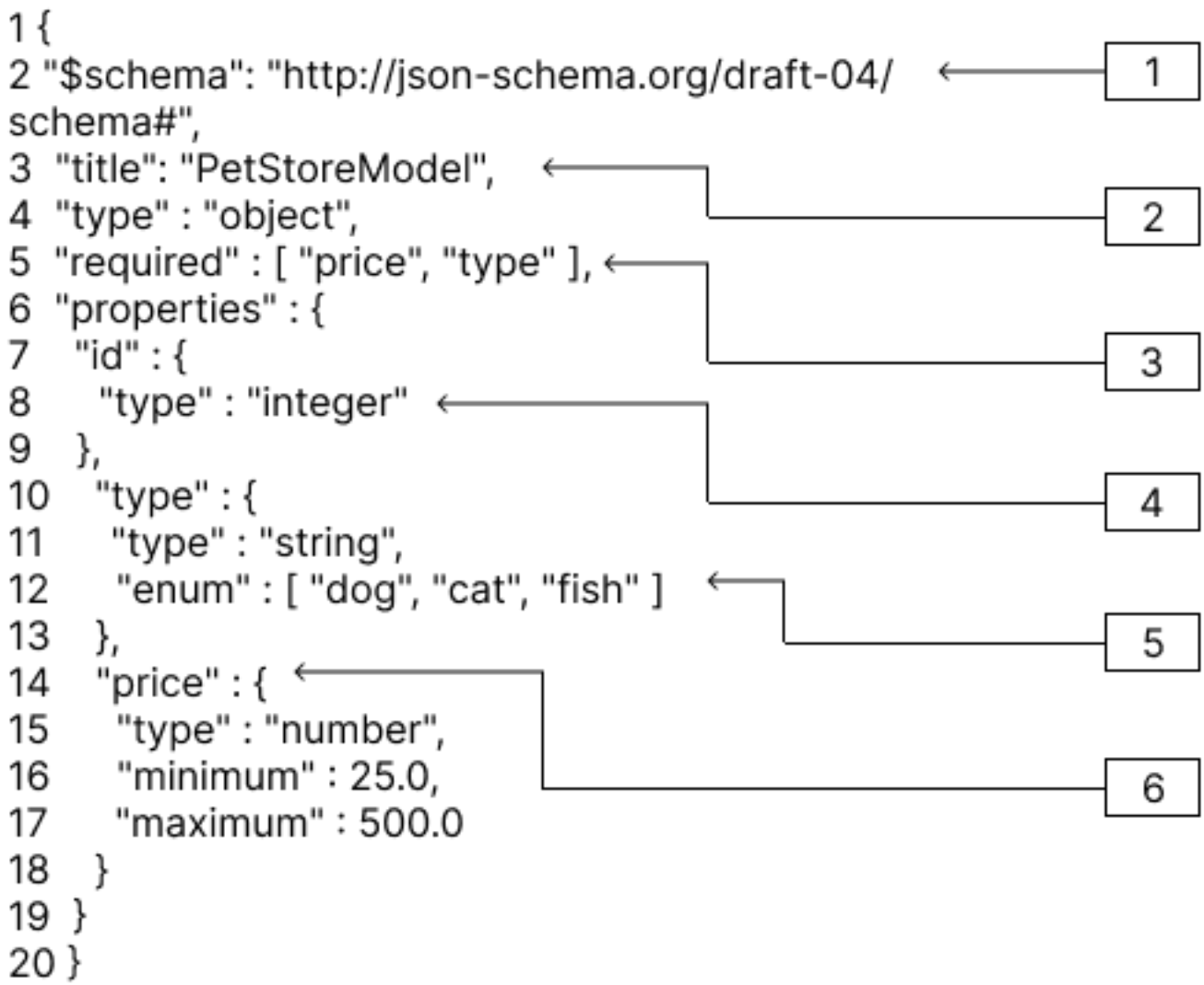
## REST API 的資料模型

在 API Gateway 中，模型會定義承載的資料結構。在 API Gateway 中，模型以 JSON 結構描述草稿第 4 版定義。下列 JSON 物件是 Pet Store 範例中的範例資料。

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

資料包含寵物的 `id`、`type` 和 `price`。此資料的模型可讓您：

- 使用基本請求驗證。
- 建立對應範本進行資料轉換。
- 產生 SDK 時，建立使用者定義的資料類型 (UDT)。




在這個模型中：

1. `$schema` 物件代表有效的 JSON 結構描述版本識別符。此結構描述為 JSON 結構描述草稿第 4 版。
2. `title` 物件是人類看得懂的模型識別符。這個標題是 `PetStoreModel`。
3. `required` 驗證關鍵字需要 `type` 和 `price`，才能進行基本請求驗證。
4. 模型的 `properties` 是 `id`、`type` 和 `price`。每個物件都有模型中描述的屬性。
5. 物件 `type` 只能具有值 `dog`、`cat` 或 `fish`。
6. 物件 `price` 是一個數字，且受到 `minimum` 為 25 且 `maximum` 為 500 的限制。

PetStore 模型

```
1 {
2 "$schema": "http://json-schema.org/draft-04/schema#",
3  "title": "PetStoreModel",
4  "type" : "object",
5  "required" : [ "price", "type" ],
6  "properties" : {
7    "id" : {
8      "type" : "integer"
9    },
10    "type" : {
11      "type" : "string",
12      "enum" : [ "dog", "cat", "fish" ]
13    },
14    "price" : {
15      "type" : "number",
16      "minimum" : 25.0,
17      "maximum" : 500.0
18    }
19  }
20 }
```

在這個模型中：

1. 在第 2 行，$schema 物件代表有效的 JSON 結構描述版本識別符。此結構描述為 JSON 結構描述草稿第 4 版。
2. 在第 3 行，title 物件是人類看得懂的模型識別符。這個標題是 PetStoreModel。
3. 在第 5 行，required 驗證關鍵字需要 type 和 price，才能進行基本請求驗證。
4. 在第 6 至 17 行，模型的 properties 是 id、type 和 price。每個物件都有模型中描述的屬性。
5. 在第 12 行，物件 type 只能具有值 dog、cat 或 fish。
6. 在第 14 至 17 行，物件 price 是一個數字，且受到 minimum 為 25 且 maximum 為 500 的限制。

### 建立更複雜的模型

您可以使用 $ref 基本值，為較長的模型建立可重複使用的定義。例如，您可以在描述 price 物件的 definitions 區段中建立稱為 Price 的定義。$ref 的值是 Price 定義。

```
{
  "$schema" : "http://json-schema.org/draft-04/schema#",
```

```
  "title" : "PetStoreModelReUsableRef",
  "required" : ["price", "type" ],
  "type" : "object",
  "properties" : {
    "id" : {
      "type" : "integer"
    },
    "type" : {
      "type" : "string",
      "enum" : [ "dog", "cat", "fish" ]
    },
    "price" : {
        "$ref": "#/definitions/Price"
    }
  },
  "definitions" : {
      "Price": {
        "type" : "number",
        "minimum" : 25.0,
        "maximum" : 500.0
            }
      }
}
```

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。將 $ref 屬性的值設定為模型的位置。在下列範例中，Price 模型是在 API a1234 的 PetStorePrice 模型中定義的。

```
{
  "$schema" : "http://json-schema.org/draft-04/schema#",
  "title" : "PetStorePrice",
  "type": "number",
  "minimum": 25,
  "maximum": 500
}
```

較長的模型可以參考 PetStorePrice 模型。

```
{
  "$schema" : "http://json-schema.org/draft-04/schema#",
  "title" : "PetStoreModelReusableRefAPI",
  "required" : [ "price", "type" ],
  "type" : "object",
  "properties" : {
```

```
    "id" : {
      "type" : "integer"
    },
    "type" : {
      "type" : "string",
      "enum" : [ "dog", "cat", "fish" ]
    },
    "price" : {
        "$ref": "https://apigateway.amazonaws.com/restapis/a1234/models/PetStorePrice"
    }
  }
}
```

使用輸出資料模型

如果轉換您的資料，您可以在整合回應中定義承載模型。產生 SDK 時，可以使用承載模型。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)。如果您在產生 SDK 時透過資料模型提供 UDT，則 API Gateway 會建立該 UDT。如需資料轉型的詳細資訊，請參閱 API Gateway 中 REST API 的映射範本轉換。

下列範例是整合回應的輸出資料。

```
{
[
  {
    "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
  }
]
}
```

下列範例是描述輸出資料的承載模型。

```
{
```

```
"$schema": "http://json-schema.org/draft-04/schema#",
  "title": "PetStoreOutputModel",
  "type" : "object",
  "required" : [ "description", "askingPrice" ],
  "properties" : {
    "description" : {
      "type" : "string"
    },
    "askingPrice" : {
      "type" : "number",
      "minimum" : 25.0,
      "maximum" : 500.0
    }
  }
}
```

有了此模型，您可以呼叫 SDK，透過讀取 `PetStoreOutputModel[i].description` 和 `PetStoreOutputModel[i].askingPrice` 屬性，來擷取 `description` 和 `askingPrice` 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。

後續步驟

- 本節提供的資源可讓您深入了解本主題中呈現的概念。

  您可以遵循請求驗證教學課程：

  - 使用 API Gateway 主控台設定請求驗證
  - 使用 設定基本請求驗證 AWS CLI
  - 使用 OpenAPI 定義來設定基本請求驗證
- 如需資料轉換和映射範本的詳細資訊，請參閱 API Gateway 中 REST API 的映射範本轉換。

## 在 API Gateway 中設定基本請求驗證

本節說明如何使用 主控台 AWS CLI和 OpenAPI 定義來設定 API Gateway 的請求驗證。

主題

- 使用 API Gateway 主控台設定請求驗證
- 使用 設定基本請求驗證 AWS CLI
- 使用 OpenAPI 定義來設定基本請求驗證

使用 API Gateway 主控台設定請求驗證

您可以使用 API Gateway 主控台來驗證請求，方法是為 API 請求選取三個驗證程式之一：

- 驗證內文。
- 驗證查詢字串參數和標頭。
- 驗證內文、查詢字串參數和標頭。

當您在 API 方法上套用其中一個驗證程式時，API Gateway 主控台會將驗證程式新增至 API 的 RequestValidators 對應。

若要遵循本教學課程，您將使用 CloudFormation 範本來建立不完整的 API Gateway API。此 API 具有 `/validator` 資源，其中含有 `GET` 和 `POST` 方法。這兩種方法會與 `http://petstore-demo-endpoint.execute-api.com/petstore/pets` HTTP 端點整合。您將設定兩種請求驗證：

- 在 `GET` 方法中，您將設定 URL 查詢字串參數的請求驗證。
- 在 `POST` 方法中，您將設定請求內文的請求驗證。

這將只允許某些 API 呼叫傳遞至 API。

下載並解壓縮 的應用程式建立範本 CloudFormation。您將使用此範本建立不完整的 API。您將完成 API Gateway 主控台中的其餘步驟。

建立 CloudFormation 堆疊

1. 在 https：//https://console.aws.amazon.com/cloudformation 開啟 CloudFormation 主控台。
2. 選擇 Create stack (建立堆疊)，然後選擇 With new resources (standard) (使用新資源 (標準))。
3. 對於 Specify template (指定範本)，選擇 Upload a template file (上傳範本檔案)。
4. 選取您下載的範本。
5. 選擇 Next (下一步)。
6. 針對 Stack name (堆疊名稱)，輸入 **`request-validation-tutorial-console`**，然後選擇 Next (下一步)。
7. 針對 Configure stack options (設定堆疊選項)，選擇 Next (下一步)。
8. 對於 功能，確認 CloudFormation 可以在您的帳戶中建立 IAM 資源。
9. 選擇下一步，然後選擇提交。

CloudFormation 會佈建範本中指定的資源。完成資源佈建可能需要幾分鐘的時間。當您的 CloudFormation 堆疊狀態為 CREATE_COMPLETE 時，您就可以繼續進行下一個步驟。

選取新建立的 API

1. 選取新建立的 **request-validation-tutorial-console** 堆疊。
2. 選擇資源。
3. 在實體 ID 下，選擇您的 API。此連結會將您導向至 API Gateway 主控台。

在修改 GET 和 POST 方法之前，您必須建立模型。

建立裝置

1. 需有模型才能在傳入請求的內文上使用請求驗證。若要建立模型，請在主導覽窗格中選擇模型。
2. 選擇建立模型。
3. 對於名稱，輸入 **PetStoreModel**。
4. 針對內容類型，輸入 **application/json**。如果找不到相符的內容類型，則不會執行請求驗證。若要使用相同的模型，而不論內容類型為何，請輸入 **$default**。
5. 對於描述，輸入 **My PetStore Model** 作為模型描述。
6. 對於模型結構描述，將下列模型貼入程式碼編輯器中，然後選擇建立。

```
{
  "type" : "object",
  "required" : [ "name", "price", "type" ],
  "properties" : {
    "id" : {
      "type" : "integer"
    },
    "type" : {
      "type" : "string",
      "enum" : [ "dog", "cat", "fish" ]
    },
    "name" : {
      "type" : "string"
    },
    "price" : {
      "type" : "number",
      "minimum" : 25.0,
      "maximum" : 500.0
```

```
    }
  }
}
```

如需關於模型的詳細資訊，請參閱REST API 的資料模型。

設定 **GET** 方法的請求驗證

1. 在主導覽窗格中，選擇資源，然後選取 GET 方法。
2. 在方法請求索引標籤的方法請求設定下，選擇編輯。
3. 對於請求驗證程式，選取驗證查詢字串參數與標頭。
4. 在 URL 查詢字串參數下，執行下列動作：
   a. 選擇新增查詢字串。
   b. 對於名稱，輸入 **petType**。
   c. 開啟必要。
   d. 讓快取保持關閉。
5. 選擇儲存。
6. 在整合請求索引標籤上，於整合請求設定下，選擇編輯。
7. 在 URL 查詢字串參數下，執行下列動作：
   a. 選擇新增查詢字串。
   b. 對於名稱，輸入 **petType**。
   c. 對於映射自，輸入 **method.request.querystring.petType**。這會將 **petType** 對應到寵物的類型。

      如需資料對應的詳細資訊，請參閱資料對應教學課程。
   d. 讓快取保持關閉。
8. 選擇儲存。

測試 **GET** 方法的請求驗證

1. 選擇測試標籤。您可能需要選擇向右箭頭按鈕才能顯示此索引標籤。
2. 對於查詢字串，輸入 **petType=dog**，然後選擇測試。
3. 方法測試會傳回 `200 OK` 及狗的清單。

如需如何轉換此輸出資料的相關資訊，請參閱資料對應教學課程。

4. 移除 **petType=dog** 並選擇測試。
5. 方法測試會傳回 400 錯誤，並顯示下列錯誤訊息：

```
{
  "message": "Missing required request parameters: [petType]"
}
```

設定 **POST** 方法的請求驗證

1. 在主導覽窗格中，選擇資源，然後選取 POST 方法。
2. 在方法請求索引標籤的方法請求設定下，選擇編輯。
3. 對於請求驗證程式，選取驗證內文。
4. 在請求內文下，選擇新增模型。
5. 針對內容類型，輸入 **application/json**。如果找不到相符的內容類型，則不會執行請求驗證。若要使用相同的模型，而不論內容類型為何，請輸入 $default。

   對於模型，選取 PetStoreModel。
6. 選擇儲存。

測試 **POST** 方法的請求驗證

1. 選擇測試標籤。您可能需要選擇向右箭頭按鈕才能顯示此索引標籤。
2. 對於請求內文，將下列內容貼入程式碼編輯器中：

```
{
  "id": 2,
  "name": "Bella",
  "type": "dog",
  "price": 400
}
```

   選擇測試。
3. 方法測試會傳回 200 OK 和成功訊息。
4. 對於請求內文，將下列內容貼入程式碼編輯器中：

```
{
  "id": 2,
  "name": "Bella",
  "type": "dog",
  "price": 4000
}
```

選擇測試。

5. 方法測試會傳回 400 錯誤，並顯示下列錯誤訊息：

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

測試日誌底部會傳回請求內文無效的原因。在此案例中，寵物的價格超出了模型中指定的最高價格。

### 刪除 CloudFormation 堆疊

1. 在 https：//https://console.aws.amazon.com/cloudformation 開啟 CloudFormation 主控台。
2. 選取您的 CloudFormation 堆疊。
3. 選擇刪除，然後確認您的選擇。

### 後續步驟

- 如需如何轉換輸出資料並執行其他資料對應的相關資訊，請參閱資料對應教學課程。
- 遵循使用 AWS CLI設定基本請求驗證教學課程，以使用 AWS CLI執行類似的步驟。

## 使用 設定基本請求驗證 AWS CLI

您可以使用 AWS CLI建立驗證程式來設定請求驗證。若要遵循本教學課程，您將使用 CloudFormation 範本來建立不完整的 API Gateway API。

> **Note**
>
> 這與主控台教學課程的 CloudFormation 範本不同。

使用預先公開的 /validator 資源，您將建立 GET 和 POST 方法。這兩種方法會與 http://petstore-demo-endpoint.execute-api.com/petstore/pets HTTP 端點整合。您將設定下列兩個請求驗證：

- 在 GET 方法上，您將建立 params-only 驗證程式來驗證 URL 查詢字串參數。
- 在 POST 方法上，您將建立 body-only 驗證程式來驗證請求內文。

這將只允許某些 API 呼叫傳遞至 API。

建立 CloudFormation 堆疊

下載並解壓縮 的應用程式建立範本 CloudFormation。

若要完成下列教學課程，您需要 AWS Command Line Interface (AWS CLI) 第 2 版。

對於長命令，逸出字元 (\) 用於將命令分割為多行。

> **Note**
>
> 在 Windows 中，作業系統的內建終端機不支援您常用的某些 Bash CLI 命令 (例如 zip)。若要取得 Ubuntu 和 Bash 的 Windows 整合版本，請安裝適用於 Linux 的 Windows 子系統。本指南中的 CLI 命令範例使用 Linux 格式。如果您使用的是 Windows CLI，必須重新格式化包含內嵌 JSON 文件的命令。

1. 使用下列命令來建立 CloudFormation 堆疊。

```
aws cloudformation create-stack --stack-name request-validation-tutorial-cli
 --template-body file://request-validation-tutorial-cli.zip --capabilities
 CAPABILITY_NAMED_IAM
```

2. CloudFormation 會佈建範本中指定的資源。完成資源佈建可能需要幾分鐘的時間。使用下列命令來查看 CloudFormation 堆疊的狀態。

```
aws cloudformation describe-stacks --stack-name request-validation-tutorial-cli
```

3. 當您的 CloudFormation 堆疊狀態為 時StackStatus: "CREATE_COMPLETE"，請使用下列命令來擷取未來步驟的相關輸出值。

```
aws cloudformation describe-stacks --stack-name request-validation-tutorial-cli
--query "Stacks[*].Outputs[*].{OutputKey: OutputKey, OutputValue: OutputValue,
Description: Description}"
```

輸出值如下：

- ApiId，這是 API 的 ID。對於本教學課程，API ID 為 abc123。
- ResourceId，這是 GET 和 POST 方法公開所在驗證程式資源的 ID。對於本教學課程，資源 ID 為 efg456

建立請求驗證程式並匯入模型

1. 需有驗證程式才能搭配 AWS CLI使用請求驗證。使用下列命令建立僅驗證請求參數的驗證程式。

```
aws apigateway create-request-validator --rest-api-id abc123 \
      --no-validate-request-body \
      --validate-request-parameters \
      --name params-only
```

請記下 params-only 驗證程式的 ID。

2. 使用下列命令建立僅驗證請求內文的驗證程式。

```
aws apigateway create-request-validator --rest-api-id abc123 \
      --validate-request-body \
      --no-validate-request-parameters \
      --name body-only
```

請記下 body-only 驗證程式的 ID。

3. 需有模型才能在傳入請求的內文上使用請求驗證。使用下列命令匯入模型。

```
aws apigateway create-model --rest-api-id abc123 --name PetStoreModel --description
 'My PetStore Model' --content-type 'application/json' --schema '{"type":
 "object", "required" : [ "name", "price", "type" ], "properties" : { "id" :
 {"type" : "integer"},"type" : {"type" : "string", "enum" : [ "dog", "cat",
 "fish" ]},"name" : { "type" : "string"},"price" : {"type" : "number","minimum" :
 25.0, "maximum" : 500.0}}}}'
```

如果找不到相符的內容類型，則不會執行請求驗證。若要使用相同的模型，而不論內容類型為何，請指定 $default 為索引鍵。

### 建立 GET 和 POST 方法

1. 使用下列命令，在 /validate 資源上新增 GET HTTP 方法。此命令會建立 GET 方法、新增 params-only 驗證程式，並視需要設定查詢字串 petType。

```
aws apigateway put-method --rest-api-id abc123 \
       --resource-id efg456 \
       --http-method GET \
       --authorization-type "NONE" \
       --request-validator-id aaa111 \
       --request-parameters "method.request.querystring.petType=true"
```

使用下列命令，在 /validate 資源上新增 POST HTTP 方法。此命令會建立 POST 方法、新增 body-only 驗證程式，並將模型附加至僅內文驗證程式。

```
aws apigateway put-method --rest-api-id abc123 \
       --resource-id efg456 \
       --http-method POST \
       --authorization-type "NONE" \
       --request-validator-id bbb222 \
       --request-models 'application/json'=PetStoreModel
```

2. 使用下列命令設定 GET /validate 方法的 200 OK 回應。

```
aws apigateway put-method-response --rest-api-id abc123  \
            --resource-id efg456 \
            --http-method GET \
            --status-code 200
```

使用下列命令設定 POST /validate 方法的 200 OK 回應。

```
aws apigateway put-method-response --rest-api-id abc123  \
            --resource-id efg456 \
            --http-method POST \
            --status-code 200
```

3. 使用下列命令，為 GET /validation 方法設定具有所指定 HTTP 端點的 Integration。

```
aws apigateway put-integration --rest-api-id abc123  \
            --resource-id efg456 \
            --http-method GET \
            --type HTTP \
            --integration-http-method GET \
            --request-parameters '{"integration.request.querystring.type" :
 "method.request.querystring.petType"}' \
            --uri 'http://petstore-demo-endpoint.execute-api.com/petstore/pets'
```

使用下列命令，為 POST /validation 方法設定具有所指定 HTTP 端點的 Integration。

```
aws apigateway put-integration --rest-api-id abc123  \
              --resource-id efg456 \
              --http-method POST \
              --type HTTP \
              --integration-http-method GET \
              --uri 'http://petstore-demo-endpoint.execute-api.com/petstore/pets'
```

4. 使用下列命令設定 GET /validation 方法的整合回應。

```
aws apigateway put-integration-response --rest-api-id abc123 \
              --resource-id efg456\
              --http-method GET \
              --status-code 200 \
              --selection-pattern ""
```

使用下列命令設定 POST /validation 方法的整合回應。

```
aws apigateway put-integration-response --rest-api-id abc123 \
            --resource-id efg456 \
            --http-method POST \
            --status-code 200 \
            --selection-pattern ""
```

若要測試 API

1. 若要測試將為查詢字串執行請求驗證的 GET 方法，請使用下列命令：

```
aws apigateway test-invoke-method --rest-api-id abc123 \
            --resource-id efg456 \
            --http-method GET \
            --path-with-query-string '/validate?petType=dog'
```

結果將傳回 200 OK 和狗清單。

2. 在不包括查詢字串 petType 的情況下，使用下列命令進行測試，

```
aws apigateway test-invoke-method --rest-api-id abc123 \
            --resource-id efg456 \
            --http-method GET
```

結果將傳回 400 錯誤。

3. 若要測試將為請求內文執行請求驗證的 POST 方法，請使用下列命令：

```
 aws apigateway test-invoke-method --rest-api-id abc123 \
            --resource-id efg456 \
            --http-method POST \
            --body '{"id": 1, "name": "bella", "type": "dog", "price" : 400 }'
```

結果將傳回 200 OK 和成功訊息。

4. 使用下列命令，以利用無效內文進行測試。

```
 aws apigateway test-invoke-method --rest-api-id abc123 \
              --resource-id efg456 \
              --http-method POST \
              --body '{"id": 1, "name": "bella", "type": "dog", "price" : 1000 }'
```

結果將傳回 400 錯誤，因為狗的價格超過了模型定義的最高價格。

### 刪除 CloudFormation 堆疊

- 使用下列命令來刪除您的 CloudFormation 資源。

```
aws cloudformation delete-stack  --stack-name request-validation-tutorial-cli
```

使用 OpenAPI 定義來設定基本請求驗證

您可以在 API 層級宣告請求驗證程式，方法是在 x-amazon-apigateway-request-validators 物件 對應中指定一組 x-amazon-apigateway-request-validators.requestValidator 物件 物件，以選取要驗證請求的哪一部分。在範例 OpenAPI 定義中，有兩個驗證程式：

- `all` 驗證程式，其會同時驗證內文 (使用 `RequestBodyModel` 資料模型) 和參數。

  此 `RequestBodyModel` 資料模型要求輸入 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。
- `param-only`，其僅會驗證參數。

若要在 API 的所有方法上開啟請求驗證程式，請在 OpenAPI 定義的 API 層級指定 x-amazon-apigateway-request-validator 屬性 屬性。在範例 OpenAPI 定義中，除非特別覆寫，否則會在所有 API 方法上使用 `all` 驗證程式。使用模型驗證主體時，如果找不到相符的內容類型，則不會執行請求驗證。若要使用相同的模型，而不論內容類型為何，請指定 `$default` 為索引鍵。

若要在個別方法上開啟請求驗證程式，請在方法層級指定 `x-amazon-apigateway-request-validator` 屬性。在範例 (OpenAPI 定義) 中，`param-only` 驗證程式會覆寫 `GET` 方法上的 `all` 驗證程式。

若要將 OpenAPI 範例匯入至 API Gateway，請參閱下列指示，將區域 API 匯入至 API Gateway 或 將邊緣最佳化 API 匯入至 API Gateway。

OpenAPI 3.0

```
{
  "openapi" : "3.0.1",
  "info" : {
    "title" : "ReqValidators Sample",
    "version" : "1.0.0"
  },
  "servers" : [ {
    "url" : "/{basePath}",
    "variables" : {
      "basePath" : {
        "default" : "/v1"
      }
    }
```

```
 } ],
 "paths" : {
   "/validation" : {
     "get" : {
       "parameters" : [ {
         "name" : "q1",
         "in" : "query",
         "required" : true,
         "schema" : {
           "type" : "string"
         }
       } ],
       "responses" : {
         "200" : {
           "description" : "200 response",
           "headers" : {
             "test-method-response-header" : {
               "schema" : {
                 "type" : "string"
               }
             }
           },
           "content" : {
             "application/json" : {
               "schema" : {
                 "$ref" : "#/components/schemas/ArrayOfError"
               }
             }
           }
         }
       },
       "x-amazon-apigateway-request-validator" : "params-only",
       "x-amazon-apigateway-integration" : {
         "httpMethod" : "GET",
         "uri" : "http://petstore-demo-endpoint.execute-api.com/petstore/pets",
         "responses" : {
           "default" : {
             "statusCode" : "400",
             "responseParameters" : {
               "method.response.header.test-method-response-header" : "'static
value'"
             },
             "responseTemplates" : {
               "application/xml" : "xml 400 response template",
```

```
              "application/json" : "json 400 response template"
            }
          },
          "2\\d{2}" : {
            "statusCode" : "200"
          }
        },
        "requestParameters" : {
          "integration.request.querystring.type" : "method.request.querystring.q1"
        },
        "passthroughBehavior" : "when_no_match",
        "type" : "http"
      }
    },
    "post" : {
      "parameters" : [ {
        "name" : "h1",
        "in" : "header",
        "required" : true,
        "schema" : {
          "type" : "string"
        }
      } ],
      "requestBody" : {
        "content" : {
          "application/json" : {
            "schema" : {
              "$ref" : "#/components/schemas/RequestBodyModel"
            }
          }
        },
        "required" : true
      },
      "responses" : {
        "200" : {
          "description" : "200 response",
          "headers" : {
            "test-method-response-header" : {
              "schema" : {
                "type" : "string"
              }
            }
          },
          "content" : {
```

```
            "application/json" : {
              "schema" : {
                "$ref" : "#/components/schemas/ArrayOfError"
              }
            }
          }
        }
      },
      "x-amazon-apigateway-request-validator" : "all",
      "x-amazon-apigateway-integration" : {
        "httpMethod" : "POST",
        "uri" : "http://petstore-demo-endpoint.execute-api.com/petstore/pets",
        "responses" : {
          "default" : {
            "statusCode" : "400",
            "responseParameters" : {
              "method.response.header.test-method-response-header" : "'static
value'"
            },
            "responseTemplates" : {
              "application/xml" : "xml 400 response template",
              "application/json" : "json 400 response template"
            }
          },
          "2\\d{2}" : {
            "statusCode" : "200"
          }
        },
        "requestParameters" : {
          "integration.request.header.custom_h1" : "method.request.header.h1"
        },
        "passthroughBehavior" : "when_no_match",
        "type" : "http"
      }
    }
  }
},
"components" : {
  "schemas" : {
    "RequestBodyModel" : {
      "required" : [ "name", "price", "type" ],
      "type" : "object",
      "properties" : {
        "id" : {
```

```
            "type" : "integer"
          },
          "type" : {
            "type" : "string",
            "enum" : [ "dog", "cat", "fish" ]
          },
          "name" : {
            "type" : "string"
          },
          "price" : {
            "maximum" : 500.0,
            "minimum" : 25.0,
            "type" : "number"
          }
        }
      },
      "ArrayOfError" : {
        "type" : "array",
        "items" : {
          "$ref" : "#/components/schemas/Error"
        }
      },
      "Error" : {
        "type" : "object"
      }
    }
  },
  "x-amazon-apigateway-request-validators" : {
    "all" : {
      "validateRequestParameters" : true,
      "validateRequestBody" : true
    },
    "params-only" : {
      "validateRequestParameters" : true,
      "validateRequestBody" : false
    }
  }
}
```

## OpenAPI 2.0

```
{
  "swagger" : "2.0",
```

```
  "info" : {
    "version" : "1.0.0",
    "title" : "ReqValidators Sample"
  },
  "basePath" : "/v1",
  "schemes" : [ "https" ],
  "paths" : {
    "/validation" : {
      "get" : {
        "produces" : [ "application/json", "application/xml" ],
        "parameters" : [ {
          "name" : "q1",
          "in" : "query",
          "required" : true,
          "type" : "string"
        } ],
        "responses" : {
          "200" : {
            "description" : "200 response",
            "schema" : {
              "$ref" : "#/definitions/ArrayOfError"
            },
            "headers" : {
              "test-method-response-header" : {
                "type" : "string"
              }
            }
          }
        },
        "x-amazon-apigateway-request-validator" : "params-only",
        "x-amazon-apigateway-integration" : {
          "httpMethod" : "GET",
          "uri" : "http://petstore-demo-endpoint.execute-api.com/petstore/pets",
          "responses" : {
            "default" : {
              "statusCode" : "400",
              "responseParameters" : {
                "method.response.header.test-method-response-header" : "'static
value'"
              },
              "responseTemplates" : {
                "application/xml" : "xml 400 response template",
                "application/json" : "json 400 response template"
              }
```

```
            },
            "2\\d{2}" : {
              "statusCode" : "200"
            }
          },
          "requestParameters" : {
            "integration.request.querystring.type" : "method.request.querystring.q1"
          },
          "passthroughBehavior" : "when_no_match",
          "type" : "http"
        }
      },
      "post" : {
        "consumes" : [ "application/json" ],
        "produces" : [ "application/json", "application/xml" ],
        "parameters" : [ {
          "name" : "h1",
          "in" : "header",
          "required" : true,
          "type" : "string"
        }, {
          "in" : "body",
          "name" : "RequestBodyModel",
          "required" : true,
          "schema" : {
            "$ref" : "#/definitions/RequestBodyModel"
          }
        } ],
        "responses" : {
          "200" : {
            "description" : "200 response",
            "schema" : {
              "$ref" : "#/definitions/ArrayOfError"
            },
            "headers" : {
              "test-method-response-header" : {
                "type" : "string"
              }
            }
          }
        },
        "x-amazon-apigateway-request-validator" : "all",
        "x-amazon-apigateway-integration" : {
          "httpMethod" : "POST",
```

```
          "uri" : "http://petstore-demo-endpoint.execute-api.com/petstore/pets",
          "responses" : {
            "default" : {
              "statusCode" : "400",
              "responseParameters" : {
                "method.response.header.test-method-response-header" : "'static
value'"
              },
              "responseTemplates" : {
                "application/xml" : "xml 400 response template",
                "application/json" : "json 400 response template"
              }
            },
            "2\\d{2}" : {
              "statusCode" : "200"
            }
          },
          "requestParameters" : {
            "integration.request.header.custom_h1" : "method.request.header.h1"
          },
          "passthroughBehavior" : "when_no_match",
          "type" : "http"
        }
      }
    }
  },
  "definitions" : {
    "RequestBodyModel" : {
      "type" : "object",
      "required" : [ "name", "price", "type" ],
      "properties" : {
        "id" : {
          "type" : "integer"
        },
        "type" : {
          "type" : "string",
          "enum" : [ "dog", "cat", "fish" ]
        },
        "name" : {
          "type" : "string"
        },
        "price" : {
          "type" : "number",
          "minimum" : 25.0,
```

```
          "maximum" : 500.0
        }
      }
    },
    "ArrayOfError" : {
      "type" : "array",
      "items" : {
        "$ref" : "#/definitions/Error"
      }
    },
    "Error" : {
      "type" : "object"
    }
  },
  "x-amazon-apigateway-request-validators" : {
    "all" : {
      "validateRequestParameters" : true,
      "validateRequestBody" : true
    },
    "params-only" : {
      "validateRequestParameters" : true,
      "validateRequestBody" : false
    }
  }
}
```

## AWS CloudFormation 具有基本請求驗證的範例 API 範本

下列 CloudFormation 範例範本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。

這裡會宣告兩種請求驗證程式：

**GETValidator**

此驗證程式會在 GET 方法上啟用。它可讓 API Gateway 驗證傳入請求中已包含不是空白的必要查詢參數 (q1)。

**POSTValidator**

此驗證程式會在 POST 方法上啟用。它可讓 API Gateway 驗證，當內容類型為 application/json 時，承載請求格式是否遵循指定的 RequestBodyModel，如果找不到相符的內容類型，則不會執行請求驗證。若無論內容類型為何都要使用相同的模型，可指定

$default。RequestBodyModel 包含一個額外的模型 RequestBodyModelId，用於定義寵物 ID。

```
AWSTemplateFormatVersion: 2010-09-09
Parameters:
  StageName:
    Type: String
    Default: v1
    Description: Name of API stage.
Resources:
  Api:
    Type: 'AWS::ApiGateway::RestApi'
    Properties:
      Name: ReqValidatorsSample
  RequestBodyModelId:
    Type: 'AWS::ApiGateway::Model'
    Properties:
      RestApiId: !Ref Api
      ContentType: application/json
      Description: Request body model for Pet ID.
      Schema:
        $schema: 'http://json-schema.org/draft-04/schema#'
        title: RequestBodyModelId
        properties:
            id:
              type: integer
  RequestBodyModel:
    Type: 'AWS::ApiGateway::Model'
    Properties:
      RestApiId: !Ref Api
      ContentType: application/json
      Description: Request body model for Pet type, name, price, and ID.
      Schema:
        $schema: 'http://json-schema.org/draft-04/schema#'
        title: RequestBodyModel
        required:
          - price
          - name
          - type
        type: object
        properties:
            id:
```

```
              "$ref": !Sub
                - 'https://apigateway.amazonaws.com/restapis/${Api}/models/
${RequestBodyModelId}'
                - Api: !Ref Api
                  RequestBodyModelId: !Ref RequestBodyModelId
            price:
              type: number
              minimum: 25
              maximum: 500
            name:
              type: string
            type:
              type: string
              enum:
                - "dog"
                - "cat"
                - "fish"
  GETValidator:
    Type: AWS::ApiGateway::RequestValidator
    Properties:
      Name: params-only
      RestApiId: !Ref Api
      ValidateRequestBody: False
      ValidateRequestParameters: True
  POSTValidator:
    Type: AWS::ApiGateway::RequestValidator
    Properties:
      Name: body-only
      RestApiId: !Ref Api
      ValidateRequestBody: True
      ValidateRequestParameters: False
  ValidationResource:
    Type: 'AWS::ApiGateway::Resource'
    Properties:
      RestApiId: !Ref Api
      ParentId: !GetAtt Api.RootResourceId
      PathPart: 'validation'
  ValidationMethodGet:
    Type: 'AWS::ApiGateway::Method'
    Properties:
      RestApiId: !Ref Api
      ResourceId: !Ref ValidationResource
      HttpMethod: GET
      AuthorizationType: NONE
```

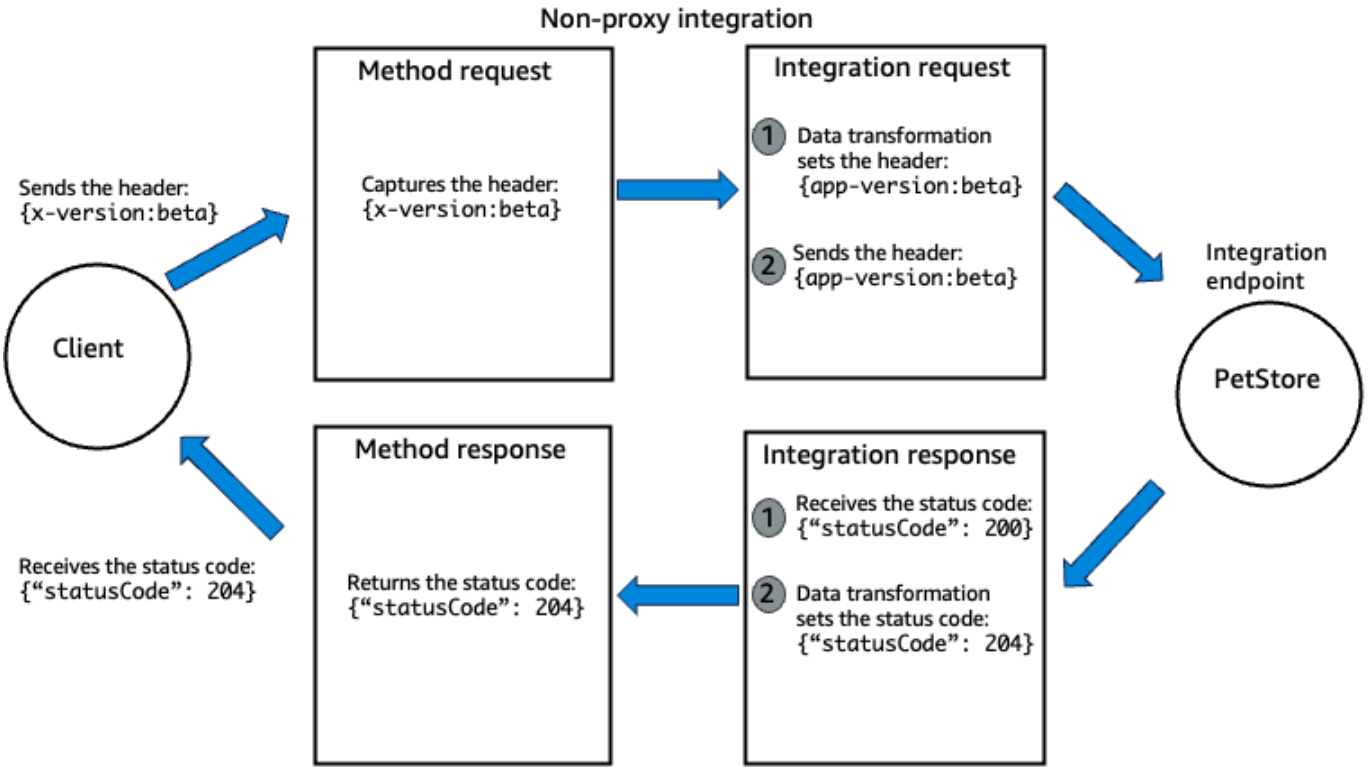
```
      RequestValidatorId: !Ref GETValidator
      RequestParameters:
        method.request.querystring.q1: true
      Integration:
        Type: HTTP_PROXY
        IntegrationHttpMethod: GET
        Uri: http://petstore-demo-endpoint.execute-api.com/petstore/pets/
  ValidationMethodPost:
    Type: 'AWS::ApiGateway::Method'
    Properties:
      RestApiId: !Ref Api
      ResourceId: !Ref ValidationResource
      HttpMethod: POST
      AuthorizationType: NONE
      RequestValidatorId: !Ref POSTValidator
      RequestModels:
        application/json : !Ref RequestBodyModel
      Integration:
        Type: HTTP_PROXY
        IntegrationHttpMethod: POST
        Uri: http://petstore-demo-endpoint.execute-api.com/petstore/pets/
  ApiDeployment:
    Type: 'AWS::ApiGateway::Deployment'
    DependsOn:
      - ValidationMethodGet
      - RequestBodyModel
    Properties:
      RestApiId: !Ref Api
      StageName: !Sub '${StageName}'
Outputs:
  ApiRootUrl:
    Description: Root Url of the API
    Value: !Sub 'https://${Api}.execute-api.${AWS::Region}.amazonaws.com/${StageName}'
```

# API Gateway 中用於 REST API 的資料轉換

> **Note**
>
> 本節說明可搭配非代理整合使用的功能。不過，我們建議您盡可能針對 REST API 使用代理整合。代理整合採用簡化的整合設定，並且可隨著後端演進，而不必縮減現有的設定。如需詳細資訊，請參閱the section called “選擇 API 整合類型”。

如果您使用非代理整合，您可以使用 API Gateway 的兩項功能來轉換方法請求和整合回應。如果方法請求採用與整合請求承載不同的承載格式，則您可以轉換方法請求。如果整合回應傳回的承載格式與方法回應中需要傳回的格式不同，則您可以轉換整合回應。如需有關請求生命週期的詳細資訊，請參閱 the section called “REST API 的範例資源”。

下列範例說明資料轉換，其中標頭 "x-version:beta" 的 x-version 標頭參數會轉換成 app-version 標頭參數。整合請求中會進行 x-version 到 app-version 的資料轉換。如此一來，整合端點就會收到轉換後的標頭參數值。當整合端點傳回狀態碼時，狀態碼會在方法回應之前，從 200 轉換為 204。




若要建立資料轉換，您可以使用下列功能：

參數映射

在參數映射中，您可以修改整合請求 URL 路徑參數、URL 查詢字串參數或 HTTP 標頭值，但無法修改整合請求承載。您也可以修改 HTTP 回應標頭值。使用參數映射建立跨來源資源分享 (CORS) 的靜態標頭值。

您可以在代理和非代理整合的整合請求中使用參數映射，但若要將參數映射用於整合回應，則需要非代理整合。參數映射不需要使用 Velocity 範本語言 (VTL) 編寫任何指令碼。如需詳細資訊，請參閱the section called “參數映射”。

映射範本轉換

在映射範本轉換中，您可以使用映射範本來對應 URL 路徑參數、URL 查詢字串參數、HTTP 標頭，以及整合請求或整合回應內文。映射範本是以 Velocity 範本語言 (VTL) 表達的指令碼，會使用 JSONPath 表達式並根據 `Content-type` 標頭套用至承載。

您可以使用映射範本執行下列操作：

- 選取要使用整合傳送的資料 AWS 服務，例如 Amazon DynamoDB 或 Lambda 函數或 HTTP 端點。如需詳細資訊，請參閱the section called “教學課程：修改 AWS 服務整合的整合請求和回應”。
- 有條件地覆寫 API 的整合請求和整合回應參數、建立新的標頭值，以及覆寫狀態碼。如需詳細資訊，請參閱the section called “覆寫 API 請求和回應參數及狀態碼”。

若整合請求內文的 `Content-type` 標頭沒有相符的映射範本，您也可以指定 API 的行為。這稱為整合傳遞行為。如需詳細資訊，請參閱the section called “沒有映射範本時，承載的方法請求行為”。

## 選擇參數映射或映射範本轉換

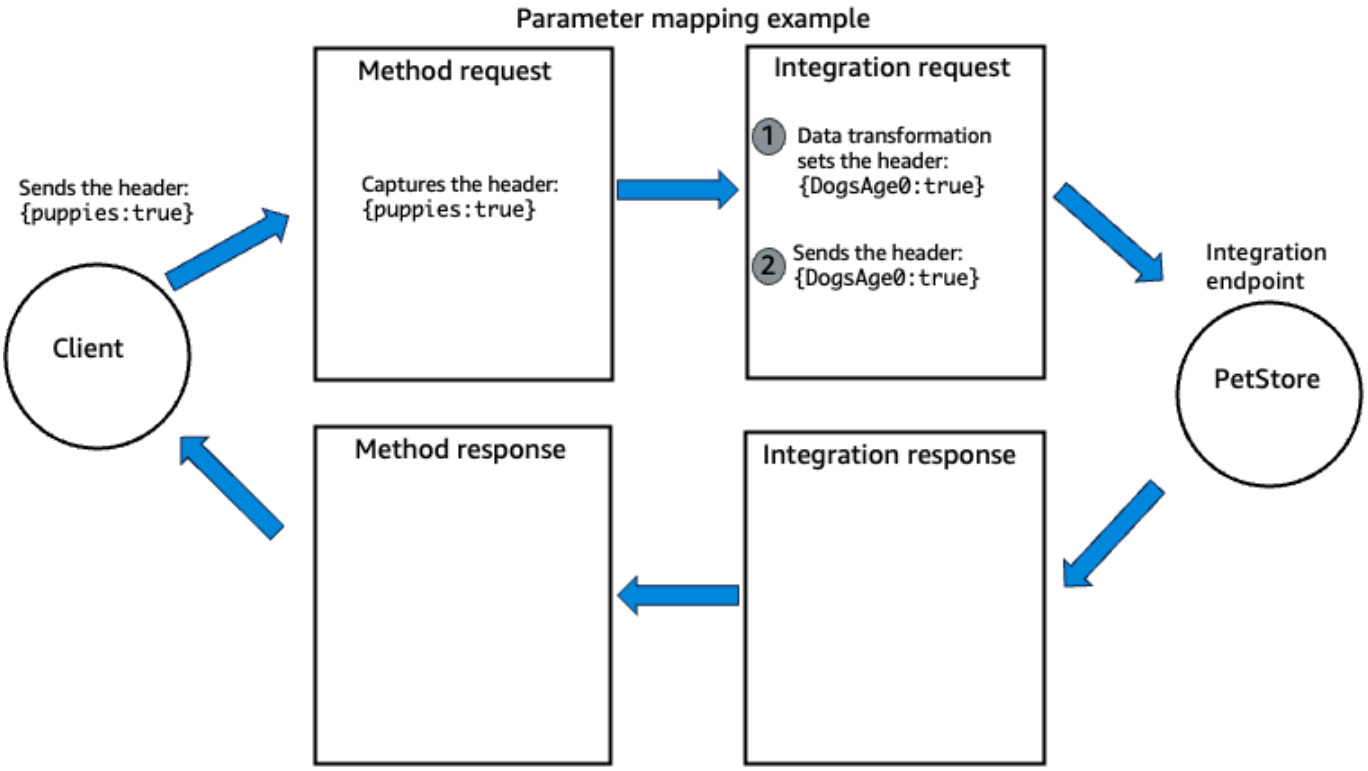
我們建議您盡可能使用參數映射來轉換資料。如果您的 API 要求您變更內文，或要求您根據傳入的整合請求或整合回應執行條件式覆寫和修改，而且您無法使用代理整合，則使用映射範本轉換。

## API Gateway 中 REST API 的參數映射

> **Note**
>
> 如果您使用的是 HTTP API，請參閱 the section called “參數映射”。

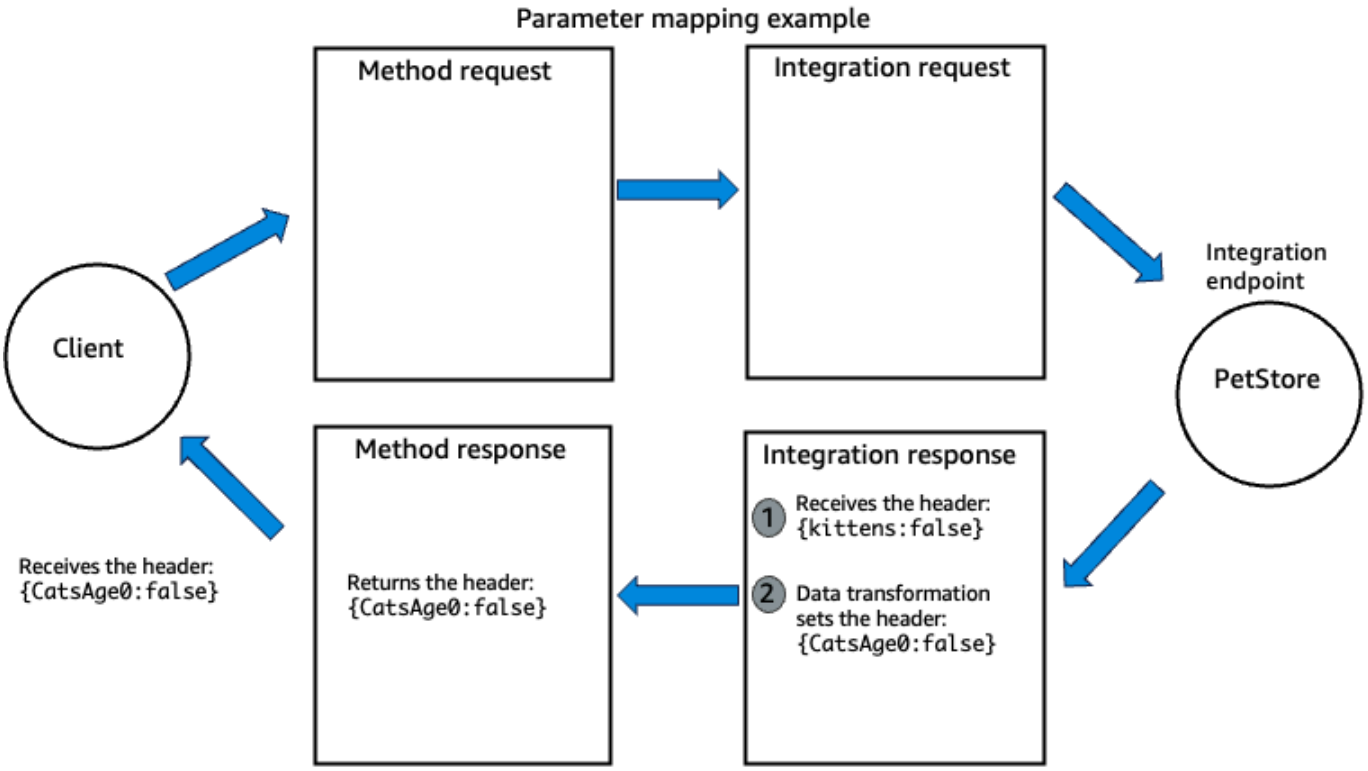
在參數映射中，您可以對應請求或回應參數。您可以使用參數映射表達式或靜態值來對應參數。如需映射表達式的清單，請參閱 the section called “參數映射來源參考”。您可以在代理和非代理整合的整合請求中使用參數映射，但若要將參數映射用於整合回應，則需要非代理整合。

例如，您可以將方法請求標頭參數 `puppies` 對應到整合請求標頭參數 `DogsAge0`。之後如果用戶端將標頭 `puppies:true` 傳送至您的 API，則整合請求會將請求標頭 `DogsAge0:true` 傳送至整合端點。下圖顯示此範例的請求生命週期。




若要使用 API Gateway 建立此範例，請參閱 the section called “範例 1：將方法請求參數對應至整合請求參數”。

另一個範例是，您也可以將整合回應標頭參數 `kittens` 對應至方法回應標頭參數 `CatsAge0`。之後如果整合端點傳回 `kittens:false`，則用戶端會收到標頭 `CatsAge0:false`。下圖顯示此範例的請求生命週期。




主題

- API Gateway 中 REST API 的參數映射範例
- API Gateway 中 REST API 的參數映射來源參考

## API Gateway 中 REST API 的參數映射範例

下列範例示範如何使用 API Gateway 主控台、OpenAPI 和 CloudFormation 範本建立參數映射表達式。如需如何使用參數映射來建立所需 CORS 標頭的範例，請參閱 the section called “CORS”。

### 範例 1：將方法請求參數對應至整合請求參數

下列範例會將方法請求標頭參數 `puppies` 對應至整合請求標頭參數 `DogsAge0`。

#### AWS 管理主控台

對應方法請求參數

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 選擇 REST API。
3. 選擇一個方法。

您的方法必須有非代理整合。

4. 針對方法請求設定，選擇編輯。
5. 選擇 HTTP 請求標頭。
6. 選擇新增標頭。
7. 對於名稱，輸入 **puppies**。
8. 選擇儲存。
9. 選擇整合請求索引標籤，然後針對整合請求設定，選擇編輯。

   AWS 管理主控台 會自動puppies為您從 新增參數映射method.request.header.puppies 至 ，但您需要變更名稱以符合整合端點預期的請求標頭參數。
10. 對於名稱，輸入 **DogsAge0**。
11. 選擇儲存。
12. 重新部署 API 以使變更生效。

下列步驟說明如何驗證參數映射是否成功。

(選用) 測試參數映射

1. 選擇測試標籤。您可能需要選擇向右箭頭按鈕才能顯示此索引標籤。
2. 針對標頭，輸入 **puppies:true**。
3. 選擇測試。
4. 在日誌中，結果應如下所示：

```
Tue Feb 04 00:28:36 UTC 2025 : Method request headers: {puppies=true}
Tue Feb 04 00:28:36 UTC 2025 : Method request body before transformations:
Tue Feb 04 00:28:36 UTC 2025 : Endpoint request URI: http://petstore-demo-
endpoint.execute-api.com/petstore/pets
Tue Feb 04 00:28:36 UTC 2025 : Endpoint request headers: {DogsAge0=true,
 x-amzn-apigateway-api-id=abcd1234, Accept=application/json, User-
Agent=AmazonAPIGateway_aaaaaaa, X-Amzn-Trace-Id=Root=1-abcd-12344}
```

請求標頭參數已從 puppies 變更為 DogsAge0。

CloudFormation

在此範例中，您使用 body 屬性將 OpenAPI 定義檔案匯入 API Gateway 中。

```
AWSTemplateFormatVersion: 2010-09-09
Resources:
  Api:
    Type: 'AWS::ApiGateway::RestApi'
    Properties:
      Body:
        openapi: 3.0.1
        info:
          title: ParameterMappingExample
          version: "2025-02-04T00:30:41Z"
        paths:
          /pets:
            get:
              parameters:
                - name: puppies
                  in: header
                  schema:
                    type: string
              responses:
                "200":
                  description: 200 response
              x-amazon-apigateway-integration:
                httpMethod: GET
                uri: http://petstore-demo-endpoint.execute-api.com/petstore/pets
                responses:
                  default:
                    statusCode: "200"
                requestParameters:
                  integration.request.header.DogsAge0: method.request.header.puppies
                passthroughBehavior: when_no_match
                type: http
  ApiGatewayDeployment:
    Type: 'AWS::ApiGateway::Deployment'
    DependsOn: Api
    Properties:
      RestApiId: !Ref Api
  ApiGatewayDeployment20250219:
    Type: 'AWS::ApiGateway::Deployment'
    DependsOn: Api
    Properties:
```

```
      RestApiId: !Ref Api
  Stage:
    Type: 'AWS::ApiGateway::Stage'
    Properties:
       DeploymentId: !Ref ApiGatewayDeployment20250219
       RestApiId: !Ref Api
       StageName: prod
```

### OpenAPI

```
{
  "openapi" : "3.0.1",
  "info" : {
    "title" : "ParameterMappingExample",
    "version" : "2025-02-04T00:30:41Z"
  },
  "paths" : {
    "/pets" : {
      "get" : {
        "parameters" : [ {
          "name" : "puppies",
          "in" : "header",
          "schema" : {
            "type" : "string"
          }
        } ],
        "responses" : {
          "200" : {
            "description" : "200 response"
          }
        },
        "x-amazon-apigateway-integration" : {
          "httpMethod" : "GET",
          "uri" : "http://petstore-demo-endpoint.execute-api.com/petstore/pets",
          "responses" : {
            "default" : {
              "statusCode" : "200"
            }
          },
          "requestParameters" : {
            "integration.request.header.DogsAge0" : "method.request.header.puppies"
          },
          "passthroughBehavior" : "when_no_match",
```

```
          "type" : "http"
        }
      }
    }
  }
}
```

範例 2：將多個方法請求參數對應至不同的整合請求參數

下列範例會將多值方法請求查詢字串參數 `methodRequestQueryParam` 對應至整合請求查詢字串參數 `integrationQueryParam`，並將方法請求標頭參數 `methodRequestHeaderParam` 對應至整合請求路徑參數 `integrationPathParam`。

AWS 管理主控台

對應方法請求參數

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 選擇 REST API。
3. 選擇一個方法。

   您的方法必須有非代理整合。
4. 針對方法請求設定，選擇編輯。
5. 選擇 URL 查詢字串參數。
6. 選擇新增查詢字串。
7. 對於名稱，輸入 **methodRequestQueryParam**。
8. 選擇 HTTP 請求標頭。
9. 選擇新增標頭。
10. 對於名稱，輸入 **methodRequestHeaderParam**。
11. 選擇儲存。
12. 選擇整合請求索引標籤，然後針對整合請求設定，選擇編輯。
13. 選擇 URL 路徑參數。
14. 選擇新增路徑參數。
15. 對於名稱，輸入 **integrationPathParam**。
16. 對於映射自，輸入 **method.request.header.methodRequestHeaderParam**。

這樣做會將您在方法請求中指定的方法請求標頭，對應至新的整合請求路徑參數。

17. 選擇 URL 查詢字串參數。
18. 選擇新增查詢字串。
19. 對於名稱，輸入 **integrationQueryParam**。
20. 對於映射自，輸入 **method.request.multivaluequerystring.methodRequestQueryParam**。

    這樣做會將多值查詢字串參數對應至新的單一值整合請求查詢字串參數。
21. 選擇儲存。
22. 重新部署 API 以使變更生效。

### CloudFormation

在此範例中，您使用 body 屬性將 OpenAPI 定義檔案匯入 API Gateway 中。

以下 OpenAPI 定義會針對 HTTP 整合建立下列參數映射：

- 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

```
AWSTemplateFormatVersion: 2010-09-09
Resources:
  Api:
    Type: 'AWS::ApiGateway::RestApi'
    Properties:
      Body:
        openapi: 3.0.1
        info:
          title: Parameter mapping example 2
          version: "2025-01-15T19:12:31Z"
        paths:
          /:
            post:
              parameters:
                - name: methodRequestQueryParam
                  in: query
```

```
                schema:
                  type: string
              - name: methodRequestHeaderParam
                in: header
                schema:
                  type: string
            responses:
              "200":
                description: 200 response
            x-amazon-apigateway-integration:
              httpMethod: GET
              uri: http://petstore-demo-endpoint.execute-api.com/petstore/pets
              responses:
                default:
                  statusCode: "200"
              requestParameters:
                integration.request.querystring.integrationQueryParam:
method.request.multivaluequerystring.methodRequestQueryParam
                integration.request.path.integrationPathParam:
method.request.header.methodRequestHeaderParam
              requestTemplates:
                application/json: '{"statusCode": 200}'
              passthroughBehavior: when_no_templates
              timeoutInMillis: 29000
              type: http
 ApiGatewayDeployment:
   Type: 'AWS::ApiGateway::Deployment'
   DependsOn: Api
   Properties:
     RestApiId: !Ref Api
 ApiGatewayDeployment20250219:
   Type: 'AWS::ApiGateway::Deployment'
   DependsOn: Api
   Properties:
     RestApiId: !Ref Api
 Stage:
   Type: 'AWS::ApiGateway::Stage'
   Properties:
      DeploymentId: !Ref ApiGatewayDeployment20250219
      RestApiId: !Ref Api
      StageName: prod
```

### OpenAPI

以下 OpenAPI 定義會針對 HTTP 整合建立下列參數映射：

- 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

```
{
  "openapi" : "3.0.1",
  "info" : {
    "title" : "Parameter mapping example 2",
    "version" : "2025-01-15T19:12:31Z"
  },
  "paths" : {
    "/" : {
      "post" : {
        "parameters" : [ {
          "name" : "methodRequestQueryParam",
          "in" : "query",
          "schema" : {
            "type" : "string"
          }
        }, {
          "name" : "methodRequestHeaderParam",
          "in" : "header",
          "schema" : {
            "type" : "string"
          }
        } ],
        "responses" : {
          "200" : {
            "description" : "200 response"
          }
        },
        "x-amazon-apigateway-integration" : {
          "httpMethod" : "GET",
          "uri" : "http://petstore-demo-endpoint.execute-api.com/petstore/pets",
          "responses" : {
            "default" : {
              "statusCode" : "200"
```

```
            }
          },
          "requestParameters" : {
            "integration.request.querystring.integrationQueryParam" :
 "method.request.multivaluequerystring.methodRequestQueryParam",
            "integration.request.path.integrationPathParam" :
 "method.request.header.methodRequestHeaderParam"
          },
          "requestTemplates" : {
            "application/json" : "{\"statusCode\": 200}"
          },
          "passthroughBehavior" : "when_no_templates",
          "timeoutInMillis" : 29000,
          "type" : "http"
        }
      }
    }
  }
}
```

範例 3：將欄位從 JSON 請求內文對應至整合請求參數

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。下列範例會將方法請求內文對應至名為 body-header 的整合請求標頭，並將部分請求內文 (如 JSON 表達式所示) 對應至名為 pet-price 的整合請求標頭。

若要測試此範例，請提供包含價格類別的輸入，如下所示：

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

AWS 管理主控台

對應方法請求參數

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。

2. 選擇 REST API。
3. 選擇 POST、PUT、PATCH 或 ANY 方法。

   您的方法必須有非代理整合。
4. 針對整合請求設定，選擇編輯。
5. 選擇 URL 請求標頭參數。
6. 選擇新增請求標頭參數。
7. 對於名稱，輸入 **body-header**。
8. 對於映射自，輸入 **method.request.body**。

   這樣做會將方法請求內文對應至新的整合請求標頭參數。
9. 選擇新增請求標頭參數。
10. 對於名稱，輸入 **pet-price**。
11. 對於映射自，輸入 **method.request.body[0].price**。

    這樣做會將部分方法請求內文對應至新的整合請求標頭參數。
12. 選擇儲存。
13. 重新部署 API 以使變更生效。

### CloudFormation

在此範例中，您使用 body 屬性將 OpenAPI 定義檔案匯入 API Gateway 中。

```
AWSTemplateFormatVersion: 2010-09-09
Resources:
  Api:
    Type: 'AWS::ApiGateway::RestApi'
    Properties:
      Body:
        openapi: 3.0.1
        info:
          title: Parameter mapping example 3
          version: "2025-01-15T19:19:14Z"
        paths:
          /:
            post:
              responses:
                "200":
```

```
                description: 200 response
            x-amazon-apigateway-integration:
              httpMethod: GET
              uri: http://petstore-demo-endpoint.execute-api.com/petstore/pets
              responses:
                default:
                  statusCode: "200"
              requestParameters:
                integration.request.header.pet-price: method.request.body[0].price
                integration.request.header.body-header: method.request.body
              requestTemplates:
                application/json: '{"statusCode": 200}'
              passthroughBehavior: when_no_templates
              timeoutInMillis: 29000
              type: http
  ApiGatewayDeployment:
    Type: 'AWS::ApiGateway::Deployment'
    DependsOn: Api
    Properties:
      RestApiId: !Ref Api
  ApiGatewayDeployment20250219:
    Type: 'AWS::ApiGateway::Deployment'
    DependsOn: Api
    Properties:
      RestApiId: !Ref Api
  Stage:
    Type: 'AWS::ApiGateway::Stage'
    Properties:
       DeploymentId: !Ref ApiGatewayDeployment20250219
       RestApiId: !Ref Api
       StageName: prod
```

### OpenAPI

以下 OpenAPI 定義會從 JSON 請求內文中的欄位對應整合請求參數。

```
{
  "openapi" : "3.0.1",
  "info" : {
    "title" : "Parameter mapping example 3",
    "version" : "2025-01-15T19:19:14Z"
  },
  "paths" : {
    "/" : {
```

```
      "post" : {
        "responses" : {
          "200" : {
            "description" : "200 response"
          }
        },
        "x-amazon-apigateway-integration" : {
          "httpMethod" : "GET",
          "uri" : "http://petstore-demo-endpoint.execute-api.com/petstore/pets",
          "responses" : {
            "default" : {
              "statusCode" : "200"
            }
          },
          "requestParameters" : {
            "integration.request.header.pet-price" : "method.request.body[0].price",
            "integration.request.header.body-header" : "method.request.body"
          },
          "requestTemplates" : {
            "application/json" : "{\"statusCode\": 200}"
          },
          "passthroughBehavior" : "when_no_templates",
          "timeoutInMillis" : 29000,
          "type" : "http"
        }
      }
    }
  }
}
```

範例 4：將整合回應對應至方法回應

您也可以將整合回應對應至方法回應。下列範例會將整合回應內文對應至名為 `location` 的方法回應標頭、將整合回應標頭 `x-app-id` 對應至方法回應標頭 `id`，並將多值整合回應標頭 `item` 對應至方法回應標頭 `items`。

AWS 管理主控台

對應整合回應

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 選擇 REST API。

3. 選擇一個方法。

   您的方法必須有非代理整合。
4. 選擇方法回應索引標籤，然後針對回應 200 選擇編輯。
5. 針對標頭名稱，選擇新增標頭。
6. 建立三個分別名為 **id**、**item** 和 **location** 的標頭。
7. 選擇儲存。
8. 選擇整合回應索引標籤，然後針對預設 - 回應選擇編輯。
9. 在標頭映射下，輸入以下內容。

   a. 針對 id，輸入 **integration.response.header.x-app-id**

   b. 針對 item，輸入 **integration.response.multivalueheader.item**

   c. 針對 location，輸入 **integration.response.body.redirect.url**
10. 選擇儲存。
11. 重新部署 API 以使變更生效。

CloudFormation

在此範例中，您使用 body 屬性將 OpenAPI 定義檔案匯入 API Gateway 中。

```
AWSTemplateFormatVersion: 2010-09-09
Resources:
  Api:
    Type: 'AWS::ApiGateway::RestApi'
    Properties:
      Body:
        openapi: 3.0.1
        info:
          title: Parameter mapping example
          version: "2025-01-15T19:21:35Z"
        paths:
          /:
            post:
              responses:
                "200":
                  description: 200 response
                  headers:
                    item:
```

```
                      schema:
                        type: string
                    location:
                      schema:
                        type: string
                    id:
                      schema:
                        type: string
              x-amazon-apigateway-integration:
                type: http
                httpMethod: GET
                uri: http://petstore-demo-endpoint.execute-api.com/petstore/pets
                responses:
                  default:
                    statusCode: "200"
                    responseParameters:
                      method.response.header.id: integration.response.header.x-app-
id
                      method.response.header.location:
 integration.response.body.redirect.url
                      method.response.header.item:
 integration.response.multivalueheader.item
                requestTemplates:
                  application/json: '{"statusCode": 200}'
                passthroughBehavior: when_no_templates
                timeoutInMillis: 29000
  ApiGatewayDeployment:
    Type: 'AWS::ApiGateway::Deployment'
    DependsOn: Api
    Properties:
      RestApiId: !Ref Api
  ApiGatewayDeployment20250219:
    Type: 'AWS::ApiGateway::Deployment'
    DependsOn: Api
    Properties:
      RestApiId: !Ref Api
  Stage:
    Type: 'AWS::ApiGateway::Stage'
    Properties:
       DeploymentId: !Ref ApiGatewayDeployment20250219
       RestApiId: !Ref Api
       StageName: prod
```

OpenAPI

以下 OpenAPI 定義會將整合回應對應至方法回應。

```
{
  "openapi" : "3.0.1",
  "info" : {
    "title" : "Parameter mapping example",
    "version" : "2025-01-15T19:21:35Z"
  },
  "paths" : {
    "/" : {
      "post" : {
        "responses" : {
          "200" : {
            "description" : "200 response",
            "headers" : {
              "item" : {
                "schema" : {
                  "type" : "string"
                }
              },
              "location" : {
                "schema" : {
                  "type" : "string"
                }
              },
              "id" : {
                "schema" : {
                  "type" : "string"
                }
              }
            }
          }
        },
        "x-amazon-apigateway-integration" : {
          "type" : "http",
          "httpMethod" : "GET",
          "uri" : "http://petstore-demo-endpoint.execute-api.com/petstore/pets",
          "responses" : {
            "default" : {
              "statusCode" : "200",
              "responseParameters" : {
```

```
                "method.response.header.id" : "integration.response.header.x-app-
id",
                "method.response.header.location" :
 "integration.response.body.redirect.url",
                "method.response.header.item" :
 "integration.response.multivalueheader.item"
              }
            }
          },
          "requestTemplates" : {
            "application/json" : "{\"statusCode\": 200}"
          },
          "passthroughBehavior" : "when_no_templates",
          "timeoutInMillis" : 29000
        }
      }
    }
  }
}
```

## API Gateway 中 REST API 的參數映射來源參考

當您建立參數映射時，請指定要修改的方法請求或整合回應參數，並指定如何修改這些參數。

下表顯示您可以對應的方法請求參數，以及建立映射的表達式。在這些表達式中，*name* 是方法請求參數的名稱。例如，若要對應請求標頭參數 puppies，則使用表達式 `method.request.header.puppies`。您的表達式必須符合規則表達式 `'^[a-zA-Z0-9._$-]+$]'`。您可以在代理和非代理整合的整合請求中使用參數映射。

| 映射的資料來源 | 映射表達式 |
| --- | --- |
| 方法請求路徑 | `method.request.path.` *name* |
| 方法請求查詢字串 | `method.request.querystring.` *name* |
| 多值方法請求查詢字串 | `method.request.multivaluequerystring.` *name* |
| 方法請求標頭 | `method.request.header.` *name* |

| 映射的資料來源 | 映射表達式 |
| --- | --- |
| 多值方法請求標頭 | `method.request.multivalueheader.` *`name`* |
| 方法請求內文 | `method.request.body` |
| 方法請求內文 (JsonPath) | `method.request.body.` *`JSONPath_EXPRESSION`* .<br><br>*`JSONPath_EXPRESSION`* 是請求的內文中，JSON 欄位的 JSONPath 表達式。如需詳細資訊，請參閱 JSONPath 表達式。 |
| 階段變數 | `stageVariables.` *`name`* |
| 環境變數 | `context.`*`name`*<br><br>name 必須是支援的環境變數之一。 |
| 靜態值 | *`'static_value'`* .<br><br>*`static_value`* 是字串常值，而且前後必須加上單引號。例如 `'https://www.example.com'` 。 |

下表顯示您可以對應的整合回應參數，以及建立映射的表達式。在這些表達式中，*`name`* 是整合回應參數的名稱。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。若要針對整合回應使用參數映射，則您需要非代理整合。

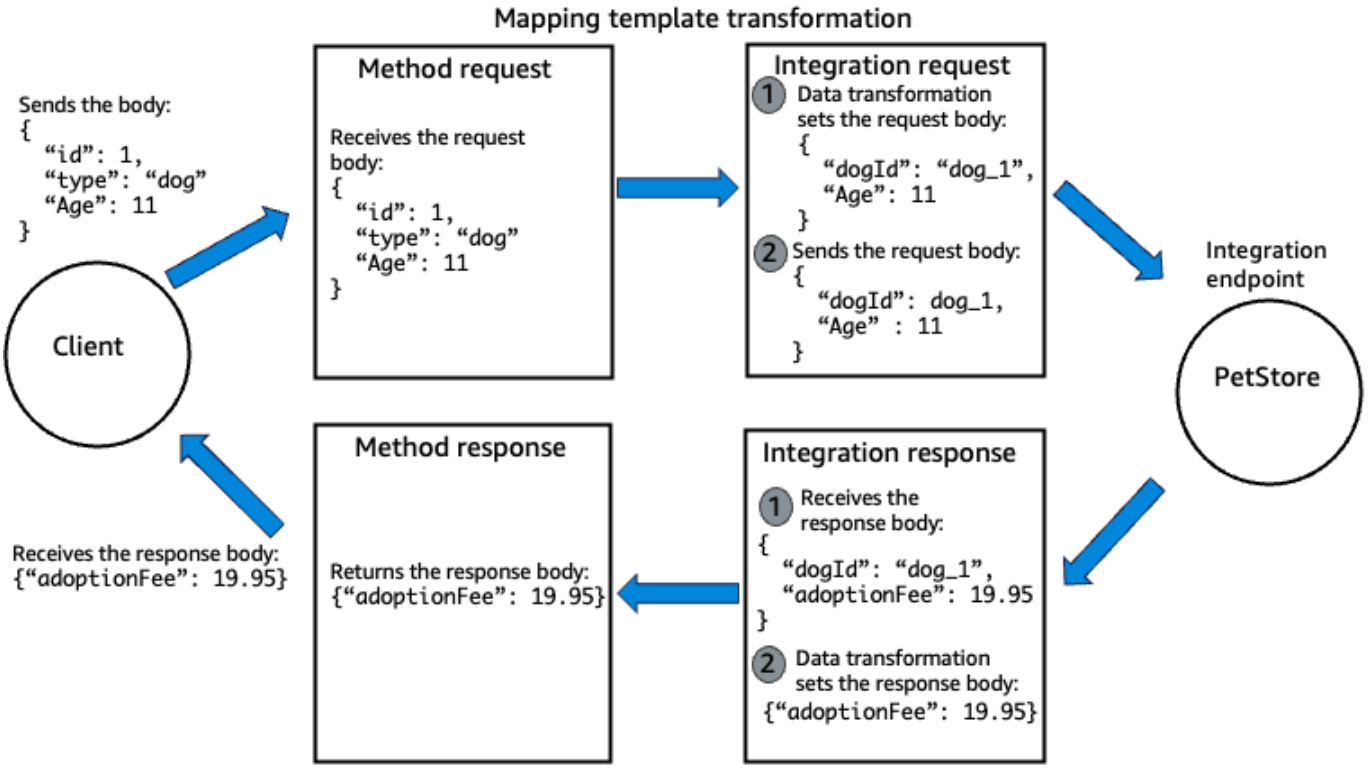
| 映射的資料來源 | 對應表達式 |
| --- | --- |
| 整合回應標頭 | `integration.response.header.` *`name`* |
| 整合回應標頭 | `integration.response.multivalueheader.` *`name`* |
| 整合回應內文 | `integration.response.body` |

| 映射的資料來源 | 對應表達式 |
| --- | --- |
| 整合回應內文 (JsonPath) | `integration.response.body.` *JSONPath_EXPRESSION*<br><br>*JSONPath_EXPRESSION* 是回應的內文中，JSON 欄位的 JSONPath 表達式。如需詳細資訊，請參閱 JSONPath 表達式。 |
| 階段變數 | `stageVariables.` *name* |
| 環境變數 | `context.`*name*<br><br>name 必須是支援的環境變數之一。 |
| 靜態值 | *'static_value'*<br><br>*static_value* 是字串常值，而且前後必須加上單引號。例如 `'https://www.example.com'` 。 |

## API Gateway 中 REST API 的映射範本轉換

映射範本轉換會使用映射範本來修改您的整合請求或整合回應。映射範本是以 Velocity 範本語言 (VTL) 表達的指令碼，會使用 JSONPath 並根據 `Content-type` 標頭套用至承載。您會使用映射範本來進行映射範本轉換。本節說明與映射範本相關的概念性資訊。

下圖顯示與 PetStore 整合端點整合之 `POST /pets` 資源的請求生命週期。在此 API 中，使用者會傳送有關寵物的資料，而整合端點會傳回與寵物相關聯的認養費。在此請求生命週期中，映射範本轉換會篩選傳送至整合端點的請求內文，以及篩選來自整合端點的回應內文。




以下各節說明請求和回應生命週期。

方法請求和整合請求

在上述範例中，如果這是傳送至方法請求的請求內文：

```
POST /pets
    HTTP/1.1
    Host:abcd1234.us-west-2.amazonaws.com
    Content-type: application/json

  {
    "id": 1,
    "type": "dog",
    "Age": 11,
  }
```

此請求內文的格式不正確，無法供整合端點使用，因此 API Gateway 會執行映射範本轉換。API Gateway 只會執行映射範本轉換，因為有針對 Content-Type `application/json` 定義的映射範本。如果您未針對 Content-Type 定義映射範本，則根據預設，API Gateway 會透過整合請求將內文傳遞至整合端點。若要修改此行為，請參閱 the section called “沒有映射範本時，承載的方法請求行為”。

下列映射範本會先轉換整合請求中的方法請求資料，再將其傳送至整合端點：

```
#set($inputRoot = $input.path('$'))
  {
    "dogId" : "dog_"$elem.id,
    "Age": $inputRoot.Age
  }
```

1. $inputRoot 變數代表上一節中原始 JSON 資料的根物件。指令以 # 符號開始。
2. dog 是使用者的 id 和字串值的串連。
3. Age 來自方法請求內文。

然後，下列輸出會轉送至整合端點：

```
{
    "dogId" : "dog_1",
    "Age": 11
  }
```

整合回應和方法回應

成功對整合端點發出請求後，端點會將回應傳送至 API Gateway 的整合回應。以下是來自整合端點的輸出資料範例：

```
{
    "dogId" : "dog_1",
    "adoptionFee": 19.95,
}
```

方法回應預期的承載與整合回應傳回的承載不同。API Gateway 會執行映射範本轉換。API Gateway 只會執行映射範本轉換，因為有針對 Content-Type application/json 定義的映射範本。如果您未針對 Content-Type 定義映射範本，則根據預設，API Gateway 會透過整合回應將內文傳遞至方法回應。若要修改此行為，請參閱 the section called “沒有映射範本時，承載的方法請求行為”。

```
#set($inputRoot = $input.path('$'))
  {
    "adoptionFee" : $inputRoot.adoptionFee,
  }
```

下列輸出會傳送至方法回應：

```
{"adoptionFee": 19.95}
```

映射範本轉換範例至此完成。我們建議您盡可能使用代理整合來轉換資料，而不要使用映射範本轉換。如需詳細資訊，請參閱the section called “選擇 API 整合類型”。

API Gateway 中 REST API 沒有映射範本時，承載的方法請求行為

如果您的方法請求具有承載，而您未針對 Content-Type 標頭定義映射範本，則您可以選擇透過整合請求將用戶端提供的請求承載傳遞至後端，而不進行轉換。這個過程稱為整合傳遞。

傳入請求的實際傳遞行為是由此設定決定。共有三個選項：

當沒有範本符合請求 Content-Type 標頭時

當方法請求內容類型不符合與對應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，請選擇此選項。

呼叫 API Gateway API 時，您可以透過在整合上將 WHEN_NO_MATCH 設定為 passthroughBehavior 屬性值來選擇此選項。

未定義範本時 (建議)

當整合請求中未定義任何對應範本時，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，請選擇此選項。如果選取此選項時已定義範本，則具有承載但內容類型不符合任何已定義映射範本的方法請求將會遭拒，並顯示 HTTP 415 Unsupported Media Type (不支援的媒體類型) 回應。

呼叫 API Gateway API 時，您可以透過在整合上將 WHEN_NO_TEMPLATES 設定為 passthroughBehavior 屬性值來選擇此選項。

從不

當整合請求中未定義任何對應範本時，如果您不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，請選擇此選項。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。

呼叫 API Gateway API 時，您可以透過在整合上將 NEVER 設定為 passthroughBehavior 屬性值來選擇此選項。

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 | 傳遞選項 | Behavior (行為) |
| --- | --- | --- |
| 無<br><br>API Gateway 預設為 application/json | WHEN_NO_MATCH | 使用此範本轉換請求承載。 |
| 無<br><br>API Gateway 預設為 application/json | WHEN_NO_TEMPLATES | 使用此範本轉換請求承載。 |
| 無<br><br>API Gateway 預設為 application/json | NEVER | 使用此範本轉換請求承載。 |
| application/json | WHEN_NO_MATCH | 使用此範本轉換請求承載。 |
| application/json | WHEN_NO_TEMPLATES | 使用此範本轉換請求承載。 |
| application/json | NEVER | 使用此範本轉換請求承載。 |
| application/xml | WHEN_NO_MATCH | 請求承載未轉換，而且會依現狀傳送到後端。 |
| application/xml | WHEN_NO_TEMPLATES | 請求遭到拒絕，回應為 HTTP 415 Unsupported Media Type。 |
| application/xml | NEVER | 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 | 傳遞選項 | Behavior (行為) |
| --- | --- | --- |
| 無<br><br>API Gateway 預設為 `application/json` | `WHEN_NO_MATCH` | 請求承載未轉換，而且會依現狀傳送到後端。 |
| 無<br><br>API Gateway 預設為 `application/json` | `WHEN_NO_TEMPLATES` | 請求遭到拒絕，回應為 HTTP `415 Unsupported Media Type`。 |
| 無<br><br>API Gateway 預設為 `application/json` | `NEVER` | 請求遭到拒絕，回應為 HTTP `415 Unsupported Media Type`。 |
| `application/json` | `WHEN_NO_MATCH` | 請求承載未轉換，而且會依現狀傳送到後端。 |
| `application/json` | `WHEN_NO_TEMPLATES` | 請求遭到拒絕，回應為 HTTP `415 Unsupported Media Type`。 |
| `application/json` | `NEVER` | 請求遭到拒絕，回應為 HTTP `415 Unsupported Media Type`。 |
| `application/xml` | `WHEN_NO_MATCH` | 使用此範本轉換請求承載。 |
| `application/xml` | `WHEN_NO_TEMPLATES` | 使用此範本轉換請求承載。 |
| `application/xml` | `NEVER` | 使用此範本轉換請求承載。 |

範例 3：整合請求中未定義任何映射範本。

| Content-type | 傳遞選項 | Behavior (行為) |
| --- | --- | --- |
| 無<br><br>API Gateway 預設為 `application/json` | `WHEN_NO_MATCH` | 請求承載未轉換，而且會依現狀傳送到後端。 |
| 無<br><br>API Gateway 預設為 `application/json` | `WHEN_NO_TEMPLATES` | 請求承載未轉換，而且會依現狀傳送到後端。 |
| 無<br><br>API Gateway 預設為 `application/json` | `NEVER` | 請求遭到拒絕，回應為 HTTP `415 Unsupported Media Type`。 |
| `application/json` | `WHEN_NO_MATCH` | 請求承載未轉換，而且會依現狀傳送到後端。 |
| `application/json` | `WHEN_NO_TEMPLATES` | 請求承載未轉換，而且會依現狀傳送到後端。 |
| `application/json` | `NEVER` | 請求遭到拒絕，回應為 HTTP `415 Unsupported Media Type`。 |
| `application/xml` | `WHEN_NO_MATCH` | 請求承載未轉換，而且會依現狀傳送到後端。 |
| `application/xml` | `WHEN_NO_TEMPLATES` | 請求承載未轉換，而且會依現狀傳送到後端。 |
| `application/xml` | `NEVER` | 請求遭到拒絕，回應為 HTTP `415 Unsupported Media Type`。 |

## API Gateway 中 REST API 的其他映射範本範例

下列範例顯示 API Gateway 中的相簿 API，其使用映射範本來轉換整合請求和整合回應資料。此 API 也會使用資料模型來定義方法請求和整合回應承載。如需更多有關資料模型的詳細資訊，請參閱 REST API 的資料模型。

### 方法請求和整合請求

以下是定義方法請求內文的模型。此輸入模型會要求發起人上傳一張相片，並要求每頁至少 10 張相片。您可以使用此輸入模型來產生 SDK，或針對您的 API 使用請求驗證。使用請求驗證時，如果方法請求內文未遵循模型的資料結構，則 API Gateway 會讓請求失敗。

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "title": "PhotosInputModel",
  "type": "object",
  "properties": {
    "photos": {
      "type": "object",
      "required" : [
      "photo"
      ],
      "properties": {
        "page": { "type": "integer" },
        "pages": { "type": "string" },
        "perpage": { "type": "integer", "minimum" : 10 },
        "total": { "type": "string" },
        "photo": {
          "type": "array",
          "items": {
            "type": "object",
            "properties": {
              "id": { "type": "string" },
              "owner": { "type": "string" },
              "photographer_first_name" : {"type" : "string"},
              "photographer_last_name" : {"type" : "string"},
              "secret": { "type": "string" },
              "server": { "type": "string" },
              "farm": { "type": "integer" },
              "title": { "type": "string" },
              "ispublic": { "type": "boolean" },
              "isfriend": { "type": "boolean" },
              "isfamily": { "type": "boolean" }
```

```
            }
          }
        }
      }
    }
  }
}
```

以下是遵循上述資料模型之資料結構的方法請求內文範例。

```
{
  "photos": {
    "page": 1,
    "pages": "1234",
    "perpage": 100,
    "total": "123398",
    "photo": [
      {
        "id": "12345678901",
        "owner": "23456789@A12",
        "photographer_first_name" : "Saanvi",
        "photographer_last_name" : "Sarkar",
        "secret": "abc123d456",
        "server": "1234",
        "farm": 1,
        "title": "Sample photo 1",
        "ispublic": true,
        "isfriend": false,
        "isfamily": false
      },
      {
        "id": "23456789012",
        "owner": "34567890@B23",
        "photographer_first_name" : "Richard",
        "photographer_last_name" : "Roe",
        "secret": "bcd234e567",
        "server": "2345",
        "farm": 2,
        "title": "Sample photo 2",
        "ispublic": true,
        "isfriend": false,
        "isfamily": false
      }
```

```
    ]
  }
}
```

在此範例中，如果用戶端提交了上述方法請求內文，則此映射範本會轉換承載，使其符合整合端點要求的格式。

```
#set($inputRoot = $input.path('$'))
{
  "photos": [
#foreach($elem in $inputRoot.photos.photo)
    {
      "id": "$elem.id",
      "photographedBy": "$elem.photographer_first_name $elem.photographer_last_name",
      "title": "$elem.title",
      "ispublic": $elem.ispublic,
      "isfriend": $elem.isfriend,
      "isfamily": $elem.isfamily
    }#if($foreach.hasNext),#end

#end
  ]
}
```

下列範例是轉換的輸出資料：

```
{
  "photos": [
    {
      "id": "12345678901",
      "photographedBy": "Saanvi Sarkar",
      "title": "Sample photo 1",
      "ispublic": true,
      "isfriend": false,
      "isfamily": false
    },
    {
      "id": "23456789012",
      "photographedBy": "Richard Roe",
      "title": "Sample photo 2",
      "ispublic": true,
      "isfriend": false,
      "isfamily": false
```

```
    }
  ]
}
```

此資料會傳送到整合請求，然後再傳送到整合端點。

整合回應和方法回應

以下是整合端點中相片資料的範例輸出模型。您可以使用此模型作為方法回應模型，當您為 API 產生強型別 SDK 時，需要這樣做。這會導致輸出在 Java 或 Objective-C 中轉換成適當的類別。

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "title": "PhotosOutputModel",
  "type": "object",
  "properties": {
    "photos": {
      "type": "array",
      "items": {
        "type": "object",
        "properties": {
          "id": { "type": "string" },
          "photographedBy": { "type": "string" },
          "title": { "type": "string" },
          "ispublic": { "type": "boolean" },
          "isfriend": { "type": "boolean" },
          "isfamily": { "type": "boolean" }
        }
      }
    }
  }
}
```

整合端點的回應可能無法使用遵循此模型的資料結構。例如，整合回應可能如下所示：

```
  "photos": [
    {
      "id": "12345678901",
      "photographedBy": "Saanvi Sarkar",
      "title": "Sample photo 1",
      "description": "My sample photo 1",
      "public": true,
```

```
      "friend": false,
      "family": false
    },
    {
      "id": "23456789012",
      "photographedBy": "Richard Roe",
      "title": "Sample photo 2",
      "description": "My sample photo 1",
      "public": true,
      "friend": false,
      "family": false
    }
  ]
}
```

下列範例映射範本會將整合回應資料轉換為方法回應預期的格式：

```
#set($inputRoot = $input.path('$'))
{
  "photos": [
#foreach($elem in $inputRoot.photos.photo)
    {
      "id": "$elem.id",
      "photographedBy": "$elem.photographer_first_name $elem.photographer_last_name",
      "title": "$elem.title",
      "ispublic": $elem.public,
      "isfriend": $elem.friend,
      "isfamily": $elem.family
    }#if($foreach.hasNext),#end

#end
  ]
}
```

下列範例是轉換的輸出資料：

```
{
  "photos": [
    {
      "id": "12345678901",
      "photographedBy": "Saanvi Sarkar",
      "title": "Sample photo 1",
      "ispublic": true,
```

```
      "isfriend": false,
      "isfamily": false
    },
    {
      "id": "23456789012",
      "photographedBy": "Richard Roe",
      "title": "Sample photo 2",
      "ispublic": true,
      "isfriend": false,
      "isfamily": false
    }
  ]
}
```

此資料會傳送到方法回應，然後再傳回用戶端。

覆寫 API Gateway 中 REST API 的 API 請求和回應參數及狀態碼

您可以使用映射範本轉換來覆寫任何類型的請求參數、回應標頭或回應狀態碼。使用映射範本執行下列操作：

- 執行多對一參數映射
- 在套用標準 API Gateway 映射之後覆寫參數
- 根據內文內容或其他參數值，有條件地映射參數
- 以程式設計方式建立新參數
- 覆寫由您的整合端點所傳回的狀態碼

覆寫是最終。覆寫只能套用到每個參數一次。如果您嘗試覆寫相同的參數多次，API Gateway 會傳回 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。只會在整個範本剖析後才套用範本。

範例 1：根據整合內文覆寫狀態碼

下列範例使用範例 API，根據整合回應內文覆寫狀態碼。

AWS 管理主控台

根據整合回應內文覆寫狀態碼

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 選擇 Create API (建立 API)。

3. 針對 REST API，選擇組建。
4. 針對 API 詳細資訊，選擇範例 API。
5. 選擇建立 API。

   API Gateway 會建立範例寵物商店 API。若要擷取有關寵物的資訊，您可以使用 API 方法請求 `GET /pets/{petId}`，其中 `{petId}` 是對應寵物 ID 號碼的路徑參數。

   在此範例中，您會在偵測到錯誤條件時，將 GET 方法的回應程式碼覆寫為 400。
6. 在資源樹狀結構中的 `/{petId}` 下，選擇 GET 方法。
7. 首先，測試目前的 API 實作。

   選擇測試標籤。您可能需要選擇向右箭頭按鈕才能顯示此索引標籤。
8. 針對 petId，輸入 **-1**，然後選擇測試。

   回應內文指出超出範圍錯誤：

   ```
   {
     "errors": [
       {
         "key": "GetPetRequest.petId",
         "message": "The value is out of range."
       }
     ]
   }
   ```

   此外，日誌下的最後一行結尾為：`Method completed with status: 200`。

   整合已成功完成，但發生錯誤。現在您將根據整合回應覆寫狀態碼。
9. 在整合回應索引標籤上，針對預設 - 回應，選擇編輯。
10. 選擇對應範本。
11. 選擇新增對應範本。
12. 針對內容類型，輸入 **application/json**。
13. 針對範本內文，輸入下列內容：

    ```
    #set($inputRoot = $input.path('$'))
    $input.json("$")
    #if($inputRoot.toString().contains("error"))
    #set($context.responseOverride.status = 400)
    ```

```
#end
```

如果整合回應包含字串 `error`，則此映射範本會使用 `$context.responseOverride.status` 變數將狀態碼覆寫為 `400`。

14. 選擇儲存。
15. 選擇測試標籤。
16. 針對 petId，輸入 **-1**。
17. 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，日誌下的最後一列結尾現在會是：`Method completed with status: 400`。

### CloudFormation

在此範例中，您使用 body 屬性將 OpenAPI 定義檔案匯入 API Gateway 中。

```
AWSTemplateFormatVersion: 2010-09-09
Resources:
  Api:
    Type: 'AWS::ApiGateway::RestApi'
    Properties:
      Body:
        openapi: 3.0.1
        info:
          title: PetStore Example 1
          description: Example pet store API.
          version: "2025-01-14T00:13:18Z"
        paths:
          /pets/{petId}:
            get:
              parameters:
                - name: petId
```

```
                in: path
                required: true
                schema:
                  type: string
            responses:
              "200":
                description: 200 response
            x-amazon-apigateway-integration:
              httpMethod: GET
              uri: http://petstore.execute-api.us-east-1.amazonaws.com/petstore/
pets/{petId}
              responses:
                default:
                  statusCode: "200"
                  responseTemplates:
                    application/json: |-
                      #set($inputRoot = $input.path('$'))
                      $input.json("$")
                      #if($inputRoot.toString().contains("error"))
                      #set($context.responseOverride.status = 400)
                      #end
              requestParameters:
                integration.request.path.petId: method.request.path.petId
              passthroughBehavior: when_no_match
              type: http
        components:
          schemas:
            Pet:
              type: object
              properties:
                id:
                  type: integer
                type:
                  type: string
                price:
                  type: number
  ApiGatewayDeployment:
    Type: 'AWS::ApiGateway::Deployment'
    DependsOn: Api
    Properties:
      RestApiId: !Ref Api
  ApiGatewayDeployment20250219:
    Type: 'AWS::ApiGateway::Deployment'
    DependsOn: Api
```

```
    Properties:
      RestApiId: !Ref Api
  Stage:
    Type: 'AWS::ApiGateway::Stage'
    Properties:
       DeploymentId: !Ref ApiGatewayDeployment20250219
       RestApiId: !Ref Api
       StageName: prod
```

### OpenAPI

以下 OpenAPI 定義會建立 GET pets/{petId} 資源，並根據整合內文覆寫狀態碼。

```
{
  "openapi" : "3.0.1",
  "info" : {
    "title" : "PetStore Example 1",
    "description" : "Example pet store API.",
    "version" : "2025-01-14T00:13:18Z"
  },
  "paths" : {
    "/pets/{petId}" : {
      "get" : {
        "parameters" : [ {
          "name" : "petId",
          "in" : "path",
          "required" : true,
          "schema" : {
            "type" : "string"
          }
        } ],
        "responses" : {
          "200" : {
            "description" : "200 response"
          }
        },
        "x-amazon-apigateway-integration" : {
          "httpMethod" : "GET",
          "uri" : "http://petstore.execute-api.us-east-1.amazonaws.com/petstore/
pets/{petId}",
          "responses" : {
            "default" : {
              "statusCode" : "200",
              "responseTemplates" : {
```

```
                "application/json" : "#set($inputRoot = $input.path('$'))\n
$input.json(\"$\")\n#if($inputRoot.toString().contains(\"error
\"))\n#set($context.responseOverride.status = 400)\n#end"
              }
            }
          },
          "requestParameters" : {
            "integration.request.path.petId" : "method.request.path.petId"
          },
          "passthroughBehavior" : "when_no_match",
          "type" : "http"
        }
      }
    }
  },
  "components" : {
    "schemas" : {
      "Pet" : {
        "type" : "object",
        "properties" : {
          "id" : {
            "type" : "integer"
          },
          "type" : {
            "type" : "string"
          },
          "price" : {
            "type" : "number"
          }
        }
      }
    }
  }
}
```

### 範例 2：覆寫請求標頭並建立新標頭

下列範例使用範例 API 覆寫請求標頭並建立新標頭。

AWS 管理主控台

透過建立新標頭來覆寫方法的請求標頭

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 選擇您在上述教學課程中建立的範例 API。API 的名稱應為 PetStore。
3. 在資源樹狀結構中的 /pet 下，選擇 GET 方法。
4. 在方法請求索引標籤上，針對方法請求設定，選擇編輯。
5. 選擇 HTTP 請求標頭，然後選擇新增標頭。
6. 對於名稱，輸入 **header1**。
7. 選擇新增標頭，然後建立名為 **header2** 的第二個標頭。
8. 選擇儲存。

   現在，請使用映射範本將這些標頭合併成一個標頭值。
9. 在整合請求索引標籤上，針對整合請求設定，選擇編輯。
10. 針對請求內文傳遞，選取未定義範本時 (建議)。
11. 選擇對應範本，然後執行下列動作：

    a. 選擇新增映射範本。

    b. 針對內容類型，輸入 **application/json**。

    c. 針對範本內文，輸入下列內容：

    ```
    #set($header1Override = "pets")
    #set($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])
    ```

    此映射範本會將 header1 覆寫為字串 pets，並建立結合 header1 和 header2 且名為 $header3Value 的多值標頭。
12. 選擇儲存。
13. 選擇測試標籤。
14. 在標頭下，複製下列程式碼：

```
header1:header1Val
header2:header2Val
```

15. 選擇 Test (測試)。

    在日誌中，您應該會看到包括此文字的項目：

```
Endpoint request headers: {header3=header1Valheader2Val,
header2=header2Val, header1=pets, x-amzn-apigateway-api-id=api-id,
Accept=application/json, multivalueheader=pets,header1Valheader2Val}
```

## CloudFormation

在此範例中，您使用 body 屬性將 OpenAPI 定義檔案匯入 API Gateway 中。

```
AWSTemplateFormatVersion: 2010-09-09
Resources:
  Api:
    Type: 'AWS::ApiGateway::RestApi'
    Properties:
      Body:
        openapi: 3.0.1
        info:
          title: PetStore Example 2
          description: Example pet store API.
          version: "2025-01-14T00:36:18Z"
        paths:
          /pets:
            get:
              parameters:
                - name: header2
                  in: header
                  schema:
                    type: string
                - name: page
                  in: query
                  schema:
                    type: string
                - name: type
                  in: query
                  schema:
```

```
                    type: string
                - name: header1
                  in: header
                  schema:
                    type: string
              responses:
                "200":
                  description: 200 response
              x-amazon-apigateway-integration:
                httpMethod: GET
                uri: http://petstore.execute-api.us-east-1.amazonaws.com/petstore/
pets
                responses:
                  default:
                    statusCode: "200"
                requestParameters:
                  integration.request.header.header1: method.request.header.header1
                  integration.request.header.header2: method.request.header.header2
                  integration.request.querystring.page:
 method.request.querystring.page
                  integration.request.querystring.type:
 method.request.querystring.type
                requestTemplates:
                  application/json: |-
                    #set($header1Override = "pets")
                    #set($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])
                passthroughBehavior: when_no_match
                type: http
        components:
          schemas:
            Pet:
              type: object
              properties:
                id:
                  type: integer
                type:
                  type: string
```

```
                price:
                  type: number
  ApiGatewayDeployment:
    Type: 'AWS::ApiGateway::Deployment'
    DependsOn: Api
    Properties:
      RestApiId: !Ref Api
  ApiGatewayDeployment20250219:
    Type: 'AWS::ApiGateway::Deployment'
    DependsOn: Api
    Properties:
      RestApiId: !Ref Api
  Stage:
    Type: 'AWS::ApiGateway::Stage'
    Properties:
       DeploymentId: !Ref ApiGatewayDeployment20250219
       RestApiId: !Ref Api
       StageName: prod
```

### OpenAPI

以下 OpenAPI 定義會建立 GET pets 資源，以及覆寫請求標頭並建立新標頭。

```
{
  "openapi" : "3.0.1",
  "info" : {
    "title" : "PetStore Example 2",
    "description" : "Example pet store API.",
    "version" : "2025-01-14T00:36:18Z"
  },
  "paths" : {
    "/pets" : {
      "get" : {
        "parameters" : [ {
          "name" : "header2",
          "in" : "header",
          "schema" : {
            "type" : "string"
          }
        }, {
          "name" : "page",
          "in" : "query",
          "schema" : {
            "type" : "string"
```

```
          }
        }, {
          "name" : "type",
          "in" : "query",
          "schema" : {
            "type" : "string"
          }
        }, {
          "name" : "header1",
          "in" : "header",
          "schema" : {
            "type" : "string"
          }
        } ],
        "responses" : {
          "200" : {
            "description" : "200 response"
          }
        },
        "x-amazon-apigateway-integration" : {
          "httpMethod" : "GET",
          "uri" : "http://petstore.execute-api.us-east-1.amazonaws.com/petstore/
pets",
          "responses" : {
            "default" : {
              "statusCode" : "200"
            }
          },
          "requestParameters" : {
            "integration.request.header.header1" : "method.request.header.header1",
            "integration.request.header.header2" : "method.request.header.header2",
            "integration.request.querystring.page" :
 "method.request.querystring.page",
            "integration.request.querystring.type" :
 "method.request.querystring.type"
          },
          "requestTemplates" : {
            "application/json" : "#set($header1Override = \"pets
\")\n#set($header3Value = \"$input.params('header1')$input.params('header2')\")\n
$input.json(\"$\")\n#set($context.requestOverride.header.header3
 = $header3Value)\n#set($context.requestOverride.header.header1 =
 $header1Override)\n#set($context.requestOverride.header.multivalueheader=[$header1Override,
 $header3Value])"
          },
```

```
          "passthroughBehavior" : "when_no_match",
          "type" : "http"
        }
      }
    }
  }
}
```

若要使用映射範本覆寫，請新增一或多個下列 $context 變數。如需 $context 變數清單，請參閱 the section called “資料轉換的內容變數”。

教學課程：修改 AWS 服務整合的整合請求和回應

下列教學課程說明如何使用映射範本轉換來設定映射範本，以使用主控台和 CLI AWS 轉換整合請求和回應。

主題

- 使用 API Gateway 主控台設定資料轉換
- 使用 CLI AWS 設定資料轉換
- 已完成的資料轉換 CloudFormation 範本

使用 API Gateway 主控台設定資料轉換

在本教學課程中，您將使用下列 .zip 檔案 data-transformation-tutorial-console.zip，建立不完整的 API 和 DynamoDB 資料表。此不完整的 API 具有 /pets 資源，其中含有 GET 和 POST 方法。

- GET 方法將從 http://petstore-demo-endpoint.execute-api.com/petstore/pets HTTP 端點取得資料。輸出資料將根據 the section called “映射範本轉換” 中的對應範本進行轉換。
- POST 方法可讓使用者使用對應範本，將寵物資訊 POST 至 Amazon DynamoDB 資料表。

下載並解壓縮 的應用程式建立範本 CloudFormation。您將使用此範本建立 DynamoDB 資料表，以發布寵物資訊和不完整的 API。您將完成 API Gateway 主控台中的其餘步驟。

建立 CloudFormation 堆疊

1. 在 https：//https://console.aws.amazon.com/cloudformation 開啟 CloudFormation 主控台。
2. 選擇 Create stack (建立堆疊)，然後選擇 With new resources (standard) (使用新資源 (標準))。

3. 對於 Specify template (指定範本)，選擇 Upload a template file (上傳範本檔案)。
4. 選取您下載的範本。
5. 選擇 Next (下一步)。
6. 針對 Stack name (堆疊名稱)，輸入 **data-transformation-tutorial-console**，然後選擇 Next (下一步)。
7. 針對 Configure stack options (設定堆疊選項)，選擇 Next (下一步)。
8. 針對 功能，確認 CloudFormation 可以在您的帳戶中建立 IAM 資源。
9. 選擇下一步，然後選擇提交。

CloudFormation 會佈建範本中指定的資源。完成資源佈建可能需要幾分鐘的時間。當 CloudFormation 堆疊的狀態為 CREATE_COMPLETE 時，您就可以繼續進行下一個步驟。

測試 **GET** 整合回應

1. 在 CloudFormation 堆疊的資源索引標籤上**data-transformation-tutorial-console**，選取 API 的實體 ID。
2. 在主導覽窗格中，選擇資源，然後選取 GET 方法。
3. 選擇測試標籤。您可能需要選擇向右箭頭按鈕才能顯示此索引標籤。

   測試的輸出將顯示下列內容：

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

您將根據 the section called “映射範本轉換”中的對應範本轉換此輸出。

### 轉換 GET 整合回應

1. 選擇整合回應索引標籤。

   目前，沒有已定義的對應範本，因此不會轉換整合回應。
2. 針對預設 - 回應，選擇編輯。
3. 選擇對應範本，然後執行下列動作：

   a. 選擇新增映射範本。

   b. 針對內容類型，輸入 **application/json**。

   c. 針對範本內文，輸入下列內容：

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

#end
]
```

   選擇儲存。

### 測試 GET 整合回應

- 選擇測試索引標籤，然後選擇測試。

  測試的輸出將顯示轉換後的回應。

```
[
  {
    "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
  }
]
```

### 從 **POST** 方法轉換輸入資料

1. 選擇 POST 方法。
2. 選擇整合請求索引標籤，然後針對整合請求設定，選擇編輯。

   CloudFormation 範本已填入一些整合請求欄位。

   - 整合類型為 AWS 服務。
   - AWS 服務 是 DynamoDB。
   - HTTP 方法為 POST。
   - 動作為 `PutItem`。
   - 允許 API Gateway 將項目放入 DynamoDB 資料表的執行角色是 `data-transformation-tutorial-console-APIGatewayRole`. CloudFormation created this role，以允許 API Gateway 具有與 DynamoDB 互動的最低許可。

   尚未指定 DynamoDB 資料表的名稱。您將在下列步驟中指定此名稱。
3. 針對請求內文傳遞，選取永不。

   這意味著 API 將拒絕 Content-Type 沒有對應範本的資料。
4. 選擇對應範本。
5. 內容類型會設定為 `application/json`。這表示不是 application/json 的內容類型將遭 API 拒絕。如需整合傳遞行為的詳細資訊，請參閱 API Gateway 中 REST API 沒有映射範本時，承載的方法請求行為。
6. 將下列程式碼輸入至文字編輯器。

```
{
    "TableName":"data-transformation-tutorial-console-ddb",
```

```
    "Item": {
        "id": {
            "N": $input.json("$.id")
        },
        "type": {
            "S": $input.json("$.type")
        },
        "price": {
            "N": $input.json("$.price")
        }
    }
}
```

此範本會將資料表指定為 `data-transformation-tutorial-console-ddb`，並將項目設為 `id`、`type` 和 `price`。這些項目將來自 POST 方法的內文。您也可以使用資料模型來協助建立對應範本。如需詳細資訊，請參閱API Gateway 中的 REST API 的請求驗證。

7. 選擇儲存以儲存您的映射範本。

### 從 **POST** 方法新增方法和整合回應

CloudFormation 已建立空白方法和整合回應。您將編輯此回應，以提供詳細資訊。如需如何編輯回應的詳細資訊，請參閱 API Gateway 中 REST API 的參數映射範例。

1. 在整合回應索引標籤上，針對預設 - 回應選擇編輯。
2. 選擇對應範本，然後選擇新增對應範本。
3. 針對 Content-Type，輸入 **application/json**。
4. 在程式碼編輯器中，輸入下列輸出對應範本以傳送輸出訊息：

```
{ "message" : "Your response was recorded at $context.requestTime" }
```

如需內容變數的詳細資訊，請參閱 資料轉換的內容變數。

5. 選擇儲存以儲存您的對應範本。

### 測試 **POST** 方法

選擇測試標籤。您可能需要選擇向右箭頭按鈕才能顯示此索引標籤。

1. 在請求內文中，輸入下列範例。

```
{
          "id": "4",
          "type" : "dog",
          "price": "321"
}
```

2. 選擇測試。

   輸出應該會顯示您的成功訊息。

   您可以在 https://console.aws.amazon.com/dynamodb/ 開啟 DynamoDB 主控台，以驗證範例項目是否位於您的資料表中。

### 刪除 CloudFormation 堆疊

1. 在 https：//https://console.aws.amazon.com/cloudformation 開啟 CloudFormation 主控台。
2. 選取您的 CloudFormation 堆疊。
3. 選擇刪除，然後確認您的選擇。

### 使用 CLI AWS 設定資料轉換

在本教學課程中，您將使用下列 .zip 檔案 data-transformation-tutorial-cli.zip，建立不完整的 API 和 DynamoDB 資料表。這個不完整的 API 具有 `/pets` 資源，其中含有與 `http://petstore-demo-endpoint.execute-api.com/petstore/pets` HTTP 端點整合的 `GET` 方法。您將建立 `POST` 方法，以連線至 DynamoDB 資料表，並使用對應範本將資料輸入至 DynamoDB 資料表。

- 您將根據 the section called “映射範本轉換”中的對應範本轉換輸出資料。
- 您將建立 `POST` 方法，允許使用者使用對應範本，將寵物資訊 `POST` 至 Amazon DynamoDB 資料表。

### 建立 CloudFormation 堆疊

下載並解壓縮 的應用程式建立範本 CloudFormation。

若要完成下列教學課程，您需要 AWS Command Line Interface (AWS CLI) 第 2 版。

對於長命令，逸出字元 (\) 用於將命令分割為多行。

> **Note**
>
> 在 Windows 中，作業系統的內建終端機不支援您常用的某些 Bash CLI 命令 (例如 `zip`)。若要取得 Ubuntu 和 Bash 的 Windows 整合版本，請安裝適用於 Linux 的 Windows 子系統。本指南中的 CLI 命令範例使用 Linux 格式。如果您使用的是 Windows CLI，必須重新格式化包含內嵌 JSON 文件的命令。

1. 使用下列命令來建立 CloudFormation 堆疊。

```
aws cloudformation create-stack --stack-name data-transformation-tutorial-cli
 --template-body file://data-transformation-tutorial-cli.zip --capabilities
 CAPABILITY_NAMED_IAM
```

2. CloudFormation 會佈建範本中指定的資源。完成資源佈建可能需要幾分鐘的時間。使用下列命令來查看 CloudFormation 堆疊的狀態。

```
aws cloudformation describe-stacks --stack-name data-transformation-tutorial-cli
```

3. 當您的 CloudFormation 堆疊狀態為 時`StackStatus: "CREATE_COMPLETE"`，請使用下列命令來擷取未來步驟的相關輸出值。

```
aws cloudformation describe-stacks --stack-name data-transformation-tutorial-cli
--query "Stacks[*].Outputs[*].{OutputKey: OutputKey, OutputValue: OutputValue,
Description: Description}"
```

   輸出值如下：

   - ApiRole，這是允許 API Gateway 將項目放入 DynamoDB 資料表的角色名稱。對於本教學課程，角色名稱為 `data-transformation-tutorial-cli-APIGatewayRole-ABCDEFG`。
   - DDBTableName，這是 DynamoDB 資料表的名稱。對於本教學課程，資料表名稱為 `data-transformation-tutorial-cli-ddb`。
   - ResourceId，這是 GET 和 POST 方法公開所在寵物資源的 ID。對於本教學課程，資源 ID 為 `efg456`
   - ApiId，這是 API 的 ID。對於本教學課程，API ID 為 `abc123`。

### 在資料轉換之前測試 GET 方法

- 使用下列命令測試 GET 方法。

```
aws apigateway test-invoke-method --rest-api-id abc123 \
          --resource-id efg456 \
          --http-method GET
```

測試的輸出將顯示下列內容。

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

您將根據 the section called “映射範本轉換”中的對應範本轉換此輸出。

### 轉換 GET 整合回應

- 使用下列命令更新 GET 方法的整合回應。將 *rest-api-id* 和 *resource-id* 取代為您的值。

使用下列命令建立整合回應。

```
aws apigateway put-integration-response --rest-api-id abc123 \
  --resource-id efg456 \
  --http-method GET \
  --status-code 200 \
  --selection-pattern "" \
```

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

### 測試 GET 方法

- 使用下列命令測試 GET 方法。

```
aws apigateway test-invoke-method --rest-api-id abc123 \
  --resource-id efg456 \
  --http-method GET \
```

測試的輸出將顯示轉換後的回應。

```
[
  {
    "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
  }
]
```

### 建立 POST 方法

1. 使用下列命令，在 /pets 資源上建立新方法。

```
aws apigateway put-method --rest-api-id abc123 \
  --resource-id efg456 \
  --http-method POST \
  --authorization-type "NONE" \
```

此方法可讓您將寵物資訊傳送至您在 CloudFormation 堆疊中建立的 DynamoDB 資料表。

2. 使用下列命令在 POST方法上建立 AWS 服務 整合。

```
aws apigateway put-integration --rest-api-id abc123 \
  --resource-id efg456 \
  --http-method POST \
  --type AWS \
  --integration-http-method POST \
  --uri "arn:aws:apigateway:us-east-2:dynamodb:action/PutItem" \
  --credentials arn:aws:iam::111122223333:role/data-transformation-tutorial-cli-APIGatewayRole-ABCDEFG \
  --request-templates '{"application/json":"{\"TableName\":\"data-transformation-tutorial-cli-ddb\",\"Item\":{\"id\":{\"N\":$input.json(\"$.id\")},\"type\":{\"S\":$input.json(\"$.type\")},\"price\":{\"N\":$input.json(\"$.price\")} }}"}'
```

3. 使用下列命令，為 POST 方法的成功呼叫建立方法回應。

```
aws apigateway put-method-response --rest-api-id abc123 \
  --resource-id efg456 \
  --http-method POST \
  --status-code 200
```

4. 使用下列命令，為 POST 方法的成功呼叫建立整合回應。

```
aws apigateway put-integration-response --rest-api-id abc123 \
  --resource-id efg456 \
  --http-method POST \
  --status-code 200 \
  --selection-pattern "" \
  --response-templates '{"application/json": "{\"message\": \"Your response was recorded at $context.requestTime\"}"}'
```

### 測試 POST 方法

- 使用下列命令測試 POST 方法。

```
aws apigateway test-invoke-method --rest-api-id abc123 \
  --resource-id efg456 \
  --http-method POST \
  --body '{\"id\": \"4\", \"type\": \"dog\", \"price\": \"321\"}'
```

輸出將顯示成功訊息。

### 刪除 CloudFormation 堆疊

- 使用下列命令來刪除您的 CloudFormation 資源。

```
aws cloudformation delete-stack  --stack-name data-transformation-tutorial-cli
```

### 已完成的資料轉換 CloudFormation 範本

下列範例是已完成的 CloudFormation 範本，其會使用 和 POST方法建立具有 /pets 資源的 API GET 和 DynamoDB 資料表。

- GET 方法將從 http://petstore-demo-endpoint.execute-api.com/petstore/pets HTTP 端點取得資料。輸出資料將根據 the section called “映射範本轉換”中的對應範本進行轉換。
- POST 方法可讓使用者使用對應範本，將寵物資訊 POST 至 DynamoDB 資料表。

### 範本範例 CloudFormation

```
AWSTemplateFormatVersion: 2010-09-09
Description: A completed Amazon API Gateway REST API that uses non-proxy integration
 to POST to an Amazon DynamoDB table and non-proxy integration to GET transformed pets
 data.
Parameters:
  StageName:
    Type: String
    Default: v1
    Description: Name of API stage.
Resources:
  DynamoDBTable:
    Type: 'AWS::DynamoDB::Table'
    Properties:
      TableName: !Sub data-transformation-tutorial-complete
      AttributeDefinitions:
        - AttributeName: id
          AttributeType: N
      KeySchema:
        - AttributeName: id
          KeyType: HASH
```

```
      ProvisionedThroughput:
        ReadCapacityUnits: 5
        WriteCapacityUnits: 5
  APIGatewayRole:
    Type: 'AWS::IAM::Role'
    Properties:
      AssumeRolePolicyDocument:
        Version: 2012-10-17
        Statement:
          - Action:
              - 'sts:AssumeRole'
            Effect: Allow
            Principal:
              Service:
                - apigateway.amazonaws.com
      Policies:
        - PolicyName: APIGatewayDynamoDBPolicy
          PolicyDocument:
            Version: 2012-10-17
            Statement:
              - Effect: Allow
                Action:
                  - 'dynamodb:PutItem'
                Resource: !GetAtt DynamoDBTable.Arn
  Api:
    Type: 'AWS::ApiGateway::RestApi'
    Properties:
      Name: data-transformation-complete-api
      ApiKeySourceType: HEADER
  PetsResource:
    Type: 'AWS::ApiGateway::Resource'
    Properties:
      RestApiId: !Ref Api
      ParentId: !GetAtt Api.RootResourceId
      PathPart: 'pets'
  PetsMethodGet:
    Type: 'AWS::ApiGateway::Method'
    Properties:
      RestApiId: !Ref Api
      ResourceId: !Ref PetsResource
      HttpMethod: GET
      ApiKeyRequired: false
      AuthorizationType: NONE
      Integration:
```

```
        Type: HTTP
        Credentials: !GetAtt APIGatewayRole.Arn
        IntegrationHttpMethod: GET
        Uri: http://petstore-demo-endpoint.execute-api.com/petstore/pets/
        PassthroughBehavior: WHEN_NO_TEMPLATES
        IntegrationResponses:
          - StatusCode: '200'
            ResponseTemplates:
              application/json: "#set($inputRoot = $input.path(\"$
\"))\n[\n#foreach($elem in $inputRoot)\n {\n  \"description\": \"Item $elem.id is a
 $elem.type\",\n  \"askingPrice\": \"$elem.price\"\n }#if($foreach.hasNext),#end\n
\n#end\n]"
      MethodResponses:
        - StatusCode: '200'
  PetsMethodPost:
    Type: 'AWS::ApiGateway::Method'
    Properties:
      RestApiId: !Ref Api
      ResourceId: !Ref PetsResource
      HttpMethod: POST
      ApiKeyRequired: false
      AuthorizationType: NONE
      Integration:
        Type: AWS
        Credentials: !GetAtt APIGatewayRole.Arn
        IntegrationHttpMethod: POST
        Uri: arn:aws:apigateway:us-west-1:dynamodb:action/PutItem
        PassthroughBehavior: NEVER
        RequestTemplates:
          application/json: "{\"TableName\":\"data-transformation-tutorial-complete
\",\"Item\":{\"id\":{\"N\":$input.json(\"$.id\")},\"type\":{\"S\":$input.json(\"$.type
\")},\"price\":{\"N\":$input.json(\"$.price\")} }}"
        IntegrationResponses:
          - StatusCode: 200
            ResponseTemplates:
              application/json: "{\"message\": \"Your response was recorded at
 $context.requestTime\"}"
      MethodResponses:
        - StatusCode: '200'

  ApiDeployment:
    Type: 'AWS::ApiGateway::Deployment'
    DependsOn:
      - PetsMethodGet
```

```
    Properties:
      RestApiId: !Ref Api
      StageName: !Sub '${StageName}'
Outputs:
  ApiId:
    Description: API ID for CLI commands
    Value: !Ref Api
  ResourceId:
    Description: /pets resource ID for CLI commands
    Value: !Ref PetsResource
  ApiRole:
    Description: Role ID to allow API Gateway to put and scan items in DynamoDB table
    Value: !Ref APIGatewayRole
  DDBTableName:
    Description: DynamoDB table name
    Value: !Ref DynamoDBTable
```

使用變數進行 API Gateway 映射範本轉換的範例

以下範例說明如何在映射範本中使用 `$context`、`input` 和 `util` 變數。您可以使用模擬整合或 Lambda 非代理整合，將輸入事件傳回至 API Gateway。如需資料轉換支援的所有變數的清單，請參閱the section called “用於資料轉換的變數”。

範例 1：將多個 **`$context`** 變數傳遞至整合端點

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：

```
{
    "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"
}
```

此映射範本的輸出應如下所示：

```
{
  stage: 'prod',
  request_id: 'abcdefg-000-000-0000-abcdefg',
  api_id: 'abcd1234',
  resource_path: '/',
  resource_id: 'efg567',
  http_method: 'GET',
  source_ip: '192.0.2.1',
  user-agent: 'curl/7.84.0',
  account_id: '111122223333',
  api_key: 'MyTestKey',
  caller: 'ABCD-0000-12345',
  user: 'ABCD-0000-12345',
  user_arn: 'arn:aws:sts::111122223333:assumed-role/Admin/carlos-salazar'
}
```

其中一個變數是 API 金鑰。此範例假設該方法需要 API 金鑰。

範例 2：透過 JSON 承載將所有請求參數傳遞至整合端點

下列範例會透過 JSON 承載將所有請求參數 (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
  }
}
```

如果請求具有下列輸入參數：

- 名為 myparam 的路徑參數
- 查詢字串參數 querystring1=value1,value2
- 標頭 "header1" : "value1"。

此映射範本的輸出應如下所示：

```
{"params":{"path":{"example2":"myparamm"},"querystring":
{"querystring1":"value1,value2"},"header":{"header1":"value1"}}}
```

範例 3：將方法請求的子區段傳遞至整合端點

下列範例使用輸入參數 name 僅擷取 name 參數，並使用輸入參數 input.json('$') 擷取方法請求的整個內文：

```
{
    "name" : "$input.params('name')",
    "body" : $input.json('$')
}
```

對於包含查詢字串參數 name=Bella&type=dog 和下列內文的請求：

```
{
    "Price" : "249.99",
    "Age": "6"
}
```

此映射範本的輸出應如下所示：

```
{
    "name" : "Bella",
    "body" : {"Price":"249.99","Age":"6"}
}
```

此映射範本會移除查詢字串參數 type=dog。

如果 JSON 輸入包含 JavaScript 無法剖析的非逸出字元，則 API Gateway 可能會傳回 400 回應。套用 $util.escapeJavaScript($input.json('$')) 以確保 JSON 輸入可正確剖析。

套用 $util.escapeJavaScript($input.json('$')) 的前一個範例如下所示：

```
{
    "name" : "$input.params('name')",
    "body" : "$util.escapeJavaScript($input.json('$'))"
}
```

在這種情況下，此映射範本的輸出應如下所示：

```
{
    "name" : "Bella",
    "body": {"Price":"249.99","Age":"6"}
}
```

範例 4：使用 JSONPath 表達式將方法請求的子區段傳遞至整合端點

下列範例使用 JSONPath 表達式，僅從請求內文擷取輸入參數 name 和 Age：

```
{
    "name" : "$input.params('name')",
    "body" : $input.json('$.Age')
}
```

對於包含查詢字串參數 name=Bella&type=dog 和下列內文的請求：

```
{
    "Price" : "249.99",
    "Age": "6"
}
```

此映射範本的輸出應如下所示：

```
{
    "name" : "Bella",
    "body" : "6"
}
```

此映射範本會從內文中移除查詢字串參數 type=dog 和 Price 欄位。

如果方法請求承載包含 JavaScript 無法剖析的非逸出字元，API Gateway 可能會傳回 400 回應。套用 $util.escapeJavaScript() 以確保 JSON 輸入可正確剖析。

套用 $util.escapeJavaScript($input.json('$.Age')) 的前一個範例如下所示：

```
{
    "name" : "$input.params('name')",
    "body" : "$util.escapeJavaScript($input.json('$.Age'))"
}
```

在這種情況下，此映射範本的輸出應如下所示：

```
{
    "name" : "Bella",
    "body": "\"6\""
}
```

範例 5：使用 JSONPath 表達式將有關方法請求的資訊傳遞至整合端點

下列範例使用 `$input.params()`、`$input.path()` 和 `$input.json()` 將有關方法請求的資訊傳送至整合端點。此映射範本使用 `size()` 方法來提供清單中的元素數量。

```
{
    "id" : "$input.params('id')",
    "count" : "$input.path('$.things').size()",
    "things" : $input.json('$.things')
}
```

對於包含路徑參數 `123` 和下列內文的請求：

```
{
      "things": {
            "1": {},
            "2": {},
            "3": {}
      }
}
```

此映射範本的輸出應如下所示：

```
{"id":"123","count":"3","things":{"1":{},"2":{},"3":{}}}
```

如果方法請求承載包含 JavaScript 無法剖析的非逸出字元，API Gateway 可能會傳回 `400` 回應。套用 `$util.escapeJavaScript()` 以確保 JSON 輸入可正確剖析。

套用 $util.escapeJavaScript($input.json('$.things')) 的前一個範例如下所示：

```
{
     "id" : "$input.params('id')",
     "count" : "$input.path('$.things').size()",
     "things" : "$util.escapeJavaScript($input.json('$.things'))"
}
```

此映射範本的輸出應如下所示：

```
{"id":"123","count":"3","things":"{\"1\":{},\"2\":{},\"3\":{}}"}
```

## 用於 API Gateway 資料轉換的變數

建立參數映射時，您可以使用內容變數作為資料來源。建立映射範本轉換時，您可以在您使用 Velocity 範本語言 (VTL) 編寫的指令碼中使用內容變數、輸入和 util 變數。如需使用這些參考變數的映射範本範例，請參閱 the section called “使用變數進行映射範本轉換的範例”。

如需存取記錄的參考變數清單，請參閱 the section called “API Gateway 存取記錄的變數”。

資料轉換的內容變數

您可以使用下列區分大小寫的 $context 變數進行資料轉換。

| 參數 | 說明 |
| --- | --- |
| $context.accountId | API 擁有者 AWS 的帳戶 ID。 |
| $context.apiId | API Gateway 指派給您 API 的識別碼。 |
| $context.authorizer.claims.*property* | 成功驗證方法發起人之後，從 Amazon Cognito 使用者集區傳回之宣告的屬性。如需詳細資訊，請參閱 the section called “針對 REST API 使用 Amazon Cognito 使用者集區做為授權方”。<br><br>**Note**<br>呼叫 $context.authorizer.claims 會傳回 null。 |

| 參數 | 說明 |
| --- | --- |
| `$context.authorizer.principalId` | 與用戶端所傳送並從 API Gateway Lambda 授權方 (先前稱作自訂授權方) 所傳回之字符相關聯的主要使用者身分。如需詳細資訊，請參閱 the section called “使用 Lambda 授權方”。 |
| `$context.authorizer.`*property* | API Gateway Lambda 授權方函數所傳回 `context` 映射之指定索引鍵/值組的字串化值。例如，如果授權方傳回下列 `context` 映射：<br>`"context" : { "key": "value", "numKey": 1, "boolKey": true }`<br>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 `$context.authorizer.boolKey` 會傳回 `"true"` 字串。<br>對於*##*，唯一支援的特殊字元是底線 (`_`) 字元。<br>如需詳細資訊，請參閱the section called “使用 Lambda 授權方”。 |
| `$context.awsEndpointRequestId` | AWS 端點的請求 ID。 |
| `$context.deploymentId` | API 部署的 ID。 |
| `$context.domainName` | 用來調用 API 的完整網域名稱。這應該與傳入的 `Host` 標頭相同。 |
| `$context.domainPrefix` | `$context.domainName` 的第一個標籤。 |

| 參數 | 說明 |
| --- | --- |
| $context.error.message | 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 GatewayResponse 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the section called “指標”及the section called “設定閘道回應以自訂錯誤回應”。 |
| $context.error.messageString | $context.error.message 的引用值，即 "$context.error.message"。 |
| $context.error.responseType | GatewayResponse 的類型。此變數只能用於 Velocity 範本語言引擎無法處理的 GatewayResponse 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the section called “指標”及the section called “設定閘道回應以自訂錯誤回應”。 |
| $context.error.validationErrorString | 字串，其中包含詳細的驗證錯誤訊息。 |
| $context.extendedRequestId | API Gateway 產生和指派給 API 請求的延伸 ID。延伸請求 ID 包含有助於偵錯和疑難排解的資訊。 |
| $context.httpMethod | 使用的 HTTP 方法。有效值包含：DELETE、GET、HEAD、OPTIONS、PATCH、POST 和 PUT。 |
| $context.identity.accountId | 與請求相關聯的 AWS 帳戶 ID。 |
| $context.identity.apiKey | 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 the section called “用量計劃”。 |

| 參數 | 說明 |
| --- | --- |
| `$context.identity.apiKeyId` | 與需要 API 金鑰之 API 請求相關聯的 API 金鑰 ID。 |
| `$context.identity.caller` | 已簽署請求之發起人的主體識別符。支援使用 IAM 授權的資源。 |
| `$context.identity.cognitoAuthenticationProvider` | 提出請求的發起人所使用的所有 Amazon Cognito 驗證提供者清單 (以逗號分隔)。僅在使用 Amazon Cognito 登入資料簽署請求時才可使用。<br><br>例如，適用於 Amazon Cognito 使用者集區的身分，`cognito-idp.`*`region`*`.amazonaws.com/`*`user_pool_id`*`,cognito-idp.`*`region`*`.amazonaws.com/`*`user_pool_id`*`:CognitoSignIn:`*`token subject claim`*<br><br>如需有關 Amazon Cognito 驗證提供者的詳細資訊，請參閱《Amazon Cognito 開發人員指南》中的使用聯合身分。 |
| `$context.identity.cognitoAuthenticationType` | 提出請求的發起人的 Amazon Cognito 驗證類型。僅在使用 Amazon Cognito 登入資料簽署請求時才可使用。可能的值包括用於已驗證身分的 `authenticated` 和未經驗證身分的 `unauthenticated` 。 |
| `$context.identity.cognitoIdentityId` | 提出請求的發起人的 Amazon Cognito 身分 ID。僅在使用 Amazon Cognito 登入資料簽署請求時才可使用。 |
| `$context.identity.cognitoIdentityPoolId` | 提出請求的發起人的 Amazon Cognito 身分集區 ID。僅在使用 Amazon Cognito 登入資料簽署請求時才可使用。 |
| `$context.identity.principalOrgId` | AWS 組織 ID。 |

| 參數 | 說明 |
| --- | --- |
| $context.identity.sourceIp | 對 API Gateway 端點提出請求之即時 TCP 連線的來源 IP 位址。 |
| $context.identity.clientCert.clientCertPem | 用戶端在交互 TLS 驗證期間所呈現的 PEM 編碼用戶端憑證。當用戶端使用已啟用交互 TLS 的自訂網域名稱存取 API 時會顯示。只有在交互 TLS 驗證失敗時才會顯示在存取記錄檔中。 |
| $context.identity.clientCert.subjectDN | 用戶端提供之憑證主體的辨別名稱。當用戶端使用已啟用交互 TLS 的自訂網域名稱存取 API 時會顯示。只有在交互 TLS 驗證失敗時才會顯示在存取記錄檔中。 |
| $context.identity.clientCert.issuerDN | 用戶端提供之憑證發行者的辨別名稱。當用戶端使用已啟用交互 TLS 的自訂網域名稱存取 API 時會顯示。只有在交互 TLS 驗證失敗時才會顯示在存取記錄檔中。 |
| $context.identity.clientCert.serialNumber | 憑證的序號。當用戶端使用已啟用交互 TLS 的自訂網域名稱存取 API 時會顯示。只有在交互 TLS 驗證失敗時才會顯示在存取記錄檔中。 |
| $context.identity.clientCert.validity.notBefore | 憑證無效之前的日期。當用戶端使用已啟用交互 TLS 的自訂網域名稱存取 API 時會顯示。只有在交互 TLS 驗證失敗時才會顯示在存取記錄檔中。 |
| $context.identity.clientCert.validity.notAfter | 憑證無效之後的日期。當用戶端使用已啟用交互 TLS 的自訂網域名稱存取 API 時會顯示。只有在交互 TLS 驗證失敗時才會顯示在存取記錄檔中。 |
| $context.identity.vpcId | 對 API Gateway 端點提出請求之 VPC 的 VPC ID。 |
| $context.identity.vpceId | 對 API Gateway 端點提出請求之 VPC 端點的 VPC 端點 ID。只會在您擁有私有 API 時顯示。 |

| 參數 | 說明 |
| --- | --- |
| `$context.identity.user` | 將針對資源存取授權之使用者的主體識別符。支援使用 IAM 授權的資源。 |
| `$context.identity.userAgent` | API 發起人的 `User-Agent` 標頭。 |
| `$context.identity.userArn` | 身分驗證之後識別之有效使用者的 Amazon Resource Name (ARN)。如需詳細資訊，請參閱https://docs.aws.amazon.com/IAM/latest/UserGuide/id_users.html。 |
| `$context.isCanaryRequest` | 如果請求導向 Canary，則傳回 `true`；如果請求未導向 Canary，則傳回 `false`。只會在啟用 Canary 時顯示。 |
| `$context.path` | 請求路徑。例如，對於 `https://{rest-api-id}.execute-api.{region}.amazonaws.com/{stage}/root/child` 的非代理請求 URL，`$context.path` 值是 `/{stage}/root/child`。 |
| `$context.protocol` | 請求通訊協定，例如 `HTTP/1.1`。<br><br>**Note**<br>API Gateway API 可以接受 HTTP/2 請求，但 API Gateway 會使用 HTTP/1.1 將請求傳送至後端整合。因此，即使用戶端傳送使用 HTTP/2 的請求，請求通訊協定也會記錄為 HTTP/1.1。 |
| `$context.requestId` | 請求的 ID。用戶端可以覆寫此請求 ID。使用 `$context.extendedRequestId` 即可取得 API Gateway 產生的唯一請求 ID。 |

| 參數 | 說明 |
| --- | --- |
| `$context.requestOverride.header. header_name` | 請求標題會覆寫。如果此參數已經定義，它包含要使用的標頭 (而不是在 Integration Request (整合請求) 窗格中定義的 HTTP Headers (HTTP 標頭))。如需詳細資訊，請參閱 覆寫 API Gateway 中 REST API 的 API 請求和回應參數及狀態碼。 |
| `$context.requestOverride.path. path_name` | 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 覆寫 API Gateway 中 REST API 的 API 請求和回應參數及狀態碼。 |
| `$context.requestOverride.querystring. querystring_name` | 請求查詢字串會覆寫。如果此參數已經定義，它包含要使用的請求查詢字串 (而不是在 Integration Request (整合請求) 窗格中定義的 URL Query String (URL 查詢字串))。如需詳細資訊，請參閱 覆寫 API Gateway 中 REST API 的 API 請求和回應參數及狀態碼。 |
| `$context.responseOverride.header. header_name` | 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 覆寫 API Gateway 中 REST API 的 API 請求和回應參數及狀態碼。 |
| `$context.responseOverride.status` | 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 覆寫 API Gateway 中 REST API 的 API 請求和回應參數及狀態碼。 |

| 參數 | 說明 |
| --- | --- |
| $context.requestTime | CLF 格式化請求時間 (dd/MMM/yyyy:HH:mm:ss +-hhmm )。 |
| $context.requestTimeEpoch | Epoch 格式化的要求時間，以毫秒為單位。 |
| $context.resourceId | API Gateway 指派給您的資源的識別碼。 |
| $context.resourcePath | 您資源的路徑。例如，對於非代理請求 URI https://{rest-api-id}.execute-api.{region}.amazonaws.com/{stage}/root/child ，$context.resourcePath 值是 /root/child 。如需詳細資訊，請參閱 教學：建立具有 HTTP 非代理整合的 REST API。 |
| $context.stage | API 請求的部署階段 (例如，Beta 或 Prod)。 |
| $context.wafResponseCode | 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 the section called “AWS WAF”。 |
| $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the section called “AWS WAF”。 |

輸入變數

您可以使用下列區分大小寫的 $input 變數來參考方法請求承載和方法請求參數。可使用以下函數：

| 變數和函數 | 描述 |
| --- | --- |
| $input.body | 傳回原始請求承載做為字串。您可以使用 $input.body 保留整個浮點數，例如 10.00。 |

| 變數和函數 | 描述 |
| --- | --- |
| `$input.json(x)` | 此函數會評估 JSONPath 表達式，並傳回結果作為 JSON 字串。<br><br>例如，`$input.json('$.pets')` 會傳回代表 `pets` 結構的 JSON 字串。<br><br>如需 JSONPath 的詳細資訊，請參閱 JSONPath 或適用於 Java 的 JSONPath。 |
| `$input.params()` | 傳回所有請求參數的映射。我們建議您使用 `$util.escapeJavaScript` 清理結果，以避免潛在的注入攻擊。為了完全控制請求清理，請使用沒有模板的代理整合，並處理整合中的請求清理。 |
| `$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。我們建議您使 `$util.escapeJavaScript` 用清理參數，以避免潛在的注入攻擊。為了完全控制參數清理，請使用沒有模板的代理整合，並處理整合中的請求清理。 |

| 變數和函數 | 描述 |
| --- | --- |
| `$input.path(x)` | 採用 JSONPath 表達式字串 (x)，並傳回結果的 JSON 物件呈現。這可讓您存取和運用 Apache Velocity 範本語言 (VTL) 中原生承載的元素。<br><br>例如，如果表達式 `$input.path('$.pets')` 傳回如下物件：<br><br>`[ { "id": 1, "type": "dog", "price": 249.99 }, { "id": 2, "type": "cat", "price": 124.99 }, { "id": 3, "type": "fish", "price": 0.99 } ]`<br><br>`$input.path('$.pets').size()` 會傳回 "3"。<br><br>如需 JSONPath 的詳細資訊，請參閱 JSONPath 或適用於 Java 的 JSONPath。 |

### 階段變數

您可以使用下列階段變數作為方法整合中 ARN 和 URL 的預留位置。如需詳細資訊，請參閱the section called “使用階段變數”。

| 語法 | 說明 |
| --- | --- |
| `$stageVariables.` *variable_name* 、`$stageVariables['` *variable_name* `']` 或 `${stageVariables['` *variable_name* `']}` | *variable_name* 代表階段變數名稱。 |

Util 變數

您可以使用下列區分大小寫的 $util 變數，針對映射範本使用公用程式函數。除非特別指定，否則預設字元集為 UTF-8。

| 函數 | 描述 |
| --- | --- |
| `$util.escapeJavaScript()` | 使用 JavaScript 字串規則來逸出字串中的字元。<br><br>**Note**<br>此函數會將任何一般單引號 (') 轉換為逸出單引號 (\')。不過，逸出單引號不適用於 JSON。因此，將此函數的輸出用於 JSON 屬性時，您必須將任何逸出單引號 (\') 轉換為一般單引號 (')。下列範例顯示這種情況：<br><br>`"input" : "$util.escapeJavaScript( data).replaceAll("\\'",'')"` |
| `$util.parseJson()` | 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： |

| 函數 | 描述 |
| --- | --- |
| | `{"errorMessage":"{\"key1\":\"var1\", \"key2\":{\"arr\":[1,2,3]}}"}`<br><br>並使用下列映射範本<br><br>`#set ($errorMessageObj = $util.parseJson($input.path('$.errorMessage')))`<br>`{`<br>`  "errorMessageObjKey2ArrVal" : $errorMessageObj.key2.arr[0]`<br>`}`<br><br>您將會收到下列輸出：<br><br>`{`<br>`  "errorMessageObjKey2ArrVal" : 1`<br>`}` |
| `$util.urlEncode()` | 將字串轉換為 "application/x-www-form-urlencoded" 格式。 |
| `$util.urlDecode()` | 解碼 "application/x-www-form-urlencoded" 字串。 |
| `$util.base64Encode()` | 將資料編碼為 base64 編碼字串。 |
| `$util.base64Decode()` | 解碼 base64 編碼字串中的資料。 |

## API Gateway 中 REST API 的閘道回應

閘道回應以 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，閘道回應以 GatewayResponse 表示。在 OpenAPI 中，`GatewayResponse` 執行個體是以 x-amazon-apigateway-gateway-responses.gatewayResponse 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映射與承載映射範本，將映射的結果傳回給 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。

## 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 在外部交換和 AWS 雲端之間協調時，您可以使用 VTL 映射範本進行整合請求或整合回應，將承載從一個格式映射到另一個格式。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 `$context` 變數的詳細資訊，請參閱「資料轉換的內容變數」。如需有關 `$stageVariables` 的詳細資訊，請參閱 階段變數。如需有關方法請求參數的詳細資訊，請參閱 the section called “輸入變數”。

主題

- 使用 API Gateway 主控台設定 REST API 的閘道回應
- 使用 API Gateway REST API 設定閘道回應
- 設定 OpenAPI 中的閘道回應自訂
- API Gateway 的閘道回應類型

## 使用 API Gateway 主控台設定 REST API 的閘道回應

下列範例顯示如何使用 API Gateway 主控台設定 REST API 的閘道回應

使用 API Gateway 主控台自訂閘道回應

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 選擇 REST API。
3. 在主導覽窗格中，選擇閘道資源。
4. 選擇回應類型，然後選擇編輯。在此逐步說明中，我們使用遺漏身分驗證權杖作為範例。
5. 您可以變更 API Gateway 所產生的狀態碼，以傳回符合您的 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，請在回應標頭下，選擇新增標頭。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. 在回應範本下，將內容類型保持為 `application/json`，然後在範本內文編輯器中輸入下列內文對應範本：

   ```
   {
        "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. 選擇儲存變更。

9. 將 API 部署到新的或現有的階段。

透過呼叫下列 CURL 命令進行閘道回應測試，並假設映射 API 方法的調用 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，因此會從指定的閘道回應傳回錯誤。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: 123344566
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 441811a054e8d055b893175754efd0c3.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，並在請求酬載中提供 statusCode、responseParameters 及 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您需要的個別 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     \"statusCode\": \"'404'\"\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。

## API Gateway 的閘道回應類型

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 |
| --- | --- | --- |
| 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，或整合回應映射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 | 500 | 無法連線到自訂或 Amazon Cognito 授權方時的閘道回應。如果未指定回應類型，此回應預設為 DEFAULT_5XX 類型。 |
| 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 |

| 閘道回應類型 | 預設狀態碼 | 描述 |
| --- | --- | --- |
| | | 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。 |
| 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 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。<br><br>Note<br>AWS WAF 自訂回應優先於自訂閘道回應。 |
| 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。 |
| 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。 |

| 閘道回應類型 | 預設狀態碼 | 描述 |
| --- | --- | --- |
| 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。 |
| 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 | 無效 AWS 簽章錯誤的閘道回應。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：HTTP content length exceeded 10485760 bytes。 |

| 閘道回應類型 | 預設狀態碼 | 描述 |
| --- | --- | --- |
| 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。 |
| WAF_FILTERED | 403 | 請求遭 AWS WAF封鎖時所出現的閘道回應。如果未指定回應類型，此回應預設為 DEFAULT_4XX 類型。<br><br>Note<br>AWS WAF 自訂回應優先於自訂閘道回應。 |

# API Gateway 中 REST API 的 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。如需詳細資訊，請參閱什麼是 CORS？。

## 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`)

如果您無法存取 API 並收到包含 `Cross-Origin Request Blocked` 的錯誤訊息，您可能需要啟用 CORS。

跨來源 HTTP 請求可分為兩種類型：簡單請求和非簡單請求。

## 針對簡單請求啟用 CORS

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含標頭 `Access-Control-Allow-Origin: '*'` 或 `Access-Control-Allow-Origin:'origin'`。

所有其他跨來源 HTTP 請求都是非簡單請求。

## 針對非簡單請求啟用 CORS

如果您的 API 資源收到非簡單請求，則您必須啟用其他 CORS 支援，取決於您的整合類型。

### 針對非代理整合啟用 CORS

對於這些整合，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 (或請求憑證)，然後再傳送實際請求。您必須設定 API，才能將適當的回應傳送至預檢請求。

若要建立預檢回應：

1. 建立具有模擬整合的 OPTIONS 方法。
2. 將下列回應標頭新增至 200 方法回應：
   - `Access-Control-Allow-Headers`
   - `Access-Control-Allow-Methods`
   - `Access-Control-Allow-Origin`
3. 將整合傳遞行為設定為 NEVER。在此案例中，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。如需詳細資訊，請參閱API Gateway 中 REST API 沒有映射範本時，承載的方法請求行為。
4. 輸入回應標頭的值。若要允許所有來源、所有方法和常見的標頭，請使用下列標頭值：
   - `Access-Control-Allow-Headers: 'Content-Type,X-Amz-Date,Authorization,X-Api-Key,X-Amz-Security-Token'`
   - `Access-Control-Allow-Methods: 'DELETE,GET,HEAD,OPTIONS,PUT,POST,PATCH'`
   - `Access-Control-Allow-Origin: '*'`

在建立預檢請求之後，至少針對所有 200 回應，您必須為所有已啟用 CORS 的方法傳回 `Access-Control-Allow-Origin: '*'` 或 `Access-Control-Allow-Origin:'origin'` 標頭。

### 使用 為非代理整合啟用 CORS AWS 管理主控台

您可以使用 AWS 管理主控台 來啟用 CORS。API Gateway 會建立 OPTIONS 方法，並將 `Access-Control-Allow-Origin` 標頭新增至現有方法的整合回應。這並不一定可行，有時您需要手動修改整合回應，至少針對所有 200 回應，為所有已啟用 CORS 的方法傳回 `Access-Control-Allow-Origin` 標題。

如果您將 API 的二進位媒體類型設定為 `*/*`，當 API Gateway 建立 OPTIONS 方法時，請將 `contentHandling` 變更為 `CONVERT_TO_TEXT`。

以下 update-integration 命令會將整合請求的 `contentHandling` 變更為 `CONVERT_TO_TEXT`：

```
aws apigateway update-integration \
```

```
  --rest-api-id abc123 \
  --resource-id aaa111 \
  --http-method OPTIONS \
  --patch-operations op='replace',path='/contentHandling',value='CONVERT_TO_TEXT'
```

以下 update-integration-response 命令會將整合回應的 contentHandling 變更為 CONVERT_TO_TEXT：

```
aws apigateway update-integration-response \
  --rest-api-id abc123 \
  --resource-id aaa111 \
  --http-method OPTIONS \
  --status-code 200 \
  --patch-operations op='replace',path='/contentHandling',value='CONVERT_TO_TEXT'
```

## 針對代理整合啟用 CORS 支援

對於 Lambda 代理整合或 HTTP 代理整合，您的後端負責傳回 Access-Control-Allow-Origin、Access-Control-Allow-Methods 和 Access-Control-Allow-Headers 標頭，因為代理整合不會傳回整合回應。

下列範例 Lambda 函數會傳回必要的 CORS 標頭：

Node.js

```
export const handler = async (event) => {
    const response = {
        statusCode: 200,
        headers: {
            "Access-Control-Allow-Headers" : "Content-Type",
            "Access-Control-Allow-Origin": "https://www.example.com",
            "Access-Control-Allow-Methods": "OPTIONS,POST,GET"
        },
        body: JSON.stringify('Hello from Lambda!'),
    };
    return response;
};
```

Python 3

```
import json
```

```
def lambda_handler(event, context):
    return {
        'statusCode': 200,
        'headers': {
            'Access-Control-Allow-Headers': 'Content-Type',
            'Access-Control-Allow-Origin': 'https://www.example.com',
            'Access-Control-Allow-Methods': 'OPTIONS,POST,GET'
        },
        'body': json.dumps('Hello from Lambda!')
    }
```

主題

- 使用 API Gateway 主控台在資源上啟用 CORS
- 使用 API Gateway 匯入 API 在資源上啟用 CORS
- 測試 API Gateway API 的 CORS

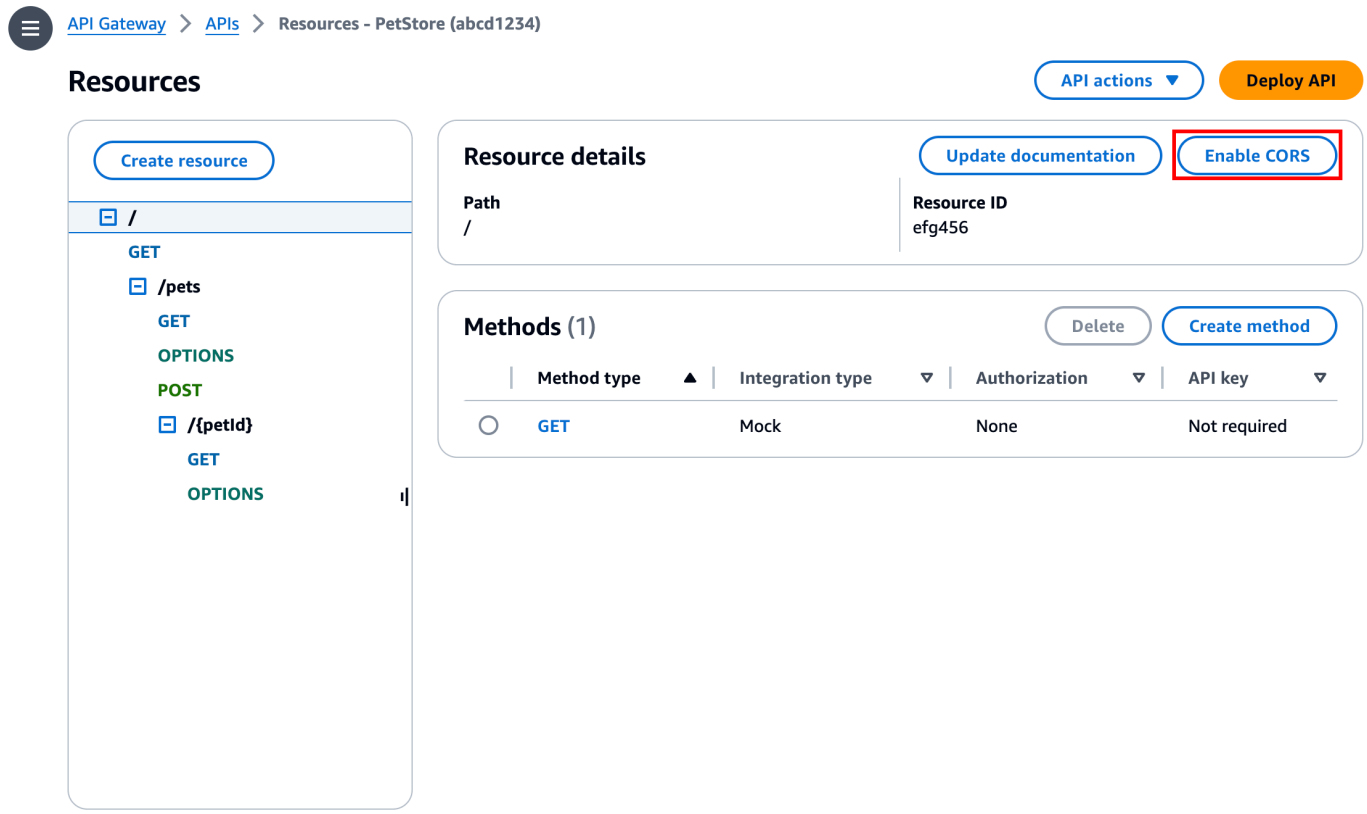
## 使用 API Gateway 主控台在資源上啟用 CORS

您可以使用 API Gateway 主控台，為您已建立的 REST API 資源上的一個或所有方法啟用 CORS 支援。啟用 COR 支援後，將整合傳遞行為設定為 NEVER。在此案例中，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。如需詳細資訊，請參閱API Gateway 中 REST API 沒有映射範本時，承載的方法請求行為

> **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。

若要在 REST API 資源上啟用 CORS 支援

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 選擇一個 API。
3. 在資源下，選擇一個資源。
4. 在資源詳細資訊區段中，選擇啟用 CORS。




5. 在啟用 CORS 方塊中，執行下列操作：

   a. (選用) 如果您已建立自訂閘道回應，並且想要針對回應啟用 CORS 支援，請選取閘道回應。

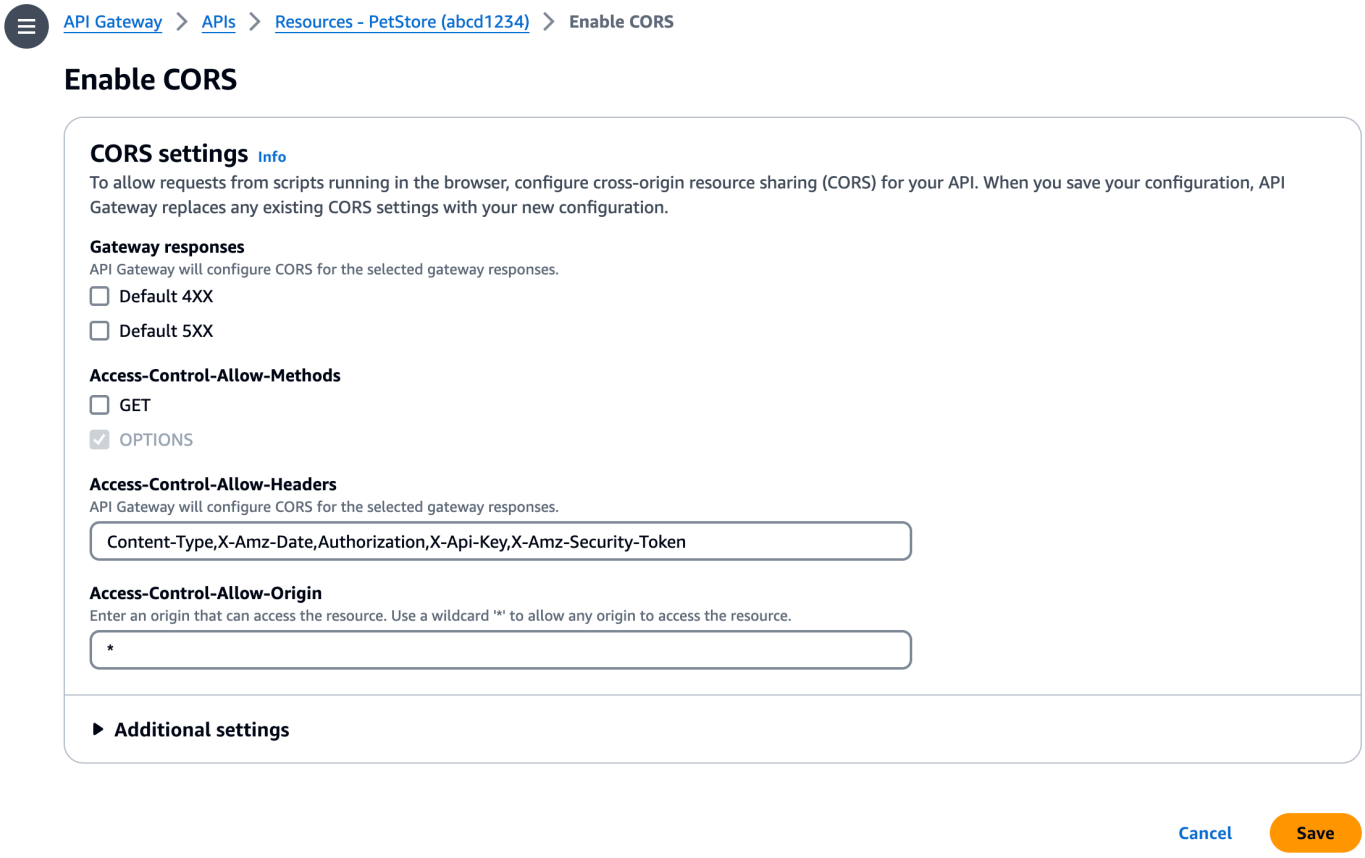
   b. 選取每一種方法以啟用 CORS 支援。OPTION 方法必須啟用 CORS。

      如果您為 ANY 方法啟用 CORS 支援，則所有方法的 CORS 都會啟用。

   c. 在 Access-Control-Allow-Headers 輸入欄位中，輸入逗號分隔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。

   d. 使用主控台提供的 `'*'` 值以做為 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起源。

   e. 選擇儲存。




> **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`。

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，以在其 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 “建立部署”。

> **Note**
>
> 如果您在遵循程序後無法在資源上啟用 CORS 支援，建議您比較 CORS 組態與範例 API `/pets` 資源。若要了解如何建立範例 API，請參閱 the section called “教學課程：匯入範例來建立 REST API”。

## 使用 API Gateway 匯入 API 在資源上啟用 CORS

如果您使用 API Gateway 匯入 API，即可透過 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。

### 示例 **`Options`** 方法

下列範例為模擬整合建立 `OPTIONS` 方法。

OpenAPI 3.0

```
/users:
  options:
    summary: CORS support
    description: |
      Enable CORS by returning correct headers
    tags:
    - CORS
    responses:
      200:
        description: Default response for CORS method
        headers:
          Access-Control-Allow-Origin:
            schema:
              type: "string"
          Access-Control-Allow-Methods:
            schema:
```

```
              type: "string"
          Access-Control-Allow-Headers:
            schema:
              type: "string"
        content: {}
    x-amazon-apigateway-integration:
      type: mock
      requestTemplates:
        application/json: "{\"statusCode\": 200}"
      passthroughBehavior: "never"
      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: "'*'"
```

## OpenAPI 2.0

```
/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: "{\"statusCode\": 200}"
        passthroughBehavior: "never"
        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 : "'*'"
    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。

   OpenAPI 3.0

   ```
     responses:
       200:
         description: Default response for CORS method
         headers:
           Access-Control-Allow-Origin:
             schema:
               type: "string"
           Access-Control-Allow-Methods:
             schema:
               type: "string"
           Access-Control-Allow-Headers:
             schema:
               type: "string"
         content: {}
   ```

   OpenAPI 2.0

   ```
     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：

   OpenAPI 3.0

```
    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: |
              {}
```

   OpenAPI 2.0

```
    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 : "'*'"
```

### API 範例

下列範例會建立具有 HTTP 整合 OPTIONS 方法和 GET 方法的完整 API。

OpenAPI 3.0

```
openapi: "3.0.1"
info:
```

```
  title: "cors-api"
  description: "cors-api"
  version: "2024-01-16T18:36:01Z"
servers:
- url: "/{basePath}"
  variables:
    basePath:
      default: "/test"
paths:
  /:
    get:
      operationId: "GetPet"
      responses:
        "200":
          description: "200 response"
          headers:
            Access-Control-Allow-Origin:
              schema:
                type: "string"
          content: {}
      x-amazon-apigateway-integration:
        httpMethod: "GET"
        uri: "http://petstore.execute-api.us-east-1.amazonaws.com/petstore/pets"
        responses:
          default:
            statusCode: "200"
            responseParameters:
              method.response.header.Access-Control-Allow-Origin: "'*'"
        passthroughBehavior: "never"
        type: "http"
    options:
      responses:
        "200":
          description: "200 response"
          headers:
            Access-Control-Allow-Origin:
              schema:
                type: "string"
            Access-Control-Allow-Methods:
              schema:
                type: "string"
            Access-Control-Allow-Headers:
              schema:
                type: "string"
```

```
          content:
            application/json:
              schema:
                $ref: "#/components/schemas/Empty"
      x-amazon-apigateway-integration:
        responses:
          default:
            statusCode: "200"
            responseParameters:
              method.response.header.Access-Control-Allow-Methods: "'GET,OPTIONS'"
              method.response.header.Access-Control-Allow-Headers: "'Content-Type,X-
Amz-Date,Authorization,X-Api-Key'"
              method.response.header.Access-Control-Allow-Origin: "'*'"
        requestTemplates:
          application/json: "{\"statusCode\": 200}"
        passthroughBehavior: "never"
        type: "mock"
components:
  schemas:
    Empty:
      type: "object"
```

## OpenAPI 2.0

```
swagger: "2.0"
info:
  description: "cors-api"
  version: "2024-01-16T18:36:01Z"
  title: "cors-api"
basePath: "/test"
schemes:
- "https"
paths:
  /:
    get:
      operationId: "GetPet"
      produces:
      - "application/json"
      responses:
        "200":
          description: "200 response"
          headers:
            Access-Control-Allow-Origin:
```

```
              type: "string"
      x-amazon-apigateway-integration:
        httpMethod: "GET"
        uri: "http://petstore.execute-api.us-east-1.amazonaws.com/petstore/pets"
        responses:
          default:
            statusCode: "200"
            responseParameters:
              method.response.header.Access-Control-Allow-Origin: "'*'"
        passthroughBehavior: "never"
        type: "http"
    options:
      consumes:
      - "application/json"
      produces:
      - "application/json"
      responses:
        "200":
          description: "200 response"
          schema:
            $ref: "#/definitions/Empty"
          headers:
            Access-Control-Allow-Origin:
              type: "string"
            Access-Control-Allow-Methods:
              type: "string"
            Access-Control-Allow-Headers:
              type: "string"
      x-amazon-apigateway-integration:
        responses:
          default:
            statusCode: "200"
            responseParameters:
              method.response.header.Access-Control-Allow-Methods: "'GET,OPTIONS'"
              method.response.header.Access-Control-Allow-Headers: "'Content-Type,X-
Amz-Date,Authorization,X-Api-Key'"
              method.response.header.Access-Control-Allow-Origin: "'*'"
        requestTemplates:
          application/json: "{\"statusCode\": 200}"
        passthroughBehavior: "never"
        type: "mock"
definitions:
  Empty:
```

```
    type: "object"
```

### 測試 API Gateway API 的 CORS

您可以透過叫用 API 來測試 API 的 CORS 組態，並在回應中檢查 CORS 標頭。下面的 `curl` 命令會傳送 OPTIONS 請求至已部署的 API。

```
curl -v -X OPTIONS https://{restapi_id}.execute-api.{region}.amazonaws.com/{stage_name}
```

```
< HTTP/1.1 200 OK
< Date: Tue, 19 May 2020 00:55:22 GMT
< Content-Type: application/json
< Content-Length: 0
< Connection: keep-alive
< x-amzn-RequestId: a1b2c3d4-5678-90ab-cdef-abc123
< Access-Control-Allow-Origin: *
< Access-Control-Allow-Headers: Content-Type,Authorization,X-Amz-Date,X-Api-Key,X-Amz-
Security-Token
< x-amz-apigw-id: Abcd=
< Access-Control-Allow-Methods: DELETE,GET,HEAD,OPTIONS,PATCH,POST,PUT
```

回應中的 `Access-Control-Allow-Origin`、`Access-Control-Allow-Headers` 和 `Access-Control-Allow-Methods` 標頭會顯示 API 支援 CORS。如需詳細資訊，請參閱API Gateway 中 REST API 的 CORS。

## API Gateway 中 REST API 的二進位媒體類型

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。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。支援二進位媒體所需的 API 組態取決於您的 API 是否使用代理或非代理整合。

如果您使用代理整合與承載回應串流，則不需要設定二進位媒體類型。如需詳細資訊，請參閱the section called “串流代理整合的整合回應”。

### AWS Lambda 代理整合

若要處理 AWS Lambda 代理整合的二進位承載，您必須對函數的回應進行 base64 編碼。您也必須為 API 配置 binaryMediaTypes。您的 API `binaryMediaTypes` 組態是 API 視為二進位資料的內容類型列表。範例二進位媒體類型包括 `image/png` 或 `application/octet-stream`。您可以使用萬用字元 (*) 來涵蓋多種媒體類型。

API Gateway 會使用來自用戶端的第一個 Accept 標頭來判斷回應是否應該傳回二進位媒體。若要在無法控制 Accept 標頭值的順序 (例如來自瀏覽器的請求) 時傳回二進位媒體，請將 API 的二進位媒體類型設定為 */*。

如需範例程式碼，請參閱 the section called “在 API Gateway 中從 Lambda 代理整合傳回二進位媒體”。

如果您使用 Lambda 代理整合與承載回應串流，則不需要設定二進位媒體類型。如需詳細資訊，請參閱the section called “設定 Lambda 代理整合與承載回應串流”。

### 非代理伺服器整合

若要處理非代理整合的二進位承載，您可以將媒體類型新增至 RestApi 資源的 binaryMediaTypes 清單。您的 API binaryMediaTypes 組態是 API 視為二進位資料的內容類型列表。或者，您可以在 Integration 與 IntegrationResponse 資源上設定 contentHandling 屬性。contentHandling 值可以是 CONVERT_TO_BINARY、CONVERT_TO_TEXT 或未定義。

> **Note**
>
> 若是 MOCK 或私有整合，不支援在 AWS 管理主控台中設定 contentHandling 屬性。您必須使用 AWS CLI CloudFormation、 或 SDK 來設定contentHandling屬性。

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 資源上的 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 屬性定址。
  - 若要讓請求承載從 base64 編碼的字串轉換為其二進位 Blob，請將屬性設定為 CONVERT_TO_BINARY。
  - 若要將請求承載從二進位 Blob 轉換為 base64 編碼的字串，請將屬性設定為 CONVERT_TO_TEXT。
  - 若要在不修改的情況下傳遞承載，請將屬性保留為未定義。若要在未經修改的情況下傳遞二進位承載，您也必須確定 Content-Type 符合其中一個 binaryMediaTypes 項目，並且已針對 API 啟用傳遞行為。

- 設定 IntegrationResponse 資源的 contentHandling 屬性。contentHandling 屬性、用戶端請求中的 Accept 標頭，以及 API 的 binaryMediaTypes 組合會決定 API Gateway 處理內容類型轉換的方式。如需詳細資訊，請參閱the section called “API Gateway 中的內容類型轉換”。

> **Important**
>
> 當請求的 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 binaryMediaTypes 清單中新增第一個 Accept 媒體類型。API Gateway 將以二進位處理此清單中的所有內容類型。
> 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，端點就能收到二進位格式的 JPEG 檔案。

如需 API Gateway 如何處理文字和二進位承載的詳細資訊，請參閱API Gateway 中的內容類型轉換。

## API Gateway 中的內容類型轉換

您的 API 的 binaryMediaTypes、用戶端請求中的標頭，以及整合 contentHandling 屬性的組合將決定 API Gateway 編碼有效承載的方式。

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酬載。

| 方法請求承載 | 請求 **Content-Type** 標頭 | **binaryMediaTypes** | **contentHandling** | 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 | CONVERT_TO_BINARY | Base64 解碼的二進位 Blob |
| 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 |

| 方法請求承載 | 請求 **Content-Type** 標頭 | **binaryMediaTypes** | **contentHandling** | 整合請求承載 |
|---|---|---|---|---|
| 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酬載。

> **Important**
>
> 當請求的 `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 `binaryMediaTypes` 清單中新增第一個 `Accept` 媒體類型。API Gateway 將以二進位處理此清單中的所有內容類型。
>
> 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，端點就能收到二進位格式的 JPEG 檔案。

| 整合回應承載 | 請求 **Accept** 標頭 | **binaryMediaTypes** | **contentHandling** | 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 |

| 整合回應承載 | 請求 **Accept** 標頭 | **binaryMediaTypes** | **contentHandling** | 方法回應承載 |
|---|---|---|---|---|
| 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 |

| 整合回應承載 | 請求 **Accept** 標頭 | **binaryMediaTypes** | **contentHandling** | 方法回應承載 |
|---|---|---|---|---|
| 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 |

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。

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。

## 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：建立 REST API 做為 Amazon S3 代理。

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 API 設定。
    c. 在 API 設定窗格的二進位媒體類型區段中，選擇管理媒體類型。
    d. 選擇新增二進位媒體類型。

e. 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。若要支援所有二進位媒體類型，請指定 `*/*`。

f. 選擇儲存變更。

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 `GET /{folder}/{item}` 動作整合的 `Object GET` 方法。

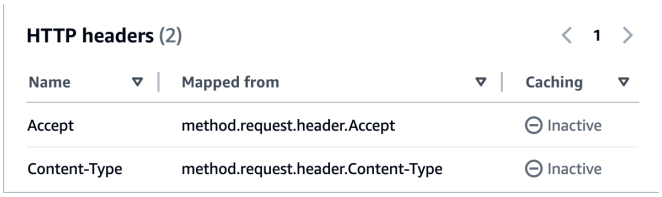
c. 針對內容處理，選擇一個選項。

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，選擇轉換為文字，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，選擇轉換為二進位。

d. 針對請求內文傳遞，選擇未定義範本時 (建議)以在請求內文上啟用傳遞行為。

您也可以選擇永不。這意味著 API 將拒絕 content-types 沒有對應範本的資料。

e. 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。




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。

以下是要轉換為文字的對應範本範例：

```
{
  "operation": "thumbnail",
  "base64Image": "$input.body"
}
```

此對應範本的格式取決於輸入的端點需求。

g. 選擇儲存。

## 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
- 設定請求承載轉換
- 設定回應承載轉換
- 將二進位資料轉換成文字資料
- 將文字資料轉換成二進位承載
- 傳遞二進位承載

### 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，請將 `RestApi` 請求提交至 `PATCH` 資源：

```
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，將 IntegrationResponse 資源的 contentHandling 屬性設定為 CONVERT_TO_BINARY。若要這樣做，請提交 PATCH 請求，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 將二進位資料做為輸入 AWS Lambda 或 Kinesis 的 JSON 屬性傳送，請執行下列動作：

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~1octet-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~1octet-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~1octet-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。如果您未定義映射範本，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 將映像儲存在 Amazon S3 儲存貯體中，請執行下列動作：

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~1octet-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 叫用傳遞範本。

```
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"
    }
  ]
}
```

## API Gateway 的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

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
- x-amazon-apigateway-integration.response 物件

## 在 API Gateway 中從 Lambda 代理整合傳回二進位媒體

要從 AWS Lambda 代理整合傳回二進位媒體，您需要對 Lambda 函數的回應進行 base64 編碼。您還必須配置 API 的二進位媒體類型。當您設定 API 的二進位媒體類型時，API 會將該內容類型視為二進位資料。承載大小上限為 10 MB。

> **Note**
>
> 若要使用網頁瀏覽器來叫用具有此範例整合的 API，請將 API 的二進位媒體類型設定為 `*/*`。API Gateway 會使用來自用戶端的第一個 `Accept` 標頭來判斷回應是否應該傳回二進位媒體。若要在無法控制 `Accept` 標頭值的順序 (例如來自瀏覽器的請求) 時傳回二進位媒體，請將 API 的二進位媒體類型設定為 `*/*` (適用於所有內容類型)。

下列範例 Lambda 函數可將二進位映像從 Amazon S3 或文字傳回給用戶端。該函數的回應會包括一個 `Content-Type` 標頭，以指示給用戶端它傳回的資料類型。該函數會有條件地在其回應中設定 `isBase64Encoded` 屬性，具體取決於它傳回的資料類型。

### Node.js

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

const client = new S3Client({region: 'us-east-2'});

export const handler = async (event) => {

  var randomint = function(max) {
    return Math.floor(Math.random() * max);
  }
  var number = randomint(2);
  if (number == 1){
    const input = {
      "Bucket" : "bucket-name",
      "Key" : "image.png"
      }
    try {
      const command = new GetObjectCommand(input)
      const response = await client.send(command);
      var str = await response.Body.transformToByteArray();
    } catch (err) {
      console.error(err);
    }
    const base64body = Buffer.from(str).toString('base64');
    return {
      'headers': { "Content-Type": "image/png" },
      'statusCode': 200,
      'body': base64body,
      'isBase64Encoded': true
      }
    } else {
        return {
        'headers': { "Content-Type": "text/html" },
        'statusCode': 200,
        'body': "<h1>This is text</h1>",
        }
    }
}
```

### Python

```
import base64
```

```
import boto3
import json
import random

s3 = boto3.client('s3')

def lambda_handler(event, context):
    number = random.randint(0,1)
    if number == 1:
        response = s3.get_object(
            Bucket='bucket-name',
            Key='image.png',
        )
        image = response['Body'].read()
        return {
            'headers': { "Content-Type": "image/png" },
            'statusCode': 200,
            'body': base64.b64encode(image).decode('utf-8'),
            'isBase64Encoded': True
        }
    else:
        return {
            'headers': { "Content-type": "text/html" },
            'statusCode': 200,
            'body': "<h1>This is text</h1>",
        }
```

若要進一步了解二進位媒體類型，請參閱API Gateway 中 REST API 的二進位媒體類型。

## 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
- 從 Amazon S3 下載影像
- 將影像上傳至 Amazon S3

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，後面接著所提供的映射範本。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

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，若要從 Amazon S3 下載 Base64 編碼字串 (格式化為 JSON 屬性) 格式的影像檔 (image.jpg)，請將回應範本新增至 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

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
```

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=
```

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 OpenAPI 範例示範如何 AWS Lambda 透過 API Gateway API 存取 中的二進位檔案。此 API 會公開用於下載及上傳指定影像檔的 GET /lambda?key={file-name} 與 PUT /lambda?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，後面接著所提供的映射範本。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。

您建立 API 呼叫的 Lambda 函數，而該函數必須傳回具有 application/json 的 Content-Type 標頭的 base64 編碼字串。

主題

- 存取 Lambda 中影像之範例 API 的 OpenAPI 檔案
- 從 Lambda 下載影像
- 將影像上傳至 Lambda

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": "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": {
              "$ref": "#/definitions/Empty"
            }
          },
          "500": {
            "description": "500 response"
          }
        },
        "x-amazon-apigateway-integration": {
          "uri": "arn:aws:apigateway:us-east-1:lambda:path/2015-03-31/functions/
arn:aws:lambda:us-east-1:123456789012:function:image/invocations",
          "type": "AWS",
          "credentials": "arn:aws:iam::123456789012:role/Lambda",
          "httpMethod": "POST",
          "contentHandling" : "CONVERT_TO_TEXT",
          "requestTemplates": {
            "application/json": "{\n   \"imageKey\": \"$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"],
  "definitions": {
    "Empty": {
      "type": "object",
      "title": "Empty Schema"
    }
  }
```

```
}
```

從 Lambda 下載影像

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]
```

從 Lambda 下載 Base64 編碼字串格式的影像檔 (image.jpg) 並格式化為 JSON 屬性：

```
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": "W3JhdyBieXRlc10="
}
```

將影像上傳至 Lambda

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
```

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

W3JhdyBieXRlc10=
```

成功回應如下所示：

```
200 OK
```

## 在 API Gateway 中調用 REST API

若要呼叫已部署的 API，用戶端會將請求提交至 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://api-id.execute-api.region.amazonaws.com/stage/
```

其中 *api-id* 是 API 識別碼，*region* 是 AWS 區域，而 *stage* 是 API 部署的階段名稱。

> **Important**
>
> 在您可以叫用 API 之前，您必須先在 API Gateway 中部署 API。如需有關部署 API 的指示，請參閱在 API Gateway 中部署 REST API。

主題

- 取得 API 的調用 URL
- 呼叫 API

- 使用 API Gateway 主控台來測試 REST API 方法
- 使用由 API Gateway 產生的 Java 軟體開發套件來執行 REST API
- 使用由 API Gateway 為 REST API 產生的 Android 軟體開發套件
- 使用 API Gateway 為 REST API 所產生的 JavaScript 軟體開發套件
- 使用由 API Gateway 為 REST API 產生的 Ruby 軟體開發套件
- 在 Objective-C 或 Swift 中使用 API Gateway 為 REST API 產生的 iOS 軟體開發套件

## 取得 API 的調用 URL

您可以使用主控台 AWS CLI，或匯出的 OpenAPI 定義來取得 API 的調用 URL。

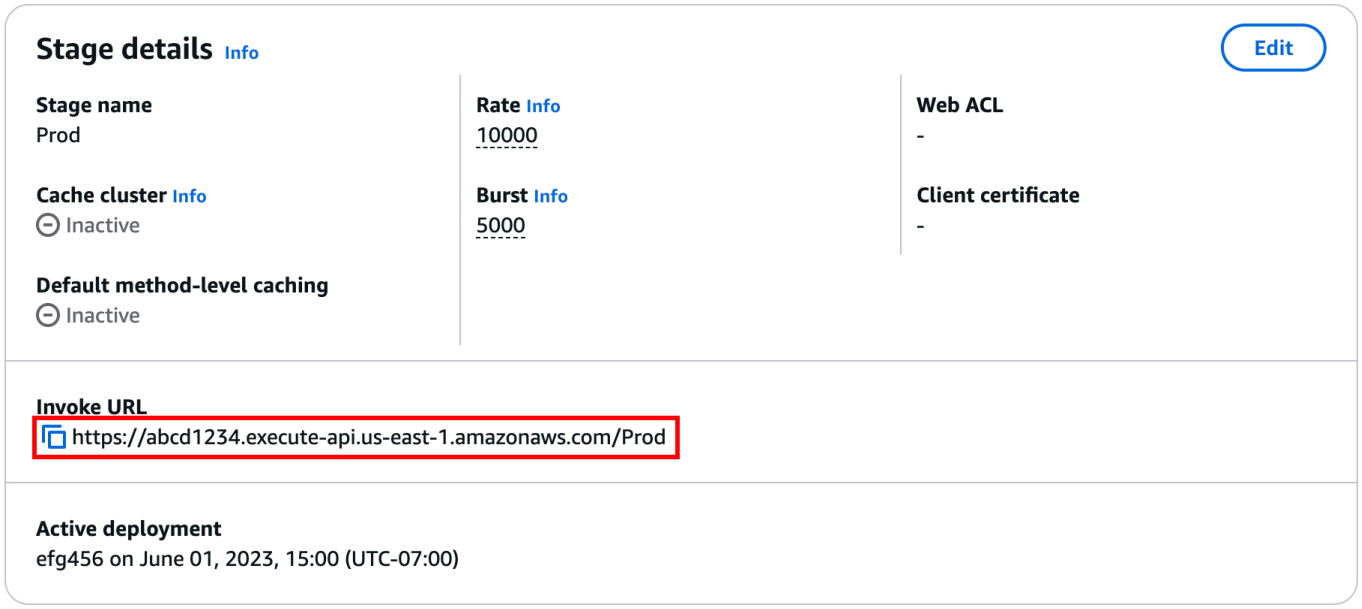
使用主控台取得 API 的調用 URL

下列程序顯示如何在 REST API 主控台取得 API 的調用 URL。

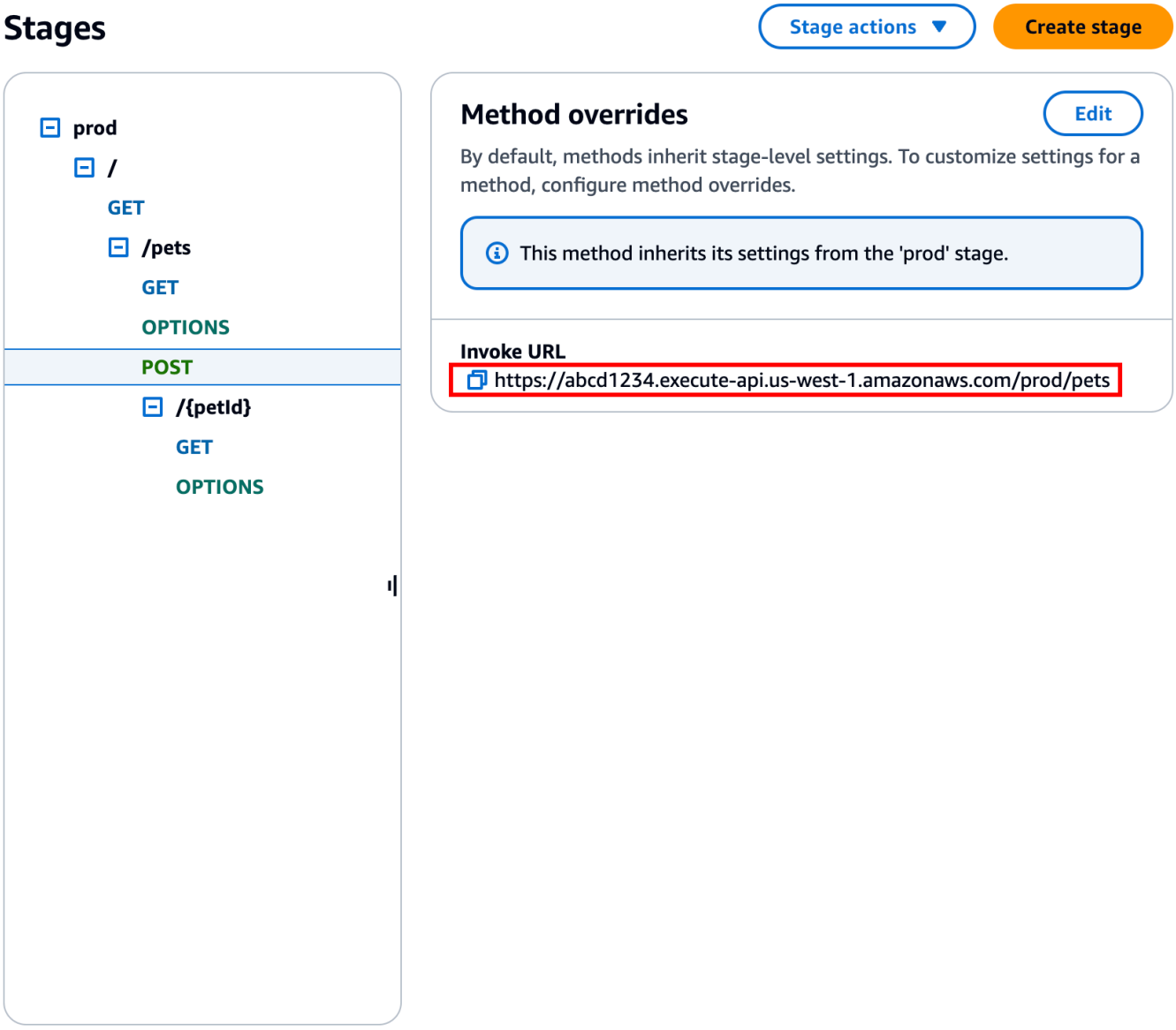
使用 REST API 主控台取得 API 的調用 URL

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 選擇部署的 API。
3. 從主導覽窗格選擇階段。
4. 在階段詳細資訊下，選擇複製圖示以複製 API 的調用 URL。

   此 URL 適用於 API 的根資源。



5. 若要取得 API 中另一個資源的 API 調用 URL，請展開次要導覽窗格下的階段，然後選擇某個方法。
6. 選擇複製圖示以複製 API 的資源層級調用 URL。




使用 AWS CLI 取得 API 的調用 URL

下列程序顯示如何使用 AWS CLI 取得 API 的調用 URL。

使用 AWS CLI 取得 API 的調用 URL

1. 使用下列命令以取得 `rest-api-id`。此命令會傳回您區域中的所有 `rest-api-id` 值。如需詳細資訊，請參閱 get-rest-apis。

```
aws apigateway get-rest-apis
```

2. 將範例 `rest-api-id` 取代為您的 `rest-api-id`，將範例 *`{stage-name}`* 取代為您的 *`{stage-name}`*，並將 *`{region}`* 取代為您的區域。

```
https://{restapi_id}.execute-api.{region}.amazonaws.com/{stage_name}/
```

使用 API 的匯出 OpenAPI 定義檔以取得 API 的調用 URL

您也可以合併 API 之已匯出 OpenAPI 定義檔的 `host` 與 `basePath` 欄位，藉以建構根 URL。如需有關如何匯出 API 的指示，請參閱 the section called “匯出 REST API”。

## 呼叫 API

您可以使用瀏覽器、curl 或其他應用程式 (例如 Postman) 呼叫已部署的 API。

此外，您可以使用 API Gateway 主控台來測試 API 呼叫。測試時會使用 API Gateway 的 `TestInvoke` 功能，其允許在部署 API 之前進行 API 測試。如需更多詳細資訊，請參閱 the section called “使用主控台測試 REST API 方法。”。

> **Note**
>
> 引動過程 URL 中的查詢字串參數值不能包含 %%。

使用 Web 瀏覽器調用 API

如果您的 API 允許匿名存取，您可以使用任何 Web 瀏覽器來調用任何 `GET` 方法。在瀏覽器的位址列中輸入完整的調用 URL。

對於其他方法或任何需要驗證的呼叫，您必須指定承載或簽署請求。您可以使用其中一個 AWS 軟體開發套件，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。

使用 curl 調用 API

您可以使用終端機中的 curl 等工具來呼叫 API。下列範例 curl 命令會在 API `prod` 階段的 `getUsers` 資源上調用 GET 方法。

Linux or Macintosh

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

Windows

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

## 使用 API Gateway 主控台來測試 REST API 方法

使用 API Gateway 主控台來測試 REST API 方法。

主題

- 先決條件
- 使用 API Gateway 主控台測試方法

### 先決條件

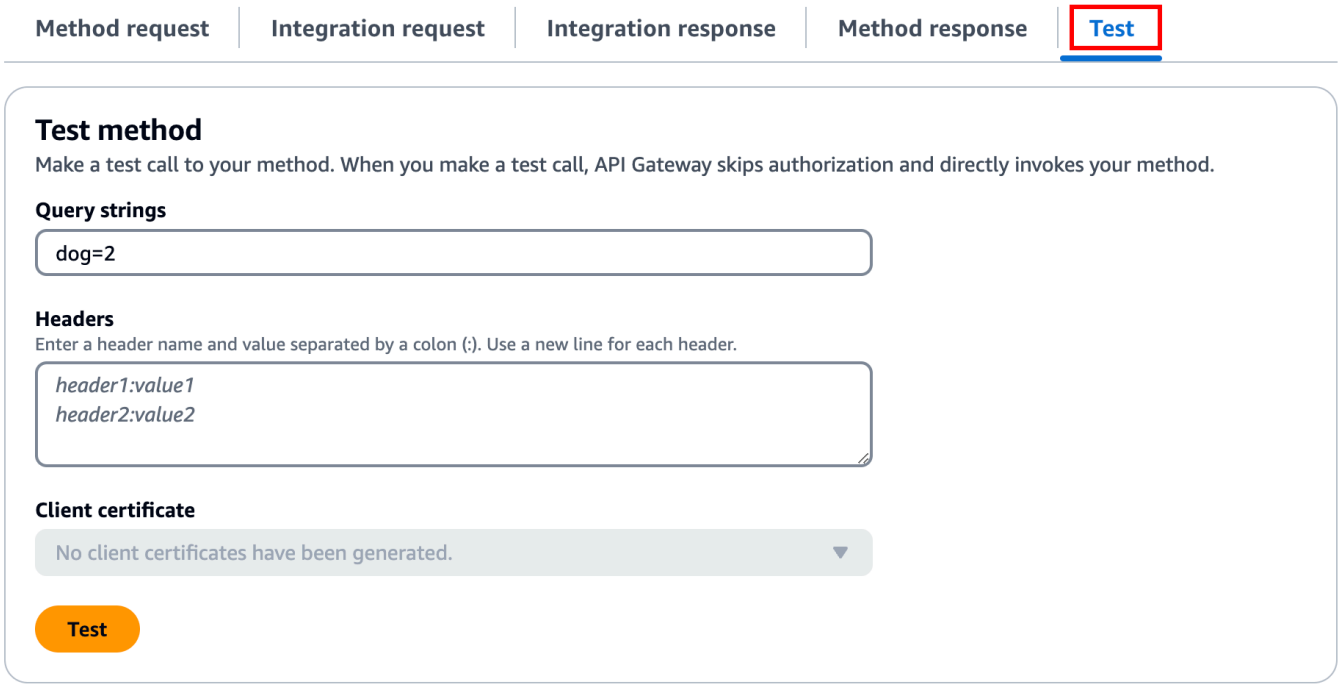
- 您必須指定所要測試之方法的設定。請遵循中的說明進行API Gateway 中 REST API 的方法

### 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 主控台之外呼叫方法是相同的。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，將會刪除 API 的資源。

若要測試方法

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 選擇 REST API。
3. 在 Resources (資源) 窗格中，選擇您要測試的方法。
4. 選擇測試標籤。您可能需要選擇向右箭頭按鈕才能顯示此索引標籤。




在顯示的任何方塊 (例如查詢字串、標頭和請求內文) 中輸入值。主控台預設應用程式/json 表單中的方法要求包含這些值。

如要了解可能需要指定的其他選項，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。
   - 延遲 (ms) 是從呼叫者收到請求，到傳回回應之間的時間。
   - 回應內文是 HTTP 回應內文。
   - 回應標頭是 HTTP 回應標頭。

   > **Tip**
   >
   > 根據映射，HTTP 狀態碼、回應內文和回應標頭可能與從 Lambda 函數、HTTP 代理 AWS 或服務代理傳送的不同。

   - Logs (日誌) 是模擬的 Amazon CloudWatch Logs 項目，如果在 API Gateway 主控台之外呼叫此方法，將會寫入該項目。

> **Note**
>
> 雖然 CloudWatch Logs 項目是模擬的，但方法呼叫的結果是真實的。

除了使用 API Gateway 主控台之外，您還可以使用 AWS CLI 或適用於 API Gateway 的 AWS SDK 來測試叫用方法。若要使用 執行此作業 AWS CLI，請參閱 test-invoke-method。

## 使用由 API Gateway 產生的 Java 軟體開發套件來執行 REST API

在本節中，我們以簡易計算機 API 為例，概述使用 API Gateway 為 REST API 所產生之 Java 軟體開發套件的步驟。繼續之前，您必須完成在 API Gateway 中為 REST API 產生 SDK中的步驟。

安裝及使用 API Gateway 所產生的 Java 軟體開發套件

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 8。
4. 設定 `JAVA_HOME` 環境變數。
5. 前往存放 pom.xml 檔案的解壓縮開發套件資料夾。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 \
    -DarchetypeGroupdId=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 DarchetypeGroupdId 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>
</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 GET /{a}/{b}/{op} API 請求承載的 GET /?a={x}&b={y}&op={operator}、POST / 與 {"a": x, "b": y, "op": "operator"} 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。

## 使用由 API Gateway 為 REST API 產生的 Android 軟體開發套件

在本節中，我們將概述使用 API Gateway 為 REST API 所產生之 Android 軟體開發套件的步驟。您必須已完成在 API Gateway 中為 REST API 產生 SDK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”。

安裝及使用 API Gateway 所產生的 Android 軟體開發套件

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 8。
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` 類別來初始化 API Gateway 產生的軟體開發套件。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 憑證提供者授權呼叫您的 API，請使用 `ApiClientFactory` 類別透過 API Gateway 所產生的軟體開發套件來傳遞一組 AWS 憑證，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);
```

9. 若要使用 API Gateway 所產生的軟體開發套件來設定 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 軟體開發套件

下列程序顯示如何使用 API Gateway 產生的 JavaScript SDK。

> **Note**
>
> 這些說明假設您已完成在 API Gateway 中為 REST API 產生 SDK中的指示。

> **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。

安裝、啟動及呼叫 API Gateway 為 REST API 所產生的 JavaScript 軟體開發套件

1. 將您稍早下載之 API Gateway 所產生的 .zip 檔案內容解壓縮。

2. 針對 API Gateway 所產生之軟體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API Gateway 中 REST API 的 CORS。

3. 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>
```

4. 在您的程式碼中，使用類似如下的程式碼來初始化 API Gateway 所產生的軟體開發套件。

```
var apigClient = apigClientFactory.newClient();
```

若要使用 AWS 登入資料初始化 API Gateway 產生的 SDK，請使用類似以下的程式碼。如果您使用 AWS 登入資料，對 API 的所有請求都會簽署。

```
var apigClient = apigClientFactory.newClient({
  accessKey: 'ACCESS_KEY',
  secretKey: 'SECRET_KEY',
});
```

若要搭配 API Gateway 所產生的軟體開發套件使用 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
```

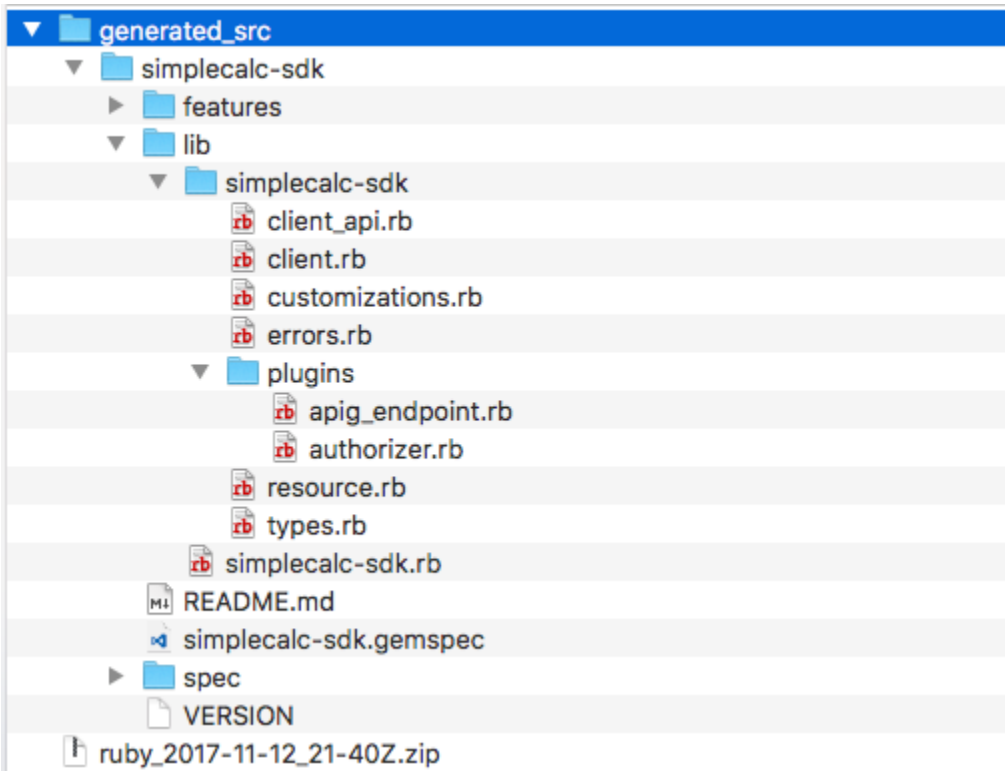
## 使用由 API Gateway 為 REST API 產生的 Ruby 軟體開發套件

下列程序顯示如何使用 API Gateway 產生的 Ruby SDK。

> **Note**
>
> 這些說明假設您已完成「在 API Gateway 中為 REST API 產生 SDK」中的說明。

### 安裝、執行個體化和呼叫 API Gateway 為 REST API 產生的 Ruby 軟體開發套件

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 已設定 AWS_IAM類型的授權，您可以在初始化secretKey期間提供 accessKey和 來包含發起人的 AWS 登入資料：

```
require 'pet-sdk'
client = Pet::Client.new(
    http_wire_trace: true,
    retry_limit: 5,
    http_read_timeout: 50,
    access_key_id: 'ACCESS_KEY',
    secret_access_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 軟體開發套件

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 軟體開發套件，以呼叫基礎 API。我們會以 SimpleCalc API 為例，說明下列主題：

- 如何在 Xcode 專案中安裝所需的 AWS Mobile SDK 元件
- 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
- 使用產生的 iOS 開發套件 (Swift) 呼叫 API

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 中完成在 API Gateway 中為 REST API 產生 SDK中的 iOS 步驟，並下載已產生之軟體開發套件的 .zip 檔案。

在 Objective-C 專案中安裝由 API Gateway 產生的 AWS 行動 SDK 和 iOS SDK

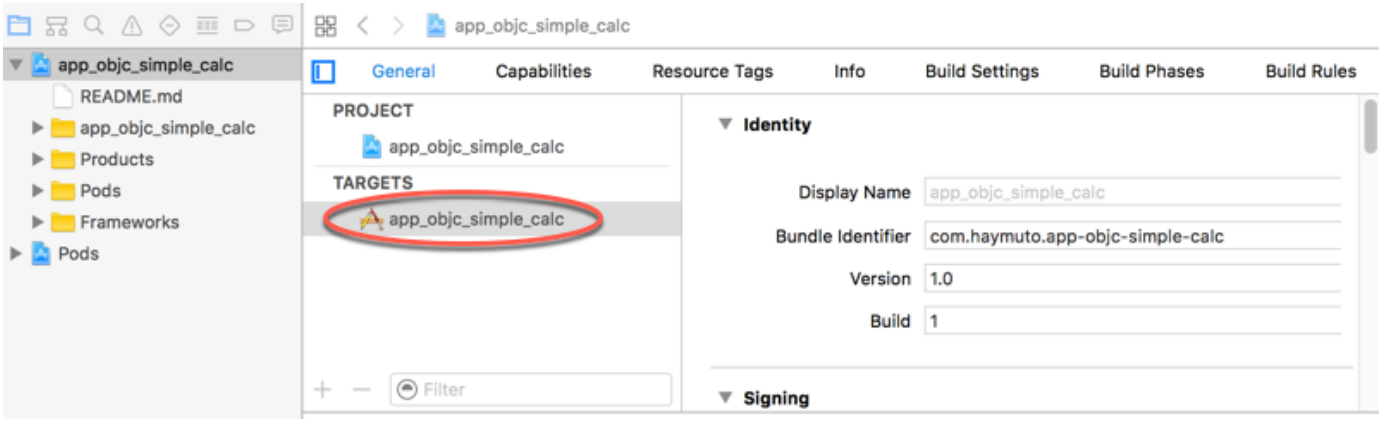
下列程序說明如何安裝開發套件。

安裝和使用 API Gateway 在 Objective-C 中產生的 iOS 軟體開發套件

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，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 AWS 的 Mobile SDK 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 AWS Mobile SDK for iOS 將 匯入 Xcode 專案，請執行下列動作：

   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'
   ```

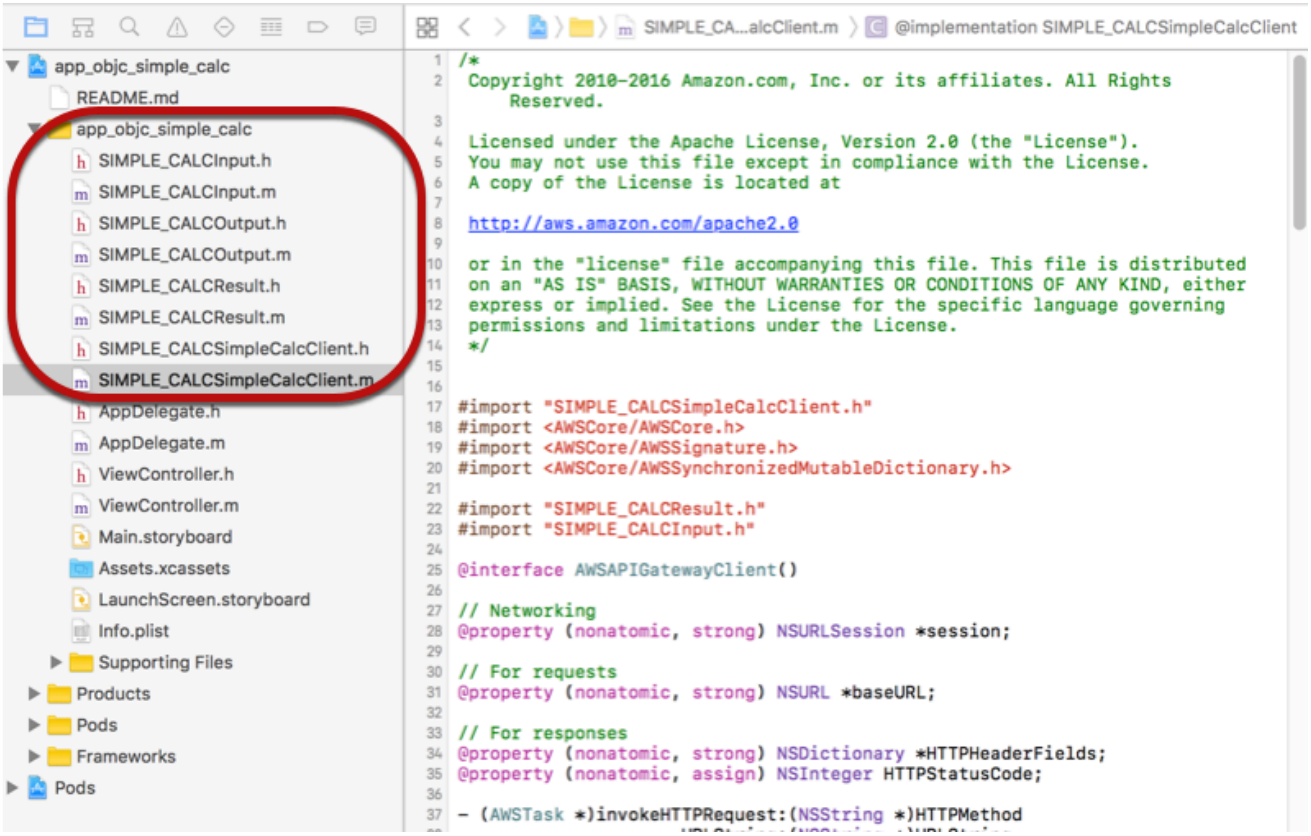
c. 開啟終端機視窗並執行下列命令：

```
pod install
```

這會安裝 API Gateway 元件和其他相依的 AWS Mobile SDK 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。




若要透過明確下載 AWS Mobile SDK 或使用 Carthage 將 AWS Mobile SDK for iOS Objective-C 匯入您的專案，請遵循 README.md 檔案中的指示。請務必僅使用其中一個選項來匯入 AWS Mobile SDK。

使用 Objective-C 專案中 API Gateway 產生的 iOS 軟體開發套件呼叫 API 方法

當您使用兩個模型來處理這些方法的輸入 (`SIMPLE_CALC`) 和輸出 (`Input`)，為這個 SimpleCalc API 產生字首為 `Result` 的開發套件後，在開發套件中，產生的 API 用戶端類別會變為

SIMPLE_CALCSimpleCalcClient，而對應的資料類別則會分別成為 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 軟體開發套件來呼叫 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，然後呼叫 `defaultClient` 方法以建立 API 用戶端物件 (如前例所示)：

```
AWSCognitoCredentialsProvider *creds = [[AWSCognitoCredentialsProvider alloc]
 initWithRegionType:AWSRegionUSEast1 identityPoolId:your_cognito_pool_id];
AWSServiceConfiguration *configuration = [[AWSServiceConfiguration 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 div 2 = 0.5。在此，div 用來代替 /，因為後端的簡單 Lambda 函數不處理 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 中完成在 API Gateway 中為 REST API 產生 SDK中的 iOS 步驟，並下載已產生之軟體開發套件的 .zip 檔案。

主題

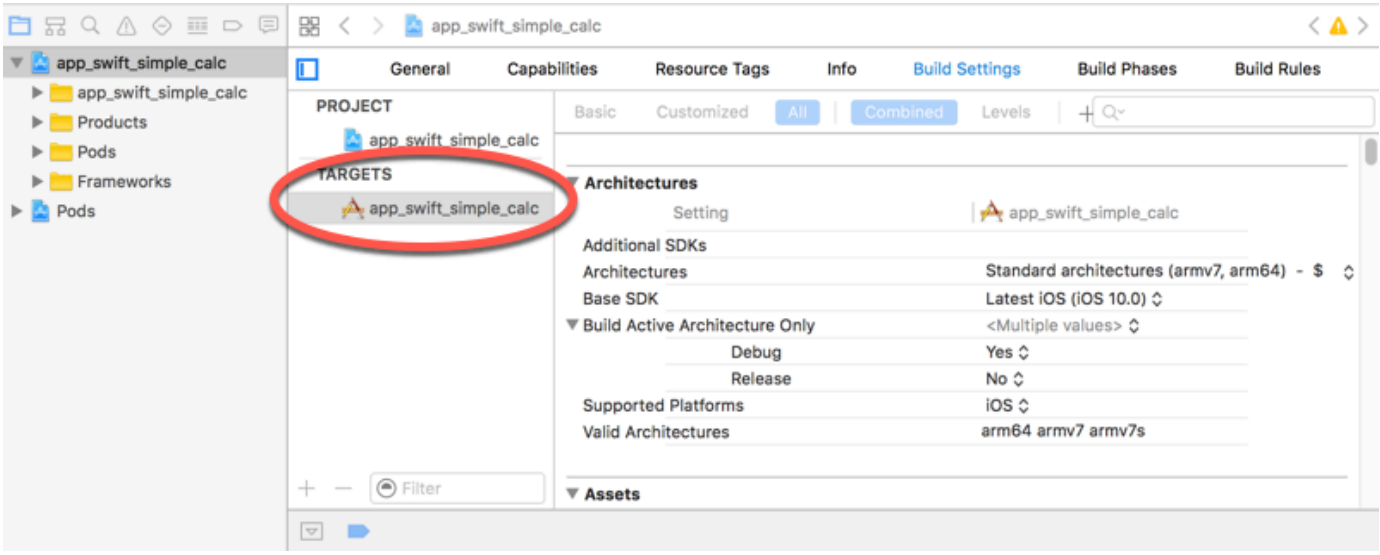
- 在 Swift 專案中安裝 AWS 行動 SDK 和 API Gateway 產生的 SDK
- 在 Swift 專案中透過 API Gateway 所產生的 iOS 軟體開發套件來呼叫 API 方法

在 Swift 專案中安裝 AWS 行動 SDK 和 API Gateway 產生的 SDK

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 軟體開發套件

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，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 SDK 元件匯入 Xcode 專案，請執行下列動作：

   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'
```

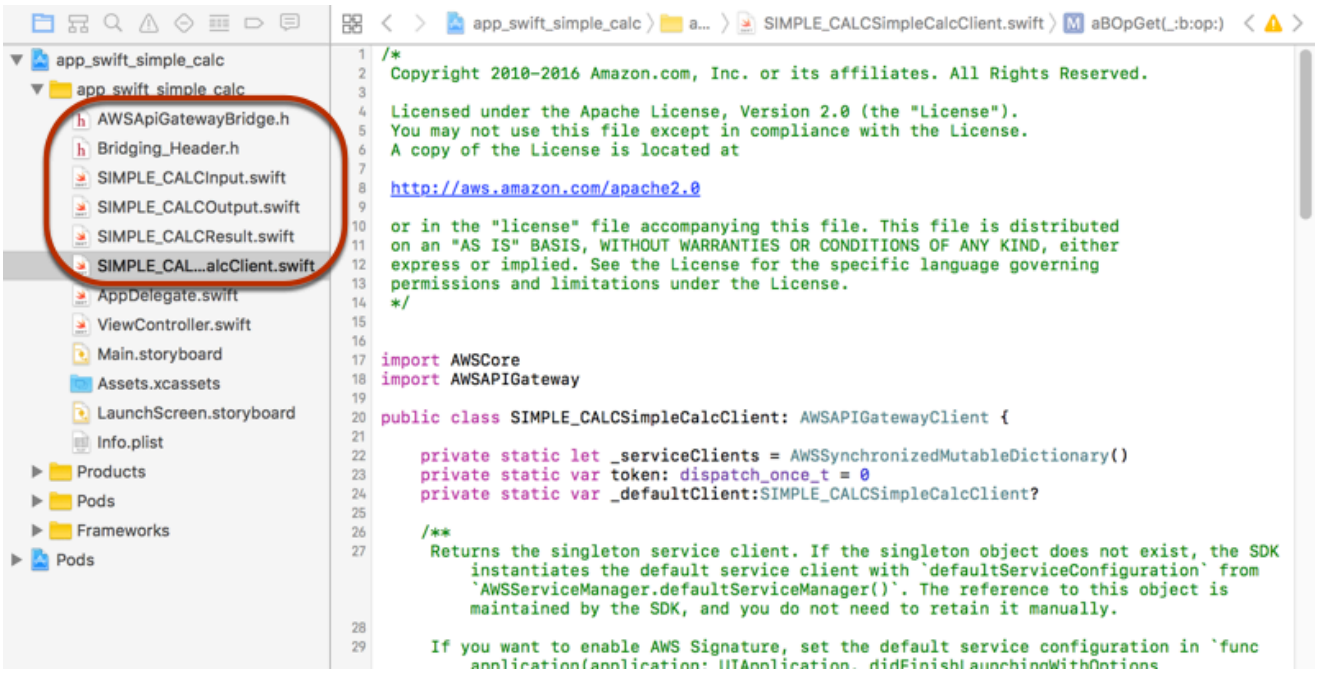
c. 開啟終端機視窗，並在應用程式目錄中執行下列命令：

```
pod install
```

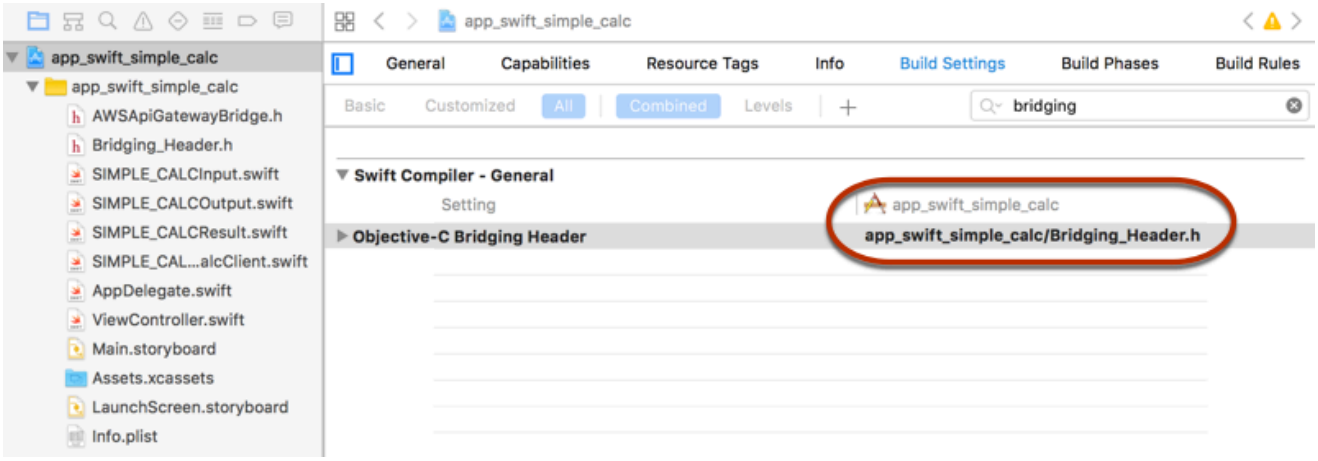
這會將 API Gateway 元件和任何相依的 AWS Mobile SDK 元件安裝到應用程式的專案中。

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。

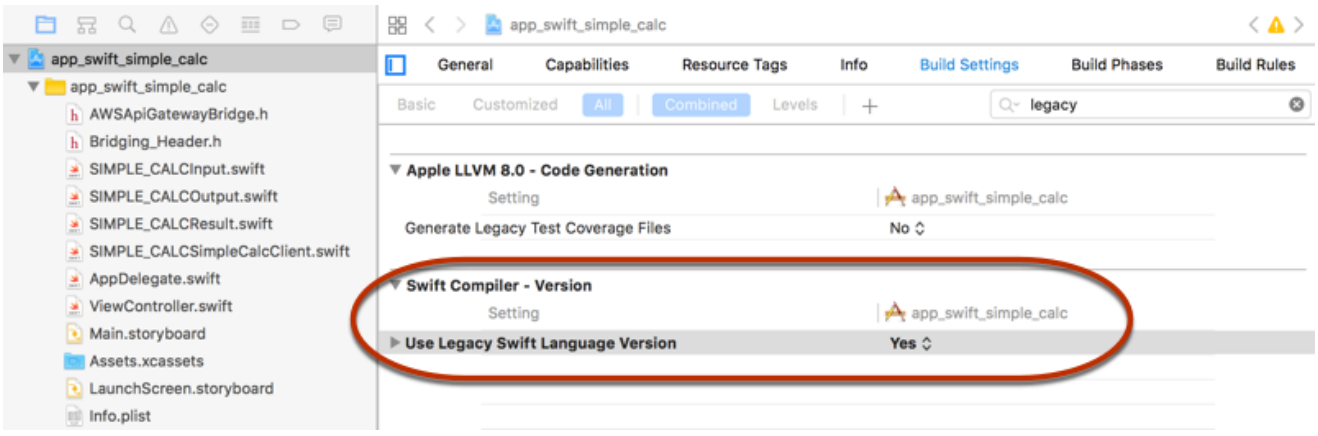


f. 若要啟用從 Swift 程式碼專案呼叫 AWS Mobile SDK 的 Objective-C 程式庫，請在您 Xcode 專案組態的 Swift 編譯器 - 一般設定下，在 Objective-C 橋接標頭屬性上設定`Bridging_Header.h`檔案路徑：




> **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 使用的 Swift 版本比 AWS Mobile SDK 支援的版本更新，您會收到 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)：



若要透過明確下載 AWS Mobile SDK 或使用 Carthage 將適用於 iOS 的 AWS Mobile SDK 匯入您的專案，請遵循 SDK 套件隨附的 README.md 檔案中的指示。請務必僅使用其中一個選項來匯入 AWS Mobile SDK。

在 Swift 專案中透過 API Gateway 所產生的 iOS 軟體開發套件來呼叫 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 產生字首為 SIMPLE_CALC 的開發套件後，在開發套件中，產生的 API 用戶端類別會變為 SIMPLE_CALCSimpleCalcClient，而對應的資料類別則會分別成為 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 軟體開發套件來呼叫 API

1. 將 API 用戶端類別執行個體化：

   ```
   let client = SIMPLE_CALCSimpleCalcClient.default()
   ```

   若要搭配 API 使用 Amazon Cognito，請先設定預設 AWS 服務組態（如下所示），再取得`default`方法（先前顯示）：

   ```
   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 函數不處理 URL 編碼的路徑變數。

# 在 API Gateway 中使用 OpenAPI 開發 REST API

您可以使用 API Gateway 將 REST API 從外部定義檔案匯入至 API Gateway。目前，API Gateway 支援 OpenAPI v2.0 和 OpenAPI v3.0 定義檔案，例外狀況列於 適用於 REST API 的 Amazon API Gateway 重要說明 中。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。

主題

- 將邊緣最佳化 API 匯入至 API Gateway
- 將區域 API 匯入至 API Gateway
- 匯入 OpenAPI 檔案以更新現有的 API 定義
- 設定 OpenAPI basePath 屬性
- AWS OpenAPI 匯入的 變數
- 將 API 匯入 API Gateway 的錯誤和警告
- 從 API Gateway 匯出 REST API

## 將邊緣最佳化 API 匯入至 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 SDK 來執行此操作。

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。

主題

- 使用 API Gateway 主控台匯入邊緣最佳化 API
- 使用 匯入邊緣最佳化 API AWS CLI

### 使用 API Gateway 主控台匯入邊緣最佳化 API

若要使用 API Gateway 主控台匯入邊緣最佳化 API，請執行以下操作：

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 選擇 Create API (建立 API)。
3. 在 REST API 下，選擇 Import (匯入)。
4. 複製 API 的 OpenAPI 定義並將它貼入程式碼編輯器，或選擇選擇檔案，從本機磁碟機載入 OpenAPI 檔案。
5. 對於 API 端點類型，選取邊緣最佳化。
6. 選擇建立 API 以開始匯入 OpenAPI 定義。

使用 匯入邊緣最佳化 API AWS CLI

以下 import-rest-api 命令會從 OpenAPI 定義檔案匯入 API，以建立新的邊緣最佳化 API：

```
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 \
    --parameters endpointConfigurationTypes=EDGE \
    --fail-on-warnings \
    --body 'file://path/to/API_OpenAPI_template.json'
```

## 將區域 API 匯入至 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 SDK。

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。

主題

- 使用 API Gateway 主控台匯入區域性 API
- 使用 匯入區域 API AWS CLI

使用 API Gateway 主控台匯入區域性 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. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 選擇 Create API (建立 API)。
3. 在 REST API 下，選擇 Import (匯入)。
4. 複製 API 的 OpenAPI 定義並將它貼入程式碼編輯器，或選擇選擇檔案，從本機磁碟機載入 OpenAPI 檔案。
5. 對於 API 端點類型，選取區域。
6. 選擇建立 API 以開始匯入 OpenAPI 定義。

使用 匯入區域 API AWS CLI

以下 import-rest-api 命令會匯入 OpenAPI 定義檔案，並將端點類型設為區域：

```
aws apigateway import-rest-api \
    --parameters 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 宣告以下列方法傳回 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
```

傳回 200 和 400 回應，且

```
POST /a
```

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。

> **Important**
>
> 使用 時mode=overwrite，會取代整個 API 定義。匯入內文中未明確包含的任何 API 組態都會重設為其預設值。這包括安全政策、授權組態和其他 API 層級屬性等設定。
> 若要保留現有的組態，請使用 mode=merge，這會保留匯入內文中未指定的現有組態。如果您使用 mode=overwrite，請確定您的 OpenAPI 定義包含所有必要的 API 組態，例如 API Gateway 擴充功能。

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
```

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 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
```

未指定 `mode` 查詢參數時，會假設使用合併。

> **Note**
>
> PUT 操作為等冪，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，重複此操作會成功將 API 放到相同的最終狀態，就像是成功進行的第一個操作。

## 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。相反地，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。

Ignore

如果 OpenAPI 檔案具有 basePath 的 /a/b/c 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 basePath 的 /a/b/c 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。

Split

如果 OpenAPI 檔案具有 basePath 的 /a/b/c 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。

## AWS OpenAPI 匯入的 變數

您可以在 OpenAPI 定義中使用下列 AWS 變數。API Gateway 會在匯入 API 時解析變數。若要指定變數，請使用 ${*variable-name*}。下表說明可用的 AWS 變數。

| 變數名稱 | 說明 |
| --- | --- |
| AWS::AccountId | 匯入 API AWS 的帳戶 ID。例如：123456789012。 |
| AWS::Partition | 匯入 API 的 AWS 分割區。對於標準 AWS 區域，分割區為 aws。 |
| AWS::Region | 匯入 API AWS 的區域。例如 us-east-2 。 |

### AWS 變數範例

下列範例使用 AWS 變數來指定 整合的 AWS Lambda 函數。

OpenAPI 3.0

```
openapi: "3.0.1"
info:
  title: "tasks-api"
  version: "v1.0"
paths:
  /:
    get:
```

```
      summary: List tasks
      description: Returns a list of tasks
      responses:
        200:
          description: "OK"
          content:
            application/json:
              schema:
                type: array
                items:
                  $ref: "#/components/schemas/Task"
        500:
          description: "Internal Server Error"
          content: {}
      x-amazon-apigateway-integration:
        uri:
          arn:${AWS::Partition}:apigateway:${AWS::Region}:lambda:path/2015-03-31/
functions/arn:${AWS::Partition}:lambda:${AWS::Region}:
${AWS::AccountId}:function:LambdaFunctionName/invocations
        responses:
          default:
            statusCode: "200"
        passthroughBehavior: "when_no_match"
        httpMethod: "POST"
        contentHandling: "CONVERT_TO_TEXT"
        type: "aws_proxy"
components:
  schemas:
    Task:
      type: object
      properties:
        id:
          type: integer
        name:
          type: string
        description:
          type: string
```

## 將 API 匯入 API Gateway 的錯誤和警告

當您將外部定義檔案匯入 API Gateway 時，API Gateway 可能會產生警告和錯誤。下列各節討論匯入期間可能發生的錯誤和警告。

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。根據預設，`failonwarnings` 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。

## 從 API Gateway 匯出 REST API

一旦您使用 API Gateway 主控台或其他方式，在 API Gateway 中建立並配置 REST API，就可以使用 API Gateway Export API (屬於 Amazon API Gateway 控制服務的一部分) 將其匯出至 OpenAPI 檔案。若要使用 API Gateway Export API，您必須簽署 API 請求。如需簽署請求的詳細資訊，請參閱《IAM 使用者指南》中的簽署 AWS API 請求。您可以選擇在匯出的 OpenAPI 定義檔中包含 API Gateway 整合延伸項目和 Postman 延伸項目。

> **Note**
>
> 使用 匯出 API 時 AWS CLI，請務必包含延伸項目參數，如下列範例所示，以確保包含`x-amazon-apigateway-request-validator`延伸項目：
>
> ```
> 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) 或是 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 Export API 提交 GET 請求的詳細資訊，請參閱 GetExport。

> **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，其錯誤訊息為「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。如需可使用 API Gateway 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
```

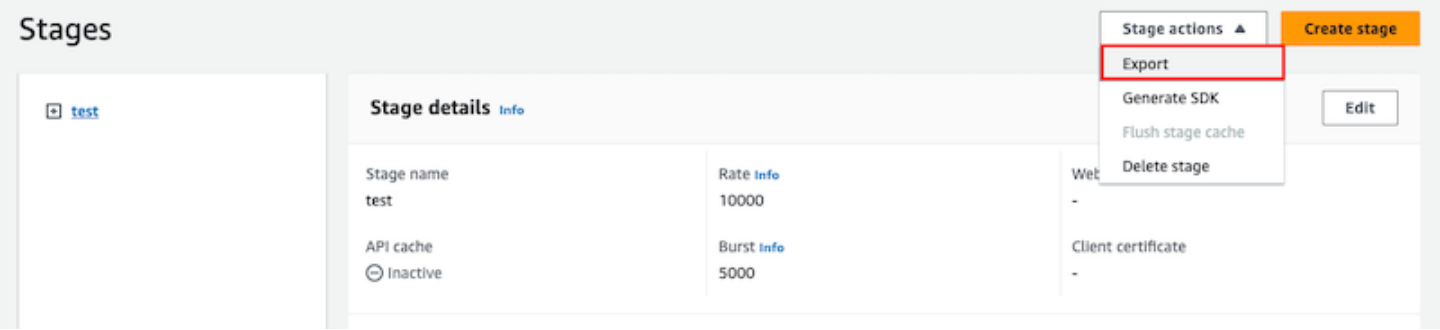
OpenAPI 2.0

```
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。

在 API Gateway 主控台的階段窗格中，選擇階段動作、匯出。




指定 API 規格類型、格式和延伸模組，以下載 API 的 OpenAPI 定義。

# 發布 REST API 以供客戶調用

只是建立和開發 API Gateway API，並不代表使用者可以自動呼叫 API。若要使其可供呼叫，您必須將 API 部署到階段。此外，您可能會想要自訂使用者用於存取 API 的 URL。您可以為它提供一個與您品牌一致的網域，或是比 API 預設 URL 更好記的網域。

在本節中，您可以了解如何部署 API，以及如何自訂提供給使用者以存取 API 的 URL。

> **Note**
>
> 為了增強 API Gateway API 的安全性，`execute-api.{region}.amazonaws.com` 網域會在公用尾碼清單 (PSL) 中註冊。為了加強安全性，如果您需要在 API Gateway API 的預設網域名稱中設定敏感性 Cookie，我們建議您使用具 `__Host-` 前置詞的 Cookie。此做法將有助於保護您的網域免受跨站請求偽造 (CSRF) 攻擊。如需更多資訊，請參閱 Mozilla 開發人員網路中的設定 Cookie 頁面。

主題

- 在 API Gateway 中部署 REST API
- API Gateway 中公有 REST API 的自訂網域名稱

# 在 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。階段是 API 生命週期狀態的邏輯參考 (例如，dev、prod、beta、v2)。可依照 API ID 和階段名稱來識別 API 階段。它們包含在您用於叫用 API 的 URL 中。每個階段都是 API 部署的具名參考，且可供用戶端應用程式呼叫。

> **Important**
>
> 每次更新 API 時，都必須將 API 重新部署到現有階段或新階段。更新 API，包括修改路由、方法、整合、授權方、資源政策，以及階段設定以外的任何其他內容。

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。這可讓您的 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 REST API (例如) 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，或將階段變數值從 `test` 的階段名稱更新為 `prod` 的階段名稱。

在本節中，我們會討論如何使用 API Gateway 主控台或呼叫 API Gateway REST API 來部署 API。若要使用其他工具，請參閱 AWS CLI 或 AWS 軟體開發套件的文件。

主題

- 在 API Gateway 中建立 REST API 的部署
- 在 API Gateway 中為 REST API 設定階段
- 設定 Amazon API Gateway Canary Release 部署
- 需要重新部署的 REST API 更新

## 在 API Gateway 中建立 REST API 的部署

在 API Gateway 中，REST API 部署由 Deployment 資源代表。它類似於由 RestApi 資源表示之 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 資源表示。表示 API 的快照，包括方法、整合、模型、對應範本、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”」中討論階段的建立。

主題

- 建立部署
- API 部署的後續步驟

### 建立部署

下列程序顯示如何建立 REST API 的部署。

AWS 管理主控台

第一次部署 REST API 之前，您必須先進行建立。如需更多詳細資訊，請參閱 在 APIs Gateway 中開發 REST API。

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。

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 在 API 導覽窗格中，選擇您要部署的 API。
3. 在 Resources (資源) 窗格中，選擇 Deploy API (部署 API)。
4. 針對階段，請從下列項目中選取：
   a. 若要建立新階段，請選取新階段，然後在階段名稱中輸入名稱。或者，您可以在部署說明中提供部署的說明。
   b. 若要選擇現有階段，請從下拉式選單選取階段名稱。建議您在部署說明中提供新部署的說明。
   c. 若要建立與階段沒有關聯的部署，請選取無階段。您之後可以再將此部署與階段建立關聯。
5. 選擇部署。

AWS CLI

當您建立部署時，您可以執行個體化 Deployment 資源。

以下 create-deployment 命令會建立新部署：

```
aws apigateway create-deployment --rest-api-id rest-api-id
```

您須先將此部署與階段建立關聯，才能呼叫 API。若現已有階段，您可使用新建立的部署 ID 更新階段的 deploymentId 屬性來完成此操作。以下 update-stage 命令會使用新部署來更新階段。在主控台中，這稱為作用中部署。

```
aws apigateway update-stage \
   --rest-api-id rest-api-id \
   --stage-name 'stage-name' \
   --patch-operations op='replace',path='/deploymentId',value='deployment-id'
```

建立部署時，您也可以同時將其與新階段建立關聯。以下 create-deployment 命令會建立新的部署，並將其與名為 beta 的新階段建立關聯：

```
aws apigateway create-deployment \
   --rest-api-id rest-api-id \
   --stage-name beta
```

若要重新部署 API，請執行相同步驟。您可以重複使用相同的階段。

### API 部署的後續步驟

以下是 API 部署的後續步驟。

#### 修改階段設定。

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，請參閱修改階段設定。

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 (例如啟用記錄功能) 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。

#### 選擇不同的部署階段組合

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. 如果您尚未在階段窗格中，請在主導覽窗格中選擇階段。
2. 選取您要更新的階段。
3. 在部署歷史記錄索引標籤上，選擇要讓階段使用的部署。
4. 選擇變更作用中部署。
5. 確認您要變更作用中部署，然後在設為作用中部署對話方塊中選擇變更作用中部署。

將部署特定資料傳遞至您的 API。

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「在 API Gateway 中使用 REST API 的階段變數」中的說明。

## 在 API Gateway 中為 REST API 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。下一節顯示如何建立和設定階段。

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 主控台建立新階段，請依照下列步驟執行：

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 選擇 REST API。
3. 在主導覽窗格中，於 API 底下選擇階段。
4. 從階段導覽窗格選擇建立。
5. 在階段名稱中輸入名稱，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。

6. (選用)。在描述中，輸入階段描述。
7. 在部署中，選取要與此階段相關聯的現有 API 部署的日期與時間。
8. 在其他設定下，您可以指定階段的其他設定。
9. 選擇建立階段。

修改階段設定。

成功部署 API 之後，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。如果您已修改 REST API 的預設端點，當您更新階段時，修改會傳播到 API 的所有階段。下列步驟示範如何使用 API Gateway 主控台的階段編輯器來執行此操作。

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 選擇 REST API。
3. 在主導覽窗格中，於 API 底下選擇階段。
4. 在 Stages (階段) 窗格中，選擇階段的名稱。
5. 在階段詳細資訊區段中，選擇編輯。
6. (選用) 針對階段描述，編輯描述。
7. 針對其他設定，請修改下列設定：

快取設定

若要針對階段啟用 API 快取，請開啟 佈建 API 快取。然後，設定預設方法層級快取、快取容量、加密快取資料、快取存活期 (TTL)，以及每個金鑰快取失效的任何要求。

快取在您針對特定方法開啟預設方法層級快取，或開啟方法層級快取之前，不會處於作用中狀態。

如需快取設定的詳細資訊，請參閱 API Gateway 中 REST API 的快取設定。

> **Note**
>
> 如果您為 API 階段啟用 API 快取，您的帳戶 AWS 可能需要支付 API 快取的費用。快取不符合 AWS 免費方案的資格。

限流設定

若要為與此 API 相關聯的所有方法設定階段層級限流目標，請開啟限流。

在 Rate (速率) 欄位，輸入目標速率。這是會將字符新增到字符儲存貯體的速率 (以每秒請求數計)。階段層級速率不得超過 在 API Gateway 中設定和執行 REST API 的配額 所指定的帳戶層級速率。

在 Burst (爆量) 欄位，輸入目標爆量率。暴量率是字符儲存貯體的容量。這可允許請求量在一段期間內超過目標速率。此階段層級爆量率不得超過 在 API Gateway 中設定和執行 REST API 的配額 所指定的帳戶層級爆量率。

> **Note**
>
> 調節率並非硬性限制，會依最佳作法來套用。在某些情況下，用戶端可能超出設定的目標。請勿依靠調節功能來控制成本或封鎖對 API 的存取。請考慮使用 AWS Budgets 來監控成本，使用 AWS WAF 來管理 API 請求量。

防火牆和憑證設定

若要將 AWS WAF Web ACL 與階段建立關聯，請從 Web ACL 下拉式清單中選取 Web ACL。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))。

若要為您的階段選取用戶端憑證，請從用戶端憑證下拉式選單中選取憑證。

8. 選擇繼續。
9. 檢閱變更，然後選擇儲存變更。
10. 若要為與此 API Gateway API 階段相關聯的所有方法啟用 Amazon CloudWatch Logs，請在日誌和追蹤區段中選擇編輯。

> **Note**
>
> 若要啟用 CloudWatch Logs，您必須同時指定 IAM 角色的 ARN，讓 API Gateway 代表您的使用者將資訊寫入 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 日誌角色中，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。
> IAM 角色也必須包含下列信任關係陳述式：
>
> JSON

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

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

如需有關 CloudWatch 的詳細資訊，請參閱 Amazon CloudWatch 使用者指南。

11. 從 CloudWatch Logs 下拉式選單中選取記錄層級。記錄層級如下：
    - 關閉 – 此階段未開啟記錄。
    - 僅限錯誤 – 僅針對錯誤啟用記錄。
    - 錯誤和資訊日誌 – 針對所有事件啟用記錄。
12. 選取資料追蹤，讓 API Gateway 向 CloudWatch 報告針對您階段所記錄的資料追蹤。這對於故障排除 API 很有用，但可能會導致記錄機密資料。

> **Note**
>
> 建議您不要針對生產 API 使用資料追蹤。

13. 若要讓 API Gateway 向 CloudWatch 回報 `API calls`、`Latency`、`Integration latency`、`400 errors` 和 `500 errors` 的 API 指標，請選取詳細指標。如需 CloudWatch 的詳細資訊，請參閱《Amazon CloudWatch 使用者指南》中的基本監控和詳細監控。

> **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 層級或階段層級指標的費用。

14. 若要啟用目的地的存取記錄，請開啟自訂存取記錄。
15. 針對存取日誌目的地 ARN，輸入日誌群組或 Firehose 串流的 ARN。

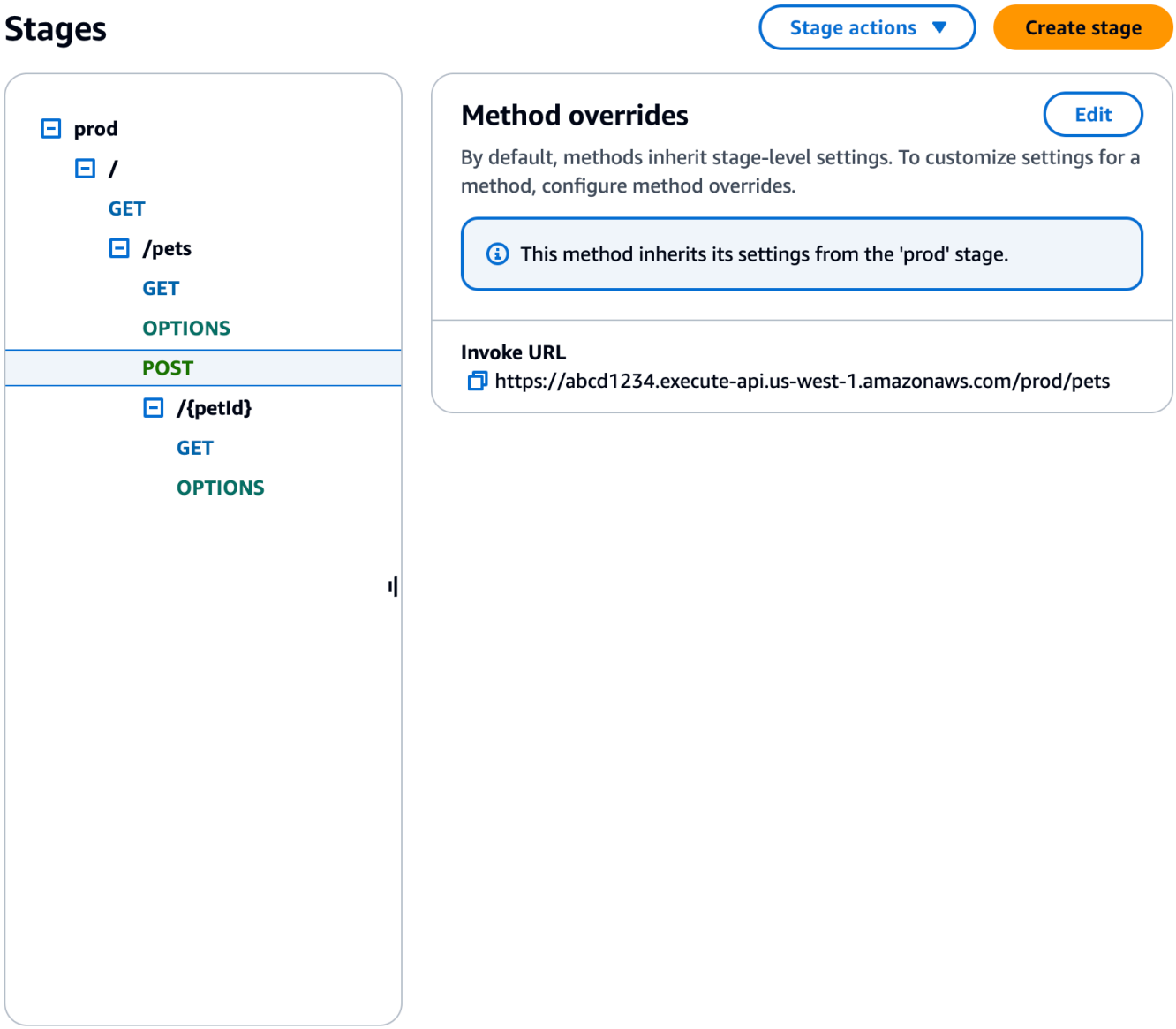
Firehose 的 ARN 格式為 `arn:aws:firehose:{region}:{account-id}:deliverystream/amazon-apigateway-{your-stream-name}`。Firehose 串流的名稱必須是 `amazon-apigateway-{your-stream-name}`。

16. 在日誌格式中，輸入日誌格式。若要進一步了解範例日誌格式，請參閱 the section called “存取日誌格式”。
17. 若要為 API 階段啟用 AWS X-Ray 追蹤，請選取 X-Ray 追蹤。如需詳細資訊，請參閱在 API Gateway 中使用 X-Ray 追蹤使用者對 REST API 的請求。
18. 選擇 Save changes (儲存變更)。重新部署 API 以使新設定生效。

### 覆寫階段層級設定

自訂階段層級設定之後，您可以針對每個 API 方法覆寫這些設定。其中一些選項可能會對您的 產生額外費用 AWS 帳戶。

1. 若要設定方法覆寫，請展開次要導覽窗格下的階段，然後選擇方法。




2. 針對方法覆寫，選擇編輯。
3. 若要開啟方法層級 CloudWatch 設定，請針對 CloudWatch Logs 選取記錄層級。
4. 若要開啟方法的資料追蹤記錄，請選取資料追蹤。

> **Note**
>
> 建議您不要針對生產 API 使用資料追蹤。

5. 若要開啟方法層級詳細指標，請選取詳細指標。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 層級或階段層級指標的費用。

6. 若要開啟方法層級限流，請選取限流。輸入適當的方法層級選項。若要進一步了解調節，請參閱 the section called “限流”。
7. 若要設定方法層級快取，請選取啟用方法快取。如果您在階段詳細資訊中變更預設方法層級快取設定，則不會影響此設定。
8. 選擇儲存。

新增 API Gateway REST API 做為 Amazon Bedrock AgentCore Gateway 的目標

Amazon Bedrock AgentCore Gateway 為 AI 代理程式開發人員提供了一種安全的方式，將您的 API Gateway REST APIs 公開為模型內容協定 (MCP) 相容工具。AgentCore Gateway 使用目標來定義工具。當您將階段新增為目標時，閘道會成為單一 MCP URL，讓 能夠存取代理程式的工具。如需詳細資訊，請參閱《Amazon Bedrock AgentCore Gateway 開發人員指南》中的 API Gateway REST API 階段作為目標。

API Gateway 目標會將 AgentCore Gateway 連線至 REST APIs的階段。您可以包含整個階段做為目標，或選取資源。建立 API Gateway 目標之後，AgentCore Gateway 會將傳入的 MCP 請求轉譯為 HTTP 請求，並處理回應格式。MCP 用戶端可以使用 `tools/list`方法擷取 API 文件，並使用 `tools/call`方法叫用 APIs。

考量事項

下列考量事項可能會影響您使用將階段新增為 AgentCore Gateway 的目標：

- 您必須已經有 AgentCore Gateway。
- 僅支援公有 REST APIs。
- 無法停用 API 的預設端點。
- API 的每個方法都必須有為其定義的操作名稱，或者當您將階段新增為目標時，需要建立名稱覆寫。此名稱用作代理程式用來與您的 方法互動的工具名稱。
- 您可以使用傳出身分驗證的 `NO_AUTH`、 `API_KEY`或 `GATEWAY_IAM_ROLE`登入資料提供者類型，以允許閘道存取您的 API。`API_KEY` 登入資料提供者由 AgentCore Gateway 定義。您可以使用現有的 API Gateway API 金鑰。如需詳細資訊，請參閱設定傳出身分驗證。
- 如果您使用 Amazon Cognito 使用者集區或 Lambda 授權方來控制對 API 的存取，則 MCP 用戶端無法存取它。
- 您的 API 必須與 AgentCore Gateway 位於相同的帳戶和區域。

新增 API 階段做為 AgentCore Gateway 的目標

下列程序說明如何將 API 的階段新增為 AgentCore Gateway 的目標。

新增 API 階段做為 AgentCore Gateway 的目標

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 選擇部署到階段的 REST API。
3. 在主導覽窗格中，選擇階段。
4. 選擇階段動作，然後選擇建立 MCP 目標。
5. 針對 AgentCore Gateway，選取 AgentCore Gateway。
6. 針對目標名稱，輸入目標名稱。
7. 針對目標描述，輸入描述。
8. 保留提供的 API 和階段。
9. 針對選取 API 資源，選取使用 AgentCore Gateway 的代理程式可存取的 API 資源。

   如果您未選取資源，則客服人員無法檢視文件或叫用端點。
10. 資源和 方法的組合是工具的操作。如果您的操作沒有名稱，請建立名稱覆寫。

    您也可以在建立方法時定義方法的操作名稱。
11. 針對傳出身分驗證組態，選擇 IAM 角色 、無授權或 API 金鑰。
12. 選擇建立目標。

若要檢視可存取 API 的所有 AgentCore Gateway，請選擇主導覽窗格中的 MCP 目標區段。 APIs 在本節中，您可以為部署到階段的區域中的任何 API 建立 MCP 目標。選擇建立 MCP 目標，然後遵循先前的步驟。

您也可以在 AgentCore Gateway 主控台中檢視目標的可用工具，並編輯目標。如需詳細資訊，請參閱將目標新增至現有的 AgentCore Gateway。

刪除階段

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。下列步驟說明如何使用 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。

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 選擇 REST API。
3. 在主導覽窗格中，選擇階段。
4. 在階段窗格中，選擇您要刪除的階段，然後選擇階段動作、刪除階段。
5. 出現提示時，輸入 **`confirm`**，然後選擇刪除。

## 在 API Gateway 中為 API 階段設定標籤

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。若要這樣做，您可以使用 API Gateway 主控台、AWS CLI/軟體開發套件或 API Gateway REST API。

階段也可以繼承其父系 REST API 的標籤。如需更多詳細資訊，請參閱 the section called “Amazon API Gateway V1 API 中的標籤繼承”。

如需標記 API Gateway 資源的詳細資訊，請參閱標記。

主題

- 使用 API Gateway 主控台設定 API 階段的標籤
- 使用 AWS CLI 設定 API 階段的標籤
- 使用 API Gateway REST API 設定 API 階段的標籤

### 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. 在主導覽窗格中，選擇階段。

5. 選擇 Tags (標籤) 索引標籤。您可能需要選擇向右箭頭按鈕才能顯示此索引標籤。
6. 選擇管理標籤。
7. 在標籤編輯器中，選擇新增標籤。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`)。選擇儲存以儲存標籤。
8. 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。
9. 若要從階段中移除現有標籤，請選擇移除。
10. 如果您先前已在 API Gateway 主控台中部署 API，就必須重新部署該 API，變更才會生效。

使用 AWS CLI 設定 API 階段的標籤

您可以使用 create-stage 命令或 tag-resource 命令，利用 AWS CLI 設定 API 階段的標籤。您可以使用 untag-resource 命令，從 API 階段刪除一或多個標籤。

以下 create-stage 命令會在建立 `test` 階段時新增標籤：

```
aws apigateway create-stage --rest-api-id abc1234 --stage-name test --description
 'Testing stage' --deployment-id efg456 --tag Department=Sales
```

以下 tag-resource 命令會將標籤新增至 `prod` 階段：

```
aws apigateway tag-resource --resource-arn arn:aws:apigateway:us-east-2::/
restapis/abc123/stages/prod --tags Department=Sales
```

以下 untag-resource 命令會從 `test` 階段移除 `Department=Sales` 標籤：

```
aws apigateway untag-resource --resource-arn arn:aws:apigateway:us-east-2::/
restapis/abc123/stages/test --tag-keys Department
```

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，請在查詢運算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 中使用 REST API 的階段變數

階段變數是鍵值對，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。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，並使用階段變數來設定 API 部署階段，以與不同的後端端點互動。

階段變數並非用於敏感資料，例如登入資料。若要將敏感資料傳遞至整合，請使用 AWS Lambda 授權方。您可以將敏感資料傳遞至 Lambda 授權方輸出中的整合。如需進一步了解，請參閱the section called “來自 API Gateway Lambda 授權方的輸出”。

階段變數的使用案例

下列為您的階段變數的使用案例。

指定不同的後端端點

您的 API 可將 GET 請求當做 HTTP 代理傳到後端 Web 主機。您可以使用階段變數，如此一來，當讓 API 呼叫者調用您的生產端點時，API Gateway 就會呼叫 `example.com`。然後，當 API 呼叫者調用 Beta 階段時，API Gateway 會呼叫不同的 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。您無法使用階段變數來設定不同的整合端點，例如將 GET 請求指向一個階段中的 HTTP 代理整合，以及在另一個階段中的 Lambda 代理整合。

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。當您在 API Gateway 主控台中指定 Lambda 函數時，將彈出 AWS CLI 命令來設定適當的許可。您也可以使用下列 AWS CLI 命令來執行此操作。

```
aws lambda add-permission --function-name "arn:aws:lambda:us-
east-2:123456789012:function:my-function" --source-arn "arn:aws:execute-
api:us-east-2:123456789012:api_id/*/HTTP_METHOD/resource" --principal
 apigateway.amazonaws.com --statement-id apigateway-access --action
 lambda:InvokeFunction
```

使用映射範本傳遞資訊

您可以存取映射範本中的階段變數，或將組態參數傳送到 AWS Lambda 或 HTTP 後端。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。

若要使用階段變數，請先設定階段變數，然後為其指派值。例如，要自訂 HTTP 整合端點時，先建立 `url` 階段變數，然後在 API 的整合請求中輸入階段變數值 **`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。如需詳細資訊，請參閱the section called “在 API Gateway 設定 REST API 的階段變數”。

在 API Gateway 設定 REST API 的階段變數

這一節顯示如何使用 Amazon API Gateway 主控台，為範例 API 的兩個部署階段設定階段變數。若要了解如何在 API Gateway 中使用階段變數，建議您遵循本節中的所有程序。

先決條件

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。請遵循中的說明進行在 APIs Gateway 中開發 REST API
- 您必須至少已部署一次 API。請遵循中的說明進行在 API Gateway 中部署 REST API
- 您必須為已部署的 API 建立第一個階段。請遵循中的說明進行建立新的階段

利用具階段變數的 API 調用 HTTP 端點

此程序說明如何為 HTTP 端點建立階段變數，並為 API 建立兩個階段。此外，您可以建立階段變數 `url`、`stageName` 和 `function`，並將其用於本節中的下列程序。

利用具階段變數的 API 調用 HTTP 端點

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 建立 API，然後在 API 根資源上建立 GET 方法。將整合類型設定為 HTTP，並將端點 URL 設定為 **`http://${stageVariables.url}`**。
3. 將 API 部署至名為 **`beta`** 的新階段。
4. 在主導覽窗格中，選擇階段，然後選取 beta 階段。
5. 在階段變數索引標籤上，選擇編輯。
6. 選擇新增階段變數。
7. 對於名稱，輸入 **`url`**。針對值，輸入 **`httpbin.org/get`**。
8. 選擇新增階段變數，然後執行下列動作：

   對於名稱，輸入 **`stageName`**。針對值，輸入 **`beta`**。
9. 選擇新增階段變數，然後執行下列動作：

對於名稱，輸入 **function**。針對值，輸入 **HelloWorld**。

10. 選擇儲存。
11. 現在建立第二個階段。從階段導覽窗格選擇建立。針對階段名稱，輸入 **prod**。從部署選取最近的部署，然後選擇建立階段。
12. 如同在 beta 階段，請將相同的三個階段變數 (url、stageName 和 function) 分別設為不同的值 (**petstore-demo-endpoint.execute-api.com/petstore/pets**、**prod** 和 **HelloEveryone**)。
13. 在 Stages (階段) 導覽窗格中，選擇 beta。在階段詳細資訊下，選擇複製圖示以複製您的 API 的調用 URL，然後在 Web 瀏覽器中輸入您的 API 調用 URL。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。在 Web 瀏覽器中輸入 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，則在 Web 瀏覽器中輸入 URL 會傳回 `{"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。

14. 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。
15. 在 Stages (階段) 導覽窗格中，選擇 prod 階段。在階段詳細資訊下，選擇複製圖示以複製您的 API 的調用 URL，然後在 Web 瀏覽器中輸入您的 API 調用 URL。這會啟動 prod 階段在 API 根資源上的 GET 請求。
16. 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，以驗證結果。此值已在 prod 階段指派給 `url` 變數。兩個回應完全相同。

### 將階段特定的中繼資料傳遞至 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。我們會使用在上一個程序中宣告的 `stageName` 階段變數。

#### 將階段特定的中繼資料傳遞至 HTTP 後端

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至方法的 URL，請選擇方法請求索引標籤，然後在方法請求設定區段中，選擇編輯。

2. 選擇 URL 查詢字串參數，然後執行下列動作：

   a. 選擇新增查詢字串。

   b. 對於名稱，輸入 **stageName**。

   c. 將必要和快取保持關閉。

3. 選擇儲存。

4. 選擇整合請求索引標籤，然後在整合請求設定區段中，選擇編輯。

5. 對於端點 URL，將 **?stageName=${stageVariables.stageName}** 附加至先前定義的 URL 值，使整個端點 URL 成為 **http://${stageVariables.url}?stageName=${stageVariables.stageName}**。

6. 選擇部署 API，並選取 beta 階段。

7. 在主導覽窗格中，選擇階段。在 Stages (階段) 導覽窗格中，選擇 beta。在階段詳細資訊下，選擇複製圖示以複製您的 API 的調用 URL，然後在 Web 瀏覽器中輸入您的 API 調用 URL。

   > **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。

8. 您會收到以下回應。請注意，指派給 `beta` 階段變數的 `stageName`，在後端傳送為 `stageName` 引數。

```
{
  "args": {
    "stageName": "beta"
  },
  "headers": {
    "Accept": "application/json",
    "Host": "httpbin.org",
    "User-Agent": "AmazonAPIGateway_abcd1234",
    "X-Amzn-Apigateway-Api-Id": "abcd1234",
    "X-Amzn-Trace-Id": "Self=1-abcd-11111111111111;Root=1-11111111-111111111111111"
  },
  "origin": "192.0.2.9",
  "url": "http://httpbin.org/get?stageName=beta"
}
```

透過 API 使用階段變數調用 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。您使用的是在 `function` 中宣告的利用具階段變數的 API 調用 HTTP 端點 階段變數。

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。

透過 API 使用階段變數調用 Lambda 函數

1. 使用預設 Node.js 執行期建立名為 **`HelloWorld`** 的 Lambda 函數。程式碼必須包含下列內容：

   ```
   export const handler = function(event, context, callback) {
       if (event.stageName)
           callback(null, 'Hello, World! I\'m calling from the ' + event.stageName + '
    stage.');
       else
           callback(null, 'Hello, World! I\'m not sure where I\'m calling from...');
   };
   ```

   如需如何建立 Lambda 函數的詳細資訊，請參閱 REST API 主控台入門。
2. 在資源窗格中，選取建立資源，然後執行下列動作：

   a. 針對資源路徑，選取 /。

   b. 針對資源名稱，輸入 **`lambdav1`**。

   c. 選擇建立資源。
3. 選擇 /lambdav1 資源，然後選擇建立方法。

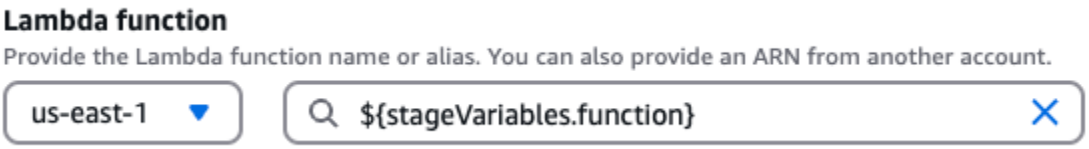
   然後，執行下列動作：

   a. 針對方法類型，選取 GET。

   b. 針對整合類型，選取 Lambda 函數。

   c. 將 Lambda 代理整合保持關閉。

   d. 對於 Lambda function (Lambda 函數)，請輸入 `${stageVariables.function}`。

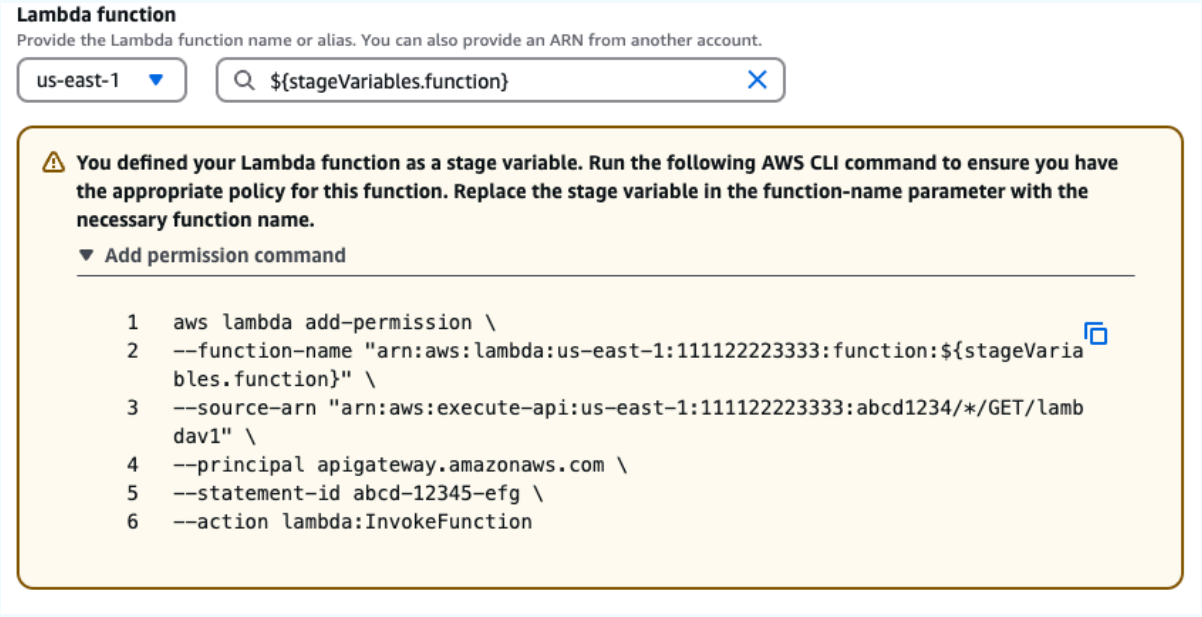



**Tip**

出現輸入新增許可命令的提示時，複製 add-permission 命令。對將要指派至 `function` 階段變數的每個 Lambda 函數執行命令。例如，如果 `$stageVariables.function` 值為 `HelloWorld`，則執行下列 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/lambdav1 --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。將 `${stageVariables.function}` 取代為指派給階段變數的 Lambda 函數名稱。




e. 選擇建立方法。

4. 將 API 同時部署到 prod 和 beta 階段。

5. 在主導覽窗格中，選擇階段。在 Stages (階段) 導覽窗格中，選擇 beta。在階段詳細資訊下，選擇複製圖示以複製您的 API 的調用 URL，然後在 Web 瀏覽器中輸入您的 API 調用 URL。將 **/lambdav1** 附加至 URL，再按下 Enter 鍵。

   您會收到以下回應。

   ```
   "Hello, World! I'm not sure where I'm calling from..."
   ```

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。您使用之前宣告的 `stageName` 階段變數，建立 POST 方法並輸入對應範本來產生承載。

透過階段變數將階段特定中繼資料傳送到 Lambda 函數

1. 選擇 /lambdav1 資源，然後選擇建立方法。

   然後，執行下列動作：

   a. 針對方法類型，選取 POST。

   b. 針對整合類型，選取 Lambda 函數。

   c. 將 Lambda 代理整合保持關閉。

   d. 對於 Lambda function (Lambda 函數)，請輸入 `${stageVariables.function}`。

   e. 出現輸入新增許可命令的提示時，複製 add-permission 命令。對將要指派至 `function` 階段變數的每個 Lambda 函數執行命令。

   f. 選擇建立方法。

2. 選擇整合請求索引標籤，然後在整合請求設定區段中，選擇編輯。

3. 選擇對應範本，然後選擇新增對應範本。

4. 針對內容類型，輸入 **application/json**。

5. 針對範本內文，輸入下列範本：

   ```
   #set($inputRoot = $input.path('$'))
   {
       "stageName" : "$stageVariables.stageName"
   }
   ```

> **Note**
>
> 在對應範本中，必須在有引號 (如 "$stageVariables.stageName" 或 "${stageVariables.stageName}") 的情況下參照階段變數。在其他地方，則必須在沒有引號的情況下參照 (如 ${stageVariables.function} 所示)。

6. 選擇儲存。
7. 將 API 同時部署到 beta 和 prod 階段。
8. 若要使用 REST API 用戶端傳遞階段特定的中繼資料，請執行下列動作：

   a. 在 Stages (階段) 導覽窗格中，選擇 beta。在階段詳細資訊下，選擇複製圖示以複製您的 API 的調用 URL，然後在 REST API 用戶端的輸入欄位中輸入您的 API 調用 URL。附加 **/lambdav1**，然後再提交請求。

      您會收到以下回應。

      ```
      "Hello, World! I'm calling from the beta stage."
      ```

   b. 在階段導覽窗格中，選擇 prod。在階段詳細資訊下，選擇複製圖示以複製您的 API 的調用 URL，然後在 REST API 用戶端的輸入欄位中輸入您的 API 調用 URL。附加 **/lambdav1**，然後再提交請求。

      您會收到以下回應。

      ```
      "Hello, World! I'm calling from the prod stage."
      ```

9. 若要使用測試功能來傳遞階段特定的中繼資料，請執行下列動作：

   a. 在資源導覽窗格中，選擇測試索引標籤。您可能需要選擇向右箭頭按鈕才能顯示此索引標籤。

   b. 針對 function，輸入 **HelloWorld**。

   c. 針對 stageName，輸入 **beta**。

   d. 選擇測試。您不需要新增內文至您的 POST 請求。

      您會收到以下回應。

      ```
      "Hello, World! I'm calling from the beta stage."
      ```

   e. 您可以重複前述步驟來測試 Prod 階段。針對 stageName，輸入 **Prod**。

您會收到以下回應。

```
"Hello, World! I'm calling from the prod stage."
```

### API Gateway 中 REST API 的 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。

- 不含通訊協定的完整 URI – http://${stageVariables.<variable_name>}
- 完整的網域 – http://${stageVariables.<variable_name>}/resource/operation
- 子網域 – http://${stageVariables.<variable_name>}.example.com/resource/operation
- 路徑 – http://example.com/${stageVariables.<variable_name>}/bar
- 查詢字串 – http://example.com/foo?q=${stageVariables.<variable_name>}

#### AWS 整合 URIs

階段變數可以用作 AWS URI 動作或路徑元件的一部分，如下列範例所示。

- `arn:aws:apigateway:<region>:<service>:${stageVariables.<variable_name>}`

AWS 整合 URIs(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:<region>:<account_id>:function:${stageVariables.<function_variable_name>}/invocations`
- `arn:aws:apigateway:<region>:lambda:path/2015-03-31/functions/arn:aws:lambda:<region>:<account_id>:function:<function_name>:${stageVariables.<version_variable_name>}/invocations`

> **Note**
>
> 若要使用 Lambda 函數的階段變數，函數必須與 API 位於相同的帳戶中。階段變數不支援跨帳戶 Lambda 函數。

Amazon Cognito 使用者集區

階段變數可用來取代 `COGNITO_USER_POOLS` 授權方的 Amazon Cognito 使用者集區。

- `arn:aws:cognito-idp:<region>:<account_id>:userpool/${stageVariables.<variable_name>}`

AWS 整合憑證

階段變數可以用作 AWS 使用者/角色登入資料 ARN 的一部分，如下列範例所示。

- `arn:aws:iam::<account_id>:${stageVariables.<variable_name>}`

## 設定 Amazon API Gateway Canary Release 部署

Canary Release 是一種軟體開發策略，此策略會部署 API (和其他軟體) 的新版本以供測試之用，而基本版本仍會部署為生產版本以供相同階段正常操作之用。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
- 建立 Canary Release 部署
- 更新 Canary Release
- 提升 Canary Release
- 關閉 Canary 版本

## 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。它們會回報至生產階段 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 SDK 在 API Gateway 中建立 Canary Release 部署。

主題

- 使用 API Gateway 主控台建立 Canary 部署
- 使用 建立 Canary 部署 AWS CLI

### 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 REST API，或建立新的 REST API。
3. 在主導覽窗格中，選擇資源，然後選擇部署 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。
4. 若要啟用 API 快取或將 AWS WAF Web ACL 與階段建立關聯，請在階段詳細資訊區段中，選擇編輯。如需詳細資訊，請參閱 the section called “快取設定” 或 the section called “使用 API Gateway 主控台將 AWS WAF Web ACL 與 API Gateway API 階段建立關聯” 。
5. 若要設定執行或存取記錄，請在日誌和追蹤區段中選擇編輯，並依照畫面上的指示進行。如需詳細資訊，請參閱在 API Gateway 中設定 REST API 的 CloudWatch 記錄功能。
6. 若要設定階段變數，請選擇階段變數索引標籤，並依照畫面上的指示新增或修改階段變數。如需詳細資訊，請參閱the section called “使用階段變數”。
7. 選擇 Canary 索引標籤，然後選擇建立 Canary。您可能需要選擇向右箭頭按鈕才能顯示 Canary 索引標籤。
8. 在 Canary 設定下，針對 Canary 輸入要轉移至 Canary 的請求百分比。
9. 如有需要，請選取階段快取以開啟 Canary 版本的快取功能。除非啟用 API 快取，否則快取不適用於 Canary Release。
10. 若要覆寫現有的階段變數，請針對 Canary 覆寫輸入新的階段變數值。

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. 每次 API 更新時，選擇部署 API。
2. 在部署 API 中，從部署階段下拉式清單中選擇包含 Canary 的階段。
3. (選用) 在部署描述中輸入描述。
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」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，而階段仍然使用的是初始 deploymentId 值。在幕後，主控台會呼叫 stage:update。

使用 建立 Canary 部署 AWS CLI

為新階段建立 Canary 部署

1. 使用以下 create-deployment 命令建立包含兩個階段變數但沒有 Canary 的部署：

```
aws apigateway create-deployment \
    --variables sv0=val0,sv1=val1 \
    --rest-api-id abcd1234 \
    --stage-name 'prod'
```

輸出將如下所示：

```
{
    "id": "du4ot1",
    "createdDate": 1511379050
}
```

2. 使用以下 create-deployment 命令在 prod 階段上建立 Canary 部署：

```
aws apigateway create-deployment \
    --rest-api-id abcd1234 \
    --canary-settings
 percentTraffic=10.5,stageVariableOverrides={sv1='val2',sv2='val3'},useStageCache=false
 \
    --stage-name 'prod'
```

輸出將如下所示：

```
{
    "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。

3. (選用) 使用以下 get-stage 命令來檢視階段表示：

```
aws apigateway get-stage --rest-api-id acbd1234 --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。

您也可以更新階段以啟用 Canary，藉此從現有的一般部署建立 Canary 發行部署。

從現有部署建立 Canary 發行部署

1. 使用 create-deployment 命令建立沒有 Canary 的部署：

```
aws apigateway create-deployment \
    --variables sv0=val0,sv1=val1 \
```

```
    --rest-api-id abcd1234 \
    --stage-name 'beta'
```

2. 使用 update-stage 命令更新階段以啟用 Canary：

```
aws apigateway update-stage \
    --rest-api-id abcd1234 \
    --stage-name 'beta' \
    --patch-operations '[{
            "op": "replace",
            "value": "0.0",
            "path": "/canarySettings/percentTraffic"
        }, {
            "op": "copy",
            "from": "/canarySettings/stageVariableOverrides",
            "path": "/variables"
        }, {
            "op": "copy",
            "from": "/canarySettings/deploymentId",
            "path": "/deploymentId"
        }]'
```

輸出將如下所示：

```
{
    "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": {}
}
```

因為您在現有 API 版本上啟用 Canary，所以生產發行 (`Stage`) 和 Canary 發行 (`canarySettings`) 會指向相同的部署。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 命令或 SDK 更新 Canary Release AWS 。

主題

- 使用 API Gateway 主控台更新 Canary Release
- 使用 更新 Canary 版本 AWS CLI

#### 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：

更新現有的 Canary 設定

1. 登入 API Gateway 主控台並選擇現有的 REST API。
2. 在主導覽窗格中，選擇階段，然後選擇現有的階段。
3. 選擇 Canary 索引標籤，然後選擇編輯。您可能需要選擇向右箭頭按鈕才能顯示 Canary 索引標籤。
4. 增加或減少 0.0 和 100.0 (含) 之間的百分比數字，以更新請求分佈。
5. 選取或清除階段快取核取方塊。
6. 新增、移除或修改 Canary 階段變數。

7. 選擇儲存。

使用 更新 Canary 版本 AWS CLI

若要使用 AWS CLI 更新 Canary，請使用 update-stage命令並修改 Canary 每個參數的修補程式操作。

如果 Canary 使用階段快取，以下 update-stage 命令就會更新：

```
aws apigateway update-stage \
    --rest-api-id {rest-api-id} \
    --stage-name '{stage-name}' \
    --patch-operations op=replace,path=/canarySettings/useStageCache,value=true
```

以下 update-stage 命令會更新 Canary 流量百分比：

```
aws apigateway update-stage \
    --rest-api-id {rest-api-id} \
    --stage-name '{stage-name}' \
    --patch-operations op=replace,path=/canarySettings/percentTraffic,value=25.0
```

以下 update-stage 會更新階段變數。此範例說明如何建立名為 newVar 的新階段變數、覆寫 var2 階段變數，以及移除 var1 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": "/canarySettings/percentTraffic",
        "value": "20.0"
    }, {
        "op": "replace",
        "path": "/canarySettings/useStageCache",
        "value": "true"
    }, {
        "op": "remove",
        "path": "/canarySettings/stageVariableOverrides/var1"
    }, {
        "op": "replace",
        "path": "/canarySettings/stageVariableOverrides/newVar",
        "value": "newVal"
    }, {
        "op": "replace",
        "path": "/canarySettings/stageVariableOverrides/val2",
        "value": "val4"
      }]'
```

### 提升 Canary Release

當您提升 Canary 版本時，Canary 版本會取代目前的階段設定。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。若要提升 Canary，請執行下列操作。

- 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%。

### 主題

- 使用 API Gateway 主控台提升 Canary Release

- 使用 提升 Canary Release AWS CLI

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：

提升 Canary 版本部署

1. 登入 API Gateway 主控台，然後在主導覽窗格中選擇現有 API。
2. 在主導覽窗格中，選擇階段，然後選擇現有的階段。
3. 選擇 Canary 索引標籤。
4. 選擇提升 Canary。
5. 確認要進行的變更，然後選擇提升 Canary。

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 驗證此項目 AWS CLI。如需範例，請參閱 the section called “使用 提升 Canary Release AWS CLI”。

使用 提升 Canary Release AWS CLI

若要使用 AWS CLI 命令將 Canary Release 提升為生產版本，請呼叫 `update-stage`命令來複製與`deploymentId`階段關聯的 Canary`deploymentId`，將 Canary 流量百分比重設為零 (`0.0`)，以及將任何 Canary 繫結階段變數複製到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"
    }
}
```

使用下列 update-stage 命令來提升 Canary：

```
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"
      }]'
```

輸出將如下所示：

```
{
    "_links": {
        ...
    },
    "accessLogSettings": {
        ...
    },
```

```
    "cacheClusterEnabled": false,
    "cacheClusterStatus": "NOT_AVAILABLE",
    "canarySettings": {
        "deploymentId": "eh1sby",
        "useStageCache": false,
        "stageVariableOverrides": {
            "sv2": "val3",
            "sv1": "val2"
        },
        "percentTraffic": 0
    },
    "createdDate": "2017-11-20T04:42:19Z",
    "deploymentId": "eh1sby",
    "lastUpdatedDate": "2017-11-22T05:29:47Z",
    "methodSettings": {
        ...
    },
    "stageName": "prod",
    "variables": {
        "sv2": "val3",
        "sv1": "val2"
    }
}
```

將 Canary 版本提升到該階段後，並不會停用 Canary，而該部署仍然是 Canary 版本部署。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 “關閉 Canary 版本”。

關閉 Canary 版本

若要關閉 Canary 版本部署，請將 `canarySettings` 設為 null，以將它從階段中移除。

您可以使用 API Gateway 主控台、 AWS CLI或 AWS SDK 停用 Canary Release 部署。

主題

- 使用 API Gateway 主控台關閉 Canary 版本
- 使用 關閉 Canary Release AWS CLI

使用 API Gateway 主控台關閉 Canary 版本

若要使用 API Gateway 主控台關閉 Canary 版本部署，請執行下列步驟：

關閉 Canary 版本部署

1. 登入 API Gateway 主控台，然後在主導覽窗格中選擇現有 API。
2. 在主導覽窗格中，選擇階段，然後選擇現有的階段。
3. 選擇 Canary 索引標籤。
4. 選擇 刪除。
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 驗證此項目 AWS CLI。如需範例，請參閱 the section called “使用 關閉 Canary Release AWS CLI”。

使用 關閉 Canary Release AWS CLI

以下 update-stage 命令會關閉 Canary 發行部署：

```
aws apigateway update-stage \
    --rest-api-id abcd1234 \
    --stage-name canary \
    --patch-operations '[{"op":"remove", "path":"/canarySettings"}]'
```

輸出看起來如下：

```
{
    "stageName": "prod",
    "accessLogSettings": {
        ...
    },
    "cacheClusterEnabled": false,
    "cacheClusterStatus": "NOT_AVAILABLE",
    "deploymentId": "nfcn0x",
    "lastUpdatedDate": 1511309280,
    "createdDate": 1511152939,
    "methodSettings": {
        ...
    }
}
```

如輸出中所示，已停用 Canary 部署的階段中不再有 canarySettings 屬性。

## 需要重新部署的 REST API 更新

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、 SDK CloudFormation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。

下表說明當您更新 API 時，需要重新部署 API 的 API 資源。

| 資源 | 備註 |
| --- | --- |
| ApiKey | 關於適用的屬性和支援的操作，請參閱 apikey:update。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 authorizer:update。更新需要重新部署 API。 |
| disableExecuteApiEndpoint | 更新需要修改 API 上的任何階段，例如將 API 重新部署到階段。 |
| DocumentationPart | 關於適用的屬性和支援的操作，請參閱 documentationpart:update。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 documentationversion:update。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 gatewayresponse:update。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 integration:update。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 integrationresponse:update。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 method:update。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 methodresponse:update。更新需要重新部署 API。 |

| 資源 | 備註 |
| --- | --- |
| 模型 | 關於適用的屬性和支援的操作，請參閱 model:update。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 requestvalidator:update。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 resource:update。更新需要重新部署 API。 |
| RestApi | 關於適用的屬性和支援的操作，請參閱 restapi:update。更新需要重新部署 API。這包括修改資源政策。 |
| VpcLink | 關於適用的屬性和支援的操作，請參閱 vpclink:update。更新需要重新部署 API。 |

下表說明在更新 API 時不需要重新部署 API 的 API 組態。

| Configuration | 備註 |
| --- | --- |
| 帳戶 | 關於適用的屬性和支援的操作，請參閱 account:update。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 deployment:update。 |
| DomainName | 關於適用的屬性和支援的操作，請參閱 domainname:update。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 basepathmapping:update。更新無須重新部署 API。 |
| IP 地址類型 | 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 stage:update。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 usage:update。更新無須重新部署 API。 |

| Configuration | 備註 |
| --- | --- |
| UsagePlan | 關於適用的屬性和支援的操作，請參閱 usageplan:update。更新無須重新部署 API。 |

## API Gateway 中公有 REST API 的自訂網域名稱

自訂網域名稱是更簡單且更直觀的 URL，可提供給 API 使用者。

部署 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* 是由 API Gateway 產生，*##*是 AWS 區域，而*##*是在部署 API 時由您指定。

URL 的主機名稱部分 (即 *api-id*.execute-api.*region*.amazonaws.com) 指的是 API 端點。預設 API 端點是隨機產生的，不容易取回，而且不方便使用。

使用自訂網域名稱，您就能設定 API 的主機名稱，並選擇基本路徑 (例如，myservice) 將替代 URL 映射至您的 API。例如，更方便使用者使用的 API 基本 URL 可以成為：

```
https://api.example.com/myservice
```

> **Note**
>
> 如需有關私有 API 的自訂網域名稱的資訊，請參閱 the section called “私有 API 的自訂網域名稱”。

### 考量事項

下列考量事項可能會影響您使用自訂網域名稱：

- 您可以停用 API 的預設端點。用戶端仍然可以連線到您的預設端點，但會收到 403 Forbidden 狀態碼。
- 區域性自訂網域名稱可與 REST API 和 HTTP API 相關聯。您可以使用 API Gateway 第 2 版 API 來建立及管理 REST API 的區域性自訂網域名稱。

- 自訂網域名稱在跨所有 AWS 帳戶的區域內必須是唯一的。
- 您可以在邊緣最佳化和區域端點之間遷移自訂網域名稱，但無法將公有自訂網域遷移至私有自訂網域名稱。
- 您必須建立或更新 DNS 提供者的資源記錄，以映射至您的 API 端點。如果沒有這種映射，送往自訂網域名稱的 API 請求無法到達 API Gateway。
- 您可以使用萬用字元憑證來支援幾乎不限數量的網域名稱，而不會超過預設配額。如需詳細資訊，請參閱the section called “萬用字元自訂網域名稱”。
- 您可以為您的自訂網域名稱選擇安全政策。如需詳細資訊，請參閱the section called “選擇安全政策”。
- 若要設定具有多個層級的 API 映射，您必須使用區域性自訂網域名稱和 TLS 1.2 安全政策。

## 自訂網域名稱的先決條件

以下是用於建立公有或私有自訂網域名稱的先決條件。如需有關私有 API 的自訂網域名稱的資訊，請參閱 the section called “私有 API 的自訂網域名稱”。

註冊網域名稱

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。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。您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。

您的網域名稱必須遵循 RFC 1035 規格，每個標籤最多可有 63 個八位元組，總共有 255 個八位元組。

指定自訂網域名稱所用的憑證。

必須為 API 設定自訂網域名稱前，您必須先在 ACM 中備妥 SSL/TLS 憑證。如果您 AWS 建立自訂網域名稱的區域中無法使用 ACM，則必須將憑證匯入該區域中的 API Gateway。

若要匯入 SSL/TLS 憑證，您必須提供 PEM 格式化 SSL/TLS 憑證內文、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。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。若要針對網域名稱使用 AWS 受管憑證，您只需參考其 ARN。

如果您的應用程式使用憑證鎖定，有時稱為 SSL 鎖定，來鎖定 ACM 憑證，則應用程式在 AWS 續約憑證後可能無法連線到您的網域。如需詳細資訊，請參閱 AWS Certificate Manager 使用者指南中的憑證關聯問題。

## 萬用字元自訂網域名稱

使用萬用字元自訂網域名稱，您可以支援幾乎無限數目的網域名稱，而不會超過預設配額。例如，您可以為每個客戶提供其自己的網域名稱，*customername*.example.com。

若要建立萬用字元自訂網域名稱，可以指定萬用字元 (*) 作為自訂網域的第一個子網域，藉以表示根網域所有可能的子網域。

例如，萬用字元自訂網域名稱 *.example.com 會產生如 a.example.com、b.example.com 和 c.example.com 等子網域。當您建立萬用字元自訂網域名稱時，其所有子網域都會透過萬用字元網域名稱的路由模式路由傳送。若要將子網域路由傳送至不同的 API，您可以執行下列其中一項操作：

- 使用 Host 標頭，以利用路由規則將傳入請求 *.example.com 路由傳送至不同的目標 REST API。如需詳細資訊，請參閱the section called “範例 4：萬用字元網域名稱的路由規則”。
- 針對您要路由傳送至不同端點的任何子網域，建立網域名稱。在單一 AWS 帳戶中，您可以同時擁有 *.example.com和 a.example.com。

您可以使用 $context.domainName 和 $context.domainPrefix 內容變數來判斷用戶端用來呼叫 API 的網域名稱。若要進一步了解環境變數，請參閱 用於 API Gateway 資料轉換的變數。

若要建立萬用字元自訂網域名稱，您必須提供由 ACM 發行並已經使用 DNS 或電子郵件驗證方法驗證的憑證。

> **Note**
>
> 如果不同的 AWS 帳戶已建立與萬用字元自訂網域名稱衝突的自訂網域名稱，則無法建立萬用字元自訂網域名稱。例如，如果帳戶 A 已建立 a.example.com，則帳戶 B 無法建立萬用字元自訂網域名稱 *.example.com。
> 如果帳戶 A 和帳戶 B 共用擁有者，您可以聯絡 AWS 支援中心，以請求例外狀況。

## 自訂網域名稱的後續步驟

以下是自訂網域名稱的後續步驟。

後續步驟

- 若要了解如何設定 SSL/TLS 憑證，請參閱 the section called “在 中備妥憑證 AWS Certificate Manager”。

- 若要了解如何建立區域性自訂網域名稱，請參閱 the section called “設定區域性自訂網域名稱”。
- 若要了解如何建立邊緣最佳化的自訂網域名稱，請參閱 the section called “設定 API Gateway 中的邊緣最佳化的自訂網域名稱”。
- 若要了解如何在區域和邊緣最佳化的自訂網域名稱之間遷移，請參閱 the section called “遷移自訂網域名稱”。
- 若要了解如何將 API 階段連線至自訂網域名稱，請參閱 the section called “在 API Gateway 中透過您的自訂網域名稱將流量傳送至 API”。
- 若要了解如何為您的自訂網域名稱選擇安全政策，請參閱 the section called “選擇安全政策”。
- 若要了解如何關閉自訂網域名稱的預設端點，請參閱 the section called “停用預設端點”。
- 若要了解如何使用 Route 53 運作狀態檢查從 API Gateway API 控制 DNS 備援，請參閱 the section called “DNS 備援”。

如果這是您第一次建立自訂網域名稱，建議您從 the section called “在 中備妥憑證 AWS Certificate Manager” 開始，指定您的憑證，然後 the section called “設定區域性自訂網域名稱” 以建立區域性自訂網域名稱。

## 在 中備妥憑證 AWS Certificate Manager

必須先在 中備妥 SSL/TLS 憑證，再設定 API 的自訂網域名稱 AWS Certificate Manager如需詳細資訊，請參閱「AWS Certificate Manager 使用者指南」。

### 考量事項

以下是 SSL/TLS 憑證的考量事項。

- 如果您建立邊緣最佳化的自訂網域名稱，API Gateway 會利用 CloudFront 支援自訂網域名稱的憑證。因此，自訂網域名稱 SSL/TLS 憑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。
- 如果您建立區域性自訂網域名稱，公有金鑰的大小上限為 2048。
- 若要搭配區域性自訂網域名稱使用 ACM 憑證，您必須在與 API 相同的區域中請求或匯入憑證。憑證必須涵蓋自訂網域名稱。
- 若要搭配邊緣最佳化的自訂網域名稱使用 ACM 憑證，您必須在美國東部 (維吉尼亞北部) – `us-east-1` 區域中請求或匯入憑證。

- 您必須具有已註冊的網域名稱，例如 *example.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。

建立 SSL/TLS 憑證或將其匯入 ACM

以下步驟顯示如何為網域名稱建立或匯入 SSL/TLS 憑證。

To request a certificate provided by ACM for a domain name

1. 登入 AWS Certificate Manager 主控台。
2. 選擇 Request a certificate (請求憑證)。
3. 對於憑證類型，選擇請求公有憑證。
4. 選擇下一步。
5. 在完整網域名稱中，輸入 API 的自訂網域名稱 (例如 `api.example.com`)。
6. (選擇性) 選擇 Add another name to this certificate (將其他名稱新增至此憑證)。
7. 對於驗證方法，選擇驗證網域擁有權的方法。
8. 對於金鑰演算法，選擇加密演算法。
9. 選擇請求。
10. 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。如果使用 Route 53 管理您的公有 DNS 記錄，您可以直接透過 ACM 更新您的記錄。

To import into ACM a certificate for a domain name

1. 取得憑證授權機構中自訂網域名稱的 PEM 編碼 SSL/TLS 憑證。如需這類 CA 的部分清單，請參閱 Mozilla 已包含 CA 清單。

   a. 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
   ```

   b. 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
   ```

   c. 將 CSR 提交至憑證授權機構，並儲存產生的憑證。

d. 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
> ```

2. 上傳憑證至 AWS Certificate Manager：

   a. 登入 AWS Certificate Manager 主控台。

   b. 選擇 Import a certificate (匯入憑證)。

   c. 針對憑證內文，輸入憑證授權單位的 PEM 格式化伺服器憑證的內文。以下示範這類憑證的縮寫範例。

   ```
   -----BEGIN CERTIFICATE-----
   EXAMPLECA+KgAwIBAgIQJ1XxJ8Pl++gOfQtj0IBoqDANBgkqhkiG9w0BAQUFADBB
   ...
   az8Cg1aicxLBQ7EaWIhhgEXAMPLE
   -----END CERTIFICATE-----
   ```

   d. 針對憑證私有金鑰，輸入 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. 針對憑證鏈，輸入 PEM 格式化中繼憑證，以及選擇性地逐一輸入根憑證，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/hEfcEejX/arxbx1BJCHBvlEPNnsdw8EXAMPLE
-----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. 選擇下一步，然後選擇下一步。

成功建立或匯入憑證後，請記下憑證 ARN。您在設定自訂網域名稱時需要此值。

## 在 API Gateway 中設定區域性自訂網域名稱

使用區域性自訂網域名稱來建立易於使用的 API 基礎 URL。有了區域性自訂網域名稱，您就可以將 HTTP 和 REST API 階段映射至相同的自訂網域名稱，並使用相互 TLS 驗證。

考量事項

以下是區域自訂網域名稱的考量事項：

- 您必須提供區域特定的 ACM 憑證。此憑證必須與您的 API 位於相同的區域。如需建立或上傳自訂網域名稱憑證的詳細資訊，請參閱「the section called “在 中備妥憑證 AWS Certificate Manager”」。
- 當您使用 ACM 憑證建立 (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。需要此服務連結角色，才能將您的 ACM 憑證附加至您的區域性端點。角色名為 AWSServiceRoleForAPIGateway，並會將 APIGatewayServiceRolePolicy 受管政策附加到該角色。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。
- 建立區域自訂網域名稱之後，您必須建立 DNS 記錄，以便將自訂網域名稱指向區域網域。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 Alias 記錄。如果您使用 Route 53 作為 DNS 供應商，請建立 A Alias 記錄。如果您使用第三方 DNS 供應商，請使用 CNAME 記錄。如果您使用 CNAME 記錄，並建立

API Gateway 介面 VPC 端點且為私有 API 啟用私有 DNS，則無法在託管您私有 API 的 VPC 內解析自訂網域名稱。

### 建立區域性自訂網域名稱

下列程序顯示如何建立區域性自訂網域名稱。完成此程序後，您會建立路由規則來將 API 的階段路由傳送至自訂網域名稱。

#### AWS 管理主控台

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 從主要導覽窗格中，選擇 Custom Domain Names (自訂網域名稱)。
3. 選擇 Create (建立)。
4. 在 Domain name (網域名稱) 中，輸入網域名稱。
5. 對於路由模式，選擇僅限路由規則。

   在此路由模式中，您只能使用路由規則將流量從自訂網域名稱傳送至 API。如需詳細資訊，請參閱the section called “在 API Gateway 中透過您的自訂網域名稱將流量傳送至 API”。
6. 針對最低 TLS 版本，選取版本。
7. 在端點組態下，針對 API 端點類型，選擇區域性。
8. 選擇 ACM 憑證。憑證必須與 API 位於相同的區域。
9. 選擇 Create (建立)。

#### AWS CLI

以下 create-domain-name 命令會建立自訂網域名稱：

```
aws apigatewayv2 create-domain-name \
    --domain-name 'regional.example.com' \
    --domain-name-configurations CertificateArn=arn:aws:acm:us-
west-2:123456789012:certificate/123456789012-1234-1234-1234-12345678 \
    --routing-mode ROUTING_RULE_ONLY
```

輸出將如下所示：

```
{
    "ApiMappingSelectionExpression": "$request.basepath",
    "DomainName": "regional.example.com",
```

```
    "DomainNameConfigurations": [
        {
            "ApiGatewayDomainName": "d-numh1z56v6.execute-api.us-
west-2.amazonaws.com",
            "CertificateArn": "arn:aws:acm:us-
west-2:123456789012:certificate/123456789012-1234-1234-1234-12345678",
            "DomainNameStatus": "AVAILABLE",
            "EndpointType": "REGIONAL",
            "HostedZoneId": "Z2OJLYMUO9EFXC",
            "SecurityPolicy": "TLS_1_2"
        }
        "RoutingMode": "ROUTING_RULE_ONLY"
    ]
}
```

DomainNameConfigurations 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。

為您的區域自訂網域名稱建立路由規則

建立自訂網域名稱後，您可以設定將流量從自訂網域名稱路由傳送至 API 的方式。由於您將路由模式設定為 ROUTING_RULE_ONLY，因此您會使用路由規則將自訂網域名稱的傳入請求路由傳送至您的 API。

在此範例中，您會建立全部擷取規則，以便將自訂網域名稱的所有傳入請求路由傳送至 API 的某一個階段。您也可以根據不同的標頭和路徑條件設定路由規則。如需詳細資訊，請參閱the section called “路由規則”。

AWS 管理主控台

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 選擇自訂網域名稱。
3. 在路由詳細資訊索引標籤上，選擇新增路由規則。
4. 選擇新增條件以新增條件。
5. 保留此規則且不設任何條件。這會將自訂網域名稱的所有請求路由傳送至您的目標 API 和目標階段。
6. 針對動作，使用下拉式清單選取您的目標 API 和目標階段。
7. 選擇下一步。

8. 在優先順序欄位中，輸入 **100**。

   API Gateway 會依優先順序評估規則，從最低值到最高值。由於這是全部擷取規則，因此您會使用高優先順序讓 API Gateway 能夠比對您先建立的任何其他規則。

9. 選擇建立路由規則。

AWS CLI

以下 `create-routing-rule` 命令會建立全部擷取路由規則：

```
aws apigatewayv2 create-routing-rule \
  --domain-name 'regional.example.com' \
  --priority 100 \
  --conditions  \
  --actions '[{
    "InvokeApi": {
      "ApiId": "a1b2c3",
      "Stage": "prod"
    }
  }]'
```

您隨時可以變更路由模式並建立新規則。如需詳細資訊，請參閱the section called “在 API Gateway 中透過您的自訂網域名稱將流量傳送至 API”。

為您的區域性自訂網域名稱建立 DNS 記錄

建立自訂網域名稱和建立基本路徑映射後，您可以建立 DNS 記錄，將自訂網域名稱指向新建立的區域性網域名稱。

AWS 管理主控台

若要使用 AWS 管理主控台，請遵循 Route 53 文件設定 Route 53 將流量路由到 API Gateway。

AWS CLI

若要設定您的 DNS 記錄，將區域性自訂網域名稱映射至指定託管區域 ID 的主機名稱，請先建立 JSON 檔案，其中包含設定區域性網域名稱 DNS 記錄的組態。

下列 `setup-dns-record.json` 顯示如何建立 DNS A 記錄，以將區域性自訂網域名稱 (`regional.example.com`) 映射至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 Amazon API Gateway 端點和配額中取得區域性 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": "Z2OJLYMUO9EFXC",
          "EvaluateTargetHealth": false
        }
      }
    }
  ]
}
```

以下 change-resource-record-sets 會為您的區域自訂網域名稱建立 DNS 記錄：

```
aws route53 change-resource-record-sets \
    --hosted-zone-id Z2OJLYMUO9EFXC \
    --change-batch file://path/to/your/setup-dns-record.json
```

將 hosted-zone-id 取代為您帳戶中所設定之 DNS 記錄的 Route 53 託管區域 ID。change-batch 參數值會指向資料夾 (*path/to/your*) 中的 JSON 檔案 (*setup-dns-record.json*)。

## 設定 API Gateway 中的邊緣最佳化的自訂網域名稱

當您針對邊緣最佳化的 API 建立自訂網域名稱，API Gateway 會設定 CloudFront 分佈和 DNS 記錄，以將 API 網域名稱映射至 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。此對應適用於要透過對應 CloudFront 分佈路由至 API Gateway 的 API 請求 (繫結至自訂網域名稱)。

### 考量事項

以下是邊緣最佳化的自訂網域名稱的考量事項：

- 若要設定邊緣最佳化自訂網域名稱或更新其憑證，您必須有更新 CloudFront 分佈的許可。

  更新 CloudFront 分佈需要下列權限：

  JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
         {
            "Sid": "AllowCloudFrontUpdateDistribution",
            "Effect": "Allow",
            "Action": [
                "cloudfront:updateDistribution"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

- 您必須為美國東部 (維吉尼亞北部) – `us-east-1` 區域中的邊緣最佳化的自訂網域名稱請求或匯入憑證。
- 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 CloudTrail 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。
- 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 開發人員指南》中的使用備用網域名稱和 HTTPS
- 邊緣最佳化的自訂網域名稱大約需要 40 分鐘才能準備就緒。
- 建立邊緣最佳化的自訂網域名稱之後，您必須建立 DNS 記錄，將自訂網域名稱映射至 CloudFront 分佈名稱。

### 建立邊緣最佳化的自訂網域名稱

下列程序說明如何建立 API 的邊緣最佳化自訂網域名稱。

AWS 管理主控台

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 從主要導覽窗格中，選擇 Custom Domain Names (自訂網域名稱)。
3. 選擇新增網域名稱。
4. 在 Domain name (網域名稱) 中，輸入網域名稱。
5. 針對路由模式，選擇 API_MAPPING_ONLY。
6. 對於 API 端點類型，選擇邊緣最佳化。
7. 選擇最低版本的 TLS。
8. 選擇 ACM 憑證。
9. 選擇新增網域名稱。

REST API

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，您需要用該名稱來設定自訂網域之 DNS 中的 A 記錄別名目標。

如需此 REST API 呼叫的程式碼範例，請參閱 domainname:create。

邊緣最佳化的自訂網域名稱大約需要 40 分鐘才能準備好，但主控台會立即以 *`distribution-id`*`.cloudfront.net` 的形式顯示關聯的 CloudFront 分佈網域名稱，以及憑證 ARN。同時，您可以建立基本路徑映射或路由規則，然後設定 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。

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

由於您將路由模式設定為 `API_MAPPING_ONLY`，因此您可以使用基本路徑映射，以將單一自訂網域名稱作為多個 API 的主機名稱使用。這可透過自訂網域名稱與相關聯基本路徑的組合，使 API 變成可存取。

例如，若您已在 API Gateway 中建立名為 PetStore 的 API 和名為 Dogs 的另一個 API，並設定 api.example.com 的自訂網域名稱，則可以將 PetStore API 的 URL 設定為 https://api.example.com。

這會建立 PetStore API 與空字串的基本路徑的關聯。如果您將 PetStore API 的 URL 設定為 https://api.example.com/PetStore，這會建立 PetStore API 與 PetStore 的基本路徑的關聯。您可以指派 Dogs API 的 MyDogList 的基本路徑。https://api.example.com/MyDogList URL 接著會是 Dogs API 的根 URL。

若要在多個層級上設定 API 映射，您只能使用區域性自訂網域名稱。不支援邊緣最佳化的自訂網域名稱。如需詳細資訊，請參閱the section called “API 映射”。

下列程序會設定 API 映射，以將自訂網域名稱的路徑映射至 API 階段。

AWS 管理主控台

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 從 API Gateway 主控台主要導覽窗格中，選擇 Custom Domain Names (自訂網域名稱)。
3. 選擇自訂網域名稱。
4. 選擇 Configure API mappings (設定 API 映射)。
5. 選擇 Add new mapping (新增對應)。
6. 指定對應的 API、Stage (階段) 及 Path (路徑) (選擇性)。
7. 選擇儲存。

REST API

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。

如要以更有彈性的方式將流量路由至 API，您可以將路由模式變更為 ROUTING_RULE_ONLY 或 ROUTING_RULE_THEN_API_MAPPING，並建立路由規則。如需詳細資訊，請參閱the section called “在 API Gateway 中透過您的自訂網域名稱將流量傳送至 API”。

為您的邊緣最佳化的自訂網域名稱建立 DNS 記錄

開始建立邊緣最佳化的自訂網域名稱後，請設定 DNS 記錄別名。

建議您使用 Route 53，為自訂網域名稱建立 A 記錄別名，並指定 CloudFront 分佈網域名稱做為別名目標。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 開發人員指南》中的在別名與非別名資源記錄集之間選擇。

有關 Amazon Route 53 的指示，請參閱《Amazon Route 53 開發人員指南》中的使用您的網域名稱將流量路由到 Amazon API Gateway API。

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。這些日誌可於 `us-east-1` 中取得。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。

| 區域 | 帳戶 ID |
| --- | --- |
| us-east-1 | 392220576650 |
| us-east-2 | 718770453195 |
| us-west-1 | 968246515281 |
| us-west-2 | 109351309407 |
| ca-central-1 | 796887884028 |
| eu-west-1 | 631144002099 |
| eu-west-2 | 544388816663 |
| eu-west-3 | 061510835048 |
| eu-central-1 | 474240146802 |
| eu-central-2 | 166639821150 |
| eu-north-1 | 394634713161 |

| 區域 | 帳戶 ID |
| --- | --- |
| eu-south-1 | 753362059629 |
| eu-south-2 | 359345898052 |
| ap-northeast-1 | 969236854626 |
| ap-northeast-2 | 020402002396 |
| ap-northeast-3 | 360671645888 |
| ap-southeast-1 | 195145609632 |
| ap-southeast-2 | 798376113853 |
| ap-southeast-3 | 652364314486 |
| ap-southeast-4 | 849137399833 |
| ap-south-1 | 507069717855 |
| ap-south-2 | 644042651268 |
| ap-east-1 | 174803364771 |
| sa-east-1 | 287228555773 |
| me-south-1 | 855739686837 |
| me-central-1 | 614065512851 |

**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。您不需要輪換自訂網域名稱的任何 ACM 發出憑證。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 the section called “建立 SSL/TLS 憑證或將其匯入 ACM” 中的步驟。

下列程序說明如何輪換網域名稱的憑證。

> **Note**
>
> 輪換匯入 ACM 的憑證大約需要 40 分鐘。

AWS 管理主控台

1. 在 ACM 中請求或匯入憑證。
2. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
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. 選擇邊緣最佳化的自訂網域名稱。
5. 針對端點組態，選擇編輯。
6. 針對 ACM 憑證，從下拉式清單選取憑證。
7. 選擇儲存變更，開始輪換自訂網域名稱的憑證。

REST API

呼叫 domainname:update 動作，並指定所指定網域名稱之新 ACM 憑證的 ARN。

AWS CLI

以下 update-domain-name 會更新邊緣最佳化網域名稱的 ACM 憑證。

```
aws apigateway update-domain-name \
    --domain-name edge.example.com \
    --patch-operations "op='replace',path='/certificateArn',value='arn:aws:acm:us-
east-2:111122223333:certificate/CERTEXAMPLE123EXAMPLE'"
```

以下 update-domain-name 會更新區域網域名稱的 ACM 憑證。

```
aws apigateway update-domain-name \
    --domain-name regional.example.com \
```

```
    --patch-operations "op='replace',path='/
regionalCertificateArn',value='arn:aws:acm:us-east-2:111122223333:certificate/
CERTEXAMPLE123EXAMPLE'"
```

當您使用基本路徑映射時，使用自訂網域名稱呼叫您的 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。

| API ID | 階段 | 預設 URL | 基本路徑 | 自訂 URL |
|---|---|---|---|---|
| udxjef | prod | https://udxjef.execute-api.us-east-1.amazonaws.com/prod | /petstore | https://api.example.com/petstore |
| udxjef | tst | https://udxjef.execute-api.us-east-1.amazonaws.com/tst | /petdepot | https://api.example.com/petdepot |
| qf3duz | dev | https://qf3duz.execute-api.us-east-1.amazonaws.com/dev | /bookstore | https://api.example.com/bookstore |
| qf3duz | tst | https://qf3duz.execute-api.us-east-1.amazonaws.com/tst | /bookstand | https://api.example.com/bookstand |

如要以更有彈性的方式將流量路由至 API，您可以建立路由規則。如需詳細資訊，請參閱the section called “在 API Gateway 中透過您的自訂網域名稱將流量傳送至 API”。

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。

## 將自訂網域名稱遷移至 API Gateway 中不同的 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。您無法將公有自訂網域名稱遷移至私有自訂網域名稱。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。最後，移除過時的自訂網域名稱端點。

### 考量事項

以下是在區域 API 端點和邊緣最佳化的 API 端點之間移轉自訂網域的考量：

- 邊緣最佳化的自訂網域名稱需要由美國東部 (維吉尼亞北部) – `us-east-1` 區域的 ACM 所提供的憑證。此憑證會分發至所有地理位置。
- 區域性自訂網域名稱需要由託管 API 的相同區域中的 ACM 所提供的憑證。您可以將不在 `us-east-1` 區域中的邊緣最佳化的自訂網域名稱，遷移至區域性自訂網域名稱，方法是請求來自 API 本端區域的新 ACM 憑證。
- 這可能需要最多 60 秒的時間才能完成邊緣最佳化的自訂網域名稱與區域性自訂網域名稱之間的遷移。遷移時間也取決於您何時更新 DNS 記錄。
- 只有在端點存取模式設定為 時，您才能新增額外的端點組態`BASIC`。一旦您有兩個端點組態，就無法變更端點存取模式。如需詳細資訊，請參閱the section called “端點存取模式”。
- 如果您的自訂網域名稱使用開頭為 的安全政策`SecurityPolicy_`，當您新增新的端點組態類型時，兩個端點類型的端點存取模式都相同，而且您必須為新的端點組態類型選擇開頭`SecurityPolicy_`為 的安全政策。

### 遷移自訂網域名稱

> **Note**
>
> 若要完成移轉，請務必從自訂網域名稱中移除過時的端點。

以下程序顯示如何將邊緣最佳化的自訂網域名稱遷移至區域性自訂網域。

AWS 管理主控台

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 從主要導覽窗格中，選擇 Custom Domain Names (自訂網域名稱)。
3. 選擇邊緣最佳化的自訂網域名稱。
4. 針對端點組態，選擇編輯。
5. 選擇新增區域端點。
6. 針對 ACM 憑證，選擇憑證。

   區域憑證必須在與區域 API 相同的區域。
7. 選擇儲存變更。
8. 設定 DNS 記錄，以將區域性自訂網域名稱指向此區域性主機名稱。如需詳細資訊，請參閱設定 Route 53 以將流量轉送至 API Gateway。
9. 確認 DNS 組態使用正確的端點後，您即可刪除邊緣最佳化的端點組態。選擇您的自訂網域名稱，然後針對邊緣最佳化的端點組態，選擇刪除。
10. 確認您的選擇並刪除端點。

AWS CLI

以下 update-domain-name 命令會將邊緣最佳化的自訂網域名稱移轉至區域自訂網域名稱：

```
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。

輸出將如下所示：

```
{
    "certificateArn": "arn:aws:acm:us-
east-1:123456789012: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-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，您可以將邊緣最佳化的自訂網域名稱移除。以下 update-domain-name 命令會移除邊緣最佳化的自訂網域名稱：

```
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"}
        ]'
```

下列程序說明如何將使用增強型安全政策的邊緣最佳化自訂網域名稱遷移至也使用增強型安全政策的區域性自訂網域名稱。

AWS 管理主控台

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 從主要導覽窗格中，選擇 Custom Domain Names (自訂網域名稱)。
3. 選擇邊緣最佳化的自訂網域名稱。
4. 針對端點組態，選擇編輯。
5. 選擇新增區域端點。

6. 針對 ACM 憑證，選擇憑證。

   區域憑證必須在與區域 API 相同的區域。
7. 針對安全政策，選擇開頭為 的安全政策`SecurityPolicy_`。
8. 針對端點存取模式，選擇基本。
9. 選擇儲存變更。
10. 設定 DNS 記錄，以將區域性自訂網域名稱指向此區域性主機名稱。如需詳細資訊，請參閱設定 Route 53 以將流量轉送至 API Gateway。
11. 確認 DNS 組態使用正確的端點後，您即可刪除邊緣最佳化的端點組態。選擇您的自訂網域名稱，然後針對邊緣最佳化的端點組態，選擇刪除。
12. 確認您的選擇並刪除端點。

AWS CLI

以下 update-domain-name 命令會將邊緣最佳化的自訂網域名稱移轉至區域自訂網域名稱：

```
aws apigateway update-domain-name \
    --domain-name 'api.example.com' \
    --patch-operations  '[
        { "op":"add", "path": "/endpointConfiguration/types","value": "REGIONAL" },
        { "op":"replace", "path": "/securityPolicy",
 "value":"SecurityPolicy_TLS13_1_3_2025_09"},
        { "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。

輸出將如下所示：

```
{
    "certificateArn": "arn:aws:acm:us-
east-1:123456789012: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"
        ]
    },
    "securityPolicy": "SecurityPolicy_TLS13_1_3_2025_09",
    "endpointAccessMode": "BASIC",
    "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，您可以將邊緣最佳化的自訂網域名稱移除。以下 update-domain-name 命令會移除邊緣最佳化的自訂網域名稱：

```
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"}
        ]'
```

以下程序顯示如何將區域性自訂網域名稱遷移至邊緣最佳化的自訂網域名稱。

AWS 管理主控台

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 在主要導覽窗格中，選擇自訂網域名稱。
3. 選擇區域性自訂網域名稱。
4. 針對端點組態，選擇編輯。
5. 選擇新增邊緣最佳化端點。
6. 針對 ACM 憑證，選擇憑證。

   在 `us-east-1` 區域中，必須建立邊緣最佳化網域憑證。
7. 選擇儲存。

8. 設定 DNS 記錄，將邊緣最佳化的自訂網域名稱指向此邊緣最佳化的主機名稱。如需詳細資訊，請參閱設定 Route 53 以將流量轉送至 API Gateway。
9. 確認 DNS 組態使用正確的端點後，您就會刪除區域性端點組態。選擇您的自訂網域名稱，然後針對區域性端點組態，選擇刪除。
10. 確認您的選擇並刪除端點。

AWS CLI

以下 update-domain-name 命令會將區域自訂網域名稱移轉至邊緣最佳化的自訂網域名稱：

```
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-1:738575810317: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。

輸出將如下所示：

```
{
    "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-cgkq2qwgzf.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。以下 update-domain-name 命令會移除區域端點類型：

```
aws apigateway update-domain-name \
    --domain-name api.example.com \
    --patch-operations '[
        {"op":"remove", "path":"/endpointConfiguration/types", value:"REGIONAL"},
        {"op":"remove", "path":"regionalCertificateArn"}
      ]'
```

輸出看起來如下：

```
{
    "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"
    }
}
```

## 在 API Gateway 中透過您的自訂網域名稱將流量傳送至 API

當您為自訂網域名稱設定路由模式時，您會設定傳入流量導向 API 的方式。您會使用路由規則、API 映射，或是路由規則與 API 映射，將流量傳送至您的 API。下節說明何時使用路由規則、何時使用 API 映射，以及如何為您的自訂網域名稱設定路由模式。

### 何時使用路由規則

當您使用路由規則時，您會將符合特定條件的傳入請求導向特定 REST API 階段。例如，如果規則包含標頭 `version:v1` 和基本路徑 `/users`，則可將請求路由至 `users` REST API 的 `production` 階段。使用路由規則來建立進階動態路由拓撲，以支援如 A/B 測試等使用案例，或增加新版 API 的使用率。

建議您將流量導向 REST API 時，針對自訂網域名稱使用路由規則。您可以使用路由規則重新建立任何 API 映射。如需詳細資訊，請參閱the section called “使用路由規則重新建立 API 映射”。

對於 REST API，您也可以同時使用路由規則和 API 映射。若您同時使用路由規則和 API 映射，則 API Gateway 一律會先評估路由規則，再評估任何 API 映射。同時使用路由規則和 API 映射來移轉您目前的自訂網域名稱，或探索路由規則。

路由規則的考量事項

下列考量事項可能會影響您使用路由規則：

- 不支援使用 WebSocket 或 HTTP API 作為路由規則的目標 API。
- 如果您的自訂網域名稱同時具有 REST 和 HTTP API 的 API 映射，則不支援路由規則。
- 您可以建立私有自訂網域至私有 REST API 的路由規則。您可以建立公有自訂網域至區域或邊緣最佳化 API 的路由規則。
- 您無法建立公有自訂網域至私有 API 的路由規則。您無法建立私有自訂網域名稱至公有 API 的路由規則。

選擇路由規則或是 API 映射

建議您盡可能使用路由規則。只有在將流量傳送至 HTTP 或 WebSocket API 時才使用 API 映射。

為您的自訂網域名稱設定路由模式

您可以選擇 API Gateway 使用何種路由模式來將流量路由傳送至您的 API。如需詳細資訊，請參閱the section called “在 API Gateway 中透過您的自訂網域名稱將流量傳送至 API”。本節討論自訂網域名稱的路由模式。您必須為自訂網域名稱設定路由模式，才能將流量路由傳送至您的 API。以下是支援的路由模式：

- ROUTING_RULE_THEN_API_MAPPING – 使用此模式時，可同時透過路由規則和 API 映射將流量傳送至 API。在此模式中，所有路由規則的優先順序都會高於任何 API 映射。如需此模式的範例，請參閱 the section called “範例 2：路由規則和 API 映射”。
- ROUTING_RULE_ONLY – 使用此模式時，只允許路由規則將流量傳送至您的 API。當您的自訂網域名稱使用此模式時，您無法建立 API 映射，但您可以使用 get-api-mappings 命令來檢視這些映射。API 發起人無法使用 API 映射來存取此網域名稱。
- API_MAPPING_ONLY – 使用此模式時，只允許 API 映射將流量傳送至您的 API。當您的自訂網域名稱使用此模式時，您無法建立路由規則，但您可以使用 `list-routing-rules` 命令來檢視這些規則。API 發起人無法使用路由規則來存取此網域名稱。

這是您所有現有網域名稱以及您建立的任何新網域名稱的預設路由模式。

當您使用 apigateway 建立自訂網域名稱時，API_MAPPING_ONLY 稱為 BASE_PATH_MAPPING_ONLY，而 ROUTING_RULE_THEN_API_MAPPING 稱為 ROUTING_RULE_THEN_BASE_PATH_MAPPING。此行為僅適用於 AWS CLI CloudFormation或任何 SDKs，而不是在 中 AWS 管理主控台。

下列程序說明如何變更現有自訂網域名稱的路由模式。當您變更自訂網域名稱的路由模式時，API 發起人無法使用任何不支援的路由模式來存取您的網域名稱。

AWS 管理主控台

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 從主要導覽窗格中，選擇 Custom Domain Names (自訂網域名稱)。
3. 選擇自訂網域名稱。
4. 針對網域詳細資訊，選擇編輯。
5. 針對路由模式，選擇 ROUTING_RULE_THEN_API_MAPPING。
6. 選擇儲存。

如果您將路由模式變更為 ROUTING_RULE_ONLY 或 API_MAPPING_ONLY，則您已建立的任何 API 映射或路由規則都會從主控台的網域名稱詳細資訊頁面中移除。如果您變更路由模式以支援路由規則或 API 映射，則會傳回這些資源。

AWS CLI - apigatewayv2

以下 update-domain-name 命令會更新網域名稱，以使用路由模式 ROUTING_RULE_THEN_API_MAPPING：

```
aws apigatewayv2 update-domain-name \
  --domain-name 'api.example.com' \
  --routing-mode "ROUTING_RULE_THEN_API_MAPPING"
```

輸出將如下所示：

```
{
"ApiMappingSelectionExpression": "$request.basepath",
"DomainName": "api.example.com",
```

```
"DomainNameArn": "arn:aws:apigateway:us-west-2::/domainnames/api.example.com",
"DomainNameConfigurations": [
  {
      "ApiGatewayDomainName": "d-abcdefg.execute-api.us-west-2.amazonaws.com",
      "CertificateArn": "arn:aws:acm:us-west-2:111122223333:certificate/
abcdefg-123456-abcdefg",
      "DomainNameStatus": "AVAILABLE",
      "EndpointType": "REGIONAL",
      "HostedZoneId": "Z2OJLYMUO9EFXC",
      "SecurityPolicy": "TLS_1_2"
   }
 ],
"RoutingMode": "ROUTING_RULE_THEN_API_MAPPING",
"Tags": {}
}
```

AWS CLI - apigateway

以下 update-domain-name 命令會更新私有自訂網域名稱，以使用路由模式 ROUTING_RULE_THEN_BASE_PATH_MAPPING：

```
aws apigateway update-domain-name \
  --domain-name 'private.example.com' \
  --patch-operations "op='replace',path='/
routingMode',value='ROUTING_RULE_THEN_BASE_PATH_MAPPING'"
```

輸出將如下所示：

```
{
"domainName": "private.example.com",
"domainNameId": "abcd1234",
"domainNameArn": "arn:aws:apigateway:us-west-2:111122223333:/domainnames/
private.example.com+abcd1234",
"certificateArn": "arn:aws:acm:us-west-2:111122223333:certificate/
a1b2c3d4-5678-90ab-cdef",
"certificateUploadDate": "2024-09-10T10:31:20-07:00",
"endpointConfiguration": {
  "types": [
    "PRIVATE"
   ],
  "ipAddressType": "dualstack"
  },
"domainNameStatus": "AVAILABLE",
```

```
"securityPolicy": "TLS_1_2",
"policy": "...",
"routingMode" : "ROUTING_RULE_THEN_BASE_PATH_MAPPING"
}
```

將 API 階段連線至 REST API 自訂網域名稱的路由規則

路由規則是一組條件，若符合條件，則會調用動作。例如，規則可以將包含 `Hello:World` 標頭且包含基本路徑 `users` 之自訂網域名稱的任何傳入請求，路由傳送至 REST API 的 `production` 階段。

規則會依優先順序加以評估，若您將路由模式設定為 `ROUTING_RULE_THEN_API_MAPPING`，則 API Gateway 一律會先評估所有路由規則，再評估任何 API 映射。下列清單說明路由規則如何使用條件、動作和優先順序。

條件

滿足規則的條件時，即會執行它的動作。API Gateway 最多可支援兩個標頭條件和一個路徑條件。API Gateway 會一起評估標頭條件和基本路徑條件。

您可以建立規則，而不設任何條件。當 API Gateway 評估此規則時，一律會執行動作。您可以建立規則，而不設任何條件，以作為全部擷取規則。

如需標頭條件的詳細資訊，請參閱 the section called “比對標頭條件”。如需路徑條件的詳細資訊，請參閱 the section called “比對基本路徑條件”。

動作

動作是符合路由規則條件的結果。目前，唯一支援的動作是調用 REST API 的階段。

每一項規則可以有一個動作。

Priority

優先順序決定評估規則的順序，從最低值到最高值。規則不可具有相同的優先順序。

您可以設定的優先順序範圍為 1 到 1,000,000。如果規則的優先順序為 1，則 API Gateway 會先評估此規則。我們建議您在建立規則時，於優先順序之間插入間隙。這樣做有助於切換規則的優先順序，並新增新規則。如需詳細資訊，請參閱the section called “變更路由規則的優先順序”。

如需 API Gateway 如何評估路由規則的範例，請參閱 the section called “API Gateway 如何評估路由規則的範例”。

### API Gateway 路由規則條件類型

下節說明路由規則條件類型。API Gateway 只有在所有條件都為 true 時才符合規則。

#### 比對標頭條件

建立標頭條件時，您可以比對標頭名稱和標頭 glob 值，例如 `Hello:World`。API Gateway 會使用常值比對來驗證比對標頭條件。透過在標頭之間使用 AND 的方式，您的條件最多能使用兩個標頭。例如，您的條件可以比對傳入請求是否包含 `Hello:World` 和 `x-version:beta`。

標頭名稱比對不區分大小寫，但標頭 glob 值區分大小寫。`Hello:World` 會比對 `hello:World`，但不會比對 `Hello:world`。

如需限制的標頭值清單，請參閱 the section called “限制”。

#### 使用萬用字元搭配標頭條件

您只能在標頭 glob 值中使用萬用字元，且萬用字元必須是 `*prefix-match`、`suffix-match*` 或 `*contains*`。下表顯示如何使用萬用字元進行標頭條件比對的範例。

| 標頭條件 | 符合路由規則的請求 | 不符合路由規則的請求 |
| --- | --- | --- |
| `x-version: a*` | • `x-version: account`<br>• `x-version: alpha` | • `x-version: backup`<br>• `x-version: beta`<br>• `x-version: users` |
| `x-version: *a` | • `x-version: alpha`<br>• `x-version: beta` | • `x-version: account`<br>• `x-version: backup`<br>• `x-version: users` |
| `x-version: *a*` | • `x-version: account`<br>• `x-version: alpha`<br>• `x-version: backup`<br>• `x-version: beta` | • `x-version: users` |
| `x-version: *a*` 和 `x-version: *b*` | • `x-version: backup`<br>• `x-version: beta` | • `x-version: account`<br>• `x-version: alpha`<br>• `x-version: users` |

| 標頭條件 | 符合路由規則的請求 | 不符合路由規則的請求 |
| --- | --- | --- |
| `x-version: b*` 和 `x-version: *a` | • `x-version: beta` | • `x-version: account`<br>• `x-version: alpha`<br>• `x-version: backup`<br>• `x-version: users` |
| `x-version: *` | • `x-version: account`<br>• `x-version: alpha`<br>• `x-version: backup`<br>• `x-version: beta`<br>• `x-version: users` | 無 |

如果您為多個標頭值建立條件，例如 `Accept:application/json,text/xml`，建議您使用 `*contains*` 作為標頭條件，並且避免使用逗號 (`,`) 字元建立條件。

由於 API Gateway 會實際比對標頭條件，因此語義比對的路由方式可能不同。下表顯示路由規則結果的差異。

| 標頭條件 | 符合路由規則的請求 | 不符合路由規則的請求 |
| --- | --- | --- |
| `Accept: *json` | • `Accept:application/json Accept:text/xml` | • `Accept:application/json,text/xml` |
| `Accept: *json*` | • `Accept:application/json Accept:text/xml`<br>• `Accept:application/json,text/xml` | 無 |

比對基本路徑條件

當您建立基本路徑條件時，如果傳入的請求包含您指定的路徑，則與規則相符。比對區分大小寫，因此路徑 `New/Users` 與 `new/users` 不相符。

您只能為一個基本路徑建立基本路徑條件。

如需受限制的基本路徑條件清單，請參閱 the section called “限制”。

使用基本路徑條件來去除基本路徑

建立基本路徑條件時，您可以選擇分割基本路徑。若去除基本路徑，則 API Gateway 會在調用目標 API 時移除傳入的相符基本路徑。此行為與您使用 API 映射時相同。若您未去除基本路徑，則 API Gateway 會將整個基本路徑轉送至目標 API。我們建議您僅在重新建立 API 映射時去除基本路徑。

下表顯示 API Gateway 如何評估去除基本路徑條件的範例。

| 條件 | 去除基本路徑 | 傳入的請求 | 結果 |
| --- | --- | --- | --- |
| 如果基本路徑包含 `PetStoreShopper/dogs` | True | `GET https://example.com/PetStoreShopper/dogs` | API Gateway 會呼叫 `/` 資源的 `GET` 方法。 |
| 如果基本路徑包含 `PetStoreShopper/dogs` 。 | False | `GET https://example.com/PetStoreShopper/dogs` | API Gateway 會呼叫 `PetStoreShopper/dogs` 資源的 `GET` 方法。 |
| 如果基本路徑包含 `PetStoreShopper` | True | `GET https://example.com/PetStoreShopper/dogs` | API Gateway 會呼叫 `dogs` 資源的 `GET` 方法。 |
| 如果基本路徑包含 `PetStoreShopper` | False | `GET https://example.com/PetStoreShopper/dogs` | API Gateway 會呼叫 `PetStoreShopper/dogs` 資源的 `GET` 方法。 |
| 如果基本路徑包含 `PetStoreShopper` | True | `GET https://example.com/PetStoreShopper?birds=available` | API Gateway 會使用查詢字串參數 `birds=available` 呼叫 `/` 資源的 `GET` 方法。 |

| 條件 | 去除基本路徑 | 傳入的請求 | 結果 |
| --- | --- | --- | --- |
| 如果基本路徑包含 PetStoreShopper | False | GET https://example.com/PetStoreShopper?birds=available | API Gateway 會使用查詢字串參數 birds=available 呼叫 /PetStoreShopper 資源的 GET 方法。 |

限制

- 目標 API 和自訂網域名稱必須位於相同的 AWS 帳戶中。
- 每一項規則可以有一個目標 API。
- 您只能針對私有自訂網域名稱至私有 API，以及針對公有自訂網域名稱至公有 API 建立路由規則。您無法混合公有和私有資源。
- 如果您的自訂網域名稱同時具有 REST 和 HTTP API 的 API 映射，則不支援路由規則。
- 最大優先順序數字為 1,000,000。
- 標頭限制：
  - 每一項 anyOf 條件只能包含一個標頭值。
  - RFC 7230 會指定下列唯一允許的標頭名稱和標頭 glob 值字元，也就是 a-z、A-Z、0-9 以及下列特殊字元：*?-!#$%&'.^_`|~。
  - 您可以在標頭 glob 值中使用萬用字元，但萬用字元必須是 *prefix-match、suffix-match* 或 *contains*。標頭 glob 值之中不可使用 *。
  - 不支援萬用字元標頭名稱。
  - 標頭名稱長度必須小於 40 個字元。
  - 標頭 glob 值長度必須小於 128 個字元。
  - infix 比對的標頭 glob 值長度必須小於 40 個字元。
  - 不支援使用下列標頭作為條件：
    - access-control-*
    - apigw-*
    - Authorization
    - Connection

    - Content-Encoding
    - Content-Length
    - Content-Location
    - Forwarded
    - Keep-Alive
    - Origin
    - Proxy-Authenticate
    - Proxy-Authorization
    - TE
    - Trailers
    - Transfer-Encoding
    - Upgrade
    - x-amz-*
    - x-amzn-*
    - x-apigw-api-id
    - X-Forwarded-For
    - X-Forwarded-Host
    - X-Forwarded-Proto
    - x-restAPI
    - Via
- 基本路徑限制：
    - 基本路徑長度必須小於 128 個字元。
    - 基本路徑只能包含字母、數字和下列字元：`$-_.+!*'()/`。

      規則表達式 (regex) 不支援這些字元。
    - 基本路徑的開頭或結尾不可為反斜線 (\)。

### API Gateway 如何評估路由規則的範例

下節說明 API Gateway 評估路由規則和 API 映射的四個範例。

### 範例 1：僅限路由規則

在此範例中，自訂網域名稱 `https://petstore.example.com` 的路由模式設定為 `ROUTING_RULE_ONLY`，且具有下列路由規則和優先順序。

| 規則 ID | Priority | 條件 | Action |
| --- | --- | --- | --- |
| `abc123` | 10 | 如果請求包含標頭：`Hello:World` | 目標 API 1 |
| `zzz000` | 50 | 如果請求包含標頭：`Accept:image/webp` 和 `Pet:Dog-*` ，且如果基本路徑包含 `PetStoreShopper` | 目標 API 2 |
| `efg456` | 100 | 無 | 目標 API 3 |

下表顯示 API Gateway 如何將上述路由規則套用至範例請求。

| 要求 | 選取的 API | 說明 |
| --- | --- | --- |
| `https://petstore.example.com -h "Hello:World"` | 目標 API 1 | 請求符合路由規則 `abc123`。 |
| `https://petstore.example.com/PetStoreShopper -h "Hello:World", "Pet:Dog-Bella", "Accept:image/webp"` | 目標 API 1 | API Gateway 會依優先順序評估所有路由規則。路由規則 `abc123` 具有第一優先順序且條件相符，因此 API Gateway 會調用目標 API 1。<br><br>雖然請求的條件也符合路由規則 `zzz000`，但 API Gateway 找到相符項目後，就不會評估任何其他路由規則。 |

| 要求 | 選取的 API | 說明 |
| --- | --- | --- |
| `https://petstore.example.com/PetStoreShopper -h "Pet:Dog-Bella", "Accept:image/webp"` | 目標 API 2 | 請求符合路由規則 `zzz000`。這是相符項目，因為 `Pet:Dog-Bella` 是 `Pet:Dog-*` 的字串相符項目 |
| `https://petstore.example.com/PetStoreShopper -h "Pet:Dog-Bella"` | 目標 API 3 | 請求不符合路由規則 `abc123`。請求不符合路由規則 `zzz000`，因為所有必要的標頭都不存在。下一個優先順序規則符合所有傳入的請求，因此 API Gateway 會調用目標 API 3。 |

範例 2：路由規則和 API 映射

在此範例中，自訂網域名稱 `https://petstore.diagram.example.com` 的路由模式設定為 `ROUTING_RULE_THEN_API_MAPPING`，且具有下列路由規則和 API 映射。

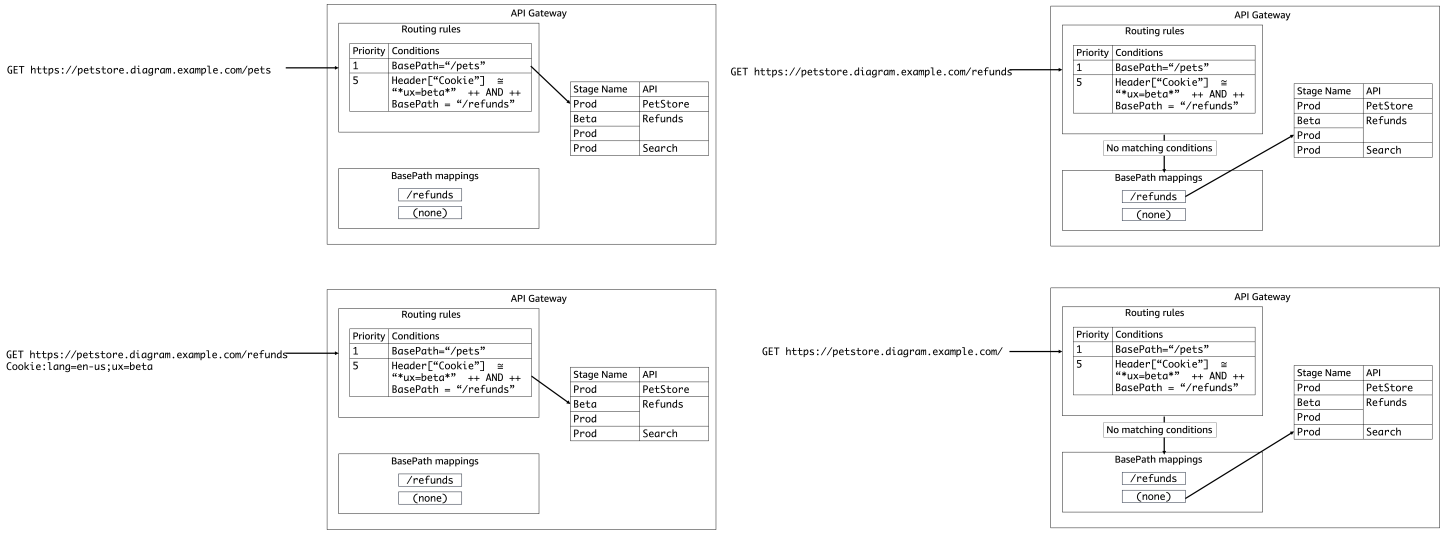
| 規則 ID | Priority | 條件 | Action |
| --- | --- | --- | --- |
| `abc123` | 1 | 如果請求包含 `pets` | 調用 `PetStore` API 的 `Prod` 階段。 |
| `000zzz` | 5 | 如果請求包含標頭：`Cookie:*ux=beta*`，且如果基本路徑包含 `/refunds` | 調用 `Refunds` API 的 `Beta` 階段。 |

下表顯示 `https://petstore.backup.example.com` 的 API 映射。

| API 映射 | 選取的 API |
| --- | --- |
| `/refunds` | 調用 `Refunds` API 的 `Prod` 階段。 |

| API 映射 | 選取的 API |
| --- | --- |
| (none) | 調用 Search API 的 Prod 階段。 |

下圖顯示 API Gateway 如何將上述路由規則和 API 映射套用至範例請求。此圖後面的表中摘要整理了範例請求。




下表顯示 API Gateway 如何將上述路由規則和 API 映射套用至範例請求。

| 要求 | 選取的 API | 說明 |
| --- | --- | --- |
| https://petstore.diagram.com/pets | PetStore API 的 Prod 階段。 | 請求符合路由規則 abc123。 |
| https://petstore.diagram.example.com/refunds -h "Cookie:lang=en-us;ux=beta" | Refunds API 的 Beta 階段。 | 請求符合路由規則 000zzz。Cookie 標頭包含此條件的正確 *contains* 相符項目和基本路徑相符項目。 |
| https://petstore.diagram.example.com/refunds | Refunds API 的 Prod 階段。 | 請求沒有比對路由規則 zzz000 所需的標頭。如果 API Gateway 無法成功比對路由規則，則會回復為 API 映射。API Gateway 可以將基本 |

| 要求 | 選取的 API | 說明 |
| --- | --- | --- |
| | | 路徑對應至 Refunds API 的 Prod 階段。 |
| https://petstore.diagram.example.com/ | Search API 的 Prod 階段。 | 請求符合空白路徑 (none) 的 API 映射。 |

範例 3：具有多層的路由規則和 API 映射

在此範例中，自訂網域名稱 https://petstore.backup.example.com 的路由模式設定為 ROUTING_RULE_THEN_API_MAPPING，且具有下列路由規則和 API 映射。

下表顯示 https://petstore.backup.example.com 的路由規則。

| 規則 ID | Priority | 條件 | Action |
| --- | --- | --- | --- |
| abc123 | 10 | 如果請求包含標頭：Hello:World | 目標 API 1 |
| 000zzz | 50 | 如果請求包含標頭：Accept:image/webp 和 Pet:Dog-* ，且如果基本路徑包含 PetStoreShopper | 目標 API 2 |

下表顯示 https://petstore.backup.example.com 的 API 映射。

| API 映射 | 選取的 API |
| --- | --- |
| PetStoreShopper | 目標 API 3 |
| PetStoreShopper/cats | 目標 API 4 |

下表顯示 API Gateway 如何將上述路由規則和 API 映射套用至範例請求。

| 要求 | 選取的 API | 說明 |
| --- | --- | --- |
| `https://petstore.example.com/PetStoreShopper -h "Accept:image/webp", "Pet:Cats"` | 目標 API 3 | 請求沒有比對路由規則 `zzz000` 所需的標頭。如果 API Gateway 無法成功比對路由規則，則會回復為 API 映射。API Gateway 可以將基本路徑對應至目標 API 3。 |
| `https://petstore.example.com/PetStoreShopper/cats -h "Hello:World"` | 目標 API 1 | 請求符合路由規則 `abc123`。如果路由模式設定為 `ROUTING_RULE_THEN_API_MAPPING` ，則路由規則一律優先於 API 映射。 |
| `https://petstore.example.com/Admin -h "Pet:Dog-Bella"` | 無 | 請求不符合任何路由規則或 API 映射。由於沒有預設的路由規則，因此 API Gateway 會拒絕呼叫並傳送 `403 Forbidden` 狀態碼給發起人。 |

範例 4：萬用字元網域名稱的路由規則

在此範例中，自訂網域名稱 `https://*.example.com` 是萬用字元網域名稱。萬用字元支援所有路由回相同網域的子網域。下列範例路由規則會變更此行為，以允許子網域使用 `Host` 標頭路由傳送至不同的目標 API。

下表顯示 `https://*.example.com` 的路由規則。

| 規則 ID | Priority | 條件 | Action |
| --- | --- | --- | --- |
| `abc123` | 10 | 如果請求包含標頭：`Host:a.example.com` | 目標 API 1 |

| 規則 ID | Priority | 條件 | Action |
| --- | --- | --- | --- |
| 000zzz | 50 | 如果請求包含標頭：Host:b.example.com | 目標 API 2 |
| efg456 | 500 | 無 | 目標 API 3 |

下表顯示 API Gateway 如何將上述路由規則套用至範例請求。

| 要求 | 選取的 API | 說明 |
| --- | --- | --- |
| https://a.example.com | 目標 API 1 | Host 標頭為 a.example.com 。此請求符合路由規則 abc123。 |
| https://b.example.com | 目標 API 2 | Host 標頭為 b.example.com 。此請求符合路由規則 000zzz。 |
| https://testing.example.com | 目標 API 3 | 這會符合全部擷取路由規則 efg456。 |

如何使用路由規則

您可以使用 AWS 管理主控台 AWS CLI或任何 AWS SDK 建立路由規則。建立規則之後，您可以變更其優先順序。

建立路由規則

下列程序說明如何針對自訂網域名稱建立路由規則，並將路由模式設定為 ROUTING_RULE_THEN_API_MAPPING 或 ROUTING_RULE_ONLY。

AWS 管理主控台

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 從主要導覽窗格中，選擇 Custom Domain Names (自訂網域名稱)。
3. 選擇自訂網域名稱。

4. 在路由詳細資訊索引標籤上，選擇新增路由規則。
5. 選擇新增條件以新增條件。

   您可以新增標頭或基本路徑條件。若要將所有傳入請求與您的自訂網域名稱進行比對，請勿新增條件。
6. 針對動作，使用下拉式清單選取您的目標 API 和目標階段。
7. 選擇下一步。
8. 在優先順序欄位中，輸入優先順序的數字。

   API Gateway 會依優先順序評估規則，從最低值到最高值。

   如果您要建立規則但不設任何條件，建議您使用高優先順序值。
9. 選擇建立路由規則。

AWS CLI

以下 create-routing-rule 命令會建立優先順序為 50 的路由規則。在此範例中，API Gateway 會將任何具有標頭 `Hello:World` 和 `x-version:beta` 以及基本路徑 `PetStoreShopper` 的傳入請求，路由傳送至目標 API `a1b2c3`。

```
aws apigatewayv2 create-routing-rule \
 --domain-name 'api.example.com' \
 --priority 50 \
 --conditions '[
   {
     "MatchHeaders": {
       "AnyOf": [
         {
           "Header": "Hello",
           "ValueGlob": "World"
         }
       ]
     }
   },
   {
     "MatchHeaders": {
       "AnyOf": [
         {
           "Header": "x-version",
           "ValueGlob": "beta"
```

```
          }
        ]
      }
    },
    {
      "MatchBasePaths": {
        "AnyOf": [
          "PetStoreShopper"
        ]
      }
    }
  ]'\
  --actions '[
  {
    "InvokeApi": {
      "ApiId": "a1b2c3",
      "Stage": "prod"
    }
  }
 ]'
```

輸出將如下所示：

```
{
    "Actions": [
        {
            "InvokeApi": {
                "ApiId": "a1b2c3",
                "Stage": "prod",
                "StripBasePath": false
            }
        }
    ],
    "Conditions": [
        {
            "MatchHeaders": {
                "AnyOf": [
                    {
                        "Header": "Hello",
                        "ValueGlob": "World"
                    }
                ]
            }
```

```
        },
        {
            "MatchHeaders": {
                "AnyOf": [
                    {
                        "Header": "x-version",
                        "ValueGlob": "beta"
                    }
                ]
            }
        },
        {
            "MatchBasePaths": {
                "AnyOf": [
                    "PetStoreShopper"
                ]
            }
        }
    ],
    "Priority": 50,
    "RoutingRuleArn": "arn:aws:apigateway:us-west-2:111122223333:/domainnames/
api.example.com/routingrules/abc123",
    "RoutingRuleId": "abc123"
}
```

變更路由規則的優先順序

您可以變更路由規則的優先順序。此變更會立即生效，且可能影響 API 取用者調用您的自訂網域名稱的方式。我們建議您在設定路由規則的優先順序時，在規則之間保留間隙。

例如，假設有兩項路由規則，規則 `abc123` 的優先順序為 50，規則 `zzz000` 的優先順序為 150。若要變更規則的優先順序，以便讓 API Gateway 先評估規則 `zzz000`，那麼您可以將規則 `zzz000` 的優先順序變更為 30。

AWS 管理主控台

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 從主要導覽窗格中，選擇 Custom Domain Names (自訂網域名稱)。
3. 選擇自訂網域名稱。
4. 在路由詳細資訊索引標籤上，選擇您的路由規則，然後選擇編輯。

5. 選擇下一步。
6. 針對優先順序，輸入新的優先順序。
7. 選擇儲存變更。

AWS CLI

以下 put-routing-rule 命令會變更路由規則 abc123 的優先順序。

```
aws apigatewayv2 put-routing-rule \
 --domain-name 'api.example.com' \
 --priority 30 \
 --routing-rule-id abc123 \
 --conditions '[
   {
     "MatchHeaders": {
       "AnyOf": [
         {
           "Header": "Hello",
           "ValueGlob": "World"
         }
       ]
     }
   },
   {
     "MatchHeaders": {
       "AnyOf": [
         {
           "Header": "x-version",
           "ValueGlob": "beta"
         }
       ]
     }
   },
   {
     "MatchBasePaths": {
       "AnyOf": [
         "PetStoreShopper"
       ]
     }
   }
 ]'\
 --actions '[
```

```
  {
    "InvokeApi": {
      "ApiId": "a1b2c3",
      "Stage": "prod"
    }
  }
 ]'
```

輸出將如下所示：

```
{
    "Actions": [
        {
            "InvokeApi": {
                "ApiId": "a1b2c3",
                "Stage": "prod",
                "StripBasePath": false
            }
        }
    ],
    "Conditions": [
        {
            "MatchHeaders": {
                "AnyOf": [
                    {
                        "Header": "Hello",
                        "ValueGlob": "World"
                    }
                ]
            }
        },
        {
            "MatchHeaders": {
                "AnyOf": [
                    {
                        "Header": "x-version",
                        "ValueGlob": "beta"
                    }
                ]
            }
        },
        {
            "MatchBasePaths": {
```

```
                "AnyOf": [
                    "PetStoreShopper"
                ]
            }
        }
    ],
    "Priority": 38,
    "RoutingRuleArn": "arn:aws:apigateway:us-west-2:111122223333:/domainnames/
api.example.com/routingrules/abc123",
    "RoutingRuleId": "abc123"
}
```

使用路由規則重新建立 API 映射

您可以使用路由規則重新建立 API 映射。若要重新建立 API 映射，請務必開啟基本路徑去除功能。這樣就會保留 API 映射的行為。如需詳細資訊，請參閱the section called “使用基本路徑條件來去除基本路徑”。

下列教學課程說明如何重新建立 API 映射 `https://` *`api.example.com`*`/orders/v2/items/categories/5` 作為路由規則，以及如何更新您的存取日誌，以記錄 API Gateway 用來將流量傳送至 API 的路由規則 ID。

AWS 管理主控台

將路由模式設定為 ROUTING_RULE_THEN_API_MAPPING

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 從主要導覽窗格中，選擇 Custom Domain Names (自訂網域名稱)。
3. 選擇您的自訂網域名稱。
4. 針對網域詳細資訊，選擇編輯。
5. 針對路由模式，選擇 ROUTING_RULE_THEN_API_MAPPING。
6. 選擇儲存

設定路由模式後，您就可以建立路由規則。

建立路由規則

1. 在路由詳細資訊索引標籤上，選擇新增路由規則。
2. 選擇新增條件，然後選擇路徑。

3. 針對路徑，輸入 **orders/v2/items/categories/5**。
4. 針對去除基本路徑，選擇作用中。
5. 針對目標 API，選擇您的目標 API。
6. 針對目標階段，選擇您的目標階段。
7. 選擇下一步。
8. 針對優先順序，輸入優先順序。

   即使您保留現有的 API 映射，API Gateway 仍會一律使用新的路由規則，因為路由規則一律優先於 API 映射。
9. 選擇儲存變更。

建立路由規則後，請更新階段的存取日誌格式或建立新的日誌，以確認 API Gateway 使用您的路由規則將流量傳送至您的 API。

更新您的存取日誌

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 選擇您的 API。
3. 在主導覽窗格中，選擇階段。
4. 針對日誌和追蹤，選擇編輯。

   如果您沒有日誌群組，請參閱 the section called “CloudWatch 日誌”。
5. 將 **$context.customDomain.routingRuleIdMatched** 新增至您的日誌格式。

   此日誌群組會記錄 API Gateway 用來將流量傳送至 API 的路由規則 ID。如需詳細資訊，請參閱the section called “我無法得知 API Gateway 如何將流量傳送至 API”。
6. 選擇 Save (儲存)。

更新存取日誌後，調用您的自訂網域名稱。以下範例 curl 命令會使用基本路徑 `orders/v2/items/categories/5` 調用自訂網域名稱 `https://api.example.com`。

```
curl "https://api.example.com/orders/v2/items/categories/5"
```

成功調用自訂網域名稱後，請確認 CloudWatch Logs 顯示 `routingRuleIdMatched`。若要了解如何使用 CloudWatch Logs 主控台檢視日誌群組，請參閱 the section called “在 CloudWatch 主控台中檢視日誌事件”。

AWS CLI

1. 使用以下 update-domain-name 命令來更新網域名稱 *api.example.com*，以使用路由模式 `ROUTING_RULE_THEN_API_MAPPING`。

```
aws apigatewayv2 update-domain-name \
  --domain-name 'api.example.com' \
  --routing-mode ROUTING_RULE_THEN_API_MAPPING
```

2. 使用以下 create-routing-rule 命令來建立新的路由規則，以重新建立 API 映射 `https://api.example.com/orders/v2/items/categories/5`。

```
aws apigatewayv2 create-routing-rule \
  --domain-name 'api.example.com' \
  --priority 50 \
  --conditions '[
  {
    "MatchBasePaths": {
      "AnyOf": [
        "orders/v2/items/categories/5"
      ]
    }
  }
]' \
  --actions '[
  {
    "InvokeApi": {
      "ApiId": "a1b2c3",
      "Stage": "prod",
      "StripBasePath": true
    }
  }
]'
```

3. 使用以下 update-stage 命令來更新存取日誌格式，以包含 `$context.customDomain.routingRuleIdMatched` 變數。此變數會記錄 API Gateway 用來將流量傳送至 API 的路由規則 ID。您可以使用此日誌來確認 API Gateway 使用您的路由

規則將流量傳送至 API。如需詳細資訊，請參閱the section called “我無法得知 API Gateway 如何將流量傳送至 API”。

```
aws apigateway update-stage \
  --rest-api-id a1bc2c3 \
  --stage-name prod \
  --patch-operations "op=replace,path=/accessLogSettings/format,value='\
$context.path \$context.customDomain.routingRuleIdMatched \$context.requestId \
$context.extendedRequestId'"
```

如果您沒有日誌群組，請參閱 the section called “CloudWatch 日誌”。

4. 使用以下範例 curl 命令，以使用基本路徑 `orders/v2/items/categories/5` 調用自訂網域名稱。

```
curl "https://api.example.com/orders/v2/items/categories/5
```

5. 使用以下 filter-log-events 命令，從包含路由規則 ID `abc123` 的日誌群組 `access-log-group-orders` 取得日誌事件。

```
aws logs filter-log-events --log-group-name access-log-group-orders --filter-
pattern abc123
```

這樣做可確認 API Gateway 使用路由規則將流量傳送至 API。

對路由規則的問題進行疑難排解

下列疑難排解指引可能有助於解決路由規則的問題。

我無法得知 API Gateway 如何將流量傳送至 API

您可以使用 REST API 階段的存取日誌來記錄路由規則並進行疑難排解。您可以使用 `$context.customDomain.routingRuleIdMatched` 變數來檢視 API Gateway 用來將流量傳送至 API 的路由規則 ID。若要檢視 API Gateway 用來將流量傳送至 API 的 API 映射，請使用 `$context.customDomain.basePathMatched` 變數。

若要記錄您的路由規則，您需要為您的帳戶設定適當的 CloudWatch Logs 角色 ARN，並建立日誌群組。

下列範例存取日誌群組可以擷取相關資訊，以對路由規則和 API 映射進行疑難排解。API Gateway 只會針對其使用的路由機制填入內容變數，否則內容變數為 -。

CLF

```
$context.path $context.customDomain.routingRuleIdMatched
 $context.customDomain.basePathMatched $context.requestId $context.extendedRequestId
```

JSON

```
{"requestPath": "$context.path", "routingRuleId" :
 "$context.customDomain.routingRuleIdMatched", "API mapping" :
 "$context.customDomain.basePathMatched", "requestId":"$context.requestId",
 "extendedRequestId":"$context.extendedRequestId"}
```

XML

```
<request id="$context.requestId"> <requestPath>$context.path</requestPath>
 <ruleId>$context.customDomain.routingRuleIdMatched</ruleId> <ApiMapping>
$context.customDomain.basePathMatched</ApiMapping> <extendedRequestId>
$context.extendedRequestId</extendedRequestId> </request>
```

CSV

```
$context.path,$context.customDomain.routingRuleIdMatched,
$context.customDomain.basePathMatched,$context.requestId,$context.extendedRequestId
```

我們也建議您確認自訂網域名稱的路由模式。如需詳細資訊，請參閱the section called “為您的自訂網域名稱設定路由模式”。

我無法在自訂網域名稱上啟用路由規則

您可能會從 API Gateway 收到下列錯誤：

```
Your account doesn’t have permission to use RoutingRules.
This might be caused by an IAM policy in your account with a deny statement on
 BasePathMapping or ApiMapping.
To grant permission for this account to use RoutingRules, use the UpdateAccount API.
This will impact any existing IAM policies that deny access to BasePathMapping or
 ApiMapping.
See API Gateway documentation for further details.
```

如果目前或過去有拒絕存取 BasePathMapping 或 ApiMapping 的 IAM 政策，則會收到此錯誤。當您針對自訂網域名稱啟用路由規則時，雖然您的政策會繼續拒絕對 BasePathMapping 或 ApiMapping 的存取，但可使用相同的政策存取 RoutingRule。如此就可讓使用者變更自訂網域名稱的路由行為。

例如，如果您的政策如下：

```
{
    "Sid": "DenyCreatingApiMappings",
    "Effect": "Deny",
    "Action": "apigateway:POST",
    "Resource": [
        "arn:aws:apigateway:us-west-2::/domainnames/example.com/apimappings"
    ]
}
```

當您針對 example.com 啟用路由規則時，此政策將繼續拒絕建立 ApiMapping 的存取，但不會拒絕建立 RoutingRule 的存取。

我們建議您稽核帳戶中的 IAM 政策。下列範例政策將會拒絕建立 ApiMapping、BasePathMapping 和 RoutingRule 的存取：

```
{
    "Sid": "DenyCreatingBasePathMappingsApiMappings",
    "Effect": "Deny",
    "Action": "apigateway:POST",
    "Resource": [
        "arn:aws:apigateway:us-west-2::/domainnames/example.com/basepathmappings",
        "arn:aws:apigateway:us-west-2::/domainnames/example.com/apimappings"
    ]
},
{
    "Sid": "DenyCreatingRoutingRules",
    "Effect": "Deny",
    "Action": "apigateway:CreateRoutingRule",
    "Resource": [
        "arn:aws:apigateway:us-west-2:111122223333:/domainnames/example.com/
routingrules/*"
    ]
}
```

您確認所有政策都已更新後，就可以更新 API 的帳戶層級設定，以針對區域啟用路由規則。

使用以下 update-account 命令來更新區域的 API 帳戶層級設定：

```
aws apigateway update-account --patch-operations 'op=remove,path=/
features,value=BlockedForRoutingRules' --region us-west-2
```

更新 API 的帳戶層級設定後，您可以變更自訂網域名稱的路由模式。您也可以繼續使用 IAM 政策來拒絕對 RoutingRules、ApiMapping 或 BasePathMapping 的存取。

使用 API 映射將 API 階段連線至 REST API 的自訂網域名稱

您可以使用 API 映射將 API 階段連線至自訂網域名稱。這會透過您的自訂網域名稱將流量傳送至 API。

API 映射指定一個 API，一個階段，以及選擇性用於映射的路徑。例如，您可以將 https://api.example.com/orders 對應至 API 的 production 階段。

您可以將 HTTP 和 REST API 階段映射至相同的自訂網域名稱。

建立 API 映射之前，您必須先擁有 API、階段和自訂網域名稱。如需進一步了解如何建立自訂網域名稱，請參閱the section called “設定區域性自訂網域名稱”。

對自訂網域名稱的傳入請求

當您將自訂網域名稱對應至 API 的階段時，API Gateway 會去除傳入的基本路徑。這樣會從 API 的調用中移除對應的基本路徑。例如，如果您的基本路徑映射是 https://api.example.com/orders/shop/5 至 test 階段，而您使用下列請求 https://api.example.com/orders/shop/5/hats，則 API Gateway 會調用 API test 階段的 /hats 資源，而非 orders/shop/5/hats 資源。

對應 API 請求

以下說明 API Gateway 如何評估 API 映射。

您可以使用單一層級映射來建立 API 映射，例如從 orders 至 API beta 階段的 API 映射，以及從 shipping 至 API alpha 階段的 API 映射。對於具有 TLS 1.2 安全政策的區域自訂網域名稱，API Gateway 可支援多層 API 映射。您可以建立從 orders/v1/items 至 API alpha 階段及 orders/v2/items 至 API beta 階段的 API 映射。當您建立有多個層級的映射時，API Gateway 會將請求傳送至具有最長相符路徑的 API 映射。

您可以建立對空白路徑 (none) 的 API 映射。如果沒有路徑與請求相符，則 API Gateway 會將請求傳送到空白路徑 (none)。

在此範例中，自訂網域名稱 `https://api.example.com` 具有下列 API 映射。

| API 映射 | 選取的 API |
| --- | --- |
| `(none)` | API 1 |
| `orders` | API 2 |
| `orders/v1/items` | API 3 |
| `orders/v2/items` | API 4 |
| `orders/v1/items/categories` | API 5 |

下表顯示 API Gateway 如何將上述 API 映射套用至範例請求。

| 要求 | 選取的 API | 說明 |
| --- | --- | --- |
| `https://api.example.com/orders` | API 2 | 請求完全符合此 API 映射。 |
| `https://api.example.com/orders/v1/items` | API 3 | 請求完全符合此 API 映射。 |
| `https://api.example.com/orders/v2/items` | API 4 | 請求完全符合此 API 映射。 |
| `https://api.example.com/orders/v1/items/123` | API 3 | API Gateway 會選擇具有最長相符路徑的映射。請求結束時的 123 不會影響選擇。請參閱 the section called “對自訂網域名稱的傳入請求”。 |
| `https://api.example.com/orders/v1/items/categories/5` | API 5 | API Gateway 會選擇具有最長相符路徑的映射。 |

| 要求 | 選取的 API | 說明 |
| --- | --- | --- |
| `https://api.example.com/customers` | API 1 | API Gateway 使用空白映射作為全部擷取。 |
| `https://api.example.com/ordersandmore` | API 2 | API Gateway 會選擇具有最長相符字首的映射。<br><br>針對設定使用單一層級映射的自訂網域名稱，例如只有 `https://api.example.com/orders` 和 `https://api.example.com/`，API Gateway 會選擇 `API 1`，因為沒有與 `ordersandmore` 相符的路徑。 |

限制

- 在 API 映射中，自訂網域名稱和映射APIs 必須在相同的 AWS 帳戶中。
- API 映射只能包含字母、數字和下列字元：`$-_.+!*'()/`。
- API 映射中路徑的最大長度為 300 個字元。
- 您可以為每個域名設定具有 200 個具多個層級的 API 映射。此限制不包括具有單一層級的 API 映射，例如 `/prod`。
- 您只能將 HTTP API 映射至具有 TLS 1.2 安全政策的區域自訂網域名稱。
- 您無法將 WebSocket API 映射至與 HTTP API 或 REST API 相同的自訂網域名稱。
- 建立您的 API 映射後，您必須建立或更新 DNS 提供者的資源記錄，以映射至您的 API 端點。
- 如果您建立具有多層的 API 映射，API Gateway 會將所有標頭名稱轉換為小寫。

建立 API 映射

若要建立 API 映射，您必須先建立自訂網域名稱、API 和階段。您自訂網域名稱的路由模式必須設定為 `ROUTING_RULE_THEN_API_MAPPING` 或 `API_MAPPING_ONLY`。如需如何設定路由模式的相關資訊，請參閱 the section called “為您的自訂網域名稱設定路由模式”。

如需建立所有資源的範例 AWS Serverless Application Model 範本，請參閱 GitHub 上的具有 SAM 的工作階段。

AWS 管理主控台

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 從主要導覽窗格中，選擇 Custom Domain Names (自訂網域名稱)。
3. 選擇自訂網域名稱。
4. 在路由詳細資訊索引標籤上，選擇設定 API 映射。
5. 輸入映射的 API、階段及路徑。
6. 選擇儲存。

AWS CLI

以下 create-api-mapping 命令會建立 API 映射。在此範例中，API Gateway 會將請求傳送至 `api.example.com/v1/orders`，到指定的 API 和階段。

> **Note**
>
> 若要建立具有多個層級的 API 映射，您必須使用 `apigatewayv2`。

```
aws apigatewayv2 create-api-mapping \
    --domain-name api.example.com \
    --api-mapping-key v1/orders \
    --api-id a1b2c3d4 \
    --stage test
```

CloudFormation

下列 CloudFormation 範例會建立 API 映射。

> **Note**
>
> 若要建立具有多個層級的 API 映射，您必須使用 `AWS::ApiGatewayV2`。

```
MyApiMapping:
```

```
  Type: 'AWS::ApiGatewayV2::ApiMapping'
  Properties:
    DomainName: api.example.com
    ApiMappingKey: 'orders/v2/items'
    ApiId: !Ref MyApi
    Stage: !Ref MyStage
```

## API Gateway 中自訂網域名稱的 IP 地址類型

當您建立自訂網域名稱時，您可以指定可叫用網域的 IP 地址類型。您可以選擇 IPv4 來解析要叫用網域的 IPv4 地址，也可以選擇雙堆疊來允許 IPv4 和 IPv6 地址叫用您的網域。我們建議您將 IP 地址類型設定為 dualstack，以減輕 IP 空間耗盡或用於安全狀態。如需雙堆疊 IP 地址類型優點的詳細資訊，請參閱 上的 IPv6 AWS。

您可以透過更新網域名稱的端點組態來變更 IP 地址類型。

### IP 地址類型的考量事項

下列考量事項可能會影響您使用 IP 地址類型。

- 公有 API 的 API Gateway APIs 自訂網域名稱預設 IP 地址類型為 IPv4。
- 私有自訂網域名稱只能具有雙堆疊 IP 地址類型。
- 您的自訂網域名稱不需要具有與其對應之所有 APIs的相同 IP 地址類型。如果您停用預設 API 端點，這可能會影響發起人如何叫用您的網域。

### 變更自訂網域名稱的 IP 地址類型

您可以更新網域名稱的端點組態來變更 IP 地址類型。您可以使用 AWS 管理主控台 AWS CLI CloudFormation、 或 AWS SDK 來更新端點組態。

#### AWS 管理主控台

變更自訂網域名稱的 IP 地址類型

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 選擇公有自訂網域名稱。
3. 選擇端點組態。
4. 針對 IP 地址類型，選取 IPv4 或 Dualstack。

5. 選擇儲存。

AWS CLI

下列 update-domain-name 命令會將 API 更新為具有 dualstack 的 IP 地址類型：

```
aws apigateway update-domain-name \
    --domain-name dualstack.example.com \
    --patch-operations "op='replace',path='/endpointConfiguration/
ipAddressType',value='dualstack'"
```

輸出將如下所示：

```
{
    "domainName": "dualstack.example.com",
    "certificateUploadDate": "2025-02-04T14:46:10-08:00",
    "regionalDomainName": "d-abcd1234.execute-api.us-east-1.amazonaws.com",
    "regionalHostedZoneId": "Z3LQWSYCGH4ADY",
    "regionalCertificateArn": "arn:aws:acm:us-east-1:111122223333:certificate/
a1b2c3d4-5678-90ab-cdef",
    "endpointConfiguration": {
        "types": [
            "REGIONAL"
        ],
        "ipAddressType": "dualstack"
    },
    "domainNameStatus": "AVAILABLE",
    "securityPolicy": "TLS_1_2",
    "tags": {}
}
```

## 在 API Gateway 中為您的自訂網域選擇安全政策

安全政策是 API Gateway 所提供的最低 TLS 版本與密碼套件的預先定義組合。當用戶端建立 API 的 TLS 交握或自訂網域名稱時，安全政策會強制執行 API Gateway 接受的 TLS 版本和密碼套件。安全政策可保護您的 API 和自訂網域名稱，使其免於發生網路安全問題，例如在用戶端和伺服器之間竄改和竊聽。

API Gateway 支援舊版安全政策和增強型安全政策。 TLS_1_0和 TLS_1_2 是舊版安全政策。將這些安全政策用於一般工作負載，或開始建立 API。任何以 開頭的政策SecurityPolicy_都是增強型安

全政策。將這些政策用於受規範的工作負載、進階治理，或使用後量子密碼學。使用增強型安全政策時，您也必須設定端點存取模式，以進行額外的治理。如需詳細資訊，請參閱the section called “端點存取模式”。

考量事項

以下是 API Gateway 中 REST APIs 自訂網域名稱的安全政策考量：

- 您無法在使用增強型安全政策的網域名稱上啟用交互 TLS。
- 您無法將 HTTP API 對應至使用增強型安全政策的網域名稱。
- 如果您啟用多層級基本路徑映射至使用增強型安全政策的 REST API，則無法為相同的網域名稱建立基本路徑映射至 HTTP API。
- 您的 API 可以映射到與 API 具有不同安全政策的自訂網域名稱。當您叫用該自訂網域名稱時，API Gateway 會使用自訂網域的安全政策來交涉 TLS 交握。如果您停用預設 API 端點，則可能會影響發起人調用 API 的方式。
- API Gateway 支援所有 APIs的安全政策。不過，您只能選擇 REST APIs的安全政策。API Gateway 僅支援 HTTP 或 WebSocket APIsTLS_1_2的安全政策。
- API Gateway 不支援更新具有多個端點類型的網域名稱的安全政策。如果您有多個端點類型的網域名稱，請刪除其中一個來更新安全政策。

API Gateway 如何套用安全政策

下列範例顯示 API Gateway 如何使用安全政策作為範例來套用`SecurityPolicy_TLS13_1_3_2025_09`安全政策。

`SecurityPolicy_TLS13_1_3_2025_09` 安全政策接受 TLS 1.3 流量，並拒絕 TLS 1.2 和 TLS 1.0 流量。對於 TLS 1.3 流量，安全政策接受下列密碼套件：

- `TLS_AES_128_GCM_SHA256`
- `TLS_AES_256_GCM_SHA384`
- `TLS_CHACHA20_POLY1305_SHA256`

API Gateway 不接受任何其他密碼套件。例如，安全政策會拒絕任何使用加密套件的 TLS 1.3 `AES128-SHA` 流量。

若要監控用於存取 API Gateway 的 TLS 通訊協定和加密用戶端，您可以使用存取日誌中的 `$context.tlsVersion`和 `$context.cipherSuite`內容變數。如需詳細資訊，請參閱the section called “監控”。

若要查看所有 REST APIs和自訂網域名稱的預設安全政策，請參閱 the section called “預設安全政策”。若要查看所有 REST APIs和自訂網域名稱支援的安全政策，請參閱 the section called “支援的安全政策”。

變更自訂網域名稱的安全政策

如果您變更安全政策，更新大約需要 15 分鐘才能完成。您可以監控自訂網域名稱`lastUpdateStatus`的 。隨著您的自訂網域名稱更新， `lastUpdateStatus` 是 `PENDING`，當它完成時，它將是 `AVAILABLE`。

當您使用開頭為 的安全政策時`SecurityPolicy_`，您還必須開啟端點存取模式。如需詳細資訊，請參閱the section called “端點存取模式”。

AWS 管理主控台

變更自訂網域名稱的安全政策

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 選擇將流量傳送至 REST APIs自訂網域名稱。

   請確定只有一種端點類型與您的自訂網域名稱相關聯。
3. 選擇自訂網域名稱設定，然後選擇編輯。
4. 針對安全政策，選取新的政策。
5. 針對端點存取模式，選擇嚴格。
6. 選擇儲存變更。

AWS CLI

下列 update-domain-name 命令會更新網域名稱以使用`SecurityPolicy_TLS13_1_3_2025_09`安全政策：

```
aws apigateway update-domain-name \
    --domain-name example.com \
    --patch-operations '[
        {
            "op": "replace",
```

```
            "path": "/securityPolicy",
            "value": "SecurityPolicy_TLS13_1_3_2025_09"
        },
        {
            "op": "replace",
            "path": "/endpointAccessMode",
            "value": "STRICT"
        }
    ]'
```

輸出將如下所示：

```
{
   "domainName": "example.com",
   "endpointConfiguration": {
      "types": [ "REGIONAL" ],
      "ipAddressType": "dualstack"
   },
   "regionalCertificateArn": "arn:aws:acm:us-west-2:111122223333:certificate/
a1b2c3d4-5678-90ab-cdef",
   "securityPolicy": "SecurityPolicy_TLS13_1_3_2025_09",
   "endpointAccessMode": "STRICT"
}
```

HTTP API 和 WebSocket API 的相關資訊

如需有關 HTTP API 和 WebSocket API 的詳細資訊，請參閱 the section called “API Gateway 中 HTTP API 的安全政策” 和 the section called “API Gateway 中適用於 WebSocket API 的安全政策”。

## 停用 REST API 的預設端點

預設情況下，用戶端可以使用 API Gateway 為 API 產生的 `execute-api` 端點來調用 API。若要確保用戶端只能使用自訂網域名稱來存取您的 API，請停用預設 `execute-api` 端點。用戶端仍然可以連線到您的預設端點，但會收到 `403 Forbidden` 狀態碼。停用預設端點會影響 API 的所有階段。當您更新任何階段的任何設定時，此設定就會生效，例如更新階段上的部署。

以下程序顯示如何停用 REST API 的預設端點。

AWS 管理主控台

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。

2. 選擇 REST API。
3. 在主要導覽窗格中，選擇 API 設定。
4. 選擇一個 API。
5. 在 API 詳細資訊上，選擇編輯。
6. 針對預設端點，選取非作用中。
7. 選擇儲存變更。
8. 在主要導覽窗格中，選擇資源。
9. 選擇部署 API。
10. 將您的 API 重新部署至階段，或更新階段上的任何設定，以讓更新生效。

AWS CLI

以下 update-rest-api 命令會停用預設端點：

```
aws apigateway update-rest-api \
    --rest-api-id abcdef123 \
    --patch-operations op=replace,path=/disableExecuteApiEndpoint,value='True'
```

停用預設端點之後，您必須部署 API，變更才會生效。

以下 create-deployment 命令會建立部署，並將其與階段建立關聯：

```
aws apigateway create-deployment \
    --rest-api-id abcdef123 \
    --stage-name dev
```

## 為 API Gateway API 設定 DNS 備援的自訂運作狀態檢查

您可以使用 Amazon Route 53 運作狀態檢查，從主要 區域中的 API Gateway API 控制 DNS 容錯移轉 AWS 區域 至次要 區域中的 API Gateway API。這可以幫助緩解發生區域問題時的影響。如果您使用自訂網域，可以執行容錯移轉而不需要用戶端變更 API 端點。

當您針對別名記錄選擇 Evaluate Target Health (評估目標運作狀態) 時，只有在區域中無法使用 API Gateway 服務時，這些記錄才會失效。在某些情況下，您自己的 API Gateway API 可能會在這段時間之前發生中斷。若要直接控制 DNS 備援，請為您的 API Gateway API 設定自訂 Route 53 運作狀態檢查。在此範例中，您可以使用 CloudWatch 警示來協助操作員控制 DNS 備援。如需設定容錯移

轉時的更多範例和其他考量事項，請參閱使用 Route 53 建立災難復原機制和使用 AWS Lambda 和 CloudWatch 對 VPC 中的私有資源執行 Route 53 運作狀態檢查。

主題

- 先決條件
- 步驟 1：設定資源
- 步驟 2：啟動容錯移轉至次要區域
- 步驟 3：測試容錯移轉
- 步驟 4：傳回主要區域
- 後續步驟：自訂和定期測試

先決條件

若要完成此程序，您必須建立並設定以下資源：

- 您擁有的網域名稱。
- 該網域名稱的 ACM 憑證為二 AWS 區域。如需更多詳細資訊，請參閱the section called “自訂網域名稱的先決條件”。
- 您網域名稱的 Route 53 託管區域。如需詳細資訊，請參閱 Amazon Route 53 開發人員指南中的使用託管區域。

如需如何為網域名稱建立 Route 53 容錯移轉 DNS 記錄的詳細資訊，請參閱《Amazon Route 53 開發人員指南》中的選擇路由政策。如需如何監控 CloudWatch 警報的詳細資訊，請參閱《Amazon Route 53 開發人員指南》中的監控 CloudWatch 警示。

步驟 1：設定資源

在此範例中，建立下列資源來設定網域名稱的 DNS 備援：

- API Gateway APIs分為兩個 AWS 區域
- 兩個名稱相同的 API Gateway 自訂網域名稱 AWS 區域
- API Gateway API 映射，將您的 API Gateway API 連線至自訂網域名稱
- 網域名稱的 Route 53 容錯移轉 DNS 記錄
- 次要區域中的 CloudWatch 警示
- 根據次要區域中的 CloudWatch 警示進行 Route 53 運作狀態檢查

首先，請確定您擁有主要和次要區域中的所有必要資源。次要區域應包含警示和運作狀態檢查。如此一來，您就不必依賴主要區域來執行容錯移轉。如需建立這些資源的範例 CloudFormation 範本，請參閱 primary.yaml和 secondary.yaml。

> **Important**
>
> 容錯移轉至次要區域之前，請確定所有必要的資源都可用。否則您的 API 將無法為次要區域的流量做好準備。

步驟 2：啟動容錯移轉至次要區域

在下列範例中，待命區域會接收 CloudWatch 指標並啟動容錯移轉。我們使用自訂指標，需要操作員介入才能啟動容錯移轉。

```
aws cloudwatch put-metric-data \
    --metric-name Failover \
    --namespace HealthCheck \
    --unit Count \
    --value 1 \
    --region us-west-1
```

針對您設定的 CloudWatch 警示將指標資料取代為對應的資料。

步驟 3：測試容錯移轉

呼叫您的 API 並驗證是否可從次要區域獲得回應。如果您在步驟 1 中使用範例範本，回應會在容錯移轉後從 {"message": "Hello from the primary Region!"} 改為 {"message": "Hello from the secondary Region!"}。

```
curl https://my-api.example.com

{"message": "Hello from the secondary Region!"}
```

步驟 4：傳回主要區域

若要傳回主要區域，請傳送讓運作狀態檢查通過的 CloudWatch 指標。

```
aws cloudwatch put-metric-data \
```

```
    --metric-name Failover \
    --namespace HealthCheck \
    --unit Count \
    --value 0 \
    --region us-west-1
```

針對您設定的 CloudWatch 警示將指標資料取代為對應的資料。

呼叫您的 API 並驗證是否可從主要區域獲得回應。如果您在步驟 1 中使用範例範本，回應會從 `{"message": "Hello from the secondary Region!"}` 改為 `{"message": "Hello from the primary Region!"}`。

```
curl https://my-api.example.com

{"message": "Hello from the primary Region!"}
```

後續步驟：自訂和定期測試

此範例示範設定 DNS 備援的一種方法。您可以將各種 CloudWatch 指標或 HTTP 端點用於管理容錯移轉的運作狀態檢查。定期測試容錯移轉機制，以確保正常運作，並且操作員熟悉您的容錯移轉程序。

# 最佳化 REST API 的效能

讓您的 API 可供呼叫之後，您可能了解 API 需要最佳化才能提高回應能力。API Gateway 提供了一些最佳化 API 的策略，如回應快取和承載壓縮功能。在本節中，您可以了解如何啟用這些功能。

主題

- API Gateway 中 REST API 的快取設定
- API Gateway 中 REST API 的承載壓縮

## API Gateway 中 REST API 的快取設定

您可以在 API Gateway 中啟用 API 快取，以快取端點的回應。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。

> **Note**
>
> 快取會盡力而為。您可使用 Amazon CloudWatch 中的 `CacheHitCount` 和 `CacheMissCount` 指標，來監控 API Gateway 從 API 快取提供的請求。

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。

此為 HIPAA 合格服務。如需 1996 年 AWS美國健康保險流通與責任法案 (HIPAA) 以及使用 AWS 服務來處理、存放和傳輸受保護醫療資訊 (PHI) 的詳細資訊，請參閱 HIPAA 概觀。

> **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，來啟用其他方法的快取。

> **Important**
>
> 快取是按小時計費，基於您選擇的快取大小。快取不符合 AWS 免費方案的資格。如需詳細資訊，請參閱 API Gateway 定價。

## 啟用 Amazon API Gateway 快取

在 API Gateway 中，您可以啟用特定階段的快取。

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。如需支援的快取大小，請參閱 API Gateway API 參考中的 cacheClusterSize。

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。

> **Note**
>
> 快取容量會影響快取執行個體的 CPU、記憶體和網路頻寬。因此，快取容量可能會影響快取的效能。

API Gateway 建議您執行 10 分鐘的負載測試，以確認快取容量適合您的工作負載。確定負載測試期間的流量會反映生產流量。例如，包括提升、持續流量和流量尖峰。負載測試應包含可從快取提供的回應，以及將項目新增至快取的唯一回應。在負載測試期間監控延遲、4xx、5xx、快取命中和快取未命中指標。根據這些指標，視需要調整快取容量。如需有關負載測試的詳細資訊，請參閱如何選取最佳 API Gateway 快取容量，以避免達到速率限制？。

### AWS 管理主控台

在 API Gateway 主控台，您可以在階段頁面上設定快取。您可以佈建階段快取，並指定預設方法層級快取設定。如果您開啟預設方法層級快取，除非該方法具有方法覆寫，否則階段上所有 GET 方法的方法層級快取都會開啟。您部署至階段的其他任何 GET 方法，都會有方法層級快取。若要為階段的特定方法設定方法層級快取設定，您可以使用方法覆寫。如需有關方法覆寫的詳細資訊，請參閱 the section called “覆寫方法快取的階段快取”。

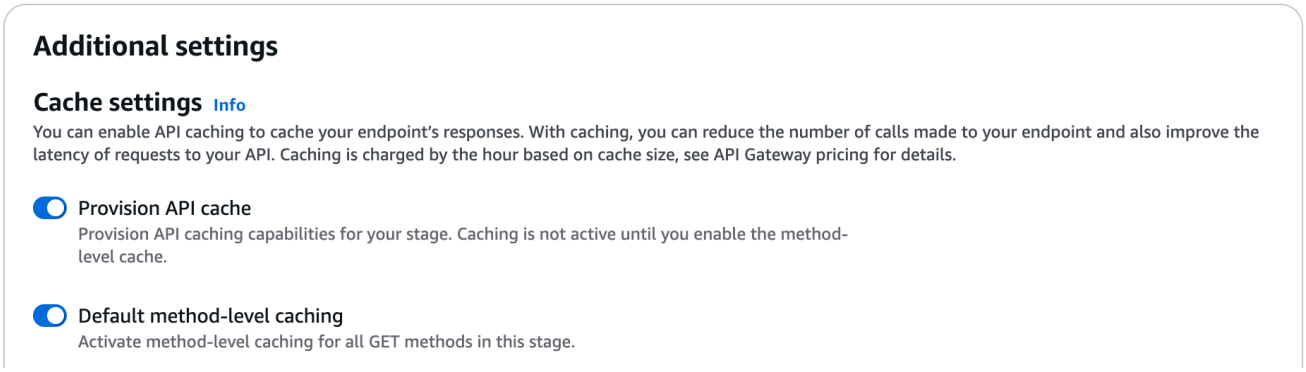
設定指定階段的 API 快取：

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 選擇 Stages (階段)。
3. 在 API 的 Stages (階段) 清單中，選擇階段。
4. 在階段詳細資訊區段中，選擇編輯。
5. 在其他設定下，針對快取設定開啟 API 快取。

   這會為您的階段佈建快取叢集。
6. 若要為您的階段啟用快取，請開啟預設方法層級快取。

   這會開啟您的階段上所有 GET 方法的方法層級快取。您部署至此階段的其他任何 GET 方法，都會有方法層級快取。

> **Note**
>
> 如果您有方法層級快取的現有設定，變更預設方法層級快取設定並不會影響現有設定。




7. 選擇儲存變更。

## AWS CLI

以下 update-stage 命令會更新階段以佈建快取，並為階段上的所有 GET 方法開啟方法層級快取：

```
aws apigateway update-stage \
    --rest-api-id a1b2c3 \
    --stage-name 'prod' \
    --patch-operations file://patch.json
```

patch.json 的內容如下：

```
[
     {
          "op": "replace",
          "path": "/cacheClusterEnabled",
          "value": "true"
     },
     {
          "op": "replace",
          "path": "/cacheClusterSize",
          "value": "0.5"
     },
     {
        "op": "replace",
        "path": "/*/*/caching/enabled",
        "value": "true"
     }
]
```

> **Note**
>
> 如果您有方法層級快取的現有設定，變更預設方法層級快取設定並不會影響現有設定。

> **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，快取叢集值會從 `Create in progress` 變更為 `Active`。快取刪除完成時，快取叢集的值會從 `Delete in progress` 變更為 `Inactive`。
> 當您開啟階段上所有方法的方法層級快取時，預設方法層級快取值會變更為 `Active`。如果您關閉階段上所有方法的方法層級快取，預設方法層級快取值會變更為 `Inactive`。如果您有方法層級快取的現有設定，變更快取的狀態不會影響該設定。

當您在階段的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，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**
>
> 請勿使用 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。

## 覆寫方法層級快取的 API Gateway 階段層級快取

您可以開啟或關閉特定方法的快取，藉以覆寫階段層級快取設定。您也可以修改 TTL 期間，或開啟或關閉快取回應的加密功能。

如果您預期正在快取的方法會在其回應中收到敏感資料，請加密快取資料。您可能需要這樣做，才能符合各種合規架構。如需詳細資訊，請參閱《AWS Security Hub 使用者指南》中的 Amazon API Gateway 控制項。

AWS 管理主控台

如果您在階段詳細資訊中變更預設方法層級快取設定，這並不影響具有覆寫權的方法層級快取設定。

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. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 選擇 API。
3. 選擇 Stages (階段)。
4. 在 API 的 Stages (階段) 清單中，展開階段，然後在 API 中選擇方法。
5. 在方法覆寫區段中，選擇編輯。
6. 在方法設定 區段中，開啟或關閉啟用方法快取，或自訂任何其他想要的選項。

> **Note**
>
> 快取在針對您的階段佈建快取叢集之前，不會起作用。

7. 選擇儲存。

AWS CLI

以下 update-stage 命令只會關閉 `GET /pets` 方法的快取：

```
aws apigateway update-stage /
    --rest-api-id a1b2c3 /
    --stage-name 'prod' /
    --patch-operations file://patch.json
```

`patch.json` 的內容如下：

```
[{
        "op": "replace",
        "path": "/~1pets/GET/caching/enabled",
        "value": "false"
}]
```

## 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

您可以使用方法或整合參數做為快取金鑰來編製快取回應的索引 這包括自訂標頭、URL 路徑或查詢字串。您可以指定部分或所有這些參數來做為快取金鑰，但必須至少指定一個值。當您有快取金鑰時，API Gateway 會分別快取每個金鑰值的回應，包括快取金鑰不存在時。

> **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：

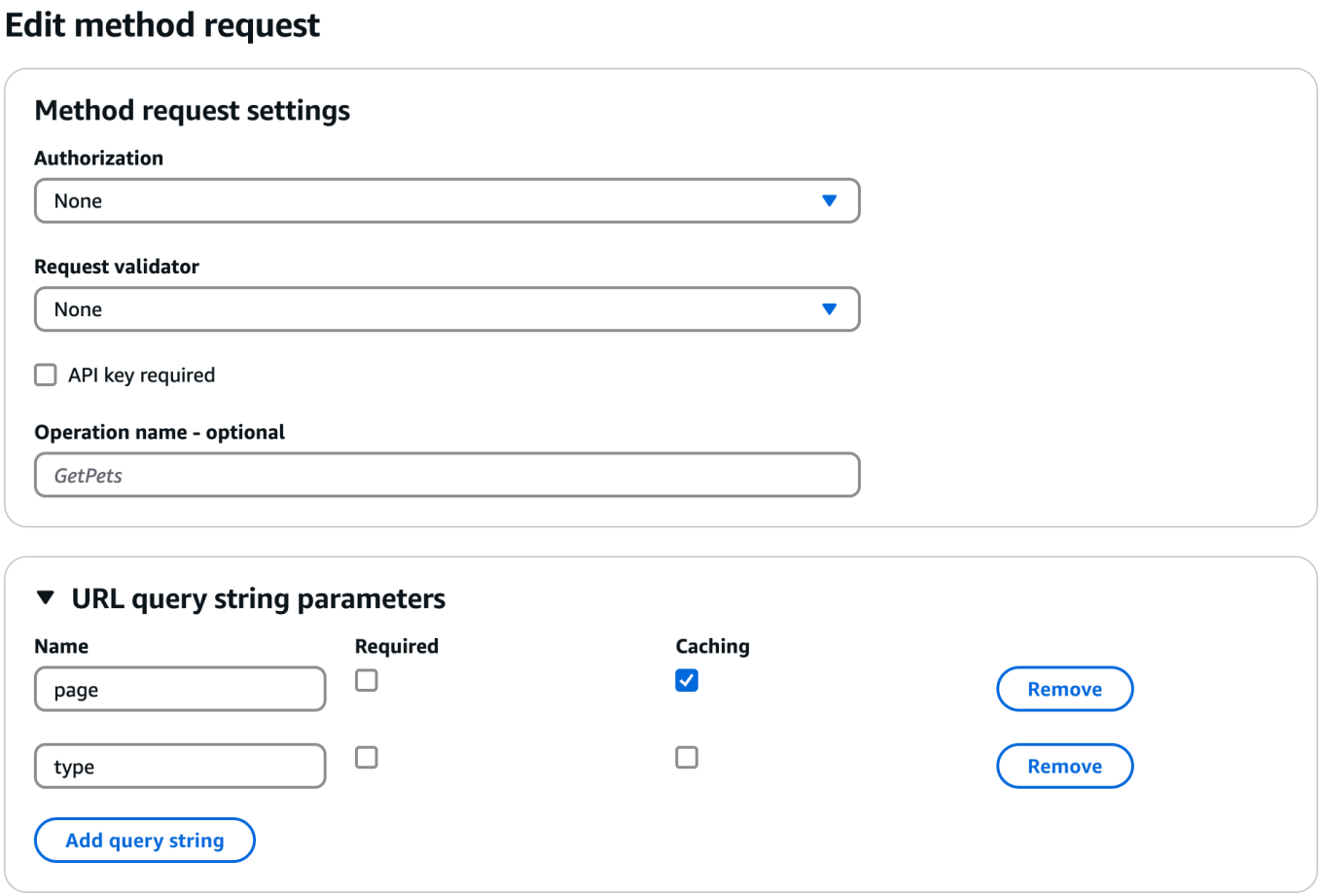
```
GET /users?type=admin&department=A HTTP/1.1
host: example.com
...
```

從這個請求的響應被緩存，並用於服務於以下請求：

```
GET /users?type=admin&department=B HTTP/1.1
host: example.com
...
```

AWS 管理主控台

若要在 API Gateway 主控台中包含方法或整合請求參數作為快取金鑰的一部分，請在新增參數之後選取 Caching (快取)。




AWS CLI

以下 put-method 命令會建立 GET 方法，並需要 type 查詢字串參數：

```
aws apigateway put-method /
    --rest-api-id a1b2c3 /
    --resource-id aaa111 /
    --http-method GET /
    --authorization-type "NONE" /
    --request-parameters "method.request.querystring.type=true"
```

以下 put-integration 命令會建立 GET 方法與 HTTP 端點的整合，並指定 API Gateway 快取 type 方法請求參數：

```
aws apigateway put-integration /
    --rest-api-id a1b2c3 /
    --resource-id aaa111 /
    --http-method GET /
    --type HTTP /
    --integration-http-method GET /
    --uri 'https://example.com' /
    --cache-key-parameters "method.request.querystring.type"
```

若要指定 API Gateway 快取整合請求參數，請使用 `integration.request.location.name` 作為快取金鑰參數。

## 在 API Gateway 中排清 API 階段快取

啟用 API 快取時，您可以排清 API 階段的快取，以確保您 API 的用戶端從您的整合端點取得最新回應。

AWS 管理主控台

排清 API 階段快取

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 選擇有快取之階段的 API。
3. 在主導覽窗格中，選擇階段，然後選擇有快取的階段。
4. 選擇階段動作功能表，然後選取排清階段快取。

AWS CLI

以下 flush-stage-cache 命令會排清階段快取：

```
aws apigateway flush-stage-cache \
    --rest-api-id a1b2c3 \
    --stage-name prod
```

> **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。

> **Note**
>
> 不支援跨帳戶快取失效。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "execute-api:InvalidateCache"
            ],
            "Resource": [
                "arn:aws:execute-api:us-east-1:111111111111: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 許可權控制 REST API 的存取。

如果您未強制執行 `InvalidateCache` 政策 (或在主控台中勾選 Require authorization (需要授權) 核取方塊)，則任何用戶端都可能使 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。

當政策設定完成時，便會啟用快取，且需要授權。

您可以從下列選項中選擇 API Gateway 處理未經授權請求的方式：

請求失敗並顯示 403 狀態碼

API Gateway 會傳回 `403 Unauthorized` 回應。

若要使用 API 來設定此選項，請使用 `FAIL_WITH_403`。

忽略快取控制標頭；在回應標頭中新增警告

API Gateway 會處理請求，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`。

忽略快取控制標頭

API Gateway 會處理請求，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`。

您可以使用 API Gateway 主控台或 AWS CLI來設定未經授權的請求處理行為。

AWS 管理主控台

指定處理未經授權請求的方式

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 選擇有快取之階段的 API。
3. 在主導覽窗格中，選擇階段，然後選擇有快取的階段。
4. 如需階段詳細資訊，請選擇編輯。
5. 針對未經授權的請求處理，選取一個選項。

6. 選擇繼續。
7. 檢閱變更，然後選擇儲存變更。

### AWS CLI

以下 update-stage 命令會更新階段，透過使請求失敗並顯示 403 狀態碼的方式來處理未經授權的請求：

```
aws apigateway update-stage /
    --rest-api-id a1b2c3 /
    --stage-name 'prod' /
    --patch-operations 'op=replace,path=/*/*/caching/
unauthorizedCacheControlHeaderStrategy,value="FAIL_WITH_403"'
```

## CloudFormation 具有快取的階段範例

下列 CloudFormation 範本會建立範例 API、佈建Prod階段的 0.5 GB 快取，以及開啟所有GET方法的方法層級快取。

> **Important**
>
> 快取是按小時計費，基於您選擇的快取大小。快取不符合 AWS 免費方案資格。如需詳細資訊，請參閱 API Gateway 定價。

### 範本範例 CloudFormation

```
AWSTemplateFormatVersion: 2010-09-09
Resources:
  Api:
    Type: 'AWS::ApiGateway::RestApi'
    Properties:
      Name: cache-example
  PetsResource:
    Type: 'AWS::ApiGateway::Resource'
    Properties:
      RestApiId: !Ref Api
      ParentId: !GetAtt Api.RootResourceId
      PathPart: 'pets'
```

```
  PetsMethodGet:
    Type: 'AWS::ApiGateway::Method'
    Properties:
      RestApiId: !Ref Api
      ResourceId: !Ref PetsResource
      HttpMethod: GET
      ApiKeyRequired: true
      AuthorizationType: NONE
      Integration:
        Type: HTTP_PROXY
        IntegrationHttpMethod: GET
        Uri: http://petstore-demo-endpoint.execute-api.com/petstore/pets/
  ApiDeployment:
    Type: 'AWS::ApiGateway::Deployment'
    DependsOn:
      - PetsMethodGet
    Properties:
      RestApiId: !Ref Api
  ApiStage:
    Type: 'AWS::ApiGateway::Stage'
    Properties:
      StageName: Prod
      Description: Prod Stage with a cache
      RestApiId: !Ref Api
      DeploymentId: !Ref ApiDeployment
      CacheClusterEnabled: True
      CacheClusterSize: 0.5
      MethodSettings:
        - ResourcePath: /*
          HttpMethod: '*'
          CachingEnabled: True
```

## API Gateway 中 REST API 的承載壓縮

API Gateway 可讓您的用戶端對使用其中一個支援的內容編碼壓縮的承載呼叫 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 minimumCompressionsSize 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、 或 API Gateway REST API AWS CLI來啟用或停用 API 的壓縮。

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

- 在 API Gateway 中啟用 API 的承載壓縮功能
- 在 API Gateway 中使用壓縮的承載呼叫 API 方法
- 在 API Gateway 中使用壓縮的承載接收 API 回應

## 在 API Gateway 中啟用 API 的承載壓縮功能

您可以使用 API Gateway 主控台 AWS CLI、 或 SDK 啟用 API 的 AWS 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。

> **Note**
>
> 優先順序最高的內容編碼必須是 API Gateway 支援的項目，否則系統不會將壓縮功能套用至回應承載。

主題

- 使用 API Gateway 主控台啟用 API 的承載壓縮功能
- 使用 啟用 API 的承載壓縮 AWS CLI
- API Gateway 支援的內容編碼

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 主控台來啟用承載壓縮功能

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 選擇現有的 API 或建立新的 API。
3. 在主導覽窗格中，選擇 API 設定。
4. 在 API 詳細資訊區段中，選擇編輯。
5. 開啟內容編碼以啟用承載壓縮。針對內文大小下限，輸入代表壓縮大小下限的數字 (位元組)。若要關閉壓縮，請關閉內容編碼選項。
6. 選擇儲存變更。

使用 啟用 API 的承載壓縮 AWS CLI

以下 create-rest-api 命令會建立具有承載壓縮的 API：

```
aws apigateway create-rest-api \
    --name "My test API" \
    --minimum-compression-size 0
```

以下 update-rest-api 命令會啟用現有 API 的承載壓縮：

```
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。

以下 update-rest-api 命令會關閉承載壓縮：

```
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;q=1.0
- Accept-Encoding:gzip;q=1.0,identity;q=0.5,*;q=0

## 在 API Gateway 中使用壓縮的承載呼叫 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: ...

[binary data]
```

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承載。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。

## 在 API Gateway 中使用壓縮的承載接收 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，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。這會開啟 API Gateway 中的承載壓縮功能。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對應範本，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�

J�)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。

# 將您的 REST API 分發至 API Gateway 中的用戶端

本節提供關於將 API Gateway API 分配給客戶的詳細資料。分配 API 包括產生開發套件供客戶下載並與用戶端應用程式整合、記錄您的 API，讓客戶知道如何從用戶端應用程式呼叫 API，並將您的 API 包含在產品項目中。

主題

- API Gateway 中 REST API 的用量計畫和 API 金鑰
- API Gateway 中 REST API 的文件
- 在 API Gateway 中為 REST API 產生 SDK
- 透過 銷售您的 API Gateway APIs AWS Marketplace

## API Gateway 中 REST API 的用量計畫和 API 金鑰

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，以便客戶存取選取的 API，並根據定義的限制和配額來調節傳送給這些 API 的請求量。您可在 API 或 API 方法層級進行設定這些內容。

### 什麼是用量計劃和 API 金鑰？

用量計劃會指定誰能存取一個或多個已部署 API 階段和方法，也可以設定目標請求率來開始調節請求量。計劃會使用 API 金鑰來識別 API 用戶端，以及每個金鑰的相關 API 階段可存取者。

API 金鑰是英數字串值，您會將其發布到應用程式開發人員客戶，以授與您 API 的存取權。您可以將 Lambda 授權方、IAM 角色或 Amazon Cognito 與 API 金鑰搭配使用，來控制對 API 的存取。API Gateway 可以代表您產生 API 金鑰，或從 CSV 檔案匯入它們。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 “在 API Gateway 中設定 REST API 的 API 金鑰”。

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。)
名稱不能超過 1024 個字元。金鑰值是介於 20 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。

調節限制即為開始進行請求調節應的目標點。您可在 API 或 API 方法層級進行設定。

配額限制即為使用特定 API 金鑰在指定時間間隔內可以提交的目標請求量上限。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。

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。

> **Note**
>
> 用量計劃調節和配額並非硬性限制，會依最佳作法來套用。在某些情況下，用戶端可能超出設定的配額。請勿依靠用量計劃配額或調節來控制成本或封鎖對 API 的存取。請考慮使用 AWS Budgets 來監控成本，使用 AWS WAF 來管理 API 請求量。

## 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。

> **Important**
>
> - 不要使用 API 金鑰進行身分驗證或授權來控制 API 的存取權。然而，若用量計劃中具有多個 API，則持有該用量計劃中某 API 有效 API 金鑰的使用者便可存取該用量計劃中的所有 API。請改用 IAM 角色、Lambda 授權方或 Amazon Cognito 使用者集區來控制 API 的存取權。
> - 使用 API Gateway 產生的 API 金鑰。API 金鑰不應包含機密資訊；用戶端通常會在可被記錄的標頭中傳輸這些資訊。

- 如果您是使用開發人員入口網站來發布 API，請注意，特定用量計畫中的所有 API 都是可訂閱的，即使您尚未讓客戶可以看到它們也一樣。

- 在某些情況下，用戶端可能超出設定的配額。請勿依靠用量計劃來控制成本。請考慮使用 AWS Budgets 來監控成本，使用 AWS WAF 來管理 API 請求量。
- 將 API 金鑰新增至用量計畫後，更新作業可能需要幾分鐘時間才能完成。

## 在 API Gateway 中選擇 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。API Gateway 會讀取金鑰，並將其與在用量計劃中的金鑰進行比對。如果有相符項目，則 API Gateway 會根據計劃的請求限制和配額來調節請求量。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 可接收來自以下兩種來源之一的 API 金鑰：

**`HEADER`**

您將 API 金鑰發布至客戶，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 “來自 API Gateway Lambda 授權方的輸出”。

> **Note**
>
> 如需考量最佳實務，請參閱 the section called “API 金鑰和用量計劃的最佳實務”。

下列程序說明如何選擇 API 的 API 金鑰來源。

AWS 管理主控台

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，選擇 API 設定。
4. 在 API 詳細資訊區段中，選擇編輯。
5. 在 API 金鑰來源下，從下拉式清單選取 `Header` 或 `Authorizer`。
6. 選擇儲存變更。

AWS CLI

以下 update-rest-api 命令會更新 API，以將 API 金鑰來源設定為 AUTHORIZER：

```
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，請在上述命令中將 value 設為 HEADER。

REST API

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/fugvjdxtri/ 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。

## 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，如以下範例所示：

```
Key,name
```

```
apikey1234abcdefghij0123456789,MyFirstApiKey
```

Key 值必須介於 20 到 128 個字元之間。Name 值不能超過 1024 個字元。

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。

> **Note**
>
> 如需考量最佳實務，請參閱 the section called “API 金鑰和用量計劃的最佳實務”。

## 在 API Gateway 中設定 REST API 的 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。在您建立 API 金鑰值之後便無法變更。

如需如何使用 API Gateway 主控台建立和部署 API 的說明，請參閱在 APIs Gateway 中開發 REST API 和在 API Gateway 中部署 REST API。

建立 API 金鑰後，您必須為該金鑰與用量計畫建立關聯。如需詳細資訊，請參閱在 API Gateway 中設定 REST API 的用量計畫。

> **Note**
>
> 如需考量最佳實務，請參閱 the section called “API 金鑰和用量計劃的最佳實務”。

主題

- 方法中需要 API 金鑰
- 建立 API 金鑰
- 匯入 API 金鑰

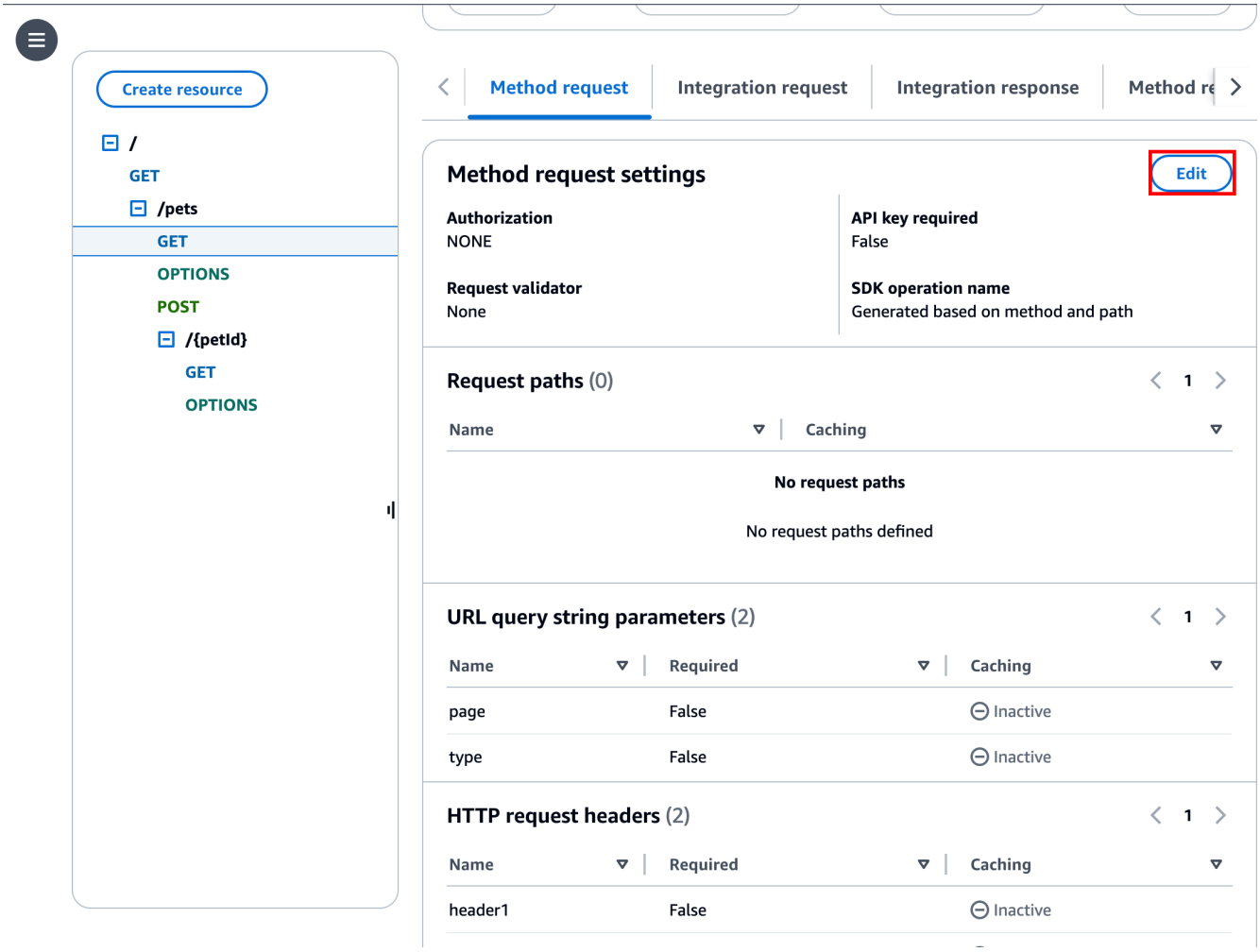
## 方法中需要 API 金鑰

下列程序說明如何設定 API 方法要求 API 金鑰。

### AWS 管理主控台

#### 設定 API 方法要求 API 金鑰

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 選擇 REST API。
3. 在 API Gateway 主導覽窗格中，選擇 Resources (資源)。
4. 在 Resources (資源) 下，建立新的方法或選擇現有的方法。
5. 在方法請求索引標籤的方法請求設定下，選擇編輯。




6. 選取需要 API 金鑰。
7. 選擇儲存。
8. 部署或重新部署 API 以使要求生效。

如果需要 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。

AWS CLI

以下 put-method 命令會建立需要 API 金鑰的 PUT 方法：

```
aws apigateway put-method \
    --rest-api-id 1234123412 \
    --resource-id a1b2c3 \
    --http-method PUT \
    --authorization-type "NONE" \
```

```
    --api-key-required
```

以下 update-method 命令會更新現有方法，以要求 API 金鑰：

```
aws apigateway update-method \
    --rest-api-id 1234123412 \
    --resource-id a1b2c3 \
    --http-method PUT \
    --patch-operations op="replace",path="/apiKeyRequired",value="true"
```

REST API

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`，將 `apiKeyRequired` 設定為 `true`。

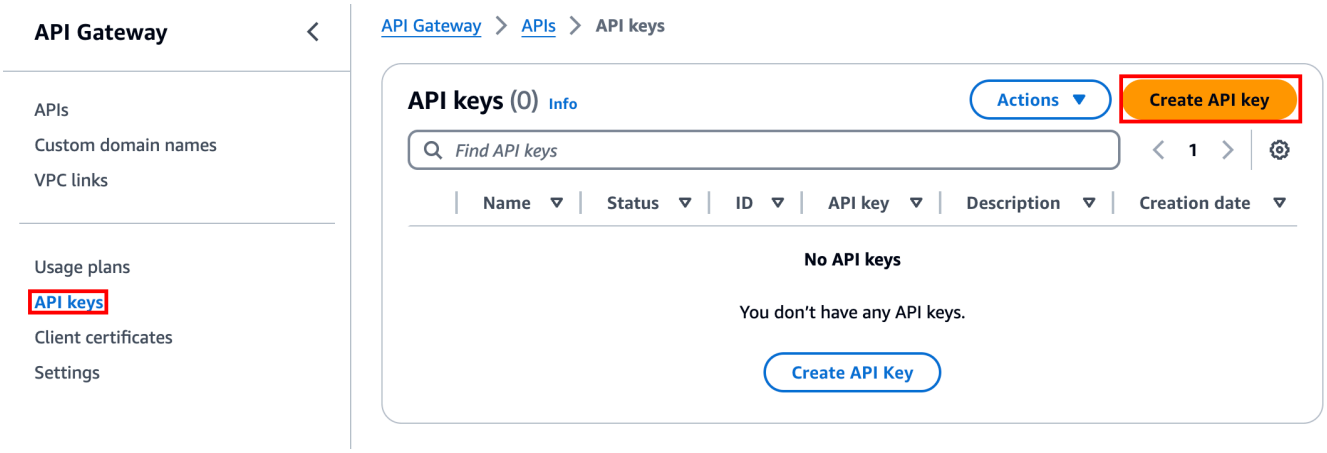
建立 API 金鑰

下列程序說明如何建立 API 金鑰。如果您想要匯入自己的 API 金鑰，請略過此步驟。

AWS 管理主控台

建立 API 金鑰

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 選擇 REST API。
3. 在 API Gateway 主導覽窗格中，選擇 API 金鑰。
4. 選擇建立 API 金鑰。



5. 針對名稱，輸入名稱。
6. 在描述，請輸入描述。
7. 針對 API 金鑰，選擇自動產生讓 API Gateway 產生金鑰值，或選擇自訂建立您自己的金鑰值。
8. 選擇儲存。

AWS CLI

以下 create-api-key 命令會建立 API 金鑰：

```
aws apigateway create-api-key \
   --name 'Dev API key' \
   --description 'API key for Devs' \
   --enabled
```

REST API

呼叫 apikey:create 來建立 API 金鑰。

匯入 API 金鑰

下列程序說明如何匯入 API 金鑰。如果您已建立 API 金鑰，請略過此步驟。

AWS 管理主控台

匯入 API 金鑰

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 選擇 REST API。
3. 在主導覽窗格中，選擇 API 金鑰。
4. 選擇動作下拉式選單，然後選擇匯入 API 金鑰。
5. 若要載入逗號分隔的金鑰檔案，請選擇選取檔案。您也可以在文字編輯器中輸入金鑰。如需檔案格式的資訊，請參閱 the section called “API Gateway API 金鑰檔案格式”。
6. 選擇發出警告時失敗，在發生錯誤時停止匯入，或選擇忽略警告，在出現警告時繼續匯入有效的金鑰項目。
7. 選擇匯入以匯入您的 API 金鑰。

AWS CLI

以下 import-api-keys 命令會匯入 API 金鑰：

```
aws apigateway import-api-key \
    a--body fileb://keys.csv \
    --format csv
```

REST API

呼叫 `apikey:import` 以從檔案匯入 API 金鑰。如需檔案格式，請參閱 API Gateway API 金鑰檔案格式。

您無法變更新 API 金鑰的值。在您建立 API 之後，請設定用量計畫。如需詳細資訊，請參閱在 API Gateway 中設定 REST API 的用量計畫。

## 在 API Gateway 中設定 REST API 的用量計畫

建立用量計畫之前，請確認您已設定 API 金鑰。如需詳細資訊，請參閱在 API Gateway 中設定 REST API 的 API 金鑰。

主題

- 遷移您的 API 到預設用量計劃 (如需要)
- 建立用量計劃
- 將階段新增至用量計畫
- 將 API 階段新增至用量計畫

### 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，則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 API 階段值 (`apiId` 和 `stage`) 與內含 API 金鑰 (透過 `UsagePlanKey`) 的關聯，而非使用 `ApiKey` `stageKeys` 屬性。

若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 APIs 遷移至預設用量計劃 AWS CLI ，如下所示：

使用 遷移至預設用量計劃 AWS CLI

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"
    }
]
```

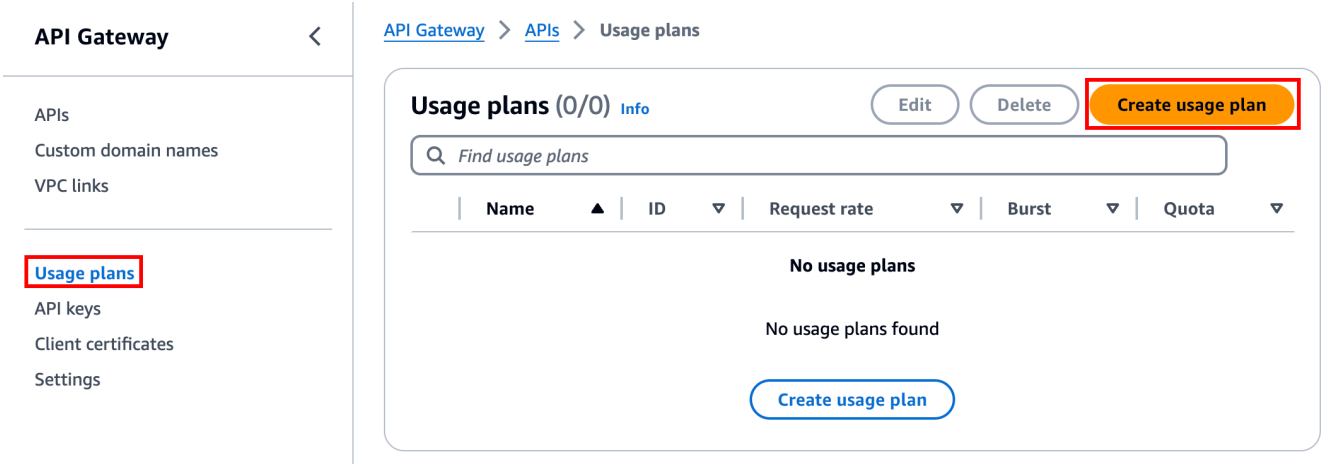
建立用量計劃

下列程序說明如何建立用量計劃。

AWS 管理主控台

建立用量計劃

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 在 API Gateway 主導覽窗格中，選擇用量計劃，然後選擇建立用量計劃。




3. 針對名稱，輸入名稱。

4. 在描述，請輸入描述。
5. 根據預設，用量計劃會啟用限流。輸入用量計劃的費率和暴增。選擇限流以關閉限流。
6. 根據預設，用量計劃會針對一段時間啟用一個配額。針對請求，輸入使用者在您的用量計劃期間內可提出的請求總數。選擇配額以關閉配額。
7. 選擇建立用量計劃。

AWS CLI

以下 create-usage-plan 命令會建立用量計畫，此計畫會在月初重設：

```
aws apigateway create-usage-plan \
    --name "New Usage Plan" \
    --description "A new usage plan" \
    --throttle burstLimit=10,rateLimit=5 \
    --quota limit=500,offset=0,period=MONTH
```

REST API

呼叫 usageplan:create 來建立用量計劃。

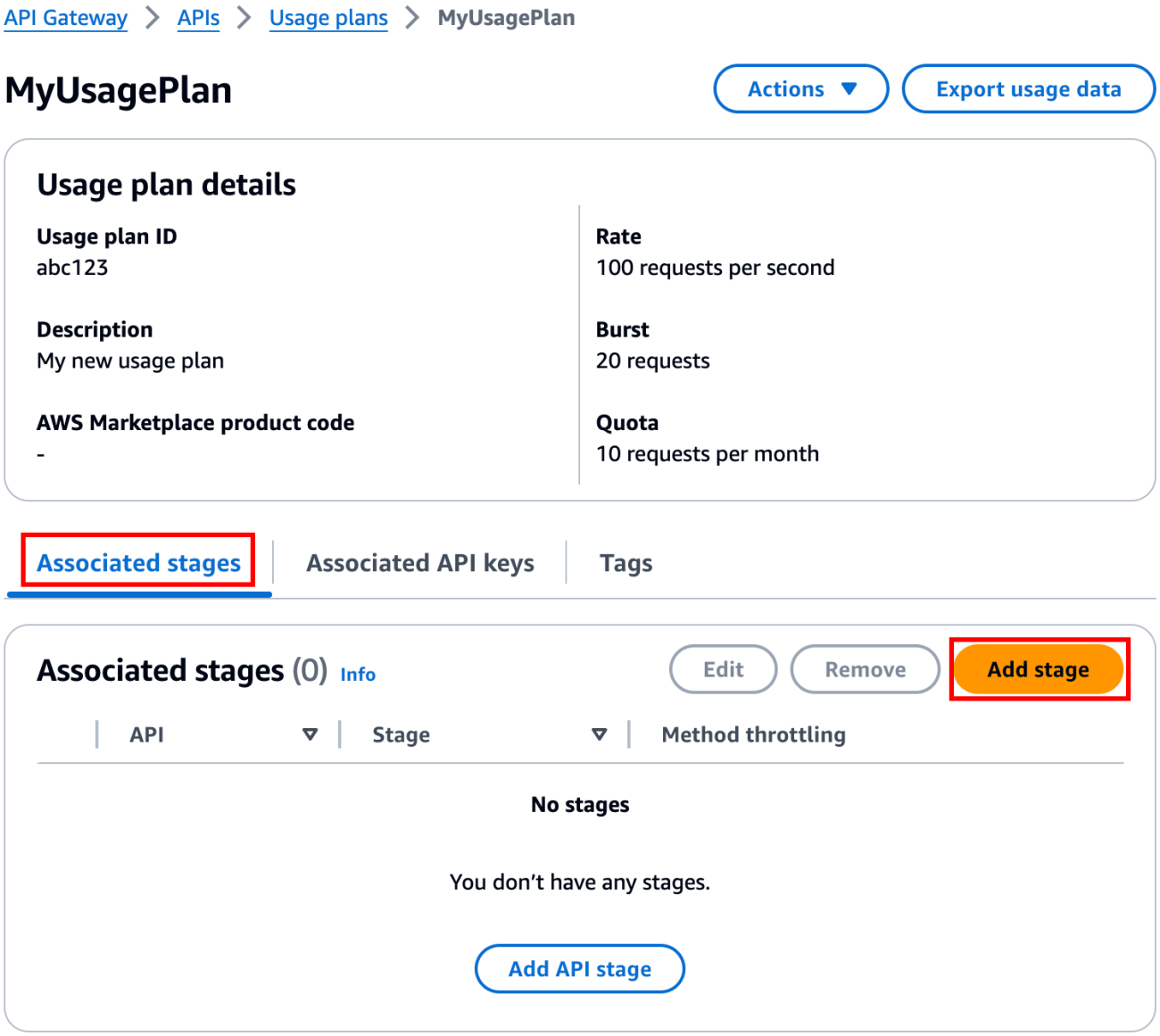
### 將階段新增至用量計畫

下列程序說明如何將階段新增至用量計畫。

AWS 管理主控台

將階段新增至用量計畫

1. 選取您的用量計劃。
2. 在關聯的階段索引標籤下，選擇新增階段。




3. 針對 API，選取 API。
4. 針對階段，選取階段。
5. (選用) 若要開啟方法層級限流，請執行下列操作：
   a. 選擇方法層級限流，然後選擇新增方法。
   b. 針對資源，從 API 中選取資源。
   c. 針對方法，從 API 中選取方法。
   d. 輸入用量計劃的費率和暴增。
6. 選擇新增至用量計劃。

AWS CLI

以下 update-usage-plan 命令會將 API 的 Prod 階段新增至用量計畫：

```
aws apigateway update-usage-plan \
    --usage-plan-id abc123 \
    --patch-operations op="add",path="/apiStages",value="a1b1c2:Prod"
```

REST API

呼叫 usageplan:update 以更新用量計畫。

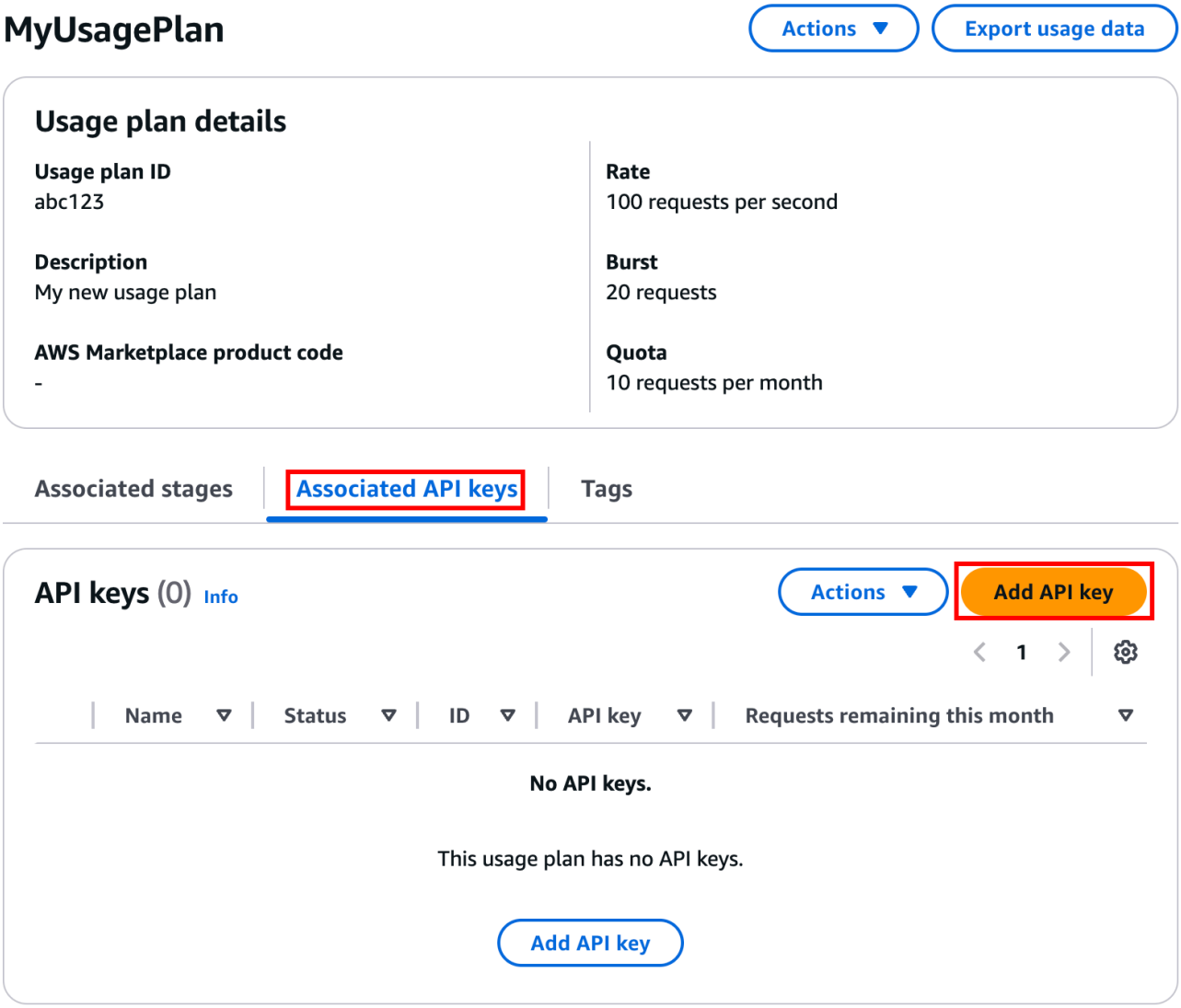
### 將 API 階段新增至用量計畫

下列程序說明如何將 API 金鑰新增至用量計畫。

AWS 管理主控台

將金鑰新增至用量計畫

1. 在關聯的 API 金鑰索引標籤下，選擇新增 API 金鑰。




2. a. 若要將現有金鑰與用量計劃建立關聯，請選取新增現有金鑰，然後從下拉式選單選取您現有的金鑰。

   b. 若要建立新的 API 金鑰，請選取建立並新增新金鑰，然後建立新金鑰。如需如何建立新金鑰的詳細資訊，請參閱 建立 API 金鑰。

3. 選擇新增 API 金鑰。

AWS CLI

以下 create-usage-plan-key 命令會將現有的 API 金鑰與用量計畫建立關聯：

```
aws apigateway create-usage-plan-key \
    --usage-plan-id a1b2c3 \
```

```
    --key-type "API_KEY" \
    --key-id aaa111bbb
```

REST API

呼叫 usageplankey:create，以將現有的 API 金鑰與用量計畫建立關聯。

您也可以在匯入 API 金鑰時，直接將金鑰與用量計畫建立關聯。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=fase 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。

## 在 API Gateway 中維護 REST API 的用量計畫

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配額。

AWS 管理主控台

監控已使用及剩餘的配額

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 在 API Gateway 主導覽窗格中，選擇用量計劃。
3. 選取用量計劃。
4. 選擇關聯的 API 金鑰索引標籤，以查看每個金鑰在期間內剩餘的請求數。
5. (選用) 選擇匯出用量資料，然後選擇從日期和到日期。接著選擇 JSON 或 CSV 作為匯出的資料格式，然後選擇匯出。

   以下範例顯示匯出的檔案。

```
{
    "px1KW6...qBazOJH": [
      [
        0,
        5000
      ],
      [
        0,
        5000
      ],
      [
        0,
        10
      ]
    ]
  }
```

   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。

下列程序說明如何修改配額。

擴充剩餘的配額

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。

2. 在 API Gateway 主導覽窗格中，選擇用量計劃。
3. 選取用量計劃。
4. 選擇關聯的 API 金鑰索引標籤，以查看每個金鑰在期間內剩餘的請求數。
5. 選取 API 金鑰，然後選擇授予用量延伸。
6. 輸入剩餘的請求配額的數量。您可以針對您的用量計劃期間增加剩餘的請求數，也可以減少剩餘的請求數。
7. 選擇更新配額。

AWS CLI

以下 update-usage-plan 範例會新增、移除或修改用量計畫中的方法層級限流設定。

> **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}}"'
```

REST API

呼叫 usageplan:update 以維護用量計畫。

## 使用 建立和設定 API 金鑰和用量計劃 CloudFormation

您可以使用 CloudFormation 在 API 方法上要求 API 金鑰，並建立 API 的使用計劃。範本範例 CloudFormation 會執行下列動作：

- 使用 GET 與 POST 方法建立 API Gateway API。
- GET 與 POST 方法需要 API 金鑰。此 API 會從每個傳入要求的 X-API-KEY 標頭接收金鑰。
- 建立 API 金鑰
- 建立用量計畫以指定每月 1,000 個要求的每月配額、每秒 100 個要求的限流速率限制，以及每秒 200 個要求的限流高載限制。
- 指定每秒 50 個請求的方法層級限流速率限制，以及 GET 方法層級限流爆量限制 (每秒 100 個請求)。
- 將 API 階段和 API 金鑰與用量計劃建立關聯。

```
AWSTemplateFormatVersion: 2010-09-09
Parameters:
  StageName:
    Type: String
    Default: v1
    Description: Name of API stage.
  KeyName:
    Type: String
    Default: MyKeyName
    Description: Name of an API key
```

```
Resources:
  Api:
    Type: 'AWS::ApiGateway::RestApi'
    Properties:
      Name: keys-api
      ApiKeySourceType: HEADER
  PetsResource:
    Type: 'AWS::ApiGateway::Resource'
    Properties:
      RestApiId: !Ref Api
      ParentId: !GetAtt Api.RootResourceId
      PathPart: 'pets'
  PetsMethodGet:
    Type: 'AWS::ApiGateway::Method'
    Properties:
      RestApiId: !Ref Api
      ResourceId: !Ref PetsResource
      HttpMethod: GET
      ApiKeyRequired: true
      AuthorizationType: NONE
      Integration:
        Type: HTTP_PROXY
        IntegrationHttpMethod: GET
        Uri: http://petstore-demo-endpoint.execute-api.com/petstore/pets/
  PetsMethodPost:
    Type: 'AWS::ApiGateway::Method'
    Properties:
      RestApiId: !Ref Api
      ResourceId: !Ref PetsResource
      HttpMethod: POST
      ApiKeyRequired: true
      AuthorizationType: NONE
      Integration:
        Type: HTTP_PROXY
        IntegrationHttpMethod: GET
        Uri: http://petstore-demo-endpoint.execute-api.com/petstore/pets/
  ApiDeployment:
    Type: 'AWS::ApiGateway::Deployment'
    DependsOn:
      - PetsMethodGet
    Properties:
      RestApiId: !Ref Api
      StageName: !Sub '${StageName}'
  UsagePlan:
```

```
    Type: AWS::ApiGateway::UsagePlan
    DependsOn:
      - ApiDeployment
    Properties:
      Description: Example usage plan with a monthly quota of 1000 calls and method-
level throttling for /pets GET
      ApiStages:
        - ApiId: !Ref Api
          Stage: !Sub '${StageName}'
          Throttle:
            "/pets/GET":
              RateLimit: 50.0
              BurstLimit: 100
      Quota:
        Limit: 1000
        Period: MONTH
      Throttle:
        RateLimit: 100.0
        BurstLimit: 200
      UsagePlanName: "My Usage Plan"
  ApiKey:
    Type: AWS::ApiGateway::ApiKey
    Properties:
      Description: API Key
      Name: !Sub '${KeyName}'
      Enabled: True
  UsagePlanKey:
    Type: AWS::ApiGateway::UsagePlanKey
    Properties:
      KeyId: !Ref ApiKey
      KeyType: API_KEY
      UsagePlanId: !Ref UsagePlan
Outputs:
  ApiRootUrl:
    Description: Root Url of the API
    Value: !Sub 'https://${Api}.execute-api.${AWS::Region}.amazonaws.com/${StageName}'
```

## 設定使用具有 OpenAPI 定義的 API 金鑰方法

您可以使用 OpenAPI 定義來要求方法上的 API 金鑰。

針對每個方法建立安全需求物件，要求 API 金鑰調用該方法。然後，在安全定義中定義 `api_key`。建立 API 之後，將新的 API 階段新增至您的用量計畫。

下列範例會建立 API，並且需要 POST 和 GET 方法的 API 金鑰：

OpenAPI 2.0

```
{
  "swagger" : "2.0",
  "info" : {
    "version" : "2024-03-14T20:20:12Z",
    "title" : "keys-api"
  },
  "basePath" : "/v1",
  "schemes" : [ "https" ],
  "paths" : {
    "/pets" : {
      "get" : {
        "responses" : { },
        "security" : [ {
          "api_key" : [ ]
        } ],
        "x-amazon-apigateway-integration" : {
          "type" : "http_proxy",
          "httpMethod" : "GET",
          "uri" : "http://petstore-demo-endpoint.execute-api.com/petstore/pets/",
          "passthroughBehavior" : "when_no_match"
        }
      },
      "post" : {
        "responses" : { },
        "security" : [ {
          "api_key" : [ ]
        } ],
        "x-amazon-apigateway-integration" : {
          "type" : "http_proxy",
          "httpMethod" : "GET",
          "uri" : "http://petstore-demo-endpoint.execute-api.com/petstore/pets/",
          "passthroughBehavior" : "when_no_match"
        }
      }
    }
  },
  "securityDefinitions" : {
    "api_key" : {
      "type" : "apiKey",
      "name" : "x-api-key",
```

```
      "in" : "header"
    }
  }
}
```

## OpenAPI 3.0

```
{
  "openapi" : "3.0.1",
  "info" : {
    "title" : "keys-api",
    "version" : "2024-03-14T20:20:12Z"
  },
  "servers" : [ {
    "url" : "{basePath}",
    "variables" : {
      "basePath" : {
        "default" : "v1"
      }
    }
  } ],
  "paths" : {
    "/pets" : {
      "get" : {
        "security" : [ {
          "api_key" : [ ]
        } ],
        "x-amazon-apigateway-integration" : {
          "httpMethod" : "GET",
          "uri" : "http://petstore-demo-endpoint.execute-api.com/petstore/pets/",
          "passthroughBehavior" : "when_no_match",
          "type" : "http_proxy"
        }
      },
      "post" : {
        "security" : [ {
          "api_key" : [ ]
        } ],
        "x-amazon-apigateway-integration" : {
          "httpMethod" : "GET",
          "uri" : "http://petstore-demo-endpoint.execute-api.com/petstore/pets/",
          "passthroughBehavior" : "when_no_match",
          "type" : "http_proxy"
```

```
        }
      }
    }
  },
  "components" : {
    "securitySchemes" : {
      "api_key" : {
        "type" : "apiKey",
        "name" : "x-api-key",
        "in" : "header"
      }
    }
  }
}
```

## 在 API Gateway 中測試 REST API 的用量計畫

以在 教學課程：匯入範例來建立 REST API 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 `xbvxlpijch`：

```
GET /testStage/pets?type=dog&page=1 HTTP/1.1
x-api-key: Hiorr45VR...c4GJc
Content-Type: application/x-www-form-urlencoded
Host: xbvxlpijch.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 `Hiorr45VR...c4GJc` 標頭中提供需要的 API 金鑰 (例如 `x-api-key`)。

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，它會在您的 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。

## 使用 API 金鑰呼叫方法

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 部署至階段。
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。
5. 使用 API 金鑰呼叫 API。下列範例 curl 命令會使用 API 金鑰，在 API prod 階段的 getUsers 資源上調用 GET 方法。

```
curl -H "X-API-Key: abcd1234" 'https://b123abcde4.execute-api.us-
west-2.amazonaws.com/prod/getUsers'
```

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。呼叫看起來與以下內容相似：

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 部署至階段。

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. 建立權杖型 Lambda 授權方。納入 `usageIdentifierKey:{api-key}` 作為授權回應的根層級屬性。如需有關建立字符型授權方的說明，請參閱 the section called “TOKEN 授權方 Lambda 函數範例”。
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。

## API Gateway 中 REST API 的文件

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，並在發布文件時分發檔案。

若要記載您的 API，您可以呼叫 API Gateway REST API、使用其中一個 AWS SDK、使用 AWS CLI for API Gateway，或使用 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。

若要與開發人員共用 API 文件，您可以使用開發人員入口網站。如需範例，請參閱 AWS 合作夥伴網路 (APN) 部落格上的將 ReadMe 與 API Gateway 整合，讓您的開發人員中樞保持在最新狀態，或如何使用 SmartBear 的 SwaggerHub 在 Amazon API Gateway 上簡化 API 開發。

主題

- API Gateway 中的 API 文件表示
- 使用 API Gateway 主控台記載 API
- 使用 API Gateway 主控台發布 API 文件
- 使用 API Gateway REST API 記載 API
- 使用 API Gateway REST API 發布 API 文件
- 匯入 API 文件
- 在 API Gateway 中控制 API 文件的存取

# 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
- 文件版本

## 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。這包括您文件的任何 UTF-8 內容和所有主要當地語系化語言。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。

### 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_ 或 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。相較之下，my-info 與 OpenAPI 規格不相容。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 映射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 | state | name | 備註 |
|---|---|---|---|---|
| 1 | /pets | * | id | 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。 |
| 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 |

| 案例 | path | state | name | 備註 | |
|---|---|---|---|---|---|
| | | | | 文件會由符合位置模式的實體繼承。 | |

| 案例 | path | state | 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，/pets 資源上任何方法的 400 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 | `{ "location": { "type": "API" }, ...` | type | 不適用 | 否 |

| 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>}</pre> | | | |
| Resource | <pre>{<br>    "location": {<br>        "type": "RESOURCE<br>",<br>        "path":<br> "*resource_path* "<br>    },<br>    ...<br>}</pre> | type | path 的預設值為 /。 | 否 |
| 方法 | <pre>{<br>    "location": {<br>        "type":<br> "METHOD",<br>        "path":<br> "*resource_path* ",<br>        "method":<br> "*http_verb* "<br>    },<br>    ...<br>}</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。 |

| 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 |
| --- | --- | --- | --- | --- |
| 查詢參數 | {<br>"location": {<br>"type": "QUERY_PARAMETER",<br>"path": "*resource_path* ",<br>"method": "*HTTP_verb* ",<br>"name": "*query_parameter_name* "<br>},<br>...<br>} | 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 | {<br>"location": {<br>"type": "REQUEST_BODY",<br>"path": "*resource_path* ",<br>"method": "*http_verb* "<br>},<br>...<br>} | 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 |
| --- | --- | --- | --- | --- |
| 請求標頭參數 | {<br>"location": {<br>"type": "REQUEST_HEADER",<br>"path": "*resource_path* ",<br>"method": "*HTTP_verb* ",<br>"name": "*header_name* "<br>},<br>...<br>} | 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 | {<br>"location": {<br>"type": "PATH_PARAMETER",<br>"path": "*resource*/{*path_parameter_name* }",<br>"method": "*HTTP_verb* ",<br>"name": "*path_parameter_name* "<br>},<br>...<br>} | 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。 |

| 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 | {<br>"location": {<br>"type": "RESPONSE",<br>"path": "*resource_path* ",<br>"method": "*http_verb* ",<br>"statusCode": "*status_code* "<br>},<br>...<br>} | 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。 |
| 回應標頭 | {<br>"location": {<br>"type": "RESPONSE_HEADER",<br>"path": "*resource_path* ",<br>"method": "*http_verb* ",<br>"statusCode": "*status_code* ",<br>"name": "*header_name* "<br>},<br>...<br>} | 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。 |

| 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 |
| --- | --- | --- | --- | --- |
| 回應內文 | {<br>"location": {<br>"type": "RESPONSE_BODY",<br>"path": "*resource_path* ",<br>"method": "*http_verb* ",<br>"statusCode": "*status_code* "<br>},<br>...<br>} | 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 | {<br>"location": {<br>"type": "AUTHORIZER",<br>"name": "*authorizer_name* "<br>},<br>...<br>} | type | 不適用 | 否 |
| 模型 | {<br>"location": {<br>"type": "MODEL",<br>"name": "*model_name* "<br>},<br>...<br>} | 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。發布 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 主控台記載 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 中的說明進行。

主題

- 記錄 API 實體
- 記錄 RESOURCE 實體
- 記錄 METHOD 實體
- 記錄 QUERY_PARAMETER 實體
- 記錄 PATH_PARAMETER 實體
- 記錄 REQUEST_HEADER 實體
- 記錄 REQUEST_BODY 實體
- 記錄 RESPONSE 實體
- 記錄 RESPONSE_HEADER 實體
- 記錄 RESPONSE_BODY 實體
- 記錄 MODEL 實體
- 記錄 AUTHORIZER 實體

### 記錄 **API** 實體

若要為 API 實體新增文件組件，請執行下列動作：

1. 在主導覽窗格中，選擇文件，然後選擇建立文件組件。
2. 針對文件類型，選取 API。

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 映射。

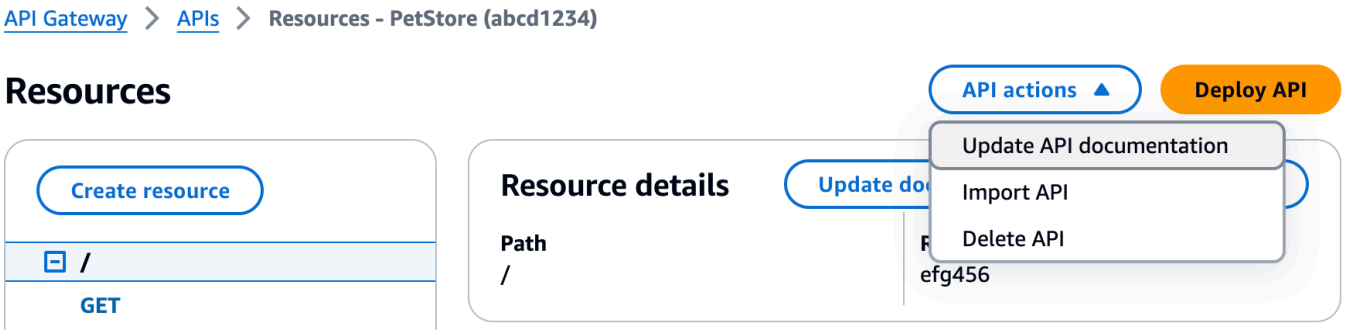
```
{
  "info": {
    "description": "Your first API Gateway API.",
    "contact": {
        "name": "John Doe",
        "email": "john.doe@api.com"
    }
  }
}
```

> **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。

3. 選擇建立文件組件。

若要在資源窗格中為 API 實體新增文件組件，請執行下列動作：

1. 在主導覽窗格中，選擇資源。
2. 選擇 API 動作選單，然後選擇更新 API 文件。




若要編輯現有文件組件，請執行下列動作：

1. 在文件窗格中，選擇資源和方法索引標籤。
2. 選取您的 API 名稱，然後在 API 卡片上選擇編輯。

### 記錄 RESOURCE 實體

若要為 RESOURCE 實體新增文件組件，請執行下列動作：

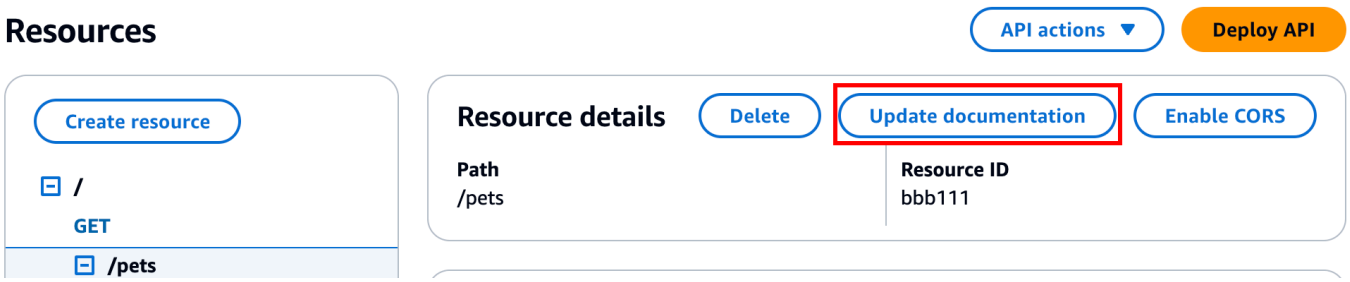
1. 在主導覽窗格中，選擇文件，然後選擇建立文件組件。
2. 針對文件類型，選取資源。
3. 在路徑中輸入路徑。
4. 在文字編輯器中輸入說明，例如：

```
{
    "description": "The PetStore's root resource."
}
```

5. 選擇建立文件組件。您可以為未列出的資源建立文件。
6. 如有需要，請重複這些步驟來新增或編輯其他文件組件。

若要在資源窗格中為 RESOURCE 實體新增文件組件，請執行下列動作：

1. 在主導覽窗格中，選擇資源。
2. 選擇資源，然後選擇更新文件。




若要編輯現有文件組件，請執行下列動作：

1. 在文件窗格中，選擇資源和方法索引標籤。
2. 選取包含文件組件的資源，然後選擇編輯。

### 記錄 METHOD 實體

若要為 METHOD 實體新增文件組件，請執行下列動作：

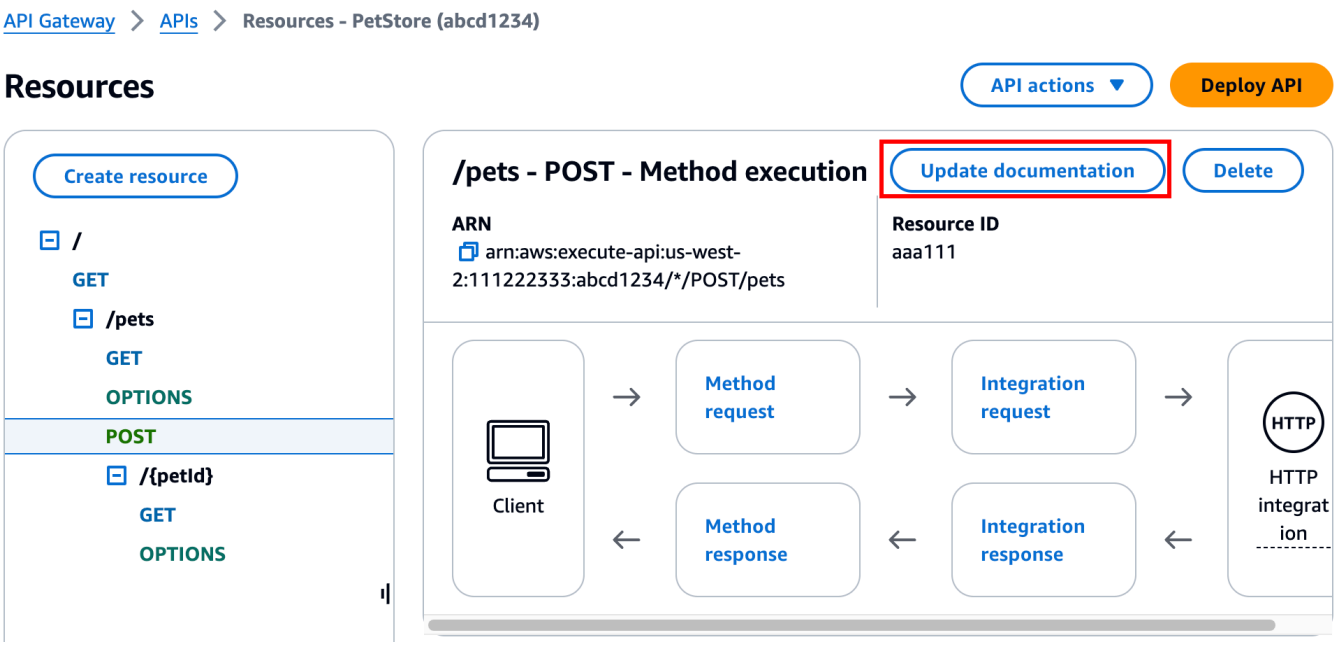
1. 在主導覽窗格中，選擇文件，然後選擇建立文件組件。
2. 針對文件類型，選取方法。
3. 在路徑中輸入路徑。
4. 針對方法，選取 HTTP 動詞。
5. 在文字編輯器中輸入說明，例如：

```
{
  "tags" : [ "pets" ],
  "summary" : "List all pets"
}
```

6. 選擇建立文件組件。您可以為未列出的方法建立文件。
7. 如有需要，請重複這些步驟來新增或編輯其他文件組件。

若要在資源窗格中為 METHOD 實體新增文件組件，請執行下列動作：

1. 在主導覽窗格中，選擇資源。
2. 選擇方法，然後選擇更新文件。




若要編輯現有文件組件，請執行下列動作：

1. 在文件窗格中，選擇資源和方法索引標籤。
2. 您可以選取方法，或是選取包含該方法的資源，然後使用搜尋列尋找並選取您的文件組件。
3. 選擇編輯。

### 記錄 QUERY_PARAMETER 實體

若要為 QUERY_PARAMETER 實體新增文件組件，請執行下列動作：

1. 在主導覽窗格中，選擇文件，然後選擇建立文件組件。
2. 針對文件類型，選取查詢參數。
3. 在路徑中輸入路徑。
4. 針對方法，選取 HTTP 動詞。
5. 針對名稱，輸入名稱。
6. 在文字編輯器中輸入說明。
7. 選擇建立文件組件。您可以為未列出的查詢參數建立文件。
8. 如有需要，請重複這些步驟來新增或編輯其他文件組件。

若要編輯現有文件組件，請執行下列動作：

1. 在文件窗格中，選擇資源和方法索引標籤。
2. 您可以選取查詢參數，或是選取包含該查詢參數的資源，然後使用搜尋列尋找並選取您的文件組件。
3. 選擇編輯。

### 記錄 PATH_PARAMETER 實體

若要為 PATH_PARAMETER 實體新增文件組件，請執行下列動作：

1. 在主導覽窗格中，選擇文件，然後選擇建立文件組件。
2. 針對文件類型，選取路徑參數。
3. 在路徑中輸入路徑。
4. 針對方法，選取 HTTP 動詞。
5. 針對名稱，輸入名稱。

6. 在文字編輯器中輸入說明。
7. 選擇建立文件組件。您可以為未列出的路徑參數建立文件。
8. 如有需要，請重複這些步驟來新增或編輯其他文件組件。

若要編輯現有文件組件，請執行下列動作：

1. 在文件窗格中，選擇資源和方法索引標籤。
2. 您可以選取路徑參數，或是選取包含該路徑參數的資源，然後使用搜尋列尋找並選取您的文件組件。
3. 選擇編輯。

### 記錄 REQUEST_HEADER 實體

若要為 REQUEST_HEADER 實體新增文件組件，請執行下列動作：

1. 在主導覽窗格中，選擇文件，然後選擇建立文件組件。
2. 針對文件類型，選取請求標頭。
3. 在路徑中，輸入請求標頭的路徑。
4. 針對方法，選取 HTTP 動詞。
5. 針對名稱，輸入名稱。
6. 在文字編輯器中輸入說明。
7. 選擇建立文件組件。您可以為未列出的請求標頭建立文件。
8. 如有需要，請重複這些步驟來新增或編輯其他文件組件。

若要編輯現有文件組件，請執行下列動作：

1. 在文件窗格中，選擇資源和方法索引標籤。
2. 您可以選取請求標頭，或是選取包含該請求標頭的資源，然後使用搜尋列尋找並選取您的文件組件。
3. 選擇編輯。

### 記錄 REQUEST_BODY 實體

若要為 REQUEST_BODY 實體新增文件組件，請執行下列動作：

1. 在主導覽窗格中，選擇文件，然後選擇建立文件組件。
2. 針對文件類型，選取請求內文。
3. 在路徑中，輸入請求內文的路徑。
4. 針對方法，選取 HTTP 動詞。
5. 在文字編輯器中輸入說明。
6. 選擇建立文件組件。您可以為未列出的請求內文建立文件。
7. 如有需要，請重複這些步驟來新增或編輯其他文件組件。

若要編輯現有文件組件，請執行下列動作：

1. 在文件窗格中，選擇資源和方法索引標籤。
2. 您可以選取請求內文，或是選取包含該請求內文的資源，然後使用搜尋列尋找並選取您的文件組件。
3. 選擇編輯。

### 記錄 RESPONSE 實體

若要為 RESPONSE 實體新增文件組件，請執行下列動作：

1. 在主導覽窗格中，選擇文件，然後選擇建立文件組件。
2. 針對文件類型，選取回應 (狀態碼)。
3. 在路徑中，輸入回應的路徑。
4. 針對方法，選取 HTTP 動詞。
5. 針對狀態碼，輸入 HTTP 狀態碼。
6. 在文字編輯器中輸入說明。
7. 選擇建立文件組件。您可以為未列出的回應狀態碼建立文件。
8. 如有需要，請重複這些步驟來新增或編輯其他文件組件。

若要編輯現有文件組件，請執行下列動作：

1. 在文件窗格中，選擇資源和方法索引標籤。
2. 您可以選取回應狀態碼，或是選取包含該回應狀態碼的資源，然後使用搜尋列尋找並選取您的文件組件。

3. 選擇編輯。

### 記錄 RESPONSE_HEADER 實體

若要為 RESPONSE_HEADER 實體新增文件組件，請執行下列動作：

1. 在主導覽窗格中，選擇文件，然後選擇建立文件組件。
2. 針對文件類型，選取回應標頭。
3. 在路徑中，輸入回應標頭的路徑。
4. 針對方法，選取 HTTP 動詞。
5. 針對狀態碼，輸入 HTTP 狀態碼。
6. 在文字編輯器中輸入說明。
7. 選擇建立文件組件。您可以為未列出的回應標頭建立文件。
8. 如有需要，請重複這些步驟來新增或編輯其他文件組件。

若要編輯現有文件組件，請執行下列動作：

1. 在文件窗格中，選擇資源和方法索引標籤。
2. 您可以選取回應標頭，或是選取包含該回應標頭的資源，然後使用搜尋列尋找並選取您的文件組件。
3. 選擇編輯。

### 記錄 RESPONSE_BODY 實體

若要為 RESPONSE_BODY 實體新增文件組件，請執行下列動作：

1. 在主導覽窗格中，選擇文件，然後選擇建立文件組件。
2. 針對文件類型，選取回應內文。
3. 在路徑中，輸入回應內文的路徑。
4. 針對方法，選取 HTTP 動詞。
5. 針對狀態碼，輸入 HTTP 狀態碼。
6. 在文字編輯器中輸入說明。
7. 選擇建立文件組件。您可以為未列出的回應內文建立文件。
8. 如有需要，請重複這些步驟來新增或編輯其他文件組件。

若要編輯現有文件組件，請執行下列動作：

1. 在文件窗格中，選擇資源和方法索引標籤。
2. 您可以選取回應內文，或是選取包含該回應內文的資源，然後使用搜尋列尋找並選取您的文件組件。
3. 選擇編輯。

### 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 DocumentPart 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"
  }
}
```

以及

```
{
  "location": {
    "type": "MODEL",
    "name": "Error.message"
  },
```

```
  "properties": {
    "description": "An error message."
  }
}
```

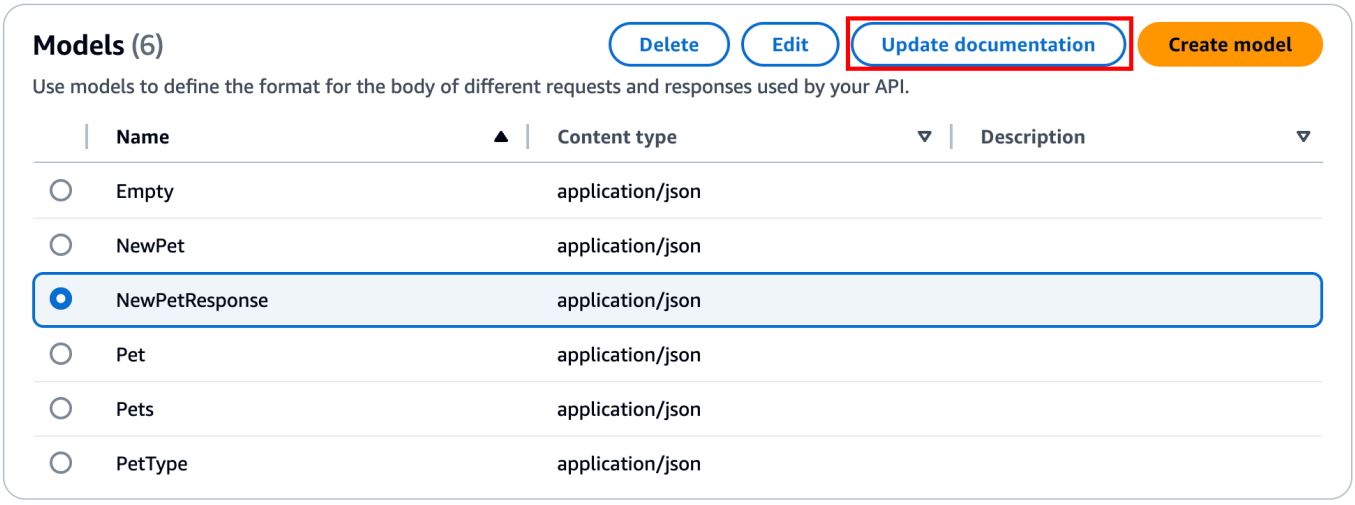
匯出 API 之後，DocumentationPart 的屬性會覆寫原始結構描述中的值。

若要為 MODEL 實體新增文件組件，請執行下列動作：

1. 在主導覽窗格中，選擇文件，然後選擇建立文件組件。
2. 針對文件類型，選取模型。
3. 針對名稱，輸入模型的名稱。
4. 在文字編輯器中輸入說明。
5. 選擇建立文件組件。您可以為未列出的模型建立文件。
6. 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。

若要在模型窗格中為 MODEL 實體新增文件組件，請執行下列動作：

1. 在主導覽窗格中，選擇模型。
2. 選擇模型，然後選擇更新文件。




若要編輯現有文件組件，請執行下列動作：

1. 在文件窗格中，選擇模型索引標籤。

2. 使用搜尋列或選取模型，然後選擇編輯。

#### 記錄 AUTHORIZER 實體

若要為 AUTHORIZER 實體新增文件組件，請執行下列動作：

1. 在主導覽窗格中，選擇文件，然後選擇建立文件組件。
2. 針對文件類型，選取授權方。
3. 在名稱中輸入授權方的名稱。
4. 在文字編輯器中輸入說明。針對授權方的有效 `location` 欄位指定值。
5. 選擇建立文件組件。您可以為未列出的授權方建立文件。
6. 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。

若要編輯現有文件組件，請執行下列動作：

1. 在文件窗格中，選擇授權方索引標籤。
2. 使用搜尋列或選取授權方，然後選擇編輯。

## 使用 API Gateway 主控台發布 API 文件

下列程序說明如何發布文件版本。

使用 API Gateway 主控台發布文件版本

1. 在主導覽窗格中，選擇文件。
2. 選擇發布文件。
3. 設定發布：
   a. 針對階段，選取階段。
   b. 在版本中輸入版本識別碼，例如 `1.0.0`。
   c. 在描述，請輸入描述。
4. 選擇 Publish (發布)。

您現在可以將文件匯出到外部 OpenAPI 檔案，繼續下載已發布的文件。如需進一步了解，請參閱the section called “匯出 REST API”。

# 使用 API Gateway REST API 記載 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 中的說明進行。

主題

- 記錄 API 實體
- 記錄 RESOURCE 實體
- 記錄 METHOD 實體
- 記錄 QUERY_PARAMETER 實體
- 記錄 PATH_PARAMETER 實體
- 記錄 REQUEST_BODY 實體
- 記錄 REQUEST_HEADER 實體
- 記錄 RESPONSE 實體
- 記錄 RESPONSE_HEADER 實體
- 記錄 AUTHORIZER 實體
- 記錄 MODEL 實體
- 更新文件組件
- 列出文件組件

## 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: 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" : "API"
    },
    "properties": "{\n\t\"info\": {\n\t\t\"description\" : \"Your first API with Amazon
 API Gateway.\"\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 Gateway.\"\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: 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\"info\": {\n\t\t\"description\" : \"Your first API with Amazon API
 Gateway.\"\n\t}\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}"
}
```

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}"
}
```

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}"
}
```

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}"
}
```

若要新增路徑參數所指定之子資源的文件，請新增以 Resource 資源為目標的 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 parameter.\"\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
 parameter.\"\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}"
}
```

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 對應：

```
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\"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，請新增以 QUERY_PARAMETER 類型為目標的 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: 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" : "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: 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" : "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  \"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: 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" : "REQUEST_BODY",
         "path" : "/pets",
         "method" : "POST"
    },
    "properties": "{\n\t\"description\" : \"A Pet object to be added to PetStore.\"\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}"
```

```
}
```

### 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: 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" : "REQUEST_HEADER",
         "path" : "/pets",
         "method" : "GET",
         "name" : "x-my-token"
    },
    "properties": "{\n\t\"description\" : \"A custom token used to authorization the
 method invocation.\"\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，請新增以指定的授權方為目標的 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" : "AUTHORIZER",
         "name" : "myAuthorizer"
    },
    "properties": "{\n\t\"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}"
}
```

> **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 DocumentPart 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"
  }
}
```

以及

```
{
  "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，請新增以指定的模型為目標的 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" : "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": {
    "curies": {
      "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 對應取代為新的對應。

```
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" : "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 發布 API 文件

若要發布 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。
- 建立文件快照
- 更新文件快照
- 取得文件快照
- 建立文件快照與 API 階段的關聯
- 下載與階段相關聯的文件快照

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: 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",
    "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: 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": "/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

若要發布 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、其中一個軟體開發套件或適用於 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 文件組件、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": "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/
{basePath}",
         "variables": {
            "basePath": {
              "default": "/test"
            }
         }
      }
   ],
   "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",
        "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 `properties` 執行個體之任何非 OpenAPI 相容的 `DocumentationPart` 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 documenationpart:by-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布的 API 文件，請遵循使用 API Gateway 主控台匯出 REST API 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 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: 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

{
  "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. 在主導覽窗格中，選擇文件。
2. 選擇匯入。
3. 如果您有現有的文件，請選取覆寫或合併您的新文件。
4. 選擇選擇檔案，從磁碟機載入檔案，或是在檔案檢視中輸入檔案內容。如需範例，請參閱使用 API Gateway REST API 匯入文件組件中的範例請求承載。
5. 選擇匯入時處理警告的方式。選取警告失敗或忽略警告。如需詳細資訊，請參閱the section called “將 API 匯入 API Gateway 的錯誤和警告”。
6. 選擇匯入。

## 在 API Gateway 中控制 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。

若要授予文件團隊建立、更新和發佈 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* 是文件團隊 AWS 的帳戶 ID。

JSON

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

    {
      "Sid": "StmtDocPartsAddEditViewDelete",
      "Effect": "Allow",
      "Action": [
        "apigateway:GET",
        "apigateway:PUT",
        "apigateway:POST",
        "apigateway:PATCH",
        "apigateway:DELETE"
      ],
      "Resource": [
        "arn:aws:apigateway:us-east-1:111111111111:/restapis/*/documentation/*"
      ]
    }
```

```
  ]
}
```

如需設定 API Gateway 資源存取許可的資訊，請參閱 the section called “Amazon API Gateway 與 IAM 搭配運作的方式”。

# 在 API Gateway 中為 REST API 產生 SDK

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。您在建立、測試和部署 API 至某個階段後產生 SDK。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 軟體開發套件。

本節說明如何產 API Gateway API 的軟體開發套件。也會示範如何在 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，這會公開此簡易計算器 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 API Gateway 中部署 REST API。

主題

- 簡易計算器的 Lambda 函數
- API Gateway 中的簡單計算器 API
- 簡易計算器 API OpenAPI 定義
- 在 API Gateway 中產生 API 的 Java SDK
- 在 API Gateway 中產生 API 的 Android SDK
- 在 API Gateway 中產生 API 的 iOS SDK
- 在 API Gateway 中產生 REST API 的 JavaScript SDK
- 在 API Gateway 中產生 API 的 Ruby SDK
- 使用 API Gateway 中的 AWS CLI 命令產生 API SDKs

## 簡易計算器的 Lambda 函數

我們將在圖中使用 Node.js Lambda 函數執行加、減、乘、除的二進位運算。

主題


- 簡易計算器 Lambda 函數的輸入格式
- 簡易計算器 Lambda 函數的輸出格式
- 簡易計算器 Lambda 函數實作


### 簡易計算器 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 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 Lambda 函數實作

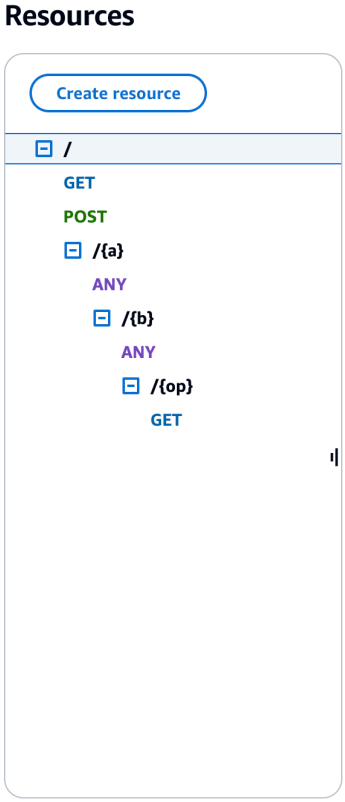
以下是 Lambda 函數的實作：

```
export const handler = async function (event, context) {
  console.log("Received event:", JSON.stringify(event));

  if (
    event.a === undefined ||
    event.b === undefined ||
    event.op === undefined
  ) {
    return "400 Invalid Input";
  }

  const res = {};
  res.a = Number(event.a);
  res.b = Number(event.b);
```

```
  res.op = event.op;
  if (isNaN(event.a) || isNaN(event.b)) {
    return "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) {
        return "400 Divide by Zero";
      } else {
        res.c = res.a / res.b;
      }
      break;
    default:
      return "400 Invalid Operator";
  }

  return res;
};
```

## API Gateway 中的簡單計算器 API

我們的簡易計算器 API 公開三種方法 (GET、POST、GET) 來呼叫the section called “簡易計算器的 Lambda 函數”。此 API 的圖形呈現如下：




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 會結合 HTTP 方法和路徑部分，以產生對應的軟體開發套件方法名稱。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，來覆寫預設軟體開發套件方法名稱。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 APIs Gateway 中開發 REST API。如果您初次使用 API Gateway，請先參閱選擇 AWS Lambda 整合教學課程。

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。為了描述回應內文資料類型，我們建立 `Output` 模型與 `Result` 模型。

建立輸入、輸出和結果的模型

1. 在主導覽窗格中，選擇模型。
2. 選擇建立模型。
3. 對於名稱，輸入 **input**。
4. 針對內容類型，輸入 **application/json**。

   如果找不到相符的內容類型，則不會執行請求驗證。若要使用相同的模型，而不論內容類型為何，請輸入 **$default**。
5. 針對模型結構描述，輸入下列模型：

```
{
    "$schema" : "$schema": "http://json-schema.org/draft-04/schema#",
    "type":"object",
    "properties":{
        "a":{"type":"number"},
        "b":{"type":"number"},
        "op":{"type":"string"}
    },
    "title":"Input"
}
```

6. 選擇建立模型。
7. 重複以下步驟來建立 `Output` 模型和 `Result` 模型。

   針對 `Output` 模型，為模型結構描述輸入下列內容：

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "type": "object",
    "properties": {
```

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

針對 Result 模型，為模型結構描述輸入下列內容。將 API ID abc123 取代為您的 API ID。

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "type":"object",
    "properties":{
        "input":{
            "$ref":"https://apigateway.amazonaws.com/restapis/abc123/models/Input"
        },
        "output":{
            "$ref":"https://apigateway.amazonaws.com/restapis/abc123/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：

設定 GET / URL 查詢字串參數

1. 在方法請求區段中，針對根 (/) 資源上的 GET 方法，選擇編輯。
2. 選擇 URL 查詢字串參數，然後執行下列動作：

   a. 選擇新增查詢字串。

   b. 對於名稱，輸入 **a**。

   c. 將必要和快取保持關閉。

   d. 讓快取保持關閉。

   重複相同的步驟，並建立名為 **b** 的查詢字串和名為 **op** 的查詢字串。
3. 選擇儲存。

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。

設定承載做為後端輸入的資料模型

1. 在方法請求區段中，針對根 (/) 資源上的 POST 方法，選擇編輯。
2. 選擇請求內文。
3. 選擇 Add model (新增模型)。
4. 針對內容類型，輸入 **application/json**。
5. 針對模型，選取輸入。
6. 選擇儲存。

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。

設定後端結果輸出的資料模型

1. 選取 /{a}/{b}/{op} 資源，然後選擇 GET 方法。
2. 在方法回應索引標籤上的回應 200 下，選擇編輯。
3. 在請求內文下，選擇新增模型。
4. 針對內容類型，輸入 **application/json**。
5. 針對模型，選取結果。
6. 選擇儲存。

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。

## 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}"
              }
            }
          },
          "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
$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"
          },
          {
            "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"
    }
  }
}
```

OpenAPI 3.0

```
{
  "openapi" : "3.0.1",
  "info" : {
    "title" : "SimpleCalc",
    "version" : "2016-09-29T20:27:30Z"
  },
  "servers" : [ {
    "url" : "https://t6dve4zn25.execute-api.us-west-2.amazonaws.com/{basePath}",
    "variables" : {
      "basePath" : {
        "default" : "demo"
      }
    }
  } ],
  "paths" : {
    "/{a}/{b}" : {
      "x-amazon-apigateway-any-method" : {
        "parameters" : [ {
          "name" : "a",
          "in" : "path",
          "required" : true,
          "schema" : {
            "type" : "string"
          }
        }, {
          "name" : "b",
          "in" : "path",
          "required" : true,
          "schema" : {
            "type" : "string"
          }
        } ],
        "responses" : {
          "404" : {
            "description" : "404 response",
            "content" : { }
          }
        },
        "x-amazon-apigateway-integration" : {
          "type" : "mock",
          "responses" : {
            "default" : {
```

```
            "statusCode" : "404",
            "responseTemplates" : {
              "application/json" : "{ \"Message\" : \"Can't $context.httpMethod
$context.resourcePath\" }"
            }
          }
        },
        "requestTemplates" : {
          "application/json" : "{\"statusCode\": 200}"
        },
        "passthroughBehavior" : "when_no_match"
      }
    }
  },
  "/{a}/{b}/{op}" : {
    "get" : {
      "parameters" : [ {
        "name" : "a",
        "in" : "path",
        "required" : true,
        "schema" : {
          "type" : "string"
        }
      }, {
        "name" : "b",
        "in" : "path",
        "required" : true,
        "schema" : {
          "type" : "string"
        }
      }, {
        "name" : "op",
        "in" : "path",
        "required" : true,
        "schema" : {
          "type" : "string"
        }
      } ],
      "responses" : {
        "200" : {
          "description" : "200 response",
          "content" : {
            "application/json" : {
              "schema" : {
```

```
                  "$ref" : "#/components/schemas/Result"
                }
              }
            }
          }
        },
        "x-amazon-apigateway-integration" : {
          "type" : "aws",
          "httpMethod" : "POST",
          "uri" : "arn:aws:apigateway:us-west-2:lambda:path/2015-03-31/functions/
arn:aws:lambda:us-west-2:111122223333:function:Calc/invocations",
          "responses" : {
            "default" : {
              "statusCode" : "200",
              "responseTemplates" : {
                "application/json" : "#set($inputRoot = $input.path('$'))\n{\n
 \"input\" : {\n    \"a\" : $inputRoot.a,\n    \"b\" : $inputRoot.b,\n    \"op\" :
 \"$inputRoot.op\"\n  },\n  \"output\" : {\n    \"c\" : $inputRoot.c\n  }\n}"
              }
            }
          },
          "requestTemplates" : {
            "application/json" : "#set($inputRoot = $input.path('$'))\n{\n
  \"a\" : $input.params('a'),\n  \"b\" : $input.params('b'),\n  \"op\" :
 \"$input.params('op')\"\n}"
          },
          "passthroughBehavior" : "when_no_templates"
        }
      }
    },
    "/" : {
      "get" : {
        "parameters" : [ {
          "name" : "op",
          "in" : "query",
          "schema" : {
            "type" : "string"
          }
        }, {
          "name" : "a",
          "in" : "query",
          "schema" : {
            "type" : "string"
          }
```

```
        }, {
          "name" : "b",
          "in" : "query",
          "schema" : {
            "type" : "string"
          }
        } ],
        "responses" : {
          "200" : {
            "description" : "200 response",
            "content" : {
              "application/json" : {
                "schema" : {
                  "$ref" : "#/components/schemas/Result"
                }
              }
            }
          }
        },
        "x-amazon-apigateway-integration" : {
          "type" : "aws",
          "httpMethod" : "POST",
          "uri" : "arn:aws:apigateway:us-west-2:lambda:path/2015-03-31/functions/
arn:aws:lambda:us-west-2:111122223333:function:Calc/invocations",
          "responses" : {
            "default" : {
              "statusCode" : "200",
              "responseTemplates" : {
                "application/json" : "#set($inputRoot = $input.path('$'))\n{\n
 \"input\" : {\n    \"a\" : $inputRoot.a,\n    \"b\" : $inputRoot.b,\n    \"op\" :
 \"$inputRoot.op\"\n  },\n  \"output\" : {\n    \"c\" : $inputRoot.c\n  }\n}"
              }
            }
          },
          "requestTemplates" : {
            "application/json" : "#set($inputRoot = $input.path('$'))\n{\n
  \"a\" : $input.params('a'),\n  \"b\" : $input.params('b'),\n  \"op\" :
 \"$input.params('op')\"\n}"
          },
          "passthroughBehavior" : "when_no_templates"
        }
      },
      "post" : {
        "requestBody" : {
```

```
          "content" : {
            "application/json" : {
              "schema" : {
                "$ref" : "#/components/schemas/Input"
              }
            }
          },
          "required" : true
        },
        "responses" : {
          "200" : {
            "description" : "200 response",
            "content" : {
              "application/json" : {
                "schema" : {
                  "$ref" : "#/components/schemas/Result"
                }
              }
            }
          }
        },
        "x-amazon-apigateway-integration" : {
          "type" : "aws",
          "httpMethod" : "POST",
          "uri" : "arn:aws:apigateway:us-west-2:lambda:path/2015-03-31/functions/
arn:aws:lambda:us-west-2:111122223333:function:Calc/invocations",
          "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}"
              }
            }
          },
          "passthroughBehavior" : "when_no_match"
        }
      }
    },
    "/{a}" : {
      "x-amazon-apigateway-any-method" : {
        "parameters" : [ {
          "name" : "a",
```

```
          "in" : "path",
          "required" : true,
          "schema" : {
            "type" : "string"
          }
        } ],
        "responses" : {
          "404" : {
            "description" : "404 response",
            "content" : { }
          }
        },
        "x-amazon-apigateway-integration" : {
          "type" : "mock",
          "responses" : {
            "default" : {
              "statusCode" : "404",
              "responseTemplates" : {
                "application/json" : "{ \"Message\" : \"Can't $context.httpMethod
$context.resourcePath\" }"
              }
            }
          },
          "requestTemplates" : {
            "application/json" : "{\"statusCode\": 200}"
          },
          "passthroughBehavior" : "when_no_match"
        }
      }
    }
  },
  "components" : {
    "schemas" : {
      "Input" : {
        "title" : "Input",
        "type" : "object",
        "properties" : {
          "a" : {
            "type" : "number"
          },
          "b" : {
            "type" : "number"
          },
          "op" : {
```

```
            "type" : "string"
          }
        }
      },
      "Output" : {
        "title" : "Output",
        "type" : "object",
        "properties" : {
          "c" : {
            "type" : "number"
          }
        }
      },
      "Result" : {
        "title" : "Result",
        "type" : "object",
        "properties" : {
          "input" : {
            "$ref" : "#/components/schemas/Input"
          },
          "output" : {
            "$ref" : "#/components/schemas/Output"
          }
        }
      }
    }
  }
}
```

## 在 API Gateway 中產生 API 的 Java SDK

下列程序顯示如何在 API Gateway 中產生 API 的 Java SDK。

在 API Gateway 中產生 API 的 Java SDK

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 選擇 REST API。
3. 選擇 Stages (階段)。
4. 在階段窗格中，選取階段的名稱。
5. 開啟階段動作選單，然後選擇產生 SDK。

6. 針對平台，選擇 Java 平台，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，輸入 **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，輸入 **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)。

7. 選擇 Generate SDK (產生軟體開發套件)，然後遵循畫面上的說明下載 API Gateway 所產生的軟體開發套件。

遵循 使用由 API Gateway 產生的 Java 軟體開發套件來執行 REST API 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。

## 在 API Gateway 中產生 API 的 Android SDK

下列程序顯示如何在 API Gateway 中產生 API 的 Android SDK。

在 API Gateway 中產生 API 的 Android SDK

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 選擇 REST API。
3. 選擇 Stages (階段)。

4. 在階段窗格中，選取階段的名稱。
5. 開啟階段動作選單，然後選擇產生 SDK。
6. 針對平台，選擇 Android 平台，然後執行下列操作：

   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`**)。
7. 選擇 Generate SDK (產生軟體開發套件)，然後遵循畫面上的說明下載 API Gateway 所產生的軟體開發套件。

遵循 使用由 API Gateway 為 REST API 產生的 Android 軟體開發套件 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。

## 在 API Gateway 中產生 API 的 iOS SDK

下列程序顯示如何在 API Gateway 中產生 API 的 iOS SDK。

在 API Gateway 中產生 API 的 iOS SDK

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 選擇 REST API。
3. 選擇 Stages (階段)。
4. 在階段窗格中，選取階段的名稱。
5. 開啟階段動作選單，然後選擇產生 SDK。
6. 針對平台，選擇 iOS (Objective-C) 或 iOS (Swift) 平台，然後執行下列操作：

   - 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 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 軟體開發套件。

7. 選擇 Generate SDK (產生軟體開發套件)，然後遵循畫面上的說明下載 API Gateway 所產生的軟體開發套件。

遵循 在 Objective-C 或 Swift 中使用 API Gateway 為 REST API 產生的 iOS 軟體開發套件 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。

## 在 API Gateway 中產生 REST API 的 JavaScript SDK

下列程序顯示如何在 API Gateway 中產生 API 的 JaveScript SDK。

在 API Gateway 中產生 API 的 JavaScript SDK

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 選擇 REST API。
3. 選擇 Stages (階段)。
4. 在階段窗格中，選取階段的名稱。
5. 開啟階段動作選單，然後選擇產生 SDK。
6. 針對平台，選擇 JavaScript 平台。
7. 選擇 Generate SDK (產生軟體開發套件)，然後遵循畫面上的說明下載 API Gateway 所產生的軟體開發套件。

遵循 使用 API Gateway 為 REST API 所產生的 JavaScript 軟體開發套件 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。

## 在 API Gateway 中產生 API 的 Ruby SDK

下列程序顯示如何在 API Gateway 中產生 API 的 Ruby SDK。

在 API Gateway 中產生 API 的 Ruby SDK

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 選擇 REST API。
3. 選擇 Stages (階段)。
4. 在階段窗格中，選取階段的名稱。
5. 開啟階段動作選單，然後選擇產生 SDK。
6. 針對平台，選擇 Ruby 平台，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，**`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`。
7. 選擇 Generate SDK (產生軟體開發套件)，然後遵循畫面上的說明下載 API Gateway 所產生的軟體開發套件。

遵循 使用由 API Gateway 為 REST API 產生的 Ruby 軟體開發套件 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 API Gateway 中的 AWS CLI 命令產生 API SDKs

您可以使用 呼叫 get-sdk 命令 AWS CLI ，為支援的平台產生和下載 API 的 SDK。以下將示範其中一些支援的平台。

主題

- 使用 產生和下載適用於 Android 的 Java 開發套件 AWS CLI
- 使用 產生和下載 JavaScript 開發套件 AWS CLI
- 使用 產生和下載 Ruby 開發套件 AWS CLI

使用 產生和下載適用於 Android 的 Java 開發套件 AWS CLI

若要在指定階段 (udpuvvzbkc) 產生及下載 API Gateway 所產生之 API (test) 的適用於 Android 的 Java 軟體開發套件，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 產生和下載 JavaScript 開發套件 AWS CLI

若要在指定階段 (udpuvvzbkc) 產生及下載 API Gateway 所產生之 API (test) 的 JavaScript 軟體開發套件，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 產生和下載 Ruby 開發套件 AWS CLI

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=0.01 \
```

```
            ~/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，請參閱在 API Gateway 中調用 REST API。

## 透過 銷售您的 API Gateway APIs AWS Marketplace

在您建置、測試和部署 APIs之後，您可以將它們封裝在 API Gateway 用量計劃中，並透過 軟體即服務 (SaaS) 產品銷售計劃 AWS Marketplace。訂閱您產品方案的 API 買方會根據對用量計劃提出的請求 AWS Marketplace 數量計費。

若要在 上銷售 APIs AWS Marketplace，您必須設定要 AWS Marketplace 與 API Gateway 整合的銷售管道。一般而言，這涉及列出您的產品 AWS Marketplace、設定具有適當政策的 IAM 角色，以允許 API Gateway 傳送用量指標給 AWS Marketplace、將 AWS Marketplace 產品與 API Gateway 用量計劃建立關聯，以及將 AWS Marketplace 買方與 API Gateway API 金鑰建立關聯。下列各節將會詳細討論。

如需將 API 做為 SaaS 產品銷售的詳細資訊 AWS Marketplace，請參閱AWS Marketplace 《使用者指南》。

主題

- 初始化與 API Gateway 的 AWS Marketplace 整合
- 處理客戶的用量計劃訂閱

### 初始化與 API Gateway 的 AWS Marketplace 整合

以下是與 API Gateway AWS Marketplace 整合的一次性初始化任務，可讓您將 APIs 銷售為 SaaS 產品。

在 上列出產品 AWS Marketplace

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 `requests` 類型的維度 `apigateway`。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。此角色允許 API Gateway AWS Marketplace 代表您將用量指標傳送至 。

計量角色的執行政策

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Action": [
        "aws-marketplace:BatchMeterUsage",
        "aws-marketplace:ResolveCustomer"
      ],
      "Resource": "*",
      "Effect": "Allow"
    }
  ]
}
```

計量角色的信任關係政策

JSON

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

### 將用量計劃與 AWS Marketplace 產品建立關聯

當您列出產品時 AWS Marketplace，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 產品代碼建立關聯。您可以使用 API Gateway 主控台、API Gateway REST API、 AWS CLI 適用於 API Gateway `UsagePlan`的 或適用於 API Gateway 的 AWS SDK，將 API Gateway `productCode`的 欄位設定為您的 AWS Marketplace 產品程式碼來啟用關聯。下列程式碼範例會使用 API Gateway REST API：

```
PATCH /usageplans/USAGE_PLAN_ID
Host: apigateway.region.amazonaws.com
Authorization: ...

{
    "patchOperations" : [{
        "path" : "/productCode",
        "value" : "MARKETPLACE_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 訂閱您的產品時 AWS Marketplace， AWS Marketplace 接著請求您在列出產品時所註冊的 POST SaaS 訂閱 URL AWS Marketplace。POST 請求會隨附於 `x-amzn-marketplace-token` 參數，其中包含買方資訊。請遵循 SaaS 客戶上線中的指示，在開發人員入口網站應用程式中處理此重新導向。

回應客戶的訂閱請求， AWS Marketplace 傳送`subscribe-success`通知到您可以訂閱的 Amazon SNS 主題。(請參閱 SaaS 客戶上線)。若要接受客戶訂閱請求，您可以為客戶建立或擷取 API Gateway API 金鑰、將客戶 AWS Marketplace佈建的 `customerId` 與 API 金鑰建立關聯，然後將 API 金鑰新增至您的用量計劃，以處理`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。

當客戶取消用量計劃的訂閱時， 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。

下列範例顯示如何呼叫 API Gateway REST API，以建立具有特定 AWS Marketplace `customerId`值 (*MARKETPLACE_CUSTOMER_ID*) 的新 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" : "MARKETPLACE_CUSTOMER_ID"
}
```

下列範例示範如何取得具有特定 AWS Marketplace `customerId`值 (*MARKETPLACE_CUSTOMER_ID*) 的 API 金鑰。

```
GET apikeys?customerId=MARKETPLACE_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 ApiKeycustomerId的欄位更新為客戶的客戶 AWS Marketplace 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 中保護您的 REST API

API Gateway 提供多種方法來保護 API 免於遭受特定威脅，例如惡意使用者或流量高峰。您可以使用產生 SSL 憑證、設定 web 應用程式防火牆、設定調節目標，並僅允許從 Virtual Private Cloud (VPC) 存取您的 API 等策略，以保護您的 API。您可以在本節中了解如何使用 API Gateway 啟用這些功能。

主題

- 需要 API Gateway 中具有交互 TLS 身分驗證的 API 用戶端憑證
- 在 API Gateway 中使用交互 TLS 將用戶端憑證呈現給後端服務
- 使用 AWS WAF 來保護 API Gateway APIs 中的 REST API
- 調節對 REST API 的請求，以提高 API Gateway 的輸送量
- API Gateway 中的私有 REST API

## 需要 API Gateway 中具有交互 TLS 身分驗證的 API 用戶端憑證

交互 TLS 驗證需要用戶端與伺服器之間的雙向驗證。使用交互 TLS 時，用戶端必須出示 X.509 憑證，以驗證其身分才能存取您的 API。交互 TLS 是物聯網 (IoT) 和企業對企業應用程式的常見需求。

交互 TLS 可與 API Gateway 支援的其他授權和身分驗證作業一起使用。API Gateway 會將用戶端提供的憑證轉送給 Lambda 授權方和後端整合系統。

> **Important**
>
> 預設情況下，用戶端可以使用 API Gateway 為 API 產生的 `execute-api` 端點來叫用 API。若要確保用戶端只能透過使用具有交互 TLS 的自訂網域名稱來存取您的 API，請停用預設 `execute-api` 端點。如需進一步了解，請參閱the section called “停用預設端點”。

主題


- 交互 TLS 的先決條件
- 設定自訂網域名稱的交互 TLS
- 使用需要交互 TLS 的自訂網域名稱叫用 API
- 更新您的信任庫
- 停用交互 TLS
- 針對 REST API 的交互 TLS 進行疑難排解


## 交互 TLS 的先決條件

若要設定交互 TLS，您需要：

- 區域性自訂網域名稱
- 在 中 AWS Certificate Manager 為您的自訂網域名稱設定至少一個憑證
- 已設定並上傳至 Amazon S3 的信任庫

### 自訂網域名稱

若要為 REST API 啟用交互 TLS，就必須設定 API 的自訂網域名稱。您可以為自訂網域名稱啟用交互 TLS，然後將自訂網域名稱提供給用戶端。若要使用已啟用交互 TLS 的自訂網域名稱來存取 API，用戶端必須提供您在 API 要求中信任的憑證。您可以在 the section called “自訂網域名稱” 中尋找詳細資訊。

#### 使用 AWS Certificate Manager 發行的憑證

您可以直接從 ACM 請求公開信任的憑證，或匯入公開或自行簽署的憑證。若要在 ACM 中設定憑證，請前往 ACM。如果想要匯入憑證，請繼續閱讀下一節。

使用匯入的 或 AWS 私有憑證授權單位 憑證

若要使用匯入 ACM 的憑證或從 的憑證 AWS 私有憑證授權單位 搭配交互 TLS，API Gateway 需要 ACM `ownershipVerificationCertificate`發行的 。此擁有權憑證僅用於確認您具有使用網域名稱的許可，不會用於 TLS 信號交換。如果您尚未擁有 `ownershipVerificationCertificate`，請前往 https://console.aws.amazon.com/acm/ 以設定一個。

您需要讓此憑證在網域名稱的有效生命週期內保持有效狀態。如果憑證過期且自動續約失敗，則網域名稱的所有更新都會遭到鎖定。您需要使用有效的 `ownershipVerificationCertificate` 更新 `ownershipVerificationCertificateArn`，才能進行任何其他變更。所以，`ownershipVerificationCertificate` 不可作為 API Gateway 中另一個交互 TLS 網域的伺服器憑證。如果直接將憑證重新匯入至 ACM，發行者必須保持不變。

設定您的信任庫

信任庫是副檔名為 `.pem` 的文字檔案。它們是來自憑證授權機構的受信任憑證清單。若要使用交互 TLS，請建立信任的 X.509 憑證信任存放區，以存取您的 API。

您必須在信任庫中包含從發行的憑證授權機構憑證到根憑證授權機構憑證完整的信任鏈。API Gateway 接受信任鏈中存在的任何憑證授權機構所發行的用戶端憑證。憑證可以來自公開或私有憑證授權單位。憑證的鏈接長度上限為四。您也可以提供自我簽署憑證。信任庫支援下列演算法：

- SHA-256 或更強的憑證
- RSA-2048 或更強的憑證
- ECDSA-256 或 ECDSA-384

API Gateway 會驗證許多憑證屬性。當用戶端叫用 API 時，您可以透過 Lambda 授權方執行其他檢查，包括檢查憑證是否已遭撤銷。如此一來，API Gateway 就會驗證下列屬性：

| 驗證 | 描述 |
| --- | --- |
| X.509 語法 | 憑證必須符合 X.509 語法需求。 |
| 完整性 | 憑證的內容不得從信任庫的憑證授權單位所簽署的內容進行變更。 |
| Validity | 憑證的有效期必須是最新的。 |

| 驗證 | 描述 |
| --- | --- |
| 名稱鏈接/金鑰鏈接 | 憑證的名稱和主體必須形成一個完整的鏈接。憑證的鏈接長度上限為四。 |

請將信任庫以單一檔案的形式上傳到 Amazon S3 儲存貯體中

以下是可能的 .pem 檔案範例。

Example certificates.pem

```
-----BEGIN CERTIFICATE-----
<Certificate contents>
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
<Certificate contents>
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
<Certificate contents>
-----END CERTIFICATE-----
...
```

下列 cp AWS CLI 命令`certificates.pem`會上傳至您的 Amazon S3 儲存貯體：

```
aws s3 cp certificates.pem s3://bucket-name
```

## 設定自訂網域名稱的交互 TLS

若要設定 REST API 的交互 TLS，您必須為您的 API 使用具有 TLS_1_2 安全政策的區域性自訂網域名稱。如需關於選擇安全政策的詳細資訊，請參閱 the section called “選擇安全政策”。

> **Note**
>
> 私有 API 不支援交互 TLS。

將信任庫上傳到 Amazon S3 後，您可以將自訂網域名稱設定為使用交互 TLS。以下 create-domain-name 命令會建立具有交互 TLS 的自訂網域名稱：

```
aws apigateway create-domain-name --region us-east-2 \
    --domain-name api.example.com \
    --regional-certificate-arn arn:aws:acm:us-
east-2:123456789012:certificate/123456789012-1234-1234-1234-12345678 \
    --endpoint-configuration types=REGIONAL \
    --security-policy TLS_1_2 \
    --mutual-tls-authentication truststoreUri=s3://bucket-name/key-name
```

建立網域名稱之後，您必須設定 API 作業的 DNS 記錄和基本路徑對應。如需進一步了解，請參閱[在 API Gateway 中設定區域性自訂網域名稱](#)。

## 使用需要交互 TLS 的自訂網域名稱叫用 API

若要調用啟用交互 TLS 的 API，用戶端必須在 API 要求中提供受信任的憑證。當用戶端嘗試叫用您的 API 時，API Gateway 會在您的信任庫中尋找用戶端憑證的發行者。若要讓 API Gateway 繼續執行要求，憑證發行者和完整信任鏈 (一路深入到根憑證授權機構憑證) 必須位於您的信任庫中。

下列範例 `curl` 命令會將要求傳送至 `api.example.com`，要求 `my-cert.pem` 中包含的要求。`my-key.key` 是憑證的私密金鑰。

```
curl -v --key ./my-key.key --cert ./my-cert.pem api.example.com
```

只有在您的信任庫信任憑證時，才會調用您的 API。下列情況會導致 API Gateway 的 TLS 信號交換失敗，並拒絕狀態碼為 `403` 的請求。如果您的憑證：

- 不受信任
- 已過期
- 未使用支援的演算法

> **Note**
>
> API Gateway 並不會驗證憑證是否已遭撤銷。

## 更新您的信任庫

若要更新信任庫中的憑證，請將新的憑證套件上傳至 Amazon S3。之後，您可以更新自訂網域名稱，以使用更新的憑證。

您可以使用 Amazon S3 版本控制來維護多個信任庫版本。在更新自訂網域名稱以使用新的信任庫版本時，如果憑證無效，則 API Gateway 會傳回警告。

API Gateway 只會在您更新網域名稱時產生憑證警告。如果先前上傳的憑證過期，API Gateway 並不會通知您。

以下 update-domain-name 命令會更新自訂網域名稱，以使用新的信任庫版本：

```
aws apigateway update-domain-name \
    --domain-name api.example.com \
    --patch-operations op='replace',path='/mutualTlsAuthentication/
truststoreVersion',value='abcdef123'
```

## 停用交互 TLS

以下 update-domain-name 會停用交互 TLS：

```
aws apigateway update-domain-name \
    --domain-name api.example.com \
    --patch-operations op='replace',path='/mutualTlsAuthentication/
truststoreUri',value=''
```

## 針對 REST API 的交互 TLS 進行疑難排解

以下針對當您開啟交互 TLS 時可能遇到錯誤和問題，所提供的疑難排解建言。

故障診斷憑證警告

建立帶有交互 TLS 的自訂網域名稱時，如果信任庫中的憑證無效，則 API Gateway 會傳回警告。在更新自訂網域名稱以使用新的信任庫時，也會發生這種情況。警告指出憑證和產生警告之憑證主體的問題。您的 API 仍然啟用交互 TLS，但有些用戶端可能無法存取您的 API。

若要識別產生警告的憑證，請解碼信任庫中的憑證。您可以使用諸如 `openssl` 解碼憑證及識別其主體之類的工具。

下列命令會顯示憑證的內容，包括其主體：

```
openssl x509 -in certificate.crt -text -noout
```

請更新或移除產生警告的憑證，然後將新的信任庫上傳至 Amazon S3。上傳新的信任庫之後，請更新您的自訂網域名稱以使用新的信任庫。

疑難排解網域名稱衝突

錯誤 "The certificate subject <certSubject> conflicts with an existing certificate from a different issuer." 表示多個憑證授權單位已發行此網域的憑證。對於憑證中的每個主體，交互 TLS 網域的 API Gateway 中只能有一個發行者。您需要透過單一發行者取得該主體的所有憑證。如果問題是您無法控制的憑證，但您可以證明您擁有網域名稱，請聯絡 支援 以開啟票證。

疑難排解網域名稱狀態訊息

PENDING_CERTIFICATE_REIMPORT：這表示您已將憑證重新匯入至 ACM，而且由於新憑證具有 SAN (主體別名)，而且這個 SAN 不屬於 ownershipVerificationCertificate 範圍或憑證中的主體或 SAN 不包含網域名稱，導致憑證驗證失敗。某些項目可能設定不正確或匯入了無效的憑證。您需要將有效憑證重新匯入 ACM。如需有關驗證的詳細資訊，請參閱驗證網域擁有權。

PENDING_OWNERSHIP_VERIFICATION：這表示您先前驗證的憑證已過期，ACM 無法對其進行自動續約。您將需要為憑證續約或申請新憑證。關於憑證續約的詳細資訊，請參閱 ACM 的疑難排解受管憑證續約指南。

針對不正確傳回的憑證進行疑難排解

將專用憑證從完整網域名稱 (FQDN) 遷移至萬用字元客戶網域名稱時，API Gateway 可能會傳回 FQDN 的憑證，而不是萬用字元網域名稱。

下列命令會顯示 API Gateway 正在傳回的憑證：

```
openssl s_client -connect hostname:port
```

如果產生的憑證適用於 FQDN，請聯絡 支援 以開通票證。

## 在 API Gateway 中使用交互 TLS 將用戶端憑證呈現給後端服務

API Gateway 可以在 TLS 交握期間向您的後端呈現用戶端憑證。您的後端接著可以驗證請求是否來自 API Gateway。這會在 API Gateway 和後端服務之間建立相互 TLS 連線。

您可以選擇兩種方法：

- AWS Certificate Manager (ACM) 憑證 – 從 ACM 匯入您自己的 CA 簽署憑證，並將其連接至您的 API 階段。透過此方法，您可以取得完整的憑證鏈、自動續約，以及與現有 PKI 基礎設施的整合。

- API Gateway 產生的憑證 – 在 API Gateway 中產生自我簽署的憑證。此方法不需要外部憑證基礎設施。憑證會在 365 天後過期，且必須手動輪換。

主題


- 後端用戶端憑證身分驗證的運作方式
- 在 API Gateway 中使用您自己的 ACM 憑證進行後端交互 TLS
- 在 API Gateway 中使用 API Gateway 產生的憑證進行後端身分驗證
- 在 API Gateway 中，API Gateway 支援的 HTTP 與 HTTP 代理整合憑證授權單位


## 後端用戶端憑證身分驗證的運作方式

透過傳出交互 TLS (mTLS)，API Gateway 充當 TLS 用戶端。它會在連線時將用戶端憑證呈現給您的後端伺服器。您的後端會檢查此憑證，以確認請求來自 API Gateway。

傳出 mTLS 與傳入 mTLS 不同。傳入 mTLS 需要 API 用戶端將憑證呈現給 API Gateway。傳出 mTLS 會保護 API Gateway 和後端之間的連結。

TLS 交握序列

下列步驟顯示 API Gateway 如何設定與後端的交互 TLS 連線：

1. API Gateway 會將 ClientHello 訊息傳送至後端。
2. 後端會將其伺服器憑證傳送至 API Gateway。
3. API Gateway 會檢查伺服器憑證。
4. 後端會將 CertificateRequest 傳送至 API Gateway。
5. API Gateway 會將用戶端憑證鏈傳送至後端。
6. 後端會對照其信任存放區檢查用戶端憑證。
7. 交互 TLS 連線已就緒。API Gateway 轉送 API 請求。

如果後端未在步驟 4 中請求用戶端憑證，API Gateway 會完成 TLS 交握，而不傳送。連線使用標準單向 TLS。

比較憑證選項

下表比較設定後端用戶端憑證的兩種方法。

與 API Gateway 產生的憑證相比，ACM 受管

| 功能 | ACM 受管憑證 | API Gateway 產生的憑證 |
| --- | --- | --- |
| 憑證來源 | 您的 PKI 或 AWS 私有憑證授權單位 | API Gateway 自我簽署 |
| 組態 | 依 ACM ARN 參考階段上的憑證；使用 ACM APIs 管理憑證生命週期。 | 在用戶端憑證頁面上或使用 API Gateway 用戶端憑證 APIs管理 。 |
| 續約 | 取決於憑證的發行方式。對於 ACM 發行的憑證 AWS 私有憑證授權單位，ACM 會自動續約它們，API Gateway 會傳播更新，無需重新部署，也不會停機。對於匯入 ACM 的憑證，您必須在過期前重新匯入它們；API Gateway 接著會自動傳播重新匯入。 | 手動。憑證會在 365 天後過期，且必須手動輪換。 |
| 鏈路支援 | 完整鏈結 （最多 5 個憑證） | 單一自我簽署憑證 |
| 後端信任模型 | 後端信任您的 CA | 後端必須鎖定 API Gateway 憑證 |

## 在 API Gateway 中使用您自己的 ACM 憑證進行後端交互 TLS

您可以設定 API Gateway，將您自己的 CA 簽署憑證呈現給後端服務。將您的憑證匯入 AWS Certificate Manager (ACM)，或透過 發行 AWS 私有憑證授權單位。然後，將 ACM 憑證 ARN 連結至您的 API 階段。

先決條件

設定 ACM 用戶端憑證之前，您必須擁有下列項目：

- AWS Certificate Manager 在與您 API 相同的區域中存取 。
- IAM 許可： `acm:ImportCertificate`和 `acm:DescribeCertificate` ( 適用於選項 A 匯入），或 `acm:RequestCertificate` ( 適用於選項 B)`acm-pca:IssueCertificate`，或 `acm-pca:GetCertificate`、 和 `acm:ImportCertificate` ( 適用於選項 C)。
- 部署至階段的 REST API。

步驟 1：匯入憑證，或透過 發行憑證 AWS 私有憑證授權單位

您可以從現有的 PKI 匯入憑證，或透過 發行新的憑證 AWS 私有憑證授權單位。這兩個路徑都會產生您在下列步驟中使用的 ACM 憑證 ARN。

> **Note**
>
> 後端用戶端身分驗證不支援 ACM 公有憑證。自 2025 年 6 月 11 日起， AWS Certificate Manager 不再發行具有`clientAuth`擴充金鑰用量 (EKU) 的公有憑證。此功能需要 EKU，因此請使用您匯入 ACM 的憑證，或透過 發行的憑證 AWS 私有憑證授權單位。

> **Note**
>
> 在與將使用它的 REST API 相同的 AWS 區域中建立 ACM 憑證。ACM 憑證是區域資源，因此憑證必須存在於 API 的區域中。

選項 A：從現有的 PKI 匯入

若要將用戶端憑證及其私有金鑰匯入 ACM，請執行下列命令。如需詳細資訊，請參閱 AWS Certificate Manager 使用者指南中的匯入憑證。

```
aws acm import-certificate \
  --certificate fileb://client-cert.pem \
  --private-key fileb://private-key.pem \
  --certificate-chain fileb://ca-chain.pem \
  --region region
```

命令會傳回 ACM 憑證 ARN。為下列步驟記錄此值。

選項 B：透過 AWS 私有憑證授權單位 (ACM 受管） 請求憑證

若要請求 ACM 管理且可自動續約的私有憑證，請執行下列命令。如需詳細資訊，請參閱AWS Certificate Manager 《 使用者指南》中的請求私有憑證。

```
aws acm request-certificate \
  --domain-name www.example.com \
  --certificate-authority-arn arn:aws:acm-pca:us-east-1:123456789012:certificate-
authority/12345678-1234-1234-1234-123456789012 \
```

```
  --region region
```

命令會傳回 ACM 憑證 ARN。為下列步驟記錄此值。

選項 C：透過 發出 AWS 私有憑證授權單位 並匯入 ACM

如果您需要直接控制憑證參數（例如自訂延伸模組或簽署演算法），您可以透過 發行憑證， AWS 私有憑證授權單位 然後將其匯入 ACM。以這種方式匯入的憑證不會由 ACM 自動續約。確保憑證符合憑證要求。如需發行私有憑證的詳細資訊，請參閱AWS 私有憑證授權單位 《 使用者指南》中的發行私有終端實體憑證。

```
aws acm-pca issue-certificate \
  --certificate-authority-arn arn:aws:acm-pca:us-east-1:123456789012:certificate-
authority/12345678-1234-1234-1234-123456789012 \
  --csr fileb://csr.pem \
  --signing-algorithm SHA256WITHRSA \
  --validity Value=365,Type=DAYS
```

> **擷取和匯入憑證**
>
> `issue-certificate` 命令會傳回 AWS 私有憑證授權單位 憑證 ARN，而不是 ACM ARN。若要搭配 API Gateway 使用此憑證，請使用 擷取憑證，`aws acm-pca get-certificate`然後使用 將其匯入 ACM`aws acm import-certificate`。匯入會產生您在下列步驟中使用的 ACM 憑證 ARN。當您執行 時`aws acm import-certificate`，`--region`請將 設定為 API 的區域，以便在該處建立 ACM 憑證。

步驟 2：設定 API 階段以使用 ACM 憑證

在您擁有 ACM 憑證 ARN 之後，請設定您的 API 階段，將憑證呈現給您的後端。

設定階段（主控台）

1. 在 https：//https://console.aws.amazon.com/apigateway 開啟 API Gateway 主控台。
2. 選擇您的 REST API。
3. 選擇 Stages (階段)。
4. 在階段詳細資訊區段中，選擇編輯。
5. 對於用戶端憑證，請從下拉式清單中選取您的 ACM 憑證。

6. 選擇儲存變更。

設定階段 (AWS CLI)

執行以下命令：

```
aws apigateway update-stage \
  --rest-api-id abc123 \
  --stage-name prod \
  --patch-operations op='replace',path=/clientCertificateId,value=arn:aws:acm:us-
east-1:123456789012:certificate/12345678-1234-1234-1234-123456789012
```

> **Note**
>
> API Gateway 對 ACM 和 API Gateway 產生的憑證使用相同的`clientCertificateId`欄位。當您提供 ACM 憑證 ARN 時，API Gateway 會自動偵測格式並使用 ACM 受管工作流程。

步驟 3：驗證組態

若要驗證 API Gateway 是否將憑證傳送至您的後端，請完成下列步驟：

> **後端必須請求用戶端憑證**
>
> 您的後端必須設定為在 TLS 交握期間請求用戶端憑證。如果後端未請求，API Gateway 不會顯示憑證。

1. 叫用您的 API 端點。
2. 檢查您的後端在 TLS 交握期間取得用戶端憑證。
3. 檢查您的後端是否接受憑證並傳回成功的回應。

如果後端拒絕憑證，請確認憑證鏈可以針對後端的信任存放區進行驗證。

憑證需求

您設定的分葉憑證必須符合下列要求。

ACM 用戶端憑證需求

| 需求 | 說明 |
| --- | --- |
| 最大鏈結長度 | 5 個憑證 |
| Validity | 當您設定憑證時，憑證不得過期或尚未有效 |
| 區域 | ACM 憑證必須與 API 位於相同的區域 |
| 帳戶 | ACM 憑證必須與 API 位於相同的帳戶中 |
| 擴充金鑰用量 (EKU) | 如果存在， 必須包含 `clientAuth` 。如果不存在，則會接受憑證。 |
| 金鑰用量 (KU) | 如果存在， 必須包含 `digitalSignature` 或 `keyAgreement` 。如果不存在，則會接受憑證。 |
| 金鑰演算法 | 必須是下列其中一項：RSA 2048、RSA 3072、RSA 4096、ECDSA P-256 (EC_prime256v1)、ECDSA P-384 (EC_secp384r1) 或 ECDSA P-521 (EC_secp521r1) |
| ACM 憑證狀態 | 必須為 `ISSUED` |

> **Note**
>
> API Gateway 不會驗證分葉和中繼憑證之間的信任鏈。API Gateway 也不會驗證中繼憑證的憑證意圖或基本限制條件 （ 例如 `CA:TRUE`)。您的後端會在 TLS 交握期間執行這些驗證。

憑證續約和傳播

當 ACM 中的憑證變更時，API Gateway 會偵測更新並自動傳播新憑證。您不需要重新部署階段，而且 API 在輪換期間不會發生停機時間。

憑證傳播最終一致。在更新期間，您的後端可能會收到舊憑證或新憑證，直到傳播完成為止。

憑證續約的方式取決於發行方式：

- 透過 AWS 私有憑證授權單位 (ACM 受管）（選項 B) 發行的憑證 – ACM 會自動續約這些憑證。API Gateway 會自動偵測續約和更新。
- 由 發行 AWS 私有憑證授權單位 和匯入的憑證（選項 C) – ACM 不會自動續約匯入的憑證。您必須重新匯入續約的憑證。重新匯入憑證後，API Gateway 會自動偵測變更和更新。
- 從 PKI（選項 A) 匯入的憑證 – 您必須將續約的憑證重新匯入 ACM。重新匯入憑證後，API Gateway 會自動偵測變更和更新。

ACM 透過 Amazon EventBridge 傳送憑證過期通知。您可以使用這些通知在憑證過期之前設定警示。

ACM 憑證行為和限制

檢視設定的憑證

ACM 憑證不會出現在 `GetClientCertificate`或 `GetClientCertificates` API 回應中。若要檢視在階段上設定的 ACM 憑證 ARN，請使用 GetStage。若要檢視憑證詳細資訊，請使用 ACM APIs DescribeCertificate 和 GetCertificate。

跨階段重複使用

您可以將相同的 ACM 憑證連接到多個階段。每個階段都會依其 ARN 獨立參考憑證。

用戶端憑證 APIs不適用於 ACM 憑證

ACM 憑證不是 API Gateway 管理的資源。當使用 ACM `GetClientCertificate`憑證 ARN 呼叫 `NotFoundException`時`UpdateClientCertificate`，、 和 `DeleteClientCertificate` APIs 會傳回 。使用 ACM APIs來管理憑證生命週期。

自動憑證關聯清除

當您從階段移除 ACM 憑證、更新階段以使用不同的憑證，或刪除階段或 REST API 時，API Gateway 會自動清除憑證關聯。不需要手動動作。

刪除 ACM 憑證

當 API Gateway 與其具有作用中關聯時，ACM 不允許您刪除憑證。若要從 ACM 刪除憑證，請先將其從參考憑證的所有階段中移除。

## 在 API Gateway 中使用 API Gateway 產生的憑證進行後端身分驗證

您可以使用 API Gateway 產生 SSL 憑證，並在後端系統中使用其公開金鑰確認對後端系統的 HTTP 請求來自 API Gateway。這可讓 HTTP 後端系統僅接受 Amazon API Gateway 所發出的請求並加以控管，即使後端系統可公開存取也一樣。

> **Note**
>
> 有些後端伺服器可能不支援 SSL 用戶端身分驗證，因為 API Gateway 可以確實傳回 SSL 憑證錯誤。如需不相容的後端伺服器清單，請參閱「the section called “重要說明”」。

> **Tip**
>
> 若要從 ACM 使用您自己的 CA 簽署憑證，而非 API Gateway 產生的自我簽署憑證，請參閱 the section called “使用您自己的 ACM 憑證”。

API Gateway 所產生的 SSL 憑證為自我簽署憑證，且只有在 API Gateway 主控台中或透過 API 才能看見憑證的公開金鑰。

主題

- 使用 API Gateway 主控台產生用戶端憑證
- 設定 API 來使用 SSL 憑證
- 測試叫用以驗證用戶端憑證組態
- 設定後端 HTTPS 伺服器以驗證用戶端憑證
- 輪換到期的用戶端憑證

使用 API Gateway 主控台產生用戶端憑證

1. 在以下網址開啟 API Gateway 主控台：https://console.aws.amazon.com/apigateway/。
2. 選擇 REST 或 WebSocket API。
3. 在主導覽窗格中，選擇用戶端憑證。
4. 從用戶端憑證頁面，選擇產生憑證。
5. 在描述，請輸入描述。
6. 選擇產生憑證以產生憑證。API Gateway 隨即會產生新的憑證，並傳回新憑證 GUID 以及 PEM 編碼的公開金鑰。

您現在已經準備好設定 API 來使用憑證。

設定 API 來使用 SSL 憑證

以下說明假設您已完成使用 API Gateway 主控台產生用戶端憑證中的步驟。

1. 在 API Gateway 主控台中，建立或開啟要使用用戶端憑證的 REST 或 WebSocket API。請確定 API 已部署至階段。
2. 在主導覽窗格中，選擇階段。
3. 在階段詳細資訊區段中，選擇編輯。
4. 針對用戶端憑證，選取一個憑證。
5. 選擇儲存變更。

選取並儲存 API 的憑證後，API Gateway 會將該憑證用於 API 中的所有 HTTP 整合呼叫。

測試叫用以驗證用戶端憑證組態

1. 選擇 REST API 方法。選擇測試標籤。您可能需要選擇向右箭頭按鈕才能顯示測試索引標籤。
2. 針對用戶端憑證，選取一個憑證。
3. 選擇測試。

API Gateway 會顯示所選擇的 SSL 憑證，讓 HTTP 後端系統對 API 進行身分驗證。

設定後端 HTTPS 伺服器以驗證用戶端憑證

以下說明假設您已完成使用 API Gateway 主控台產生用戶端憑證中的步驟，且已下載用戶端憑證的複本。若要下載用戶端憑證，您可以呼叫 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 憑證之前，請務必取得 PEM 編碼的私密金鑰，以及由受信任之憑證授權機構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” 其中之一。

例如，假設用戶端憑證檔案是 `apig-cert.pem`，而伺服器私密金鑰和憑證檔案分別為 `server-key.pem` 和 `server-cert.pem`。對於後端的 Node.js 伺服器，您可以設定類似於以下內容的伺服器：

```
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。

使用 輪換即將到期的用戶端憑證 AWS 管理主控台

下列程序說明如何在主控台中，針對先前部署的 API 輪換用戶端憑證。

1. 在主導覽窗格中，選擇用戶端憑證。
2. 從用戶端憑證窗格，選擇產生憑證。
3. 開啟您要對其使用用戶端憑證的 API。
4. 在所選取的 API 下選擇 Stages (階段)，然後選擇階段。
5. 在階段詳細資訊區段中，選擇編輯。
6. 針對用戶端憑證，選取新的憑證。
7. 若要儲存設定，請選擇儲存變更。

使用 輪換即將到期的用戶端憑證 AWS CLI

您可以呼叫 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，以檢查憑證的過期日期。

若要輪換用戶端憑證，請執行下列作業：

1. 產生新的用戶端憑證，方法是呼叫 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`。
2. 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。
3. 更新 API 階段以使用新的用戶端憑證，方法是以新的用戶端憑證 ID (`ndiqef`) 呼叫 API Gateway REST API 的 stage:update，如下所示：

   ```
   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"
       }
     ]
   }
   ```

   您也可以使用 update-stage 命令。

   如果您使用的是 WebSocket API，請使用 `apigatewayv2` update-stage 命令。
4. 更新後端伺服器以移除舊的用戶端憑證。
5. 從 API Gateway 刪除舊憑證，方法是呼叫 API Gateway REST API 的 clientcertificate:delete，並指定舊憑證的 clientCertificateId (`a1b2c3`)，如下所示：

```
DELETE /clientcertificates/a1b2c3
```

您也可以呼叫 delete-client-certificate 命令：

```
aws apigateway delete-client-certificate --client-certificate-id a1b2c3
```

## 在 API Gateway 中，API Gateway 支援的 HTTP 與 HTTP 代理整合憑證授權單位

以下清單顯示的是 API Gateway 支援的 HTTP、HTTP proxy 及私有整合憑證授權機構。

```
Alias name: accvraiz1
  SHA1: 93:05:7A:88:15:C6:4F:CE:88:2F:FA:91:16:52:28:78:BC:53:64:17
  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:D1:04:38:41:1
Alias name: acraizfnmtrcm
  SHA1: EC:50:35:07:B2:15:C4:95:62:19:E2:A8:9A:5B:42:99:2C:4C:2C:20
  SHA256:
 EB:C5:57:0C:29:01:8C:4D:67:B1:AA:12:7B:AF:12:F7:03:B4:61:1E:BC:17:B7:DA:B5:57:38:94:17:9B:93:F
Alias name: actalis
  SHA1: F3:73:B3:87:06:5A:28:84:8A:F2:F3:4A:CE:19:2B:DD:C7:8E:9C:AC
  SHA256:
 55:92:60:84:EC:96:3A:64:B9:6E:2A:BE:01:CE:0B:A8:6A:64:FB:FE:BC:C7:AA:B5:AF:C1:55:B3:7F:D7:60:6
Alias name: actalisauthenticationrootca
  SHA1: F3:73:B3:87:06:5A:28:84:8A:F2:F3:4A:CE:19:2B:DD:C7:8E:9C:AC
  SHA256:
 55:92:60:84:EC:96:3A:64:B9:6E:2A:BE:01:CE:0B:A8:6A:64:FB:FE:BC:C7:AA:B5:AF:C1:55:B3:7F:D7:60:6
Alias name: addtrustclass1ca
  SHA1: CC:AB:0E:A0:4C:23:01:D6:69:7B:DD:37:9F:CD:12:EB:24:E3:94:9D
  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:A
Alias name: addtrustexternalca
  SHA1: 02:FA:F3:E2:91:43:54:68:60:78:57:69:4D:F5:E4:5B:68:85:18:68
  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:F
Alias name: addtrustqualifiedca
  SHA1: 4D:23:78:EC:91:95:39:B5:00:7F:75:8F:03:3B:21:1E:C5:4D:8B:CF
  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:D0:60:1
Alias name: affirmtrustcommercial
  SHA1: F9:B5:B6:32:45:5F:9C:BE:EC:57:5F:80:DC:E9:6E:2C:C7:B2:78: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:A
Alias name: affirmtrustcommercialca
  SHA1: F9:B5:B6:32:45:5F:9C:BE:EC:57:5F:80:DC:E9:6E:2C:C7:B2:78: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:A
Alias name: affirmtrustnetworking
  SHA1: 29:36:21:02:8B:20:ED:02:F5:66:C5:32:D1:D6:ED:90:9F:45:00:2F
  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:B4:1
Alias name: affirmtrustnetworkingca
  SHA1: 29:36:21:02:8B:20:ED:02:F5:66:C5:32:D1:D6:ED:90:9F:45:00:2F
  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:B4:1
Alias name: affirmtrustpremium
  SHA1: D8:A6:33:2C:E0:03:6F:B1:85:F6:63:4F:7D:6A:06:65:26:32:28:27
  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:9
Alias name: affirmtrustpremiumca
  SHA1: D8:A6:33:2C:E0:03:6F:B1:85:F6:63:4F:7D:6A:06:65:26:32:28:27
  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:9
Alias name: affirmtrustpremiumecc
  SHA1: B8:23:6B:00:2F:1D:16:86:53:01:55:6C:11:A4:37:CA:EB:FF:C3:BB
  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:84:88:82:14:2
Alias name: affirmtrustpremiumeccca
  SHA1: B8:23:6B:00:2F:1D:16:86:53:01:55:6C:11:A4:37:CA:EB:FF:C3:BB
  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:84:88:82:14:2
Alias name: amazon-ca-g4-acm1
  SHA1: F2:0D:28:B6:29:C2:2C:5E:84:05:E6:02:4D:97:FE:8F:A0:84:93:A0
  SHA256:
 B0:11:A4:F7:29:6C:74:D8:2B:F5:62:DF:87:D7:28:C7:1F:B5:8C:F4:E6:73:F2:78:FC:DA:F3:FF:83:A6:8C:8
Alias name: amazon-ca-g4-acm2
  SHA1: A7:E6:45:32:1F:7A:B7:AD:C0:70:EA:73:5F:AB:ED:C3:DA:B4:D0:C8
  SHA256:
 D7:A8:7C:69:95:D0:E2:04:2A:32:70:A7:E2:87:FE:A7:E8:F4:C1:70:62:F7:90:C3:EB:BB:53:F2:AC:39:26:B
Alias name: amazon-ca-g4-acm3
  SHA1: 7A:DB:56:57:5F:D6:EE:67:85:0A:64:BB:1C:E9:E4:B0:9A:DB:9D:07
  SHA256:
 6B:EB:9D:20:2E:C2:00:70:BD:D2:5E:D3:C0:C8:33:2C:B4:78:07:C5:82:94:4E:7E:23:28:22:71:A4:8E:0E:C
Alias name: amazon-ca-g4-legacy
  SHA1: EA:E7:DE:F9:0A:BE:9F:0B:68:CE:B7:24:0D:80:74:03:BF:6E:B1:6E
```

```
  SHA256:
 CD:72:C4:7F:B4:AD:28:A4:67:2B:E1:86:47:D4:40:E9:3B:16:2D:95:DB:3C:2F:94:BB:81:D9:09:F7:91:24:5
Alias name: amazon-root-ca-ecc-384-1
  SHA1: F9:5E:4A:AB:9C:2D:57:61:63:3D:B2:57:B4:0F:24:9E:7B:E2:23:7D
  SHA256:
 C6:BD:E5:66:C2:72:2A:0E:96:E9:C1:2C:BF:38:92:D9:55:4D:29:03:57:30:72:40:7F:4E:70:17:3B:3C:9B:6
Alias name: amazon-root-ca-rsa-2k-1
  SHA1: 8A:9A:AC:27:FC:86:D4:50:23:AD:D5:63:F9:1E:AE:2C:AF:63:08:6C
  SHA256:
 0F:8F:33:83:FB:70:02:89:49:24:E1:AA:B0:D7:FB:5A:BF:98:DF:75:8E:0F:FE:61:86:92:BC:F0:75:35:CC:8
Alias name: amazon-root-ca-rsa-4k-1
  SHA1: EC:BD:09:61:F5:7A:B6:A8:76:BB:20:8F:14:05:ED:7E:70:ED:39:45
  SHA256:
 36:AE:AD:C2:6A:60:07:90:6B:83:A3:73:2D:D1:2B:D4:00:5E:C7:F2:76:11:99:A9:D4:DA:63:2F:59:B2:8B:C
Alias name: amazon1
  SHA1: 8D:A7:F9:65:EC:5E:FC:37:91:0F:1C:6E:59:FD:C1:CC:6A:6E:DE:16
  SHA256:
 8E:CD:E6:88:4F:3D:87:B1:12:5B:A3:1A:C3:FC:B1:3D:70:16:DE:7F:57:CC:90:4F:E1:CB:97:C6:AE:98:19:6
Alias name: amazon2
  SHA1: 5A:8C:EF:45:D7:A6:98:59:76:7A:8C:8B:44:96:B5:78:CF:47:4B:1A
  SHA256:
 1B:A5:B2:AA:8C:65:40:1A:82:96:01:18:F8:0B:EC:4F:62:30:4D:83:CE:C4:71:3A:19:C3:9C:01:1E:A4:6D:B
Alias name: amazon3
  SHA1: 0D:44:DD:8C:3C:8C:1A:1A:58:75:64:81:E9:0F:2E:2A:FF:B3:D2:6E
  SHA256:
 18:CE:6C:FE:7B:F1:4E:60:B2:E3:47:B8:DF:E8:68:CB:31:D0:2E:BB:3A:DA:27:15:69:F5:03:43:B4:6D:B3:A
Alias name: amazon4
  SHA1: F6:10:84:07:D6:F8:BB:67:98:0C:C2:E2:44:C2:EB:AE:1C:EF:63:BE
  SHA256:
 E3:5D:28:41:9E:D0:20:25:CF:A6:90:38:CD:62:39:62:45:8D:A5:C6:95:FB:DE:A3:C2:2B:0B:FB:25:89:70:9
Alias name: amazonrootca1
  SHA1: 8D:A7:F9:65:EC:5E:FC:37:91:0F:1C:6E:59:FD:C1:CC:6A:6E:DE:16
  SHA256:
 8E:CD:E6:88:4F:3D:87:B1:12:5B:A3:1A:C3:FC:B1:3D:70:16:DE:7F:57:CC:90:4F:E1:CB:97:C6:AE:98:19:6
Alias name: amazonrootca2
  SHA1: 5A:8C:EF:45:D7:A6:98:59:76:7A:8C:8B:44:96:B5:78:CF:47:4B:1A
  SHA256:
 1B:A5:B2:AA:8C:65:40:1A:82:96:01:18:F8:0B:EC:4F:62:30:4D:83:CE:C4:71:3A:19:C3:9C:01:1E:A4:6D:B
Alias name: amazonrootca3
  SHA1: 0D:44:DD:8C:3C:8C:1A:1A:58:75:64:81:E9:0F:2E:2A:FF:B3:D2:6E
  SHA256:
 18:CE:6C:FE:7B:F1:4E:60:B2:E3:47:B8:DF:E8:68:CB:31:D0:2E:BB:3A:DA:27:15:69:F5:03:43:B4:6D:B3:A
Alias name: amazonrootca4
  SHA1: F6:10:84:07:D6:F8:BB:67:98:0C:C2:E2:44:C2:EB:AE:1C:EF:63:BE
```

```
  SHA256:
 E3:5D:28:41:9E:D0:20:25:CF:A6:90:38:CD:62:39:62:45:8D:A5:C6:95:FB:DE:A3:C2:2B:0B:FB:25:89:70:9
Alias name: amzninternalinfoseccag3
  SHA1: B9:B1:CA:38:F7:BF:9C:D2:D4:95:E7:B6:5E:75:32:9B:A8:78:2E:F6
  SHA256:
 81:03:0B: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:6
Alias name: amzninternalrootca
  SHA1: A7:B7:F6:15:8A:FF:1E:C8:85:13:38:BC:93:EB:A2:AB:A4:09:EF:06
  SHA256:
 0E:DE:63:C1:DC:7A:8E:11:F1:AB:BC:05:4F:59:EE:49:9D:62:9A:2F:DE:9C:A7:16:32:A2:64:29:3E:8B:66:A
Alias name: aolrootca1
  SHA1: 39:21:C1:15:C1:5D:0E:CA:5C:CB:5B:C4:F0:7D:21:D8:05:0B:56:6A
  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:E
Alias name: aolrootca2
  SHA1: 85:B5:FF:67:9B:0C:79:96:1F:C8:6E:44:22:00:46:13:DB:17:92:84
  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:B
Alias name: atostrustedroot2011
  SHA1: 2B:B1:F5:3E:55:0C:1D:C5:F1:D4:E6:B7:6A:46:4B:55:06:02:AC:21
  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:A4:7
Alias name: autoridaddecertificacionfirmaprofesionalcifa62634068
  SHA1: AE:C5:FB:3F:C8:E1:BF:C4:E5:4F:03:07:5A:9A:E8:00:B7:F7:B6:FA
  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:E
Alias name: baltimorecodesigningca
  SHA1: 30:46:D8:C8:88:FF:69:30:C3:4A:FC:CD:49:27:08:7C:60:56:7B:0D
  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:46:8
Alias name: baltimorecybertrustca
  SHA1: D4:DE:20:D0:5E:66:FC:53:FE:1A:50:88:2C:78:DB:28:52:CA:E4:74
  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:E
Alias name: baltimorecybertrustroot
  SHA1: D4:DE:20:D0:5E:66:FC:53:FE:1A:50:88:2C:78:DB:28:52:CA:E4:74
  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:E
Alias name: buypassclass2ca
  SHA1: 49:0A:75:74:DE:87:0A:47:FE:58:EE:F6:C7:6B:EB:C6:0B:12:40:99
  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:4
Alias name: buypassclass2rootca
  SHA1: 49:0A:75:74:DE:87:0A:47:FE:58:EE:F6:C7:6B:EB:C6:0B:12:40:99
```

```
  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:4
Alias name: buypassclass3ca
  SHA1: DA:FA:F7:FA:66:84:EC:06:8F:14:50:BD:C7:C2:81:A5:BC:A9:64:57
  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:B2:4
Alias name: buypassclass3rootca
  SHA1: DA:FA:F7:FA:66:84:EC:06:8F:14:50:BD:C7:C2:81:A5:BC:A9:64:57
  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:B2:4
Alias name: cadisigrootr2
  SHA1: B5:61:EB:EA:A4:DE:E4:25:4B:69:1A:98:A5:57:47:C2:34:C7:D9:71
  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:F8:07:0
Alias name: camerfirmachambersca
  SHA1: 78:6A:74:AC:76:AB:14:7F:9C:6A:30:50:BA:9E:A8:7E:FE:9A:CE:3C
  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:C
Alias name: camerfirmachamberscommerceca
  SHA1: 6E:3A:55:A4:19:0C:19:5C:93:84:3C:C0:DB:72:2E:31:30:61:F0:B1
  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:C
Alias name: camerfirmachambersignca
  SHA1: 4A:BD:EE:EC:95:0D:35:9C:89:AE:C7:52:A1:2C:5B:29:F6:D6:AA:0C
  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:C
Alias name: certigna
  SHA1: B1:2E:13:63:45:86:A4:6F:1A:B2:60:68:37:58:2D:C4:AC:FD:94:97
  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:A1:2
Alias name: certignarootca
  SHA1: 2D:0D:52:14:FF:9E:AD:99:24:01:74:20:47:6E:6C:85:27:27:F5:43
  SHA256:
 D4:8D:3D:23:EE:DB:50:A4:59:E5:51:97:60:1C:27:77:4B:9D:7B:18:C9:4D:5A:05:95:11:A1:02:50:B9:31:6
Alias name: certplusclass2primaryca
  SHA1: 74:20:74:41:72:9C:DD:92:EC:79:31:D8:23:10:8D:C2:81:92:E2:BB
  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:C
Alias name: certplusclass3pprimaryca
  SHA1: 21:6B:2A:29:E6:2A:00:CE:82:01:46:D8:24:41:41:B9:25:11:B2:79
  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:02:8
Alias name: certsignrootca
  SHA1: FA:B7:EE:36:97:26:62:FB:2D:B0:2A:F6:BF:03:FD:E8:7C:4B:2F:9B
```

```
  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:B
Alias name: certsignrootcag2
  SHA1: 26:F9:93:B4:ED:3D:28:27:B0:B9:4B:A7:E9:15:1D:A3:8D:92:E5:32
  SHA256:
 65:7C:FE:2F:A7:3F:AA:38:46:25:71:F3:32:A2:36:3A:46:FC:E7:02:09:51:71:07:02:CD:FB:B6:EE:DA:33:0
Alias name: certum2
  SHA1: D3:DD:48:3E:2B:BF:4C:05:E8:AF:10:F5:FA:76:26:CF:D3:DC:30:92
  SHA256:
 B6:76:F2:ED:DA:E8:77:5C:D3:6C:B0:F6:3C:D1:D4:60:39:61:F4:9E:62:65:BA:01:3A:2F:03:07:B6:D0:B8:0
Alias name: certumca
  SHA1: 62:52:DC:40:F7:11:43:A2:2F:DE:9E:F7:34:8E:06:42:51:B1:81:18
  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:D8:14:36:2
Alias name: certumtrustednetworkca
  SHA1: 07:E0:32:E0:20:B7:2C:3F:19:2F:06:28:A2:59:3A:19:A7:0F:06:9E
  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:C4:40:8
Alias name: certumtrustednetworkca2
  SHA1: D3:DD:48:3E:2B:BF:4C:05:E8:AF:10:F5:FA:76:26:CF:D3:DC:30:92
  SHA256:
 B6:76:F2:ED:DA:E8:77:5C:D3:6C:B0:F6:3C:D1:D4:60:39:61:F4:9E:62:65:BA:01:3A:2F:03:07:B6:D0:B8:0
Alias name: cfcaevroot
  SHA1: E2:B8:29:4B:55:84:AB:6B:58:C2:90:46:6C:AC:3F:B8:39:8F:84:83
  SHA256:
 5C:C3:D7:8E:4E:1D:5E:45:54:7A:04:E6:87:3E:64:F9:0C:F9:53:6D:1C:CC:2E:F8:00:F3:55:C4:C5:FD:70:F
Alias name: chambersofcommerceroot2008
  SHA1: 78:6A:74:AC:76:AB:14:7F:9C:6A:30:50:BA:9E:A8:7E:FE:9A:CE:3C
  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:C
Alias name: chunghwaepkirootca
  SHA1: 67:65:0D:F1:7E:8E:7E:5B:82:40:A4:F4:56:4B:CF:E2:3D:69:C6:F0
  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:D
Alias name: cia-crt-g3-01-ca
  SHA1: 2B:EE:2C:BA:A3:1D:B5:FE:60:40:41:95:08:ED:46:82:39:4D:ED:E2
  SHA256:
 20:48:AD:4C:EC:90:7F:FA:4A:15:D4:CE:45:E3:C8:E4:2C:EA:78:33:DC:C7:D3:40:48:FC:60:47:27:42:99:E
Alias name: cia-crt-g3-02-ca
  SHA1: 96:4A:BB:A7:BD:DA:FC:97:34:C0:0A:2D:F0:05:98:F7:E6:C6:6F:09
  SHA256:
 93:F1:72:FB:BA:43:31:5C:06:EE:0F:9F:04:89:B8:F6:88:BC:75:15:3C:BE:B4:80:AC:A7:14:3A:F6:FC:4A:C
Alias name: comodo-ca
  SHA1: AF:E5:D2:44:A8:D1:19:42:30:FF:47:9F:E2:F8:97:BB:CD:7A:8C:B4
```

```
  SHA256:
 52:F0:E1:C4:E5:8E:C6:29:29:1B:60:31:7F:07:46:71:B8:5D:7E:A8:0D:5B:07:27:34:63:53:4B:32:B4:02:3
Alias name: comodoaaaca
  SHA1: D1:EB:23:A4:6D:17:D6:8F:D9:25:64:C2:F1:F1:60:17:64:D8:E3:49
  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:F
Alias name: comodoaaaservicesroot
  SHA1: D1:EB:23:A4:6D:17:D6:8F:D9:25:64:C2:F1:F1:60:17:64:D8:E3:49
  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:F
Alias name: comodocertificationauthority
  SHA1: 66:31:BF:9E:F7:4F:9E:B6:C9:D5:A6:0C:BA:6A:BE:D1:F7:BD:EF:7B
  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:6
Alias name: comodoecccertificationauthority
  SHA1: 9F:74:4E:9F:2B:4D:BA:EC:0F:31:2C:50:B6:56:3B:8E:2D:93:C3:11
  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:C
Alias name: comodorsacertificationauthority
  SHA1: AF:E5:D2:44:A8:D1:19:42:30:FF:47:9F:E2:F8:97:BB:CD:7A:8C:B4
  SHA256:
 52:F0:E1:C4:E5:8E:C6:29:29:1B:60:31:7F:07:46:71:B8:5D:7E:A8:0D:5B:07:27:34:63:53:4B:32:B4:02:3
Alias name: cybertrustglobalroot
  SHA1: 5F:43:E5:B1:BF:F8:78:8C:AC:1C:C7:CA:4A:9A:C6:22:2B:CC:34:C6
  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:A
Alias name: deprecateditsecca
  SHA1: 12:12:0B:03:0E:15:14:54:F4:DD:B3:F5:DE:13:6E:83:5A:29:72:9D
  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:C
Alias name: deutschetelekomrootca2
  SHA1: 85:A4:08:C0:9C:19:3E:5D:51:58:7D:CD:D6:13:30:FD:8C:DE:37:BF
  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:D
Alias name: digicertassuredidrootca
  SHA1: 05:63:B8:63:0D:62:D7:5A:BB:C8:AB:1E:4B:DF:B5:A8:99:B2:4D:43
  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:C6:32:5
Alias name: digicertassuredidrootg2
  SHA1: A1:4B:48:D9:43:EE:0A:0E:40:90:4F:3C:E0:A4:C0:91:93:51:5D:3F
  SHA256:
 7D:05:EB:B6:82:33:9F:8C:94:51:EE:09:4E:EB:FE:FA:79:53:A1:14:ED:B2:F4:49:49:45:2F:AB:7D:2F:C1:8
Alias name: digicertassuredidrootg3
  SHA1: F5:17:A2:4F:9A:48:C6:C9:F8:A2:00:26:9F:DC:0F:48:2C:AB:30:89
```

```
  SHA256:
 7E:37:CB:8B:4C:47:09:0C:AB:36:55:1B:A6:F4:5D:B8:40:68:0F:BA:16:6A:95:2D:B1:00:71:7F:43:05:3F:C
Alias name: digicertglobalrootca
  SHA1: A8:98:5D:3A:65:E5:E5:C4:B2:D7:D6:6D:40:C6:DD:2F:B1:9C:54:36
  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:C7:01:6
Alias name: digicertglobalrootg2
  SHA1: DF:3C:24:F9:BF:D6:66:76:1B:26:80:73:FE:06:D1:CC:8D:4F:82:A4
  SHA256:
 CB:3C:CB:B7:60:31:E5:E0:13:8F:8D:D3:9A:23:F9:DE:47:FF:C3:5E:43:C1:14:4C:EA:27:D4:6A:5A:B1:CB:5
Alias name: digicertglobalrootg3
  SHA1: 7E:04:DE:89:6A:3E:66:6D:00:E6:87:D3:3F:FA:D9:3B:E8:3D:34:9E
  SHA256:
 31:AD:66:48:F8:10:41:38:C7:38:F3:9E:A4:32:01:33:39:3E:3A:18:CC:02:29:6E:F9:7C:2A:C9:EF:67:31:D
Alias name: digicerthighassuranceevrootca
  SHA1: 5F:B7:EE:06:33:E2:59:DB:AD:0C:4C:9A:E6:D3:8F:1A:61:C7:DC:25
  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:C
Alias name: digicerttrustedrootg4
  SHA1: DD:FB:16:CD:49:31:C9:73:A2:03:7D:3F:C8:3A:4D:7D:77:5D:05:E4
  SHA256:
 55:2F:7B:DC:F1:A7:AF:9E:6C:E6:72:01:7F:4F:12:AB:F7:72:40:C7:8E:76:1A:C2:03:D1:D9:D2:0A:C8:99:8
Alias name: dstrootcax3
  SHA1: DA:C9:02:4F:54:D8:F6:DF:94:93:5F:B1:73:26:38:CA:6A:D7:7C:13
  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:D6:77:07:3
Alias name: dtrustrootclass3ca22009
  SHA1: 58:E8:AB:B0:36:15:33:FB:80:F7:9B:1B:6D:29:D3:FF:8D:5F:00:F0
  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:C
Alias name: dtrustrootclass3ca2ev2009
  SHA1: 96:C9:1B:0B:95:B4:10:98:42:FA:D0:D8:22:79:FE:60:FA:B9:16:83
  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:D3:48:8
Alias name: ecacc
  SHA1: 28:90:3A:63:5B:52:80:FA:E6:77:4C:0B:6D:A7:D6:BA:A6:4A:F2:E8
  SHA256:
 88:49:7F:01:60:2F:31:54:24:6A:E2:8C:4D:5A:EF:10:F1:D8:7E:BB:76:62:6F:4A:E0:B7:F9:5B:A7:96:87:9
Alias name: emsigneccrootcac3
  SHA1: B6:AF:43:C2:9B:81:53:7D:F6:EF:6B:C3:1F:1F:60:15:0C:EE:48:66
  SHA256:
 BC:4D:80:9B:15:18:9D:78:DB:3E:1D:8C:F4:F9:72:6A:79:5D:A1:64:3C:A5:F1:35:8E:1D:DB:0E:DC:0D:7E:B
Alias name: emsigneccrootcag3
  SHA1: 30:43:FA:4F:F2:57:DC:A0:C3:80:EE:2E:58:EA:78:B2:3F:E6:BB:C1
```

```
  SHA256:
 86:A1:EC:BA:08:9C:4A:8D:3B:BE:27:34:C6:12:BA:34:1D:81:3E:04:3C:F9:E8:A8:62:CD:5C:57:A3:6B:BE:6
Alias name: emsignrootcac1
  SHA1: E7:2E:F1:DF:FC:B2:09:28:CF:5D:D4:D5:67:37:B1:51:CB:86:4F:01
  SHA256:
 12:56:09:AA:30:1D:A0:A2:49:B9:7A:82:39:CB:6A:34:21:6F:44:DC:AC:9F:39:54:B1:42:92:F2:E8:C8:60:8
Alias name: emsignrootcag1
  SHA1: 8A:C7:AD:8F:73:AC:4E:C1:B5:75:4D:A5:40:F4:FC:CF:7C:B5:8E:8C
  SHA256:
 40:F6:AF:03:46:A9:9A:A1:CD:1D:55:5A:4E:9C:CE:62:C7:F9:63:46:03:EE:40:66:15:83:3D:C8:C8:D0:03:6
Alias name: entrust2048ca
  SHA1: 50:30:06:09:1D:97:D4:F5:AE:39:F7:CB:E7:92:7D:7D:65:2D:34:31
  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:B1:7
Alias name: entrustevca
  SHA1: B3:1E:B1:B7:40:E3:6C:84:02:DA:DC:37:D4:4D:F5:D4:67:49:52:F9
  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:4
Alias name: entrustnetpremium2048secureserverca
  SHA1: 50:30:06:09:1D:97:D4:F5:AE:39:F7:CB:E7:92:7D:7D:65:2D:34:31
  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:B1:7
Alias name: entrustrootcag2
  SHA1: 8C:F4:27:FD:79:0C:3A:D1:66:06:8D:E8:1E:57:EF:BB:93:22:72:D4
  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:F3:3
Alias name: entrustrootcertificationauthority
  SHA1: B3:1E:B1:B7:40:E3:6C:84:02:DA:DC:37:D4:4D:F5:D4:67:49:52:F9
  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:4
Alias name: entrustrootcertificationauthorityec1
  SHA1: 20:D8:06:40:DF:9B:25:F5:12:25:3A:11:EA:F7:59:8A:EB:14:B5:47
  SHA256:
 02:ED:0E:B2:8C:14:DA:45:16:5C:56:67:91:70:0D:64:51:D7:FB:56:F0:B2:AB:1D:3B:8E:B0:70:E5:6E:DF:F
Alias name: entrustrootcertificationauthorityg2
  SHA1: 8C:F4:27:FD:79:0C:3A:D1:66:06:8D:E8:1E:57:EF:BB:93:22:72:D4
  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:F3:3
Alias name: entrustrootcertificationauthorityg4
  SHA1: 14:88:4E:86:26:37:B0:26:AF:59:62:5C:40:77:EC:35:29:BA:96:01
  SHA256:
 DB:35:17:D1:F6:73:2A:2D:5A:B9:7C:53:3E:C7:07:79:EE:32:70:A6:2F:B4:AC:42:38:37:24:60:E6:F0:1E:8
Alias name: epkirootcertificationauthority
  SHA1: 67:65:0D:F1:7E:8E:7E:5B:82:40:A4:F4:56:4B:CF:E2:3D:69:C6:F0
```

```
  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:D
Alias name: equifaxsecureebusinessca1
  SHA1: AE:E6:3D:70:E3:76:FB:C7:3A:EB:B0:A1:C1:D4:C4:7A:A7:40:B3:F4
  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:F5:9
Alias name: equifaxsecureglobalebusinessca1
  SHA1: 3A:74:CB:7A:47:DB:70:DE:89:1F:24:35:98:64:B8:2D:82:BD:1A:36
  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:A
Alias name: eszignorootca2017
  SHA1: 89:D4:83:03:4F:9E:9A:48:80:5F:72:37:D4:A9:A6:EF:CB:7C:1F:D1
  SHA256:
 BE:B0:0B:30:83:9B:9B:C3:2C:32:E4:44:79:05:95:06:41:F2:64:21:B1:5E:D0:89:19:8B:51:8A:E2:EA:1B:9
Alias name: etugracertificationauthority
  SHA1: 51:C6:E7:08:49:06:6E:F3:92:D4:5C:A0:0D:6D:A3:62:8F:C3:52:39
  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:F2:3
Alias name: gd-class2-root.pem
  SHA1: 27:96:BA:E6:3F:18:01:E2:77:26:1B:A0:D7:77:70:02:8F:20:EE:E4
  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:E
Alias name: gd_bundle-g2.pem
  SHA1: 27:AC:93:69:FA:F2:52:07:BB:26:27:CE:FA:CC:BE:4E:F9:C3:19:B8
  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:A
Alias name: gdcatrustauthr5root
  SHA1: 0F:36:38:5B:81:1A:25:C3:9B:31:4E:83:CA:E9:34:66:70:CC:74:B4
  SHA256:
 BF:FF:8F:D0:44:33:48:7D:6A:8A:A6:0C:1A:29:76:7A:9F:C2:BB:B0:5E:42:0F:71:3A:13:B9:92:89:1D:38:9
Alias name: gdroot-g2.pem
  SHA1: 47:BE:AB:C9:22:EA:E8:0E:78:78:34:62:A7:9F:45:C2:54:FD:E6:8B
  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:D
Alias name: geotrustglobalca
  SHA1: DE:28:F4:A4:FF:E5:B9:2F:A3:C5:03:D1:A3:49:A7:F9:96:2A:82:12
  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:A7:3
Alias name: geotrustprimaryca
  SHA1: 32:3C:11:8E:1B:F7:B8:B6:52:54:E2:E2:10:0D:D6:02:90:37:F0:96
  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:D0:6
Alias name: geotrustprimarycag2
  SHA1: 8D:17:84:D5:37:F3:03:7D:EC:70:FE:57:8B:51:9A:99:E6:10:D7:B0
```

```
  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:D7:6
Alias name: geotrustprimarycag3
  SHA1: 03:9E:ED:B8:0B:E7:A0:3C:69:53:89:3B:20:D2:D9:32:3A:4C:2A:FD
  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:D
Alias name: geotrustprimarycertificationauthority
  SHA1: 32:3C:11:8E:1B:F7:B8:B6:52:54:E2:E2:10:0D:D6:02:90:37:F0:96
  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:D0:6
Alias name: geotrustprimarycertificationauthorityg2
  SHA1: 8D:17:84:D5:37:F3:03:7D:EC:70:FE:57:8B:51:9A:99:E6:10:D7:B0
  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:D7:6
Alias name: geotrustprimarycertificationauthorityg3
  SHA1: 03:9E:ED:B8:0B:E7:A0:3C:69:53:89:3B:20:D2:D9:32:3A:4C:2A:FD
  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:D
Alias name: geotrustuniversalca
  SHA1: E6:21:F3:35:43:79:05:9A:4B:68:30:9D:8A:2F:74:22:15:87:EC:79
  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:B9:1
Alias name: geotrustuniversalca2
  SHA1: 37:9A:19:7B:41:85:45:35:0C:A6:03:69:F3:3C:2E:AF:47:4F:20:79
  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:0
Alias name: globalchambersignroot2008
  SHA1: 4A:BD:EE:EC:95:0D:35:9C:89:AE:C7:52:A1:2C:5B:29:F6:D6:AA:0C
  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:C
Alias name: globalsignca
  SHA1: B1:BC:96:8B:D4:F4:9D:62:2A:A8:9A:81:F2:15:01:52:A4:1D:82:9C
  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:9
Alias name: globalsigneccrootcar4
  SHA1: 69:69:56:2E:40:80:F4:24:A1:E7:19:9F:14:BA:F3:EE:58:AB:6A:BB
  SHA256:
 BE:C9:49:11:C2:95:56:76:DB:6C:0A:55:09:86:D7:6E:3B:A0:05:66:7C:44:2C:97:62:B4:FB:B7:73:DE:22:8
Alias name: globalsigneccrootcar5
  SHA1: 1F:24:C6:30:CD:A4:18:EF:20:69:FF:AD:4F:DD:5F:46:3A:1B:69:AA
  SHA256:
 17:9F:BC:14:8A:3D:D0:0F:D2:4E:A1:34:58:CC:43:BF:A7:F5:9C:81:82:D7:83:A5:13:F6:EB:EC:10:0C:89:2
Alias name: globalsignr2ca
  SHA1: 75:E0:AB:B6:13:85:12:27:1C:04:F8:5F:DD:DE:38:E4:B7:24:2E:FE
```

```
  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:9
Alias name: globalsignr3ca
  SHA1: D6:9B:56:11:48:F0:1C:77:C5:45:78:C1:09:26:DF:5B:85:69:76:AD
  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:C5:3
Alias name: globalsignrootca
  SHA1: B1:BC:96:8B:D4:F4:9D:62:2A:A8:9A:81:F2:15:01:52:A4:1D:82:9C
  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:9
Alias name: globalsignrootcar2
  SHA1: 75:E0:AB:B6:13:85:12:27:1C:04:F8:5F:DD:DE:38:E4:B7:24:2E:FE
  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:9
Alias name: globalsignrootcar3
  SHA1: D6:9B:56:11:48:F0:1C:77:C5:45:78:C1:09:26:DF:5B:85:69:76:AD
  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:C5:3
Alias name: globalsignrootcar6
  SHA1: 80:94:64:0E:B5:A7:A1:CA:11:9C:1F:DD:D5:9F:81:02:63:A7:FB:D1
  SHA256:
 2C:AB:EA:FE:37:D0:6C:A2:2A:BA:73:91:C0:03:3D:25:98:29:52:C4:53:64:73:49:76:3A:3A:B5:AD:6C:CF:6
Alias name: godaddyclass2ca
  SHA1: 27:96:BA:E6:3F:18:01:E2:77:26:1B:A0:D7:77:70:02:8F:20:EE:E4
  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:E
Alias name: godaddyrootcertificateauthorityg2
  SHA1: 47:BE:AB:C9:22:EA:E8:0E:78:78:34:62:A7:9F:45:C2:54:FD:E6:8B
  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:D
Alias name: godaddyrootg2ca
  SHA1: 47:BE:AB:C9:22:EA:E8:0E:78:78:34:62:A7:9F:45:C2:54:FD:E6:8B
  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:D
Alias name: gtsrootr1
  SHA1: E1:C9:50:E6:EF:22:F8:4C:56:45:72:8B:92:20:60:D7:D5:A7:A3:E8
  SHA256:
 2A:57:54:71:E3:13:40:BC:21:58:1C:BD:2C:F1:3E:15:84:63:20:3E:CE:94:BC:F9:D3:CC:19:6B:F0:9A:54:7
Alias name: gtsrootr2
  SHA1: D2:73:96:2A:2A:5E:39:9F:73:3F:E1:C7:1E:64:3F:03:38:34:FC:4D
  SHA256:
 C4:5D:7B:B0:8E:6D:67:E6:2E:42:35:11:0B:56:4E:5F:78:FD:92:EF:05:8C:84:0A:EA:4E:64:55:D7:58:5C:6
Alias name: gtsrootr3
  SHA1: 30:D4:24:6F:07:FF:DB:91:89:8A:0B:E9:49:66:11:EB:8C:5E:46:E5
```

```
  SHA256:
 15:D5:B8:77:46:19:EA:7D:54:CE:1C:A6:D0:B0:C4:03:E0:37:A9:17:F1:31:E8:A0:4E:1E:6B:7A:71:BA:BC:E
Alias name: gtsrootr4
  SHA1: 2A:1D:60:27:D9:4A:B1:0A:1C:4D:91:5C:CD:33:A0:CB:3E:2D:54:CB
  SHA256:
 71:CC:A5:39:1F:9E:79:4B:04:80:25:30:B3:63:E1:21:DA:8A:30:43:BB:26:66:2F:EA:4D:CA:7F:C9:51:A4:B
Alias name: hellenicacademicandresearchinstitutionseccrootca2015
  SHA1: 9F:F1:71:8D:92:D5:9A:F3:7D:74:97:B4:BC:6F:84:68:0B:BA:B6:66
  SHA256:
 44:B5:45:AA:8A:25:E6:5A:73:CA:15:DC:27:FC:36:D2:4C:1C:B9:95:3A:06:65:39:B1:15:82:DC:48:7B:48:3
Alias name: hellenicacademicandresearchinstitutionsrootca2011
  SHA1: FE:45:65:9B:79:03:5B:98:A1:61:B5:51:2E:AC:DA:58:09:48:22:4D
  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:7
Alias name: hellenicacademicandresearchinstitutionsrootca2015
  SHA1: 01:0C:06:95:A6:98:19:14:FF:BF:5F:C6:B0:B6:95:EA:29:E9:12:A6
  SHA256:
 A0:40:92:9A:02:CE:53:B4:AC:F4:F2:FF:C6:98:1C:E4:49:6F:75:5E:6D:45:FE:0B:2A:69:2B:CD:52:52:3F:3
Alias name: hongkongpostrootca1
  SHA1: D6:DA:A8:20:8D:09:D2:15:4D:24:B5:2F:CB:34:6E:B2:58:B2:8A:58
  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:B
Alias name: hongkongpostrootca3
  SHA1: 58:A2:D0:EC:20:52:81:5B:C1:F3:F8:64:02:24:4E:C2:8E:02:4B:02
  SHA256:
 5A:2F:C0:3F:0C:83:B0:90:BB:FA:40:60:4B:09:88:44:6C:76:36:18:3D:F9:84:6E:17:10:1A:44:7F:B8:EF:D
Alias name: identrustcommercialrootca1
  SHA1: DF:71:7E:AA:4A:D9:4E:C9:55:84:99:60:2D:48:DE:5F:BC:F0:3A:25
  SHA256:
 5D:56:49:9B:E4:D2:E0:8B:CF:CA:D0:8A:3E:38:72:3D:50:50:3B:DE:70:69:48:E4:2F:55:60:30:19:E5:28:A
Alias name: identrustpublicsectorrootca1
  SHA1: BA:29:41:60:77:98:3F:F4:F3:EF:F2:31:05:3B:2E:EA:6D:4D:45:FD
  SHA256:
 30:D0:89:5A:9A:44:8A:26:20:91:63:55:22:D1:F5:20:10:B5:86:7A:CA:E1:2C:78:EF:95:8F:D4:F4:38:9F:2
Alias name: isrgrootx1
  SHA1: CA:BD:2A:79:A1:07:6A:31:F2:1D:25:36:35:CB:03:9D:43:29:A5:E8
  SHA256:
 96:BC:EC:06:26:49:76:F3:74:60:77:9A:CF:28:C5:A7:CF:E8:A3:C0:AA:E1:1A:8F:FC:EE:05:C0:BD:DF:08:C
Alias name: izenpecom
  SHA1: 2F:78:3D:25:52:18:A7:4A:65:39:71:B5:2C:A2:9C:45:15:6F:E9:19
  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:D4:53:1
Alias name: keynectisrootca
  SHA1: 9C:61:5C:4D:4D:85:10:3A:53:26:C2:4D:BA:EA:E4:A2:D2:D5:CC:97
```

```
  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:A3:3
Alias name: microseceszignorootca2009
  SHA1: 89:DF:74:FE:5C:F4:0F:4A:80:F9:E3:37:7D:54:DA:91:E1:01:31:8E
  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:F3:7
Alias name: mozillacert0.pem
  SHA1: 97:81:79:50:D8:1C:96:70:CC:34:D8:09:CF:79:44:31:36:7E:F4:74
  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:3
Alias name: mozillacert1.pem
  SHA1: 23:E5:94:94:51:95:F2:41:48:03:B4:D5:64:D2:A3:A3:F5:D8:8B:8C
  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:E
Alias name: mozillacert10.pem
  SHA1: 5F:3A:FC:0A:8B:64:F6:86:67:34:74:DF:7E:A9:A2:FE:F9:FA:7A:51
  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:D9:2
Alias name: mozillacert100.pem
  SHA1: 58:E8:AB:B0:36:15:33:FB:80:F7:9B:1B:6D:29:D3:FF:8D:5F:00:F0
  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:C
Alias name: mozillacert101.pem
  SHA1: 99:A6:9B:E6:1A:FE:88:6B:4D:2B:82:00:7C:B8:54:FC:31:7E:15:39
  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:5
Alias name: mozillacert102.pem
  SHA1: 96:C9:1B:0B:95:B4:10:98:42:FA:D0:D8:22:79:FE:60:FA:B9:16:83
  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:D3:48:8
Alias name: mozillacert103.pem
  SHA1: 70:C1:8D:74:B4:28:81:0A:E4:FD:A5:75:D7:01:9F:99:B0:3D:50:74
  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:B
Alias name: mozillacert104.pem
  SHA1: 4F:99:AA:93:FB:2B:D1:37:26:A1:99:4A:CE:7F:F0:05:F2:93:5D:1E
  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:F
Alias name: mozillacert105.pem
  SHA1: 77:47:4F:C6:30:E4:0F:4C:47:64:3F:84:BA:B8:C6:95:4A:8A:41:EC
  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:38:4
Alias name: mozillacert106.pem
  SHA1: E7:A1:90:29:D3:D5:52:DC:0D:0F:C6:92:D3:EA:88:0D:15:2E:1A:6B
```

```
  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:B2:58:01:5
Alias name: mozillacert107.pem
  SHA1: 8E:1C:74:F8:A6:20:B9:E5:8A:F4:61:FA:EC:2B:47:56:51:1A:52:C6
  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:C
Alias name: mozillacert108.pem
  SHA1: B1:BC:96:8B:D4:F4:9D:62:2A:A8:9A:81:F2:15:01:52:A4:1D:82:9C
  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:9
Alias name: mozillacert109.pem
  SHA1: B5:61:EB:EA:A4:DE:E4:25:4B:69:1A:98:A5:57:47:C2:34:C7:D9:71
  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:F8:07:0
Alias name: mozillacert11.pem
  SHA1: 05:63:B8:63:0D:62:D7:5A:BB:C8:AB:1E:4B:DF:B5:A8:99:B2:4D:43
  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:C6:32:5
Alias name: mozillacert110.pem
  SHA1: 93:05:7A:88:15:C6:4F:CE:88:2F:FA:91:16:52:28:78:BC:53:64:17
  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:D1:04:38:41:1
Alias name: mozillacert111.pem
  SHA1: 9C:BB:48:53:F6:A4:F6:D3:52:A4:E8:32:52:55:60:13:F5:AD:AF:65
  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:1
Alias name: mozillacert112.pem
  SHA1: 43:13:BB:96:F1:D5:86:9B:C1:4E:6A:92:F6:CF:F6:34:69:87:82:37
  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:A3:8
Alias name: mozillacert113.pem
  SHA1: 50:30:06:09:1D:97:D4:F5:AE:39:F7:CB:E7:92:7D:7D:65:2D:34:31
  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:B1:7
Alias name: mozillacert114.pem
  SHA1: 51:C6:E7:08:49:06:6E:F3:92:D4:5C:A0:0D:6D:A3:62:8F:C3:52:39
  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:F2:3
Alias name: mozillacert115.pem
  SHA1: 59:0D:2D:7D:88:4F:40:2E:61:7E:A5:62:32:17:65:CF:17:D8:94:E9
  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:17:05:25:5
Alias name: mozillacert116.pem
  SHA1: 2B:B1:F5:3E:55:0C:1D:C5:F1:D4:E6:B7:6A:46:4B:55:06:02:AC:21
```

```
  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:A4:7
Alias name: mozillacert117.pem
  SHA1: D4:DE:20:D0:5E:66:FC:53:FE:1A:50:88:2C:78:DB:28:52:CA:E4:74
  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:E
Alias name: mozillacert118.pem
  SHA1: 7E:78:4A:10:1C:82:65:CC:2D:E1:F1:6D:47:B4:40:CA:D9:0A:19:45
  SHA256:
 5F:0B: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:0
Alias name: mozillacert119.pem
  SHA1: 75:E0:AB:B6:13:85:12:27:1C:04:F8:5F:DD:DE:38:E4:B7:24:2E:FE
  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:9
Alias name: mozillacert12.pem
  SHA1: A8:98:5D:3A:65:E5:E5:C4:B2:D7:D6:6D:40:C6:DD:2F:B1:9C:54:36
  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:C7:01:6
Alias name: mozillacert120.pem
  SHA1: DA:40:18:8B:91:89:A3:ED:EE:AE:DA:97:FE:2F:9D:F5:B7:D1:8A:41
  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:F6:7
Alias name: mozillacert121.pem
  SHA1: CC:AB:0E:A0:4C:23:01:D6:69:7B:DD:37:9F:CD:12:EB:24:E3:94:9D
  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:A
Alias name: mozillacert122.pem
  SHA1: 02:FA:F3:E2:91:43:54:68:60:78:57:69:4D:F5:E4:5B:68:85:18:68
  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:F
Alias name: mozillacert123.pem
  SHA1: 2A:B6:28:48:5E:78:FB:F3:AD:9E:79:10:DD:6B:DF:99:72:2C:96:E5
  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:69:2
Alias name: mozillacert124.pem
  SHA1: 4D:23:78:EC:91:95:39:B5:00:7F:75:8F:03:3B:21:1E:C5:4D:8B:CF
  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:D0:60:1
Alias name: mozillacert125.pem
  SHA1: B3:1E:B1:B7:40:E3:6C:84:02:DA:DC:37:D4:4D:F5:D4:67:49:52:F9
  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:4
Alias name: mozillacert126.pem
  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:4
Alias name: mozillacert127.pem
  SHA1: DE:28:F4:A4:FF:E5:B9:2F:A3:C5:03:D1:A3:49:A7:F9:96:2A:82:12
  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:A7:3
Alias name: mozillacert128.pem
  SHA1: A9:E9:78:08:14:37:58:88:F2:05:19:B0:6D:2B:0D:2B:60:16:90:7D
  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:8
Alias name: mozillacert129.pem
  SHA1: E6:21:F3:35:43:79:05:9A:4B:68:30:9D:8A:2F:74:22:15:87:EC:79
  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:B9:1
Alias name: mozillacert13.pem
  SHA1: 06:08:3F:59:3F:15:A1:04:A0:69:A4:6B:A9:03:D0:06:B7:97:09:91
  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:9
Alias name: mozillacert130.pem
  SHA1: E5:DF:74:3C:B6:01:C4:9B:98:43:DC:AB:8C:E8:6A:81:10:9F:E4:8E
  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:01:0
Alias name: mozillacert131.pem
  SHA1: 37:9A:19:7B:41:85:45:35:0C:A6:03:69:F3:3C:2E:AF:47:4F:20:79
  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:0
Alias name: mozillacert132.pem
  SHA1: 39:21:C1:15:C1:5D:0E:CA:5C:CB:5B:C4:F0:7D:21:D8:05:0B:56:6A
  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:E
Alias name: mozillacert133.pem
  SHA1: 85:B5:FF:67:9B:0C:79:96:1F:C8:6E:44:22:00:46:13:DB:17:92:84
  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:B
Alias name: mozillacert134.pem
  SHA1: 70:17:9B:86:8C:00:A4:FA:60:91:52:22:3F:9F:3E:32:BD:E0:05:62
  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:D1:2
Alias name: mozillacert135.pem
  SHA1: 62:52:DC:40:F7:11:43:A2:2F:DE:9E:F7:34:8E:06:42:51:B1:81:18
  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:D8:14:36:2
Alias name: mozillacert136.pem
  SHA1: D1:EB:23:A4:6D:17:D6:8F:D9:25:64:C2:F1:F1:60:17:64:D8:E3:49
```

```
  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:F
Alias name: mozillacert137.pem
  SHA1: 4A:65:D5:F4:1D:EF:39:B8:B8:90:4A:4A:D3:64:81:33:CF:C7:A1:D1
  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:E
Alias name: mozillacert138.pem
  SHA1: E1:9F:E3:0E:8B:84:60:9E:80:9B:17:0D:72:A8:C5:BA:6E:14:09:BD
  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:6
Alias name: mozillacert139.pem
  SHA1: DE:3F:40:BD:50:93:D3:9B:6C:60:F6:DA:BC:07:62:01:00:89:76:C9
  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:88:7
Alias name: mozillacert14.pem
  SHA1: 5F:B7:EE:06:33:E2:59:DB:AD:0C:4C:9A:E6:D3:8F:1A:61:C7:DC:25
  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:C
Alias name: mozillacert140.pem
  SHA1: CA:3A:FB:CF:12:40:36:4B:44:B2:16:20:88:80:48:39:19:93:7C:F7
  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:8
Alias name: mozillacert141.pem
  SHA1: 31:7A:2A:D0:7F:2B:33:5E:F5:A1:C3:4E:4B:57:E8:B7:D8:F1:FC:A6
  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:D6:93:55:59:6
Alias name: mozillacert142.pem
  SHA1: 1F:49:14:F7:D8:74:95:1D:DD:AE:02:C0:BE:FD:3A:2D:82:75:51:85
  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:3
Alias name: mozillacert143.pem
  SHA1: 36:B1:2B:49:F9:81:9E:D7:4C:9E:BC:38:0F:C6:56:8F:5D:AC:B2:F7
  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:6
Alias name: mozillacert144.pem
  SHA1: 37:F7:6D:E6:07:7C:90:C5:B1:3E:93:1A:B7:41:10:B4:F2:E4:9A:27
  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:D9:2
Alias name: mozillacert145.pem
  SHA1: 10:1D:FA:3F:D5:0B:CB:BB:9B:B5:60:0C:19:55:A4:1A:F4:73:3A:04
  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:A
Alias name: mozillacert146.pem
  SHA1: 21:FC:BD:8E:7F:6C:AF:05:1B:D1:B3:43:EC:A8:E7:61:47:F2:0F:8A
```

```
  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:E
Alias name: mozillacert147.pem
  SHA1: 58:11:9F:0E:12:82:87:EA:50:FD:D9:87:45:6F:4F:78:DC:FA:D6:D4
  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:B3:4
Alias name: mozillacert148.pem
  SHA1: 04:83:ED:33:99:AC:36:08:05:87:22:ED:BC:5E:46:00:E3:BE:F9:D7
  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:D8:81:21:3
Alias name: mozillacert149.pem
  SHA1: 6E:3A:55:A4:19:0C:19:5C:93:84:3C:C0:DB:72:2E:31:30:61:F0:B1
  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:C
Alias name: mozillacert15.pem
  SHA1: 74:20:74:41:72:9C:DD:92:EC:79:31:D8:23:10:8D:C2:81:92:E2:BB
  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:C
Alias name: mozillacert150.pem
  SHA1: 33:9B:6B:14:50:24:9B:55:7A:01:87:72:84:D9:E0:2F:C3:D2:D8:E9
  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:E
Alias name: mozillacert151.pem
  SHA1: AC:ED:5F:65:53:FD:25:CE:01:5F:1F:7A:48:3B:6A:74:9F:61:78:C6
  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:F6:6
Alias name: mozillacert16.pem
  SHA1: DA:C9:02:4F:54:D8:F6:DF:94:93:5F:B1:73:26:38:CA:6A:D7:7C:13
  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:D6:77:07:3
Alias name: mozillacert17.pem
  SHA1: 40:54:DA:6F:1C:3F:40:74:AC:ED:0F:EC:CD:DB:79:D1:53:FB:90:1D
  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:4
Alias name: mozillacert18.pem
  SHA1: 79:98:A3:08:E1:4D:65:85:E6:C2:1E:15:3A:71:9F:BA:5A:D3:4A:D9
  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:A
Alias name: mozillacert19.pem
  SHA1: B4:35:D4:E1:11:9D:1C:66:90:A7:49:EB:B3:94:BD:63:7B:A7:82:B7
  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:E
Alias name: mozillacert2.pem
  SHA1: 22:D5:D8:DF:8F:02:31:D1:8D:F7:9D:B7:CF:8A:2D:64:C9:3F:6C:3A
```

```
  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:7
Alias name: mozillacert20.pem
  SHA1: D8:C5:38:8A:B7:30:1B:1B:6E:D4:7A:E6:45:25:3A:6F:9F:1A:27:61
  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:9
Alias name: mozillacert21.pem
  SHA1: 9B:AA:E5:9F:56:EE:21:CB:43:5A:BE:25:93:DF:A7:F0:40:D1:1D:CB
  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:D
Alias name: mozillacert22.pem
  SHA1: 32:3C:11:8E:1B:F7:B8:B6:52:54:E2:E2:10:0D:D6:02:90:37:F0:96
  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:D0:6
Alias name: mozillacert23.pem
  SHA1: 91:C6:D6:EE:3E:8A:C8:63:84:E5:48:C2:99:29:5C:75:6C:81:7B:81
  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:9
Alias name: mozillacert24.pem
  SHA1: 59:AF:82:79:91:86:C7:B4:75:07:CB:CF:03:57:46:EB:04:DD:B7:16
  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:B6:71:6
Alias name: mozillacert25.pem
  SHA1: 4E:B6:D5:78:49:9B:1C:CF:5F:58:1E:AD:56:BE:3D:9B:67:44:A5:E5
  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:D
Alias name: mozillacert26.pem
  SHA1: 87:82:C6:C3:04:35:3B:CF:D2:96:92:D2:59:3E:7D:44:D9:34:FF:11
  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:7
Alias name: mozillacert27.pem
  SHA1: 3A:44:73:5A:E5:81:90:1F:24:86:61:46:1E:3B:9C:C4:5F:F5:3A:1B
  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:6
Alias name: mozillacert28.pem
  SHA1: 66:31:BF:9E:F7:4F:9E:B6:C9:D5:A6:0C:BA:6A:BE:D1:F7:BD:EF:7B
  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:6
Alias name: mozillacert29.pem
  SHA1: 74:F8:A3:C3:EF:E7:B3:90:06:4B:83:90:3C:21:64:60:20:E5:DF:CE
  SHA256:
 15:F0:BA:00:A3:AC:7A:F3:AC:88:4C:07:2B:10:11:A0:77:BD:77:C0:97:F4:01:64:B2:F8:59:8A:BD:83:86:0
Alias name: mozillacert3.pem
  SHA1: 87:9F:4B:EE:05:DF:98:58:3B:E3:60:D6:33:E7:0D:3F:FE:98:71:AF
```

```
  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:03:1
Alias name: mozillacert30.pem
  SHA1: E7:B4:F6:9D:61:EC:90:69:DB:7E:90:A7:40:1A:3C:F4:7D:4F:E8:EE
  SHA256:
 A7:12:72:AE:AA:A3:CF:E8:72:7F:7F:B3:9F:0F:B3:D1:E5:42:6E:90:60:B0:6E:E6:F1:3E:9A:3C:58:33:CD:4
Alias name: mozillacert31.pem
  SHA1: 9F:74:4E:9F:2B:4D:BA:EC:0F:31:2C:50:B6:56:3B:8E:2D:93:C3:11
  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:C
Alias name: mozillacert32.pem
  SHA1: 60:D6:89:74:B5:C2:65:9E:8A:0F:C1:88:7C:88:D2:46:69:1B:18:2C
  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:89:3
Alias name: mozillacert33.pem
  SHA1: FE:B8:C4:32:DC:F9:76:9A:CE:AE:3D:D8:90:8F:FD:28:86:65:64:7D
  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:3
Alias name: mozillacert34.pem
  SHA1: 59:22:A1:E1:5A:EA:16:35:21:F8:98:39:6A:46:46:B0:44:1B:0F:A9
  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:F
Alias name: mozillacert35.pem
  SHA1: 2A:C8:D5:8B:57:CE:BF:2F:49:AF:F2:FC:76:8F:51:14:62:90:7A:41
  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:C
Alias name: mozillacert36.pem
  SHA1: 23:88:C9:D3:71:CC:9E:96:3D:FF:7D:3C:A7:CE:FC:D6:25:EC:19:0D
  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:D
Alias name: mozillacert37.pem
  SHA1: B1:2E:13:63:45:86:A4:6F:1A:B2:60:68:37:58:2D:C4:AC:FD:94:97
  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:A1:2
Alias name: mozillacert38.pem
  SHA1: CB:A1:C5:F8:B0:E3:5E:B8:B9:45:12:D3:F9:34:A2:E9:06:10:D3:36
  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:E3:43:08:5
Alias name: mozillacert39.pem
  SHA1: AE:50:83:ED:7C:F4:5C:BC:8F:61:C6:21:FE:68:5D:79:42:21:15:6E
  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:B
Alias name: mozillacert4.pem
  SHA1: E3:92:51:2F:0A:CF:F5:05:DF:F6:DE:06:7F:75:37:E1:65:EA:57:4B
```

```
  SHA256:
 0B: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:3
Alias name: mozillacert40.pem
  SHA1: 80:25:EF:F4:6E:70:C8:D4:72:24:65:84:FE:40:3B:8A:8D:6A:DB:F5
  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:B3:40:8
Alias name: mozillacert41.pem
  SHA1: 6B:2F:34:AD:89:58:BE:62:FD:B0:6B:5C:CE:BB:9D:D9:4F:4E:39:F3
  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:E
Alias name: mozillacert42.pem
  SHA1: 85:A4:08:C0:9C:19:3E:5D:51:58:7D:CD:D6:13:30:FD:8C:DE:37:BF
  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:D
Alias name: mozillacert43.pem
  SHA1: F9:CD:0E:2C:DA:76:24:C1:8F:BD:F0:F0:AB:B6:45:B8:F7:FE:D5:7A
  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:D
Alias name: mozillacert44.pem
  SHA1: 5F:43:E5:B1:BF:F8:78:8C:AC:1C:C7:CA:4A:9A:C6:22:2B:CC:34:C6
  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:A
Alias name: mozillacert45.pem
  SHA1: 67:65:0D:F1:7E:8E:7E:5B:82:40:A4:F4:56:4B:CF:E2:3D:69:C6:F0
  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:D
Alias name: mozillacert46.pem
  SHA1: 40:9D:4B:D9:17:B5:5C:27:B6:9B:64:CB:98:22:44:0D:CD:09:B8:89
  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:E9:3
Alias name: mozillacert47.pem
  SHA1: 1B:4B:39:61:26:27:6B:64:91:A2:68:6D:D7:02:43:21:2D:1F:1D:96
  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:C5:2
Alias name: mozillacert48.pem
  SHA1: A0:A1:AB:90:C9:FC:84:7B:3B:12:61:E8:97:7D:5F:D3:22:61:D3:CC
  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:3
Alias name: mozillacert49.pem
  SHA1: 61:57:3A:11:DF:0E:D8:7E:D5:92:65:22:EA:D0:56:D7:44:B3:23:71
  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:2
Alias name: mozillacert5.pem
  SHA1: B8:01:86:D1:EB:9C:86:A5:41:04:CF:30:54:F3:4C:52:B7:E5:58:C6
```

```
  SHA256:
 CE:CD:DC:90:50:99:D8:DA:DF:C5:B1:D2:09:B7:37:CB:E2:C1:8C:FB:2C:10:C0:FF:0B:CF:0D:32:86:FC:1A:A
Alias name: mozillacert50.pem
  SHA1: 8C:96:BA:EB:DD:2B:07:07:48:EE:30:32:66:A0:F3:98:6E:7C:AE:58
  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:B
Alias name: mozillacert51.pem
  SHA1: FA:B7:EE:36:97:26:62:FB:2D:B0:2A:F6:BF:03:FD:E8:7C:4B:2F:9B
  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:B
Alias name: mozillacert52.pem
  SHA1: 8B:AF:4C:9B:1D:F0:2A:92:F7:DA:12:8E:B9:1B:AC:F4:98:60:4B:6F
  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:A
Alias name: mozillacert53.pem
  SHA1: 7F:8A:B0:CF:D0:51:87:6A:66:F3:36:0F:47:C8:8D:8C:D3:35:FC:74
  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:1
Alias name: mozillacert54.pem
  SHA1: 03:9E:ED:B8:0B:E7:A0:3C:69:53:89:3B:20:D2:D9:32:3A:4C:2A:FD
  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:D
Alias name: mozillacert55.pem
  SHA1: AA:DB:BC:22:23:8F:C4:01:A1:27:BB:38:DD:F4:1D:DB:08:9E:F0:12
  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:15:5
Alias name: mozillacert56.pem
  SHA1: F1:8B:53:8D:1B:E9:03:B6:A6:F0:56:43:5B:17:15:89:CA:F3:6B:F2
  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:D3:4
Alias name: mozillacert57.pem
  SHA1: D6:DA:A8:20:8D:09:D2:15:4D:24:B5:2F:CB:34:6E:B2:58:B2:8A:58
  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:B
Alias name: mozillacert58.pem
  SHA1: 8D:17:84:D5:37:F3:03:7D:EC:70:FE:57:8B:51:9A:99:E6:10:D7:B0
  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:D7:6
Alias name: mozillacert59.pem
  SHA1: 36:79:CA:35:66:87:72:30:4D:30:A5:FB:87:3B:0F:A7:7B:B7:0D:54
  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:3
Alias name: mozillacert6.pem
  SHA1: 27:96:BA:E6:3F:18:01:E2:77:26:1B:A0:D7:77:70:02:8F:20:EE:E4
```

```
  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:E
Alias name: mozillacert60.pem
  SHA1: 3B:C4:9F:48:F8:F3:73:A0:9C:1E:BD:F8:5B:B1:C3:65:C7:D8:11:B3
  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:42:16:1
Alias name: mozillacert61.pem
  SHA1: E0:B4:32:2E:B2:F6:A5:68:B6:54:53:84:48:18:4A:50:36:87:43:84
  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:D9:40:0B:43:4
Alias name: mozillacert62.pem
  SHA1: A1:DB:63:93:91:6F:17:E4:18:55:09:40:04:15:C7:02:40:B0:AE:6B
  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:14:03:0
Alias name: mozillacert63.pem
  SHA1: 89:DF:74:FE:5C:F4:0F:4A:80:F9:E3:37:7D:54:DA:91:E1:01:31:8E
  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:F3:7
Alias name: mozillacert64.pem
  SHA1: 62:7F:8D:78:27:65:63:99:D2:7D:7F:90:44:C9:FE:B3:F3:3E:FA:9A
  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:7
Alias name: mozillacert65.pem
  SHA1: 69:BD:8C:F4:9C:D3:00:FB:59:2E:17:93:CA:55:6A:F3:EC:AA:35:FB
  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:73:01:9
Alias name: mozillacert66.pem
  SHA1: DD:E1:D2:A9:01:80:2E:1D:87:5E:84:B3:80:7E:4B:B1:FD:99:41:34
  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:D5:9
Alias name: mozillacert67.pem
  SHA1: D6:9B:56:11:48:F0:1C:77:C5:45:78:C1:09:26:DF:5B:85:69:76:AD
  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:C5:3
Alias name: mozillacert68.pem
  SHA1: AE:C5:FB:3F:C8:E1:BF:C4:E5:4F:03:07:5A:9A:E8:00:B7:F7:B6:FA
  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:E
Alias name: mozillacert69.pem
  SHA1: 2F:78:3D:25:52:18:A7:4A:65:39:71:B5:2C:A2:9C:45:15:6F:E9:19
  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:D4:53:1
Alias name: mozillacert7.pem
  SHA1: AD:7E:1C:28:B0:64:EF:8F:60:03:40:20:14:C3:D0:E3:37:0E:B5:8A
```

```
  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:B6:5
Alias name: mozillacert70.pem
  SHA1: 78:6A:74:AC:76:AB:14:7F:9C:6A:30:50:BA:9E:A8:7E:FE:9A:CE:3C
  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:C
Alias name: mozillacert71.pem
  SHA1: 4A:BD:EE:EC:95:0D:35:9C:89:AE:C7:52:A1:2C:5B:29:F6:D6:AA:0C
  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:C
Alias name: mozillacert72.pem
  SHA1: 47:BE:AB:C9:22:EA:E8:0E:78:78:34:62:A7:9F:45:C2:54:FD:E6:8B
  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:D
Alias name: mozillacert73.pem
  SHA1: B5:1C:06:7C:EE:2B:0C:3D:F8:55:AB:2D:92:F4:FE:39:D4:E7:0F:0E
  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:F
Alias name: mozillacert74.pem
  SHA1: 92:5A:8F:8D:2C:6D:04:E0:66:5F:59:6A:FF:22:D8:63:E8:25:6F:3F
  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:B
Alias name: mozillacert75.pem
  SHA1: D2:32:09:AD:23:D3:14:23:21:74:E4:0D:7F:9D:62:13:97:86:63:3A
  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:F1:7
Alias name: mozillacert76.pem
  SHA1: F9:B5:B6:32:45:5F:9C:BE:EC:57:5F:80:DC:E9:6E:2C:C7:B2:78: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:A
Alias name: mozillacert77.pem
  SHA1: 13:2D:0D:45:53:4B:69:97:CD:B2:D5:C3:39:E2:55:76:60:9B:5C:C6
  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:72:4
Alias name: mozillacert78.pem
  SHA1: 29:36:21:02:8B:20:ED:02:F5:66:C5:32:D1:D6:ED:90:9F:45:00:2F
  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:B4:1
Alias name: mozillacert79.pem
  SHA1: D8:A6:33:2C:E0:03:6F:B1:85:F6:63:4F:7D:6A:06:65:26:32:28:27
  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:9
Alias name: mozillacert8.pem
  SHA1: 3E:2B:F7:F2:03:1B:96:F3:8C:E6:C4:D8:A8:5D:3E:2D:58:47:6A:0F
```

```
  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:E
Alias name: mozillacert80.pem
  SHA1: B8:23:6B:00:2F:1D:16:86:53:01:55:6C:11:A4:37:CA:EB:FF:C3:BB
  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:84:88:82:14:2
Alias name: mozillacert81.pem
  SHA1: 07:E0:32:E0:20:B7:2C:3F:19:2F:06:28:A2:59:3A:19:A7:0F:06:9E
  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:C4:40:8
Alias name: mozillacert82.pem
  SHA1: 2E:14:DA:EC:28:F0:FA:1E:8E:38:9A:4E:AB:EB:26:C0:0A:D3:83:C3
  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:1
Alias name: mozillacert83.pem
  SHA1: A0:73:E5:C5:BD:43:61:0D:86:4C:21:13:0A:85:58:57:CC:9C:EA:46
  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:63:0
Alias name: mozillacert84.pem
  SHA1: D3:C0:63:F2:19:ED:07:3E:34:AD:5D:75:0B:32:76:29:FF:D5:9A:F2
  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:F
Alias name: mozillacert85.pem
  SHA1: CF:9E:87:6D:D3:EB:FC:42:26:97:A3:B5:A3:7A:A0:76:A9:06:23:48
  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:4
Alias name: mozillacert86.pem
  SHA1: 74:2C:31:92:E6:07:E4:24:EB:45:49:54:2B:E1:BB:C5:3E:61:74:E2
  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:7
Alias name: mozillacert87.pem
  SHA1: 5F:3B:8C:F2:F8:10:B3:7D:78:B4:CE:EC:19:19:C3:73:34:B9:C7:74
  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:F
Alias name: mozillacert88.pem
  SHA1: FE:45:65:9B:79:03:5B:98:A1:61:B5:51:2E:AC:DA:58:09:48:22:4D
  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:7
Alias name: mozillacert89.pem
  SHA1: C8:EC:8C:87:92:69:CB:4B:AB:39:E9:8D:7E:57:67:F3:14:95:73:9D
  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:96:05:61:79:15:0
Alias name: mozillacert9.pem
  SHA1: F4:8B:11:BF:DE:AB:BE:94:54:20:71:E6:41:DE:6B:BE:88:2B:40:B9
```

```
  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:B
Alias name: mozillacert90.pem
  SHA1: F3:73:B3:87:06:5A:28:84:8A:F2:F3:4A:CE:19:2B:DD:C7:8E:9C:AC
  SHA256:
 55:92:60:84:EC:96:3A:64:B9:6E:2A:BE:01:CE:0B:A8:6A:64:FB:FE:BC:C7:AA:B5:AF:C1:55:B3:7F:D7:60:6
Alias name: mozillacert91.pem
  SHA1: 3B:C0:38:0B:33:C3:F6:A6:0C:86:15:22:93:D9:DF:F5:4B:81:C0:04
  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:55:7
Alias name: mozillacert92.pem
  SHA1: A3:F1:33:3F:E2:42:BF:CF:C5:D1:4E:8F:39:42:98:40:68:10:D1:A0
  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:A9:1
Alias name: mozillacert93.pem
  SHA1: 31:F1:FD:68:22:63:20:EE:C6:3B:3F:9D:EA:4A:3E:53:7C:7C:39:17
  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:59:9
Alias name: mozillacert94.pem
  SHA1: 49:0A:75:74:DE:87:0A:47:FE:58:EE:F6:C7:6B:EB:C6:0B:12:40:99
  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:4
Alias name: mozillacert95.pem
  SHA1: DA:FA:F7:FA:66:84:EC:06:8F:14:50:BD:C7:C2:81:A5:BC:A9:64:57
  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:B2:4
Alias name: mozillacert96.pem
  SHA1: 55:A6:72:3E:CB:F2:EC:CD:C3:23:74:70:19:9D:2A:BE:11:E3:81:D1
  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:B
Alias name: mozillacert97.pem
  SHA1: 85:37:1C:A6:E5:50:14:3D:CE:28:03:47:1B:DE:3A:09:E8:F8:77:0F
  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:A8:88:63:8
Alias name: mozillacert98.pem
  SHA1: C9:A8:B9:E7:55:80:5E:58:E3:53:77:A7:25:EB:AF:C3:7B:27:CC:D7
  SHA256:
 3E:84:BA:43:42:90:85:16:E7:75:73:C0:99:2F:09:79:CA:08:4E:46:85:68:1F:F1:95:CC:BA:8A:22:9B:8A:7
Alias name: mozillacert99.pem
  SHA1: F1:7F:6F:B6:31:DC:99:E3:A3:C8:7F:FE:1C:F1:81:10:88:D9:60:33
  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:5
Alias name: netlockaranyclassgoldfotanusitvany
  SHA1: 06:08:3F:59:3F:15:A1:04:A0:69:A4:6B:A9:03:D0:06:B7:97:09:91
```

```
  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:9
Alias name: networksolutionscertificateauthority
  SHA1: 74:F8:A3:C3:EF:E7:B3:90:06:4B:83:90:3C:21:64:60:20:E5:DF:CE
  SHA256:
 15:F0:BA:00:A3:AC:7A:F3:AC:88:4C:07:2B:10:11:A0:77:BD:77:C0:97:F4:01:64:B2:F8:59:8A:BD:83:86:0
Alias name: oistewisekeyglobalrootgaca
  SHA1: 59:22:A1:E1:5A:EA:16:35:21:F8:98:39:6A:46:46:B0:44:1B:0F:A9
  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:F
Alias name: oistewisekeyglobalrootgbca
  SHA1: 0F:F9:40:76:18:D3:D7:6A:4B:98:F0:A8:35:9E:0C:FD:27:AC:CC:ED
  SHA256:
 6B:9C:08:E8:6E:B0:F7:67:CF:AD:65:CD:98:B6:21:49:E5:49:4A:67:F5:84:5E:7B:D1:ED:01:9F:27:B8:6B:D
Alias name: oistewisekeyglobalrootgcca
  SHA1: E0:11:84:5E:34:DE:BE:88:81:B9:9C:F6:16:26:D1:96:1F:C3:B9:31
  SHA256:
 85:60:F9:1C:36:24:DA:BA:95:70:B5:FE:A0:DB:E3:6F:F1:1A:83:23:BE:94:86:85:4F:B3:F3:4A:55:71:19:8
Alias name: quovadisrootca
  SHA1: DE:3F:40:BD:50:93:D3:9B:6C:60:F6:DA:BC:07:62:01:00:89:76:C9
  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:88:7
Alias name: quovadisrootca1g3
  SHA1: 1B:8E:EA:57:96:29:1A:C9:39:EA:B8:0A:81:1A:73:73:C0:93:79:67
  SHA256:
 8A:86:6F:D1:B2:76:B5:7E:57:8E:92:1C:65:82:8A:2B:ED:58:E9:F2:F2:88:05:41:34:B7:F1:F4:BF:C9:CC:7
Alias name: quovadisrootca2
  SHA1: CA:3A:FB:CF:12:40:36:4B:44:B2:16:20:88:80:48:39:19:93:7C:F7
  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:8
Alias name: quovadisrootca2g3
  SHA1: 09:3C:61:F3:8B:8B:DC:7D:55:DF:75:38:02:05:00:E1:25:F5:C8:36
  SHA256:
 8F:E4:FB:0A:F9:3A:4D:0D:67:DB:0B:EB:B2:3E:37:C7:1B:F3:25:DC:BC:DD:24:0E:A0:4D:AF:58:B4:7E:18:4
Alias name: quovadisrootca3
  SHA1: 1F:49:14:F7:D8:74:95:1D:DD:AE:02:C0:BE:FD:3A:2D:82:75:51:85
  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:3
Alias name: quovadisrootca3g3
  SHA1: 48:12:BD:92:3C:A8:C4:39:06:E7:30:6D:27:96:E6:A4:CF:22:2E:7D
  SHA256:
 88:EF:81:DE:20:2E:B0:18:45:2E:43:F8:64:72:5C:EA:5F:BD:1F:C2:D9:D2:05:73:07:09:C5:D8:B8:69:0F:4
Alias name: secomevrootca1
  SHA1: FE:B8:C4:32:DC:F9:76:9A:CE:AE:3D:D8:90:8F:FD:28:86:65:64:7D
```

```
  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:3
Alias name: secomscrootca1
  SHA1: 36:B1:2B:49:F9:81:9E:D7:4C:9E:BC:38:0F:C6:56:8F:5D:AC:B2:F7
  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:6
Alias name: secomscrootca2
  SHA1: 5F:3B:8C:F2:F8:10:B3:7D:78:B4:CE:EC:19:19:C3:73:34:B9:C7:74
  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:F
Alias name: secomvalicertclass1ca
  SHA1: E5:DF:74:3C:B6:01:C4:9B:98:43:DC:AB:8C:E8:6A:81:10:9F:E4:8E
  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:01:0
Alias name: secureglobalca
  SHA1: 3A:44:73:5A:E5:81:90:1F:24:86:61:46:1E:3B:9C:C4:5F:F5:3A:1B
  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:6
Alias name: securesignrootca11
  SHA1: 3B:C4:9F:48:F8:F3:73:A0:9C:1E:BD:F8:5B:B1:C3:65:C7:D8:11:B3
  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:42:16:1
Alias name: securetrustca
  SHA1: 87:82:C6:C3:04:35:3B:CF:D2:96:92:D2:59:3E:7D:44:D9:34:FF:11
  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:7
Alias name: securitycommunicationrootca
  SHA1: 36:B1:2B:49:F9:81:9E:D7:4C:9E:BC:38:0F:C6:56:8F:5D:AC:B2:F7
  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:6
Alias name: securitycommunicationrootca2
  SHA1: 5F:3B:8C:F2:F8:10:B3:7D:78:B4:CE:EC:19:19:C3:73:34:B9:C7:74
  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:F
Alias name: soneraclass1ca
  SHA1: 07:47:22:01:99:CE:74:B9:7C:B0:3D:79:B2:64:A2:C8:55:E9:33:FF
  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:32:31:3
Alias name: soneraclass2ca
  SHA1: 37:F7:6D:E6:07:7C:90:C5:B1:3E:93:1A:B7:41:10:B4:F2:E4:9A:27
  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:D9:2
Alias name: soneraclass2rootca
  SHA1: 37:F7:6D:E6:07:7C:90:C5:B1:3E:93:1A:B7:41:10:B4:F2:E4:9A:27
```

```
  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:D9:2
Alias name: sslcomevrootcertificationauthorityecc
  SHA1: 4C:DD:51:A3:D1:F5:20:32:14:B0:C6:C5:32:23:03:91:C7:46:42:6D
  SHA256:
 22:A2:C1:F7:BD:ED:70:4C:C1:E7:01:B5:F4:08:C3:10:88:0F:E9:56:B5:DE:2A:4A:44:F9:9C:87:3A:25:A7:C
Alias name: sslcomevrootcertificationauthorityrsar2
  SHA1: 74:3A:F0:52:9B:D0:32:A0:F4:4A:83:CD:D4:BA:A9:7B:7C:2E:C4:9A
  SHA256:
 2E:7B:F1:6C:C2:24:85:A7:BB:E2:AA:86:96:75:07:61:B0:AE:39:BE:3B:2F:E9:D0:CC:6D:4E:F7:34:91:42:5
Alias name: sslcomrootcertificationauthorityecc
  SHA1: C3:19:7C:39:24:E6:54:AF:1B:C4:AB:20:95:7A:E2:C3:0E:13:02:6A
  SHA256:
 34:17:BB:06:CC:60:07:DA:1B:96:1C:92:0B:8A:B4:CE:3F:AD:82:0E:4A:A3:0B:9A:CB:C4:A7:4E:BD:CE:BC:6
Alias name: sslcomrootcertificationauthorityrsa
  SHA1: B7:AB:33:08:D1:EA:44:77:BA:14:80:12:5A:6F:BD:A9:36:49:0C:BB
  SHA256:
 85:66:6A:56:2E:E0:BE:5C:E9:25:C1:D8:89:0A:6F:76:A8:7E:C1:6D:4D:7D:5F:29:EA:74:19:CF:20:12:3B:6
Alias name: staatdernederlandenevrootca
  SHA1: 76:E2:7E:C1:4F:DB:82:C1:C0:A6:75:B5:05:BE:3D:29:B4:ED:DB:BB
  SHA256:
 4D:24:91:41:4C:FE:95:67:46:EC:4C:EF:A6:CF:6F:72:E2:8A:13:29:43:2F:9D:8A:90:7A:C4:CB:5D:AD:C1:5
Alias name: staatdernederlandenrootcag3
  SHA1: D8:EB:6B:41:51:92:59:E0:F3:E7:85:00:C0:3D:B6:88:97:C9:EE:FC
  SHA256:
 3C:4F:B0:B9:5A:B8:B3:00:32:F4:32:B8:6F:53:5F:E1:72:C1:85:D0:FD:39:86:58:37:CF:36:18:7F:A6:F4:2
Alias name: starfieldclass2ca
  SHA1: AD:7E:1C:28:B0:64:EF:8F:60:03:40:20:14:C3:D0:E3:37:0E:B5:8A
  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:B6:5
Alias name: starfieldrootcertificateauthorityg2
  SHA1: B5:1C:06:7C:EE:2B:0C:3D:F8:55:AB:2D:92:F4:FE:39:D4:E7:0F:0E
  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:F
Alias name: starfieldrootg2ca
  SHA1: B5:1C:06:7C:EE:2B:0C:3D:F8:55:AB:2D:92:F4:FE:39:D4:E7:0F:0E
  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:F
Alias name: starfieldservicesrootcertificateauthorityg2
  SHA1: 92:5A:8F:8D:2C:6D:04:E0:66:5F:59:6A:FF:22:D8:63:E8:25:6F:3F
  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:B
Alias name: starfieldservicesrootg2ca
  SHA1: 92:5A:8F:8D:2C:6D:04:E0:66:5F:59:6A:FF:22:D8:63:E8:25:6F:3F
```

```
  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:B
Alias name: swisssigngoldcag2
  SHA1: D8:C5:38:8A:B7:30:1B:1B:6E:D4:7A:E6:45:25:3A:6F:9F:1A:27:61
  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:9
Alias name: swisssigngoldg2ca
  SHA1: D8:C5:38:8A:B7:30:1B:1B:6E:D4:7A:E6:45:25:3A:6F:9F:1A:27:61
  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:9
Alias name: swisssignplatinumg2ca
  SHA1: 56:E0:FA:C0:3B:8F:18:23:55:18:E5:D3:11:CA:E8:C2:43:31:AB:66
  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:14:3
Alias name: swisssignsilvercag2
  SHA1: 9B:AA:E5:9F:56:EE:21:CB:43:5A:BE:25:93:DF:A7:F0:40:D1:1D:CB
  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:D
Alias name: swisssignsilverg2ca
  SHA1: 9B:AA:E5:9F:56:EE:21:CB:43:5A:BE:25:93:DF:A7:F0:40:D1:1D:CB
  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:D
Alias name: szafirrootca2
  SHA1: E2:52:FA:95:3F:ED:DB:24:60:BD:6E:28:F3:9C:CC:CF:5E:B3:3F:DE
  SHA256:
 A1:33:9D:33:28:1A:0B:56:E5:57:D3:D3:2B:1C:E7:F9:36:7E:B0:94:BD:5F:A7:2A:7E:50:04:C8:DE:D7:CA:F
Alias name: teliasonerarootcav1
  SHA1: 43:13:BB:96:F1:D5:86:9B:C1:4E:6A:92:F6:CF:F6:34:69:87:82:37
  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:A3:8
Alias name: thawtepersonalfreemailca
  SHA1: E6:18:83:AE:84:CA:C1:C1:CD:52:AD:E8:E9:25:2B:45:A6:4F:B7:E2
  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:E1:8
Alias name: thawtepremiumserverca
  SHA1: E0:AB:05:94:20:72:54:93:05:60:62:02:36:70:F7:CD:2E:FC:66:66
  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:E
Alias name: thawteprimaryrootca
  SHA1: 91:C6:D6:EE:3E:8A:C8:63:84:E5:48:C2:99:29:5C:75:6C:81:7B:81
  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:9
Alias name: thawteprimaryrootcag2
  SHA1: AA:DB:BC:22:23:8F:C4:01:A1:27:BB:38:DD:F4:1D:DB:08:9E:F0:12
```

```
  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:15:5
Alias name: thawteprimaryrootcag3
  SHA1: F1:8B:53:8D:1B:E9:03:B6:A6:F0:56:43:5B:17:15:89:CA:F3:6B:F2
  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:D3:4
Alias name: thawteserverca
  SHA1: 9F:AD:91:A6:CE:6A:C6:C5:00:47:C4:4E:C9:D4:A5:0D:92:D8:49:79
  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:10:55:31:6
Alias name: trustcenterclass2caii
  SHA1: AE:50:83:ED:7C:F4:5C:BC:8F:61:C6:21:FE:68:5D:79:42:21:15:6E
  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:B
Alias name: trustcenterclass4caii
  SHA1: A6:9A:91:FD:05:7F:13:6A:42:63:0B:B1:76:0D:2D:51:12:0C:16:50
  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:3
Alias name: trustcenteruniversalcai
  SHA1: 6B:2F:34:AD:89:58:BE:62:FD:B0:6B:5C:CE:BB:9D:D9:4F:4E:39:F3
  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:E
Alias name: trustcoreca1
  SHA1: 58:D1:DF:95:95:67:6B:63:C0:F0:5B:1C:17:4D:8B:84:0B:C8:78:BD
  SHA256:
 5A:88:5D:B1:9C:01:D9:12:C5:75:93:88:93:8C:AF:BB:DF:03:1A:B2:D4:8E:91:EE:15:58:9B:42:97:1D:03:9
Alias name: trustcorrootcertca1
  SHA1: FF:BD:CD:E7:82:C8:43:5E:3C:6F:26:86:5C:CA:A8:3A:45:5B:C3:0A
  SHA256:
 D4:0E:9C:86:CD:8F:E4:68:C1:77:69:59:F4:9E:A7:74:FA:54:86:84:B6:C4:06:F3:90:92:61:F4:DC:E2:57:5
Alias name: trustcorrootcertca2
  SHA1: B8:BE:6D:CB:56:F1:55:B9:63:D4:12:CA:4E:06:34:C7:94:B2:1C:C0
  SHA256:
 07:53:E9:40:37:8C:1B:D5:E3:83:6E:39:5D:AE:A5:CB:83:9E:50:46:F1:BD:0E:AE:19:51:CF:10:FE:C7:C9:6
Alias name: trustisfpsrootca
  SHA1: 3B:C0:38:0B:33:C3:F6:A6:0C:86:15:22:93:D9:DF:F5:4B:81:C0:04
  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:55:7
Alias name: ttelesecglobalrootclass2
  SHA1: 59:0D:2D:7D:88:4F:40:2E:61:7E:A5:62:32:17:65:CF:17:D8:94:E9
  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:17:05:25:5
Alias name: ttelesecglobalrootclass2ca
  SHA1: 59:0D:2D:7D:88:4F:40:2E:61:7E:A5:62:32:17:65:CF:17:D8:94:E9
```

```
  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:17:05:25:5
Alias name: ttelesecglobalrootclass3
  SHA1: 55:A6:72:3E:CB:F2:EC:CD:C3:23:74:70:19:9D:2A:BE:11:E3:81:D1
  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:B
Alias name: ttelesecglobalrootclass3ca
  SHA1: 55:A6:72:3E:CB:F2:EC:CD:C3:23:74:70:19:9D:2A:BE:11:E3:81:D1
  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:B
Alias name: tubitakkamusmsslkoksertifikasisurum1
  SHA1: 31:43:64:9B:EC:CE:27:EC:ED:3A:3F:0B:8F:0D:E4:E8:91:DD:EE:CA
  SHA256:
 46:ED:C3:68:90:46:D5:3A:45:3F:B3:10:4A:B8:0D:CA:EC:65:8B:26:60:EA:16:29:DD:7E:86:79:90:64:87:1
Alias name: twcaglobalrootca
  SHA1: 9C:BB:48:53:F6:A4:F6:D3:52:A4:E8:32:52:55:60:13:F5:AD:AF:65
  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:1
Alias name: twcarootcertificationauthority
  SHA1: CF:9E:87:6D:D3:EB:FC:42:26:97:A3:B5:A3:7A:A0:76:A9:06:23:48
  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:4
Alias name: ucaextendedvalidationroot
  SHA1: A3:A1:B0:6F:24:61:23:4A:E3:36:A5:C2:37:FC:A6:FF:DD:F0:D7:3A
  SHA256:
 D4:3A:F9:B3:54:73:75:5C:96:84:FC:06:D7:D8:CB:70:EE:5C:28:E7:73:FB:29:4E:B4:1E:E7:17:22:92:4D:2
Alias name: ucaglobalg2root
  SHA1: 28:F9:78:16:19:7A:FF:18:25:18:AA:44:FE:C1:A0:CE:5C:B6:4C:8A
  SHA256:
 9B:EA:11:C9:76:FE:01:47:64:C1:BE:56:A6:F9:14:B5:A5:60:31:7A:BD:99:88:39:33:82:E5:16:1A:A0:49:3
Alias name: usertrustecc
  SHA1: D1:CB:CA:5D:B2:D5:2A:7F:69:3B:67:4D:E5:F0:5A:1D:0C:95:7D:F0
  SHA256:
 4F:F4:60:D5:4B:9C:86:DA:BF:BC:FC:57:12:E0:40:0D:2B:ED:3F:BC:4D:4F:BD:AA:86:E0:6A:DC:D2:A9:AD:7
Alias name: usertrustecccertificationauthority
  SHA1: D1:CB:CA:5D:B2:D5:2A:7F:69:3B:67:4D:E5:F0:5A:1D:0C:95:7D:F0
  SHA256:
 4F:F4:60:D5:4B:9C:86:DA:BF:BC:FC:57:12:E0:40:0D:2B:ED:3F:BC:4D:4F:BD:AA:86:E0:6A:DC:D2:A9:AD:7
Alias name: usertrustrsa
  SHA1: 2B:8F:1B:57:33:0D:BB:A2:D0:7A:6C:51:F7:0E:E9:0D:DA:B9:AD:8E
  SHA256:
 E7:93:C9:B0:2F:D8:AA:13:E2:1C:31:22:8A:CC:B0:81:19:64:3B:74:9C:89:89:64:B1:74:6D:46:C3:D4:CB:D
Alias name: usertrustrsacertificationauthority
  SHA1: 2B:8F:1B:57:33:0D:BB:A2:D0:7A:6C:51:F7:0E:E9:0D:DA:B9:AD:8E
```

```
  SHA256:
 E7:93:C9:B0:2F:D8:AA:13:E2:1C:31:22:8A:CC:B0:81:19:64:3B:74:9C:89:89:64:B1:74:6D:46:C3:D4:CB:D
Alias name: utndatacorpsgcca
  SHA1: 58:11:9F:0E:12:82:87:EA:50:FD:D9:87:45:6F:4F:78:DC:FA:D6:D4
  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:B3:4
Alias name: utnuserfirstclientauthemailca
  SHA1: B1:72:B1:A5:6D:95:F9:1F:E5:02:87:E1:4D:37:EA:6A:44:63:76:8A
  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:A
Alias name: utnuserfirsthardwareca
  SHA1: 04:83:ED:33:99:AC:36:08:05:87:22:ED:BC:5E:46:00:E3:BE:F9:D7
  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:D8:81:21:3
Alias name: utnuserfirstobjectca
  SHA1: E1:2D:FB:4B:41:D7:D9:C3:2B:30:51:4B:AC:1D:81:D8:38:5E:2D:46
  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:E5:8
Alias name: valicertclass2ca
  SHA1: 31:7A:2A:D0:7F:2B:33:5E:F5:A1:C3:4E:4B:57:E8:B7:D8:F1:FC:A6
  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:D6:93:55:59:6
Alias name: verisignc1g1.pem
  SHA1: 90:AE:A2:69:85:FF:14:80:4C:43:49:52:EC:E9:60:84:77:AF:55:6F
  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:33:5
Alias name: verisignc1g2.pem
  SHA1: 27:3E:E1:24:57:FD:C4:F9:0C:55:E8:2B:56:16:7F:62:F5:32:E5:47
  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:7
Alias name: verisignc1g3.pem
  SHA1: 20:42:85:DC:F7:EB:76:41:95:57:8E:13:6B:D4:B7:D1:E9:8E:46:A5
  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:6
Alias name: verisignc1g6.pem
  SHA1: 51:7F:61:1E:29:91:6B:53:82:FB:72:E7:44:D9:8D:C3:CC:53:6D:64
  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:B
Alias name: verisignc2g1.pem
  SHA1: 67:82:AA:E0:ED:EE:E2:1A:58:39:D3:C0:CD:14:68:0A:4F:60:14:2A
  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:E
Alias name: verisignc2g2.pem
  SHA1: B3:EA:C4:47:76:C9:C8:1C:EA:F2:9D:95:B6:CC:A0:08:1B:67:EC:9D
```

```
  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:A
Alias name: verisignc2g3.pem
  SHA1: 61:EF:43:D7:7F:CA:D4:61:51:BC:98:E0:C3:59:12:AF:9F:EB:63:11
  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:B
Alias name: verisignc2g6.pem
  SHA1: 40:B3:31:A0:E9:BF:E8:55:BC:39:93:CA:70:4F:4E:C2:51:D4:1D:8F
  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:F
Alias name: verisignc3g1.pem
  SHA1: A1:DB:63:93:91:6F:17:E4:18:55:09:40:04:15:C7:02:40:B0:AE:6B
  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:14:03:0
Alias name: verisignc3g2.pem
  SHA1: 85:37:1C:A6:E5:50:14:3D:CE:28:03:47:1B:DE:3A:09:E8:F8:77:0F
  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:A8:88:63:8
Alias name: verisignc3g3.pem
  SHA1: 13:2D:0D:45:53:4B:69:97:CD:B2:D5:C3:39:E2:55:76:60:9B:5C:C6
  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:72:4
Alias name: verisignc3g4.pem
  SHA1: 22:D5:D8:DF:8F:02:31:D1:8D:F7:9D:B7:CF:8A:2D:64:C9:3F:6C:3A
  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:7
Alias name: verisignc3g5.pem
  SHA1: 4E:B6:D5:78:49:9B:1C:CF:5F:58:1E:AD:56:BE:3D:9B:67:44:A5:E5
  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:D
Alias name: verisignc4g2.pem
  SHA1: 0B:77:BE:BB:CB:7A:A2:47:05:DE:CC:0F:BD:6A:02:FC:7A:BD:9B:52
  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:C7:01:19:6
Alias name: verisignc4g3.pem
  SHA1: C8:EC:8C:87:92:69:CB:4B:AB:39:E9:8D:7E:57:67:F3:14:95:73:9D
  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:96:05:61:79:15:0
Alias name: verisignclass1ca
  SHA1: CE:6A:64:A3:09:E4:2F:BB:D9:85:1C:45:3E:64:09:EA:E8:7D:60:F1
  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:B8:84:2
Alias name: verisignclass1g2ca
  SHA1: 27:3E:E1:24:57:FD:C4:F9:0C:55:E8:2B:56:16:7F:62:F5:32:E5:47
```

```
  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:7
Alias name: verisignclass1g3ca
  SHA1: 20:42:85:DC:F7:EB:76:41:95:57:8E:13:6B:D4:B7:D1:E9:8E:46:A5
  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:6
Alias name: verisignclass2g2ca
  SHA1: B3:EA:C4:47:76:C9:C8:1C:EA:F2:9D:95:B6:CC:A0:08:1B:67:EC:9D
  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:A
Alias name: verisignclass2g3ca
  SHA1: 61:EF:43:D7:7F:CA:D4:61:51:BC:98:E0:C3:59:12:AF:9F:EB:63:11
  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:B
Alias name: verisignclass3ca
  SHA1: A1:DB:63:93:91:6F:17:E4:18:55:09:40:04:15:C7:02:40:B0:AE:6B
  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:14:03:0
Alias name: verisignclass3g2ca
  SHA1: 85:37:1C:A6:E5:50:14:3D:CE:28:03:47:1B:DE:3A:09:E8:F8:77:0F
  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:A8:88:63:8
Alias name: verisignclass3g3ca
  SHA1: 13:2D:0D:45:53:4B:69:97:CD:B2:D5:C3:39:E2:55:76:60:9B:5C:C6
  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:72:4
Alias name: verisignclass3g4ca
  SHA1: 22:D5:D8:DF:8F:02:31:D1:8D:F7:9D:B7:CF:8A:2D:64:C9:3F:6C:3A
  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:7
Alias name: verisignclass3g5ca
  SHA1: 4E:B6:D5:78:49:9B:1C:CF:5F:58:1E:AD:56:BE:3D:9B:67:44:A5:E5
  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:D
Alias name: verisignclass3publicprimarycertificationauthorityg4
  SHA1: 22:D5:D8:DF:8F:02:31:D1:8D:F7:9D:B7:CF:8A:2D:64:C9:3F:6C:3A
  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:7
Alias name: verisignclass3publicprimarycertificationauthorityg5
  SHA1: 4E:B6:D5:78:49:9B:1C:CF:5F:58:1E:AD:56:BE:3D:9B:67:44:A5:E5
  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:D
Alias name: verisignroot.pem
  SHA1: 36:79:CA:35:66:87:72:30:4D:30:A5:FB:87:3B:0F:A7:7B:B7:0D:54
```

```
  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:3
Alias name: verisigntsaca
  SHA1: 20:CE:B1:F0:F5:1C:0E:19:A9:F3:8D:B1:AA:8E:03:8C:AA:7A:C7:01
  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:9
Alias name: verisignuniversalrootca
  SHA1: 36:79:CA:35:66:87:72:30:4D:30:A5:FB:87:3B:0F:A7:7B:B7:0D:54
  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:3
Alias name: verisignuniversalrootcertificationauthority
  SHA1: 36:79:CA:35:66:87:72:30:4D:30:A5:FB:87:3B:0F:A7:7B:B7:0D:54
  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:3
Alias name: xrampglobalca
  SHA1: B8:01:86:D1:EB:9C:86:A5:41:04:CF:30:54:F3:4C:52:B7:E5:58:C6
  SHA256:
 CE:CD:DC:90:50:99:D8:DA:DF:C5:B1:D2:09:B7:37:CB:E2:C1:8C:FB:2C:10:C0:FF:0B:CF:0D:32:86:FC:1A:A
Alias name: xrampglobalcaroot
  SHA1: B8:01:86:D1:EB:9C:86:A5:41:04:CF:30:54:F3:4C:52:B7:E5:58:C6
  SHA256:
 CE:CD:DC:90:50:99:D8:DA:DF:C5:B1:D2:09:B7:37:CB:E2:C1:8C:FB:2C:10:C0:FF:0B:CF:0D:32:86:FC:1A:A
```

## 使用 AWS WAF 來保護 API Gateway APIs 中的 REST API

AWS WAF 是一種 Web 應用程式防火牆，可協助保護 Web 應用程式和 APIs免受攻擊。可讓您設定一組稱為 web 存取控制清單 (web ACL) 的規則，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 AWS WAF 運作方式。

您可以使用 AWS WAF 保護您的 API Gateway REST API 免受常見的 Web 入侵，例如 SQL Injection 和跨網站指令碼 (XSS) 攻擊。這些可能會影響 API 可用性和效能、危及安全性，或耗用過多的資源。例如，您可以建立規則，允許或封鎖來自指定 IP 地址、來自 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 64 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 是您防範 Web 入侵的第一道防線。在 API 上啟用 AWS WAF 時，會在其他存取控制功能之前評估 AWS WAF 規則，例如資源政策、IAM 政策、Lambda 授權方和 Amazon Cognito 授權方。例如，如果 AWS WAF 封鎖來自資源政策允許之 CIDR 區塊的存取，則 AWS WAF 優先考慮優先順序，不會評估資源政策。

若要 AWS WAF 為您的 API 啟用 ，您需要執行下列動作：

1. 使用 AWS WAF 主控台、 AWS SDK 或 CLI 建立 Web ACL，其中包含所需的 AWS WAF 受管規則和您自己的自訂規則組合。如需詳細資訊，請參閱 AWS WAF入門和 Web 存取控制清單 (Web ACL)。

   > **Important**
   >
   > API Gateway 需要區域應用程式的 AWS WAFV2 Web ACL 或 AWS WAF Classic 區域性 Web ACL。

2. 將 AWS WAF Web ACL 與 API 階段建立關聯。您可以使用 AWS WAF 主控台、 AWS SDK、CLI 或使用 API Gateway 主控台來執行此操作。

## 使用 API Gateway 主控台將 AWS WAF Web ACL 與 API Gateway API 階段建立關聯

若要使用 API Gateway 主控台將 AWS WAF Web ACL 與現有 API Gateway API 階段建立關聯，請使用下列步驟：

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 選擇現有的 API 或建立新的 API。
3. 在主導覽窗格中，選擇階段，然後選擇一個階段。
4. 在階段詳細資訊區段中，選擇編輯。
5. 在 Web 應用程式防火牆 (AWS WAF) 下，選取您的 Web ACL。

   如果您使用的是 AWS WAFV2 AWS WAFV2 ，請選取區域應用程式的 Web ACL。Web ACL 及其使用的任何其他 AWS WAFV2 資源必須位於與您的 API 相同的區域。

   如果您使用的是 AWS WAF Classic 區域性，請選取區域 Web ACL。

6. 選擇儲存變更。

## 使用 將 AWS WAF Web ACL 與 API Gateway API 階段建立關聯 AWS CLI

下列 associate-web-acl 命令會將區域應用程式的 AWS WAFV2 Web ACL 與現有的 API Gateway API 階段建立關聯：

```
aws wafv2 associate-web-acl \
--web-acl-arn  arn:aws:wafv2:{region}:111122223333:regional/webacl/test-cli/
a1b2c3d4-5678-90ab-cdef-EXAMPLE11111 \
--resource-arn arn:aws:apigateway:{region}::/restapis/4wk1k4onj3/stages/prod
```

下列 associate-web-acl 命令會將 AWS WAF Classic 區域性 Web ACL 與現有的 API Gateway API 階段建立關聯：

```
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 將 AWS WAF Web ACL 與 API 階段建立關聯

若要使用 AWS WAFV2 REST API 將區域應用程式的 AWS WAFV2 Web ACL 與現有的 API Gateway API 階段建立關聯，請使用 AssociateWebACL 命令，如下列範例所示：

```
import boto3

wafv2 = boto3.client('wafv2')

wafv2.associate_web_acl(
    WebACLArn='arn:aws:wafv2:{region}:111122223333:regional/webacl/test/abc6aa3b-
fc33-4841-b3db-0ef3d3825b25',
    ResourceArn='arn:aws:apigateway:{region}::/restapis/4wk1k4onj3/stages/prod'
)
```

若要使用 AWS WAF REST API 將 AWS WAF Classic 區域性 Web ACL 與現有的 API Gateway API 階段建立關聯，請使用 AssociateWebACL 命令，如下列範例所示：

```
import boto3

waf = boto3.client('waf-regional')
```

```
waf.associate_web_acl(
    WebACLId='aabc123a-fb4f-4fc6-becb-2b00831cadcf',
    ResourceArn='arn:aws:apigateway:{region}::/restapis/4wk1k4onj3/stages/prod'
)
```

# 調節對 REST API 的請求，以提高 API Gateway 的輸送量

您可以為您的 API 設定調節和配額，以便防止 API 接收過多請求。調節和配額都依最佳作法來套用，它們都應該視為目標，而非確定的請求上限。

API Gateway 會使用字符儲存貯體演算法將字符計算為請求，進而調節傳送給 API 的請求量。具體而言，API Gateway 會按區域檢查帳戶中所有 API 的速率和爆量請求次數。在字符儲存貯體演算法中，爆量可以實現預先定義的超限，但在某些情況下，其他因素也可能導致超限。

提交的請求量超出穩定狀態請求率和爆量限制時，API Gateway 會開始調節請求量。此時用戶端可能會收到 `429 Too Many Requests` 的錯誤回應。發現這類例外狀況時，用戶端可以採用限制速率的方式來重新提交失敗的請求。

身為 API 開發人員，您可以設定個別 API 階段或方法的目標限制，來改善您帳戶中所有 API 的整體效能。或者，您可以啟用用量計劃，根據指定的請求速率和配額來對用戶端提交的請求量設定調節。

主題

- 如何在 API Gateway 中套用調節限制設定
- 每個區域的帳戶層級調節
- 在用量計劃中設定 API 層級和階段層級調節目標
- 設定階段層級限流目標
- 在用量計劃中設定方法層級調節目標

## 如何在 API Gateway 中套用調節限制設定

設定 API 的限流與配額設定之前，先了解 API 的限流相關設定類型，以及 API Gateway 如何套用這些設定，會很有用。

Amazon API Gateway 提供四種基本類型的調節相關設定：

- AWS 限流限制會套用至區域中的所有帳戶和用戶端。這些限制設定的存在是為防止 API 和帳戶接收的請求過多。這些限制由 設定， AWS 客戶無法變更。

- 帳戶型限制會套用至指定區域內某帳戶的所有 API。帳戶層級速率限制可按請求提高。使用較短逾時值和較小酬載的 API 可擁有更高的上限。如需請求提高區域內帳戶層級的調節限制，請與 AWS 支援中心聯絡。如需詳細資訊，請參閱配額和重要說明。請注意，這些限制不能高於限 AWS 流限制。
- API 型限制、階段型調節限制則會套用到某特定階段的 API 方法層級。您可以為所有方法做相同設定，或為每個方法做不同的調節設定。請注意，這些限制不能高於限 AWS 流限制。
- 用戶端型調節限制會套用至用戶端，這類用戶端採用與用量計劃關聯的 API 金鑰做為用戶端識別符。請注意，這些限制不能高於帳戶型限制。

API Gateway 會以如下順序套用您的限流相關設定：

1. 您在用量計劃中為 API 階段設定的用戶端型或方法型調節限制
2. 您為 API 階段設定的方法型限流
3. 每個區域的帳戶層級調節
4. AWS 區域調節

## 每個區域的帳戶層級調節

預設情況下，API Gateway 會按區域限制 AWS 帳戶內所有 API 的每秒穩定狀態請求量 (RPS)。它還會按區域限制 AWS 帳戶內所有 API 的爆量 (即儲存貯體大小上限)。在 API Gateway 中，爆量限制代表 API Gateway 傳回 `429 Too Many Requests` 錯誤回應前可實現的並行請求提交數上限。如需有關調節配額的詳細資訊，請參閱 配額和重要說明。

## 在用量計劃中設定 API 層級和階段層級調節目標

在用量計劃中，您可以在 API 或階段層級為所有方法設定方法型限流。您可以指定限流速率，也就是將權杖新增至權杖儲存貯體的速率 (以每秒請求數計算)。您也可以指定限流暴量，也就是權杖儲存貯體的容量。

您可以使用 AWS CLI、 SDKs 和 AWS 管理主控台 來建立用量計劃。如需有關如何建立用量計畫的詳細資訊，請參閱???。

## 設定階段層級限流目標

您可以使用 AWS CLI、 SDKs 和 AWS 管理主控台 來建立階段層級限流目標。

如需如何使用 AWS 管理主控台 建立階段層級限流目標的詳細資訊，請參閱 ???。如需如何使用 AWS CLI 建立階段層級限流目標的詳細資訊，請參閱 create-stage。

### 在用量計劃中設定方法層級調節目標

您可以在 Usage Plans (用量計劃) 中在方法層級設定其他調節目標，如 建立用量計劃 中的程序所示。若要在 API Gateway 主控台中設定這些限制，請在設定方法調節 設定中指定 `Resource=<resource>`、`Method=<method>`。例如，對於 PetStore 範例，您可能會指定 `Resource=/pets`，`Method=GET`。

## API Gateway 中的私有 REST API

私有 API 是只能從 Amazon VPC 內呼叫的 REST API。VPC 會使用界面 VPC 端點 (也就是您在 VPC 中建立的端點網路介面) 存取您的 API。介面端點採用 AWS PrivateLink技術，可讓您使用私有 IP 地址來私下存取 AWS 服務。

您也可以使用 Direct Connect 建立從內部部署網路到 Amazon VPC 的連線，然後透過該連線存取您的私有 API。在所有情況下，進出您私有 API 的流量都會使用安全連線，並與公有網際網路隔離。流量不會離開 Amazon 網路。

### 私有 API 的最佳實務

我們建議您在建立私有 API 時使用下列最佳實務：

- 使用單一 VPC 端點存取多個私有 API。這會使您可能需要的 VPC 端點數量減少。
- 為您的 VPC 端點與 API 建立關聯。這會建立 Route 53 別名 DNS 記錄，並簡化調用私有 API 的程序。
- 為您的 VPC 開啟私有 DNS。當您開啟 VPC 的私有 DNS 時，可以在 VPC 內調用 API，而不需傳遞 `Host` 或 `x-apigw-api-id` 標頭。

  如果開啟私有 DNS，則無法存取公有 API 的預設端點。若要存取公有 API 的預設端點，您可以關閉私有 DNS，為 VPC 中的每個私有 API 建立私有託管區域，然後在 Route 53 中佈建所需的記錄。這可讓您的私有 API 進行解析，同時仍然可以從 VPC 調用公有預設端點。如需詳細資訊，請參閱建立私有託管區域。
- 限制對特定 VPC 或 VPC 端點的私有 API 存取權。將 `aws:SourceVpc` 或 `aws:SourceVpce` 條件新增至 API 的資源政策以限制存取。
- 對於最安全的資料周邊，您可以建立 VPC 端點政策。這會控制可調用私有 API 的 VPC 端點存取權。

## 私有 API 的考量事項

下列考量事項可能會影響您使用私有 API：

- 僅支援 REST API。
- 您不能將私有 API 轉換為邊緣最佳化 API。
- 私有 API 僅支援 TLS 1.2。不支援舊版 TLS。
- 如果您使用 HTTP/2 通訊協定提出請求，則會強制該請求使用 HTTP/1.1 通訊協定。
- 您無法將私有 API 的 IP 位址類型設定為僅允許 IPv4 位址調用您的私有 API。僅支援雙堆疊。如需詳細資訊，請參閱the section called “API Gateway 中 REST API 的 IP 位址類型”。
- 若要使用私有 API 來傳送流量，您可以使用 Amazon VPC 支援的所有 IP 位址類型。您可以在 VPC 端點上配置設定值，以傳送雙堆疊和 IPv6 流量。您無法使用 API Gateway 修改此項目。如需詳細資訊，請參閱新增 VPC 的 IPv6 支援。
- 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《 AWS PrivateLink 指南》中的使用介面 VPC 端點存取 AWS 服務。如需將 API Gateway 與共用 VPC 和共用子網路搭配使用的詳細資訊，請參閱《AWS PrivateLink 指南》中的共用子網路。

## 私有 API 的後續步驟

若要了解如何建立私有 API 和建立 VPC 端點關聯，請參閱 the section called “建立私有 API”。若要遵循您在 中建立相依性的教學課程， CloudFormation 以及在 中建立私有 API AWS 管理主控台，請參閱 the section called “教學：建立私有 REST API”。

## 建立私有 API

建立私有 API 之前，會先建立 API Gateway 的 VPC 端點。接著，您會建立私有 API，並為其附加資源政策。或者，您可以為 VPC 端點與私有 API 建立關聯，以簡化您調用 API 的方式。最後，部署 API。

下列程序說明如何完成該作業。您可以使用 AWS 管理主控台 AWS CLI 或 AWS SDK 建立私有 REST API。

先決條件

若要遵循這些步驟，您必須擁有已完整設定的 VPC。如需了解如何建立 VPC，請參閱《Amazon VPC 使用者指南》中的僅建立 VPC。要在建立 VPC 時遵循所有建議步驟，請啟用私有 DNS。如此一來，您就可以在 VPC 中調用 API，而不必傳遞主機或 `x-apigw-api-id` 標頭。

若要啟用私有 DNS，VPC 的 `enableDnsSupport` 和 `enableDnsHostnames` 屬性必須設為 `true`。如需詳細資訊，請參閱 VPC 中的 DNS 支援與更新 VPC 的 DNS 支援。

步驟 1：在 VPC 中為 API Gateway 建立 VPC 端點

下列程序顯示如何為 API Gateway 建立 VPC 端點。若要為 API Gateway 建立 VPC 端點，請為 AWS 區域 建立私有 API 的 指定`execute-api`網域。用於執行 API 的 API Gateway 元件服務的是 `execute-api` 網域。

當您為 API Gateway 建立 VPC 端點時，可以指定 DNS 設定。如果關閉私有 DNS，您只能使用公有 DNS 存取您的 API。如需詳細資訊，請參閱the section called “問題：我無法從 API Gateway VPC 端點連線至我的公有 API”。

AWS 管理主控台

為 API Gateway 建立介面 VPC 端點

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/vpc/：// 開啟 Amazon VPC 主控台。
2. 在導覽窗格中的虛擬私有雲端下，選擇端點。
3. 選擇建立端點。
4. (選用) 針對名稱標籤輸入名稱，以協助識別 VPC 端點。
5. 在 Service category (服務類別) 中，選擇 AWS services。
6. 在服務下方搜尋列中輸入 **`execute-api`**。然後，在您要建立 API AWS 區域 的 中選擇 API Gateway 服務端點。服務名稱應該看起來像 `com.amazonaws.us-east-1.execute-api`，類型應該看起來像介面。
7. 針對 VPC，選擇要在其中建立端點的 VPC。
8. (選用) 若要關閉啟用私有 DNS 名稱，請選擇其他設定，然後清除啟用私有 DNS 名稱。
9. 對於子網路，請選擇要建立端點網路介面的可用區域。為了提升 API 可用性，請選擇多個子網路。
10. 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。
11. 對於政策，執行下列其中一項操作：
    - 如果您尚未建立私有 API，或不想要設定自訂 VPC 端點政策，請選擇完整存取。

- 如果您已建立私有 API，並想要設定自訂 VPC 端點政策，則可以輸入自訂 VPC 端點政策。如需詳細資訊，請參閱the section called “使用私有 API 的 VPC 端點政策”。

您可以在建立 VPC 端點後更新 VPC 端點政策。如需詳細資訊，請參閱更新 VPC 端點政策。

12. 選擇建立端點。
13. 複製產生的 VPC 端點 ID，因為您可能會在後續步驟中使用該識別碼。

AWS CLI

以下 create-vpc-endpoint 命令會建立 VPC 端點：

```
aws ec2 create-vpc-endpoint \
    --vpc-id vpc-1a2b3c4d \
    --vpc-endpoint-type Interface \
    --service-name com.amazonaws.us-east-1.execute-api \
    --subnet-ids subnet-7b16de0c \
    --security-group-id sg-1a2b3c4d
```

複製產生的 VPC 端點 ID，因為您可能會在後續步驟中使用該識別碼。

步驟 2：建立私有 API

建立 VPC 端點後，您會建立私有 REST API。下列程序顯示如何建立私有 API。

AWS 管理主控台

建立私有 API

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 選擇 Create API (建立 API)。
3. 在 REST API 下方，選擇 Build (組建)。
4. 針對名稱，輸入名稱。
5. 在描述，請輸入描述。
6. 針對 API 端點類型，選取私有。
7. (選用) 對於 VPC 端點 ID，請輸入 VPC 端點 ID。

如果您為 VPC 端點 ID 與私有 API 建立關聯，您可以從 VPC 內調用 API，而不必覆寫 `Host` 標頭或傳遞 `x-apigw-api-id header`。如需詳細資訊，請參閱 the section called “(選用) 關聯或取消關聯 VPC 端點與私有 REST API”。

8. 對於 IP 位址類型，請選擇 Dualstack。
9. 選擇建立 API。

完成先前的步驟後，您可以遵循 the section called “開始使用 REST API 主控台” 中的指示來設定此 API 的方法和整合，但無法部署 API。若要部署您的 API，請遵循步驟 3，並將資源政策連接至您的 API。

AWS CLI

以下 create-rest-api 命令會建立私有 API：

```
aws apigateway create-rest-api \
        --name 'Simple PetStore (AWS CLI, Private)' \
        --description 'Simple private PetStore API' \
        --region us-west-2 \
        --endpoint-configuration '{ "types": ["PRIVATE"], "ipAddressType":
 "dualstack" }'
```

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"
        ],
        "ipAddressType": "dualstack"
    },
    "id": "0qzs2sy7bh",
    "name": "Simple PetStore (AWS CLI, Private)"
}
```

完成先前的步驟後，您可以遵循 the section called “教學課程：使用 AWS SDKs或 建立 REST API AWS CLI” 中的指示來設定此 API 的方法和整合，但無法部署 API。若要部署您的 API，請遵循步驟 3，並將資源政策連接至您的 API。

SDK JavaScript v3

下列範例示範如何使用適用於 JavaScript v3 的 AWS SDK 建立私有 API：

```
import {APIGatewayClient, CreateRestApiCommand} from "@aws-sdk/client-api-gateway";
const apig = new APIGatewayClient({region:"us-east-1"});

const input = { // CreateRestApiRequest
  name: "Simple PetStore (JavaScript v3 SDK, private)", // required
  description: "Demo private API created using the AWS SDK for JavaScript v3",
  version: "0.00.001",
  endpointConfiguration: { // EndpointConfiguration
    types: [ "PRIVATE"],
  },
};

export const handler = async (event) => {
const command = new CreateRestApiCommand(input);
try {
  const result = await apig.send(command);
  console.log(result);
} catch (err){
  console.error(err)
 }
};
```

成功的呼叫會傳回類似如下的輸出：

```
{
  apiKeySource: 'HEADER',
  createdDate: 2024-04-03T17:56:36.000Z,
  description: 'Demo private API created using the AWS SDK for JavaScript v3',
  disableExecuteApiEndpoint: false,
  endpointConfiguration: { types: [ 'PRIVATE' ] },
  id: 'abcd1234',
  name: 'Simple PetStore (JavaScript v3 SDK, private)',
  rootResourceId: 'efg567',
  version: '0.00.001'
}
```

完成先前的步驟後，您可以遵循 the section called “教學課程：使用 AWS SDKs或 建立 REST API AWS CLI” 中的指示來設定此 API 的方法和整合，但無法部署 API。若要部署您的 API，請遵循步驟 3，並將資源政策連接至您的 API。

Python SDK

下列範例示範如何使用適用於 Python 的 AWS SDK 建立私有 API：

```
import json
import boto3
import logging

logger = logging.getLogger()
apig = boto3.client('apigateway')

def lambda_handler(event, context):
    try:
      result = apig.create_rest_api(
      name='Simple PetStore (Python SDK, private)',
      description='Demo private API created using the AWS SDK for Python',
      version='0.00.001',
      endpointConfiguration={
          'types': [
             'PRIVATE',
          ],
      },
      )
    except botocore.exceptions.ClientError as error:
            logger.exception("Couldn't create private API %s.", error)
            raise
    attribute=["id", "name", "description", "createdDate", "version",
 "apiKeySource", "endpointConfiguration"]
    filtered_data ={key:result[key] for key in attribute}
    result = json.dumps(filtered_data, default=str, sort_keys='true')
    return result
```

成功的呼叫會傳回類似如下的輸出：

```
"{\"apiKeySource\": \"HEADER\", \"createdDate\": \"2024-04-03 17:27:05+00:00\",
 \"description\": \"Demo private API created using the AWS SDK for \",
 \"endpointConfiguration\": {\"types\": [\"PRIVATE\"]}, \"id\": \"abcd1234\", \"name
\": \"Simple PetStore (Python SDK, private)\", \"version\": \"0.00.001\"}"
```

完成先前的步驟後，您可以遵循 the section called “教學課程：使用 AWS SDKs或 建立 REST API AWS CLI” 中的指示來設定此 API 的方法和整合，但無法部署 API。若要部署您的 API，請遵循步驟 3，並將資源政策連接至您的 API。

步驟 3：為私有 API 設定資源政策

所有 VPC 都無法存取您目前的私有 API。使用資源政策，授予私有 API 的 VPC 和 VPC 端點存取權。您可以授予任何 AWS 帳戶中 VPC 端點的存取權。

您的資源政策應包含 `aws:SourceVpc` 或 `aws:SourceVpce` 條件來限制存取。建議您識別特定 VPC 和 VPC 端點，不要建立允許存取所有 VPC 和 VPC 端點的資源政策。

下列程序顯示如何將資源政策連接至您的 API。

AWS 管理主控台

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 選擇 REST API。
3. 在主導覽窗格中，選擇資源政策。
4. 選擇建立政策。
5. 選擇選取來源，然後選擇來源 VPC。
6. 將 `{{vpcID}}` (包括大括號) 取代為您的 VPC ID。
7. 選擇儲存變更。

AWS CLI

以下 update-rest-api 命令會將資源政策附加至現有的 API：

```
aws apigateway update-rest-api \
    --rest-api-id a1b2c3 \
    --patch-operations op=replace,path=/
policy,value='"{\"jsonEscapedPolicyDocument\"}"'
```

您也可以控制哪些資源可以存取您的 VPC 端點。若要控制哪些資源具有 VPC 端點的存取權，可將 VPC 端點政策連接到您的 VPC 端點。如需詳細資訊，請參閱the section called “使用私有 API 的 VPC 端點政策”。

(選用) 關聯或取消關聯 VPC 端點與私有 REST API

建立 VPC 端點與私有 API 的關聯後，API Gateway 會產生新的 Route 53 別名 DNS 記錄。您可以使用此記錄調用私有 API，就像您可以執行公有 API，而無需覆寫 `Host` 標頭或傳遞 `x-apigw-api-id` 標頭。

產生的基本 URL 格式如下：

```
https://{rest-api-id}-{vpce-id}.execute-api.{region}.amazonaws.com/{stage}
```

Associate a VPC endpoint (AWS 管理主控台)

您可以在建立 VPC 端點時或建立 VPC 端點之後，為其建立與私有 API 的關聯。下列程序顯示如何為 VPC 端點與先前建立的 API 建立關聯。

為其他 VPC 端點與私有 API 建立關聯

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 選擇您的私有 API。
3. 在主導覽窗格中，選擇資源政策。
4. 編輯您的資源政策以允許來自其他 VPC 端點的呼叫。
5. 在主導覽窗格中，選擇 API 設定。
6. 在 API 詳細資訊區段中，選擇編輯。
7. 對於 VPC 端點 ID，選取其他 VPC 端點 ID。
8. 選擇儲存。
9. 重新部署 API 以使變更生效。

Dissociate a VPC endpoint (AWS 管理主控台)

將 VPC 端點與私有 REST API 取消關聯

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 選擇您的私有 API。
3. 在主導覽窗格中，選擇資源政策。
4. 編輯您的資源政策，以移除所提及要與私有 API 取消關聯的 VPC 端點。

5. 在主導覽窗格中，選擇 API 設定。
6. 在 API 詳細資訊區段中，選擇編輯。
7. 對於 VPC 端點 ID，選擇 X 以將 VPC 端點取消關聯。
8. 選擇儲存。
9. 重新部署 API 以使變更生效。

### Associate a VPC endpoint (AWS CLI)

以下 create-rest-api 命令會在建立 API 時與 VPC 端點建立關聯：

```
aws apigateway create-rest-api \
    --name Petstore \
    --endpoint-configuration '{ "types": ["PRIVATE"], "vpcEndpointIds" :
 ["vpce-0212a4ababd5b8c3e", "vpce-0393a628149c867ee"] }' \
    --region us-west-2
```

輸出將如下所示：

```
{
    "apiKeySource": "HEADER",
    "endpointConfiguration": {
        "types": [
            "PRIVATE"
        ],
        "vpcEndpointIds": [
            "vpce-0212a4ababd5b8c3e",
            "vpce-0393a628149c867ee"
        ]
    },
    "id": "u67n3ov968",
    "createdDate": 1565718256,
    "name": "Petstore"
}
```

以下 update-rest-api 命令會將 VPC 端點與您已建立的 API 建立關聯：

```
aws apigateway update-rest-api \
    --rest-api-id u67n3ov968 \
```

```
    --patch-operations "op='add',path='/endpointConfiguration/
vpcEndpointIds',value='vpce-01d622316a7df47f9'" \
    --region us-west-2
```

輸出將如下所示：

```
{
    "name": "Petstore",
    "apiKeySource": "1565718256",
    "tags": {},
    "createdDate": 1565718256,
    "endpointConfiguration": {
        "vpcEndpointIds": [
            "vpce-0212a4ababd5b8c3e",
            "vpce-0393a628149c867ee",
            "vpce-01d622316a7df47f9"
        ],
        "types": [
            "PRIVATE"
        ]
    },
    "id": "u67n3ov968"
}
```

重新部署 API 以使變更生效。

Disassociate a VPC endpoint (AWS CLI)

以下 update-rest-api 命令會取消 VPC 端點與私有 API 的關聯：

```
aws apigateway update-rest-api \
    --rest-api-id u67n3ov968 \
    --patch-operations "op='remove',path='/endpointConfiguration/
vpcEndpointIds',value='vpce-0393a628149c867ee'" \
    --region us-west-2
```

輸出將如下所示：

```
{
    "name": "Petstore",
    "apiKeySource": "1565718256",
    "tags": {},
```

```
    "createdDate": 1565718256,
    "endpointConfiguration": {
        "vpcEndpointIds": [
            "vpce-0212a4ababd5b8c3e",
            "vpce-01d622316a7df47f9"
        ],
        "types": [
            "PRIVATE"
        ]
    },
    "id": "u67n3ov968"
}
```

重新部署 API 以使變更生效。

步驟 4：部署私有 API

若要部署您的 API，請建立 API 部署，並為其建立與階段的關聯。下列程序顯示如何部署您的私有 API。

AWS 管理主控台

部署私有 API

1. 選擇您的 API。
2. 選擇部署 API。
3. 針對階段，選取新階段。
4. 針對階段名稱，輸入階段名稱。
5. 在描述，請輸入描述。
6. 選擇部署。

AWS CLI

以下 create-deployment 命令會部署私有 API：

```
aws apigateway create-deployment --rest-api-id a1b2c3 \
  --stage-name test \
  --stage-description 'Private API test stage' \
  --description 'First deployment'
```

針對私有 API 進行疑難排解

以下建議說明如何針對在建立私有 API 時可能會遇到的錯誤和問題，進行疑難排解。

問題：我無法從 API Gateway VPC 端點連線至我的公有 API

建立 VPC 時，您可以設定 DNS 設定。建議您為 VPC 開啟私有 DNS。如果選擇關閉私有 DNS，您將僅能透過公有 DNS 存取您的 API。

如果啟用私有 DNS，則您無法從 VPC 端點存取公有 API Gateway API 的預設端點。您可以使用自訂網域名稱存取 API。

如果您建立區域性自訂網域名稱，請使用 A 型別名記錄，如果建立邊緣最佳化的自訂網域名稱，則記錄類型沒有限制。您可以在啟用私有 DNS 的情況下存取這些公有 API。如需詳細資訊，請參閱問題：我從 API Gateway VPC 端點連線至我的公有 API。

問題：我的 API 傳回 **{"Message":"User: anonymous is not authorized to perform: execute-api:Invoke on resource: arn:aws:execute-api:us-east-1:********/****/****/"}**

在您的資源政策中，如果您將主體設定為 AWS 主體，如下所示：

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": [
                    "arn:aws:iam::111122223333:role/developer",
                    "arn:aws:iam::111122223333:role/Admin"
                ]
            },
            "Action": "execute-api:Invoke",
            "Resource": [
                "execute-api:/stage/GET/pets"
            ]
        }
    ]
}
```

您必須對 API 中的每個方法使用 AWS_IAM 授權，否則您的 API 會傳回先前的錯誤訊息。如需有關如何為方法開啟 AWS_IAM 授權的詳細資訊，請參閱 the section called “方法”。

問題：我無法判斷我的 VPC 端點是否與我的 API 相關聯

如果您為 VPC 端點與您的私有 API 建立關聯，或取消它們的關聯，則需要重新部署 API。由於 DNS 傳播之故，更新操作可能需要幾分鐘時間才能完成。在這段期間，您的 API 將可供使用，但新產生的 DNS URL 的 DNS 傳播可能仍在進行中。如果幾分鐘後，您的新 URL 未在 DNS 中解析，建議您重新部署 API。

## API Gateway 中的 API 私有 API 的自訂網域名稱

您也可以針對私有 API 建立自訂網域名稱。使用私有自訂網域名稱，為 API 呼叫者提供更簡單、更直覺的 URL。使用私有自訂網域名稱，您可以降低複雜性、在 TLS 交握期間設定安全措施，以及使用 AWS Certificate Manager (ACM) 控制網域名稱的憑證生命週期。如需詳細資訊，請參閱the section called “保護自訂網域名稱的憑證私有金鑰”。

私有 API 的自訂網域名稱在多個帳戶中不必是唯一的。您可以在帳戶 111122223333 和帳戶 555555555555 中建立 example.private.com，只要 ACM 憑證包含網域名稱即可。若要識別私有自訂網域名稱，請使用私有自訂網域名稱 ARN。此識別碼對私有自訂網域名稱是唯一的。

當您在 API Gateway 中建立私有自訂網域名稱時，您是 API 提供者。您可以使用 AWS 帳戶 API Gateway 或 AWS Resource Access Manager () 將您的私有自訂網域名稱提供給其他AWS RAM。

當您調用私有自訂網域名稱時，您是 API 取用者。您可以從自己的 AWS 帳戶 或另一個 使用私有自訂網域名稱 AWS 帳戶。

當您取用私有自訂網域名稱時，您會在 VPC 端點與私有自訂網域名稱之間建立網域名稱存取關聯。有了網域名稱存取關聯，API 取用者就可以在與公有網際網路隔離時，調用您的私有自訂網域名稱。如需詳細資訊，請參閱the section called “API 提供者和 API 取用者”。

保護自訂網域名稱的憑證私有金鑰

當您使用 ACM 請求 SSL/TLS 憑證來建立私有 API 的自訂網域名稱時，ACM 會產生公有/私有金鑰對。當您匯入憑證時，會產生金鑰對。公有金鑰會成為憑證的一部分。為了安全地存放私有金鑰，ACM 會使用 AWS KMS名為 KMS 金鑰的另一個金鑰，別名為 aws/acm。 AWS KMS 使用此金鑰來加密憑證的私有金鑰。如需詳細資訊，請參閱《 AWS Certificate Manager 使用者指南》中的 AWS Certificate Manager中的資料保護。

API Gateway AWS 使用 TLS Connection Manager，這是只能存取的服務 AWS 服務，用於保護和使用憑證的私有金鑰。當您使用 ACM 憑證建立 API Gateway 自訂網域名稱時，API Gateway 會將您的

憑證與 AWS TLS Connection Manager 建立關聯。我們會針對您的 AWS 受 AWS KMS 管金鑰在 中建立授予，藉此達成此目的。此授權允許 TLS Connection Manager 使用 AWS KMS 解密憑證的私有金鑰。TLS Connection Manager 使用憑證和解密的 (純文字) 私有金鑰，建立與 API Gateway 服務用戶端的安全連線 (SSL/TLS 工作階段)。當憑證與 API Gateway 服務解除關聯時，授權便會失效。如需詳細資訊，請參閱《AWS Key Management Service 開發人員指南》中的授權。

如需詳細資訊，請參閱the section called “靜態加密”。

私有自訂網域名稱的考量事項

下列考量事項可能會影響您使用私有自訂網域名稱：

- API Gateway 需要約 15 分鐘來佈建您的私有自訂網域名稱。
- 如果您更新 ACM 憑證，API Gateway 需要約 15 分鐘才能完成更新。在此期間，您的網域名稱處於 UPDATING 狀態，且您仍然可以存取它。
- 若要調用私有自訂網域名稱，您必須建立網域名稱存取關聯。建立網域名稱存取關聯後，大約需要 15 分鐘才能準備就緒。
- 私有自訂網域名稱 ARN 包含 *account-id* 和 *domain-name-id*。當您建立網域名稱時，API Gateway 會使用 ARN 格式 `arn:`*partition*`:apigateway:`*region*`::/domainnames/domain-name`。當您存取私有自訂網域名稱時，請使用 ARN 格式 `arn:`*partition*`:apigateway:`*region*`:`*account-id*`:/domainnames/domain-name+`*domain-name-id*。

  您可能需要修改 IAM 許可，以便在建立私有網域名稱之後允許存取。
- 您無法從相同的 VPC 端點調用具有相同名稱的私有自訂網域名稱。例如，若您想要調用 `arn:aws:apigateway:us-west-2:111122223333:/domainnames/private.example.com+abcd1234` 和 `arn:aws:apigateway:us-west-2:111122223333:/domainnames/private.example.com+xyz000`，請為每個私有自訂網域名稱與不同的 VPC 端點建立關聯。
- 支援萬用字元憑證，例如 `*.private.example.com` 的憑證。
- 不支援萬用字元自訂網域名稱。
- 僅支援具 2048 位元金鑰長度的 RSA 憑證，以及具 256 位元和 384 位元金鑰長度的 ECDSA 憑證。
- 您無法將私有 API 的 IP 位址類型設定為僅允許 IPv4 位址調用您的私有 API。僅支援雙堆疊。如需詳細資訊，請參閱the section called “API Gateway 中 REST API 的 IP 位址類型”。
- 若要使用私有 API 來傳送流量，您可以使用 Amazon VPC 支援的所有 IP 位址類型。您可以在 VPC 端點上配置設定值，以傳送雙堆疊和 IPv6 流量。您無法使用 API Gateway 修改此項目。如需詳細資訊，請參閱新增 VPC 的 IPv6 支援。

- 不支援多層基本路徑映射，例如將私有 API 映射至 `/developers/feature`，但您可以使用路由規則來建立多層路徑條件。如需詳細資訊，請參閱the section called “在 API Gateway 中透過您的自訂網域名稱將流量傳送至 API”。
- 您無法為私有自訂網域名稱設定最低 TLS 版本。所有私有自訂網域名稱都有 `TLS-1-2` 的安全政策。
- 您可以使用 VPC 端點政策來控制對私有自訂網域名稱的存取。如需詳細資訊，請參閱 the section called “使用私有 API 的 VPC 端點政策” 中的範例 4 和 5。
- 您必須為私有 API 和私有自訂網域名稱建立不同的資源政策。若要調用私有自訂網域名稱，API 取用者需要存取私有自訂網域名稱資源政策、私有 API 資源政策，以及私有 API 上的任何 VPC 端點政策或授權。

### 對於搭配其他 API Gateway 資源使用私有自訂網域名稱的考量事項

下列考量可能會影響您如何搭配其他 API Gateway 資源使用私有自訂網域名稱：

- 您無法將流量從私有自訂網域名稱傳送至公有 API。
- 當私有 API 映射至私有自訂網域名稱時，您無法變更 API 的端點類型。
- 您無法將公有自訂網域名稱遷移至私有自訂網域名稱。
- 如果您具有用於存取公有自訂網域名稱的 VPC 端點，請勿使用該端點來建立與私有自訂網域名稱的網域名稱存取關聯。

### 私有自訂網域名稱與公有自訂網域名稱之間的差異

以下說明私有和公有自訂網域名稱之間的差異：

- 私有自訂網域名稱在多個帳戶中不必是唯一的。
- 私有網域名稱有網域名稱 ID。此 ID 可作為唯一識別私有自訂網域名稱的方式，而公有自訂網域名稱不會產生此 ID。
- 當您使用 AWS CLI 更新或刪除私有自訂網域名稱時，必須提供網域名稱 ID。如果您具有名為 `example.com` 的私有自訂網域名稱，以及名為 `example.com` 的公有自訂網域名稱，而且您沒有提供網域名稱 ID，則 API Gateway 會修改或刪除您的公有自訂網域名稱。

### 對於私有 API 的自訂網域名稱的後續步驟

如需 API 提供者和 API 取用者的任務相關資訊，請參閱 the section called “API 提供者和 API 取用者”。

如需建立可在自己的私有自訂網域名稱的說明 AWS 帳戶，請參閱 the section called “教學：建立和調用私有 API 的自訂網域名稱”。

如需提供另一個私有自訂網域名稱 AWS 帳戶 存取權的說明，請參閱 the section called “API 提供者：使用 共用您的私有自訂網域名稱 AWS RAM”。如需將您的 VPC 端點與另一個私有自訂網域名稱建立關聯的說明 AWS 帳戶，請參閱 the section called “API 取用者：將 VPC 端點與和您分享的私有自訂網域名稱建立關聯”。

私有 API 之自訂網域名稱的 API 提供者和 API 取用者的任務

當您建立私有自訂網域名稱時，您是 API 提供者。當您調用私有自訂網域名稱時，您是 API 取用者。您可以從自己的 AWS 帳戶 或另一個 使用私有自訂網域名稱 AWS 帳戶。

下一節主要說明：API 提供者和 API 取用者使用私有自訂網域名稱所需的任務。如果您想要在自己的 中叫用私有自訂網域名稱 AWS 帳戶，您同時是 API 供應商和 API 取用者。如果您想要在另一個 中叫用私有自訂網域 AWS 帳戶，根據 API 提供者與 中的 API 取用者之間的信任關係 AWS Organizations， AWS RAM 可能會為您完成一些任務。

API 提供者的任務

API 提供者會建立私有 API，並將其映射至自訂網域名稱。

API 提供者會管理兩個資源政策，以保護其私有自訂網域名稱。第一個政策用於 `execute-api` 服務，並控制哪些 VPC 端點可以調用您的私有自訂網域名稱。這在私有自訂網域名稱組態中，稱為 `policy`。

第二個政策適用於 Amazon API Gateway Management 服務，並控制其他 中的哪些 VPC 端點 AWS 帳戶 可與私有自訂網域名稱形成網域名稱存取關聯。VPC 端點需要與私有自訂網域名稱建立網域名稱存取關聯，才能調用該名稱。這在私有自訂網域名稱組態中就是 `managementPolicy`。您可以使用 AWS RAM 或 API Gateway 來更新此政策。如果您不打算允許其他 中的 VPC 端點 AWS 帳戶 叫用您的自訂網域名稱，則不會編輯 `managementPolicy`。

如果您是 API 提供者，您必須執行下列操作：

1. 建立私有 API。
2. 更新私有 API `policy` 以授予 VPC 端點存取私有 API 的權限。
3. 建立私有自訂網域名稱。
4. 更新您的私有自訂網域名稱的 `policy`，以授予 VPC 端點存取您的私有自訂網域名稱的權限。
5. 建立基本路徑映射或路由規則，將流量從私有 API 傳送至私有自訂網域名稱。如需詳細資訊，請參閱the section called “在 API Gateway 中透過您的自訂網域名稱將流量傳送至 API”。

如果您想要允許其他 中的 API 取用者 AWS 帳戶 存取您的私有自訂網域名稱，請執行下列動作：

1. 更新私有自訂網域名稱的 `managementPolicy`，以允許其他帳戶中的 API 取用者為其 VPC 端點與您的私有自訂網域名稱建立關聯。您可以使用下列方式執行這項操作：

   AWS RAM

   使用 時 AWS RAM，如果 API 提供者和 API 取用者位於相同的組織中 AWS Organizations，則會自動接受提供者和取用者之間的資源共用。否則，您應該等到 API 取用者接受資源分享。我們建議您使用 AWS RAM 來共用私有自訂網域名稱。

   API Gateway

   透過 API Gateway，僅支援 AWS CLI 。您必須利用修補程式的運作來更新私有自訂網域名稱，並為 `managementPolicy` 提供您自己的政策文件。
2. 更新私有自訂網域名稱的 `policy`，以及對其映射的任何私有 API，以授予對 API 取用者 VPC 端點的存取權。

如需如何將 API 提供給另一個 API 的說明 AWS 帳戶，請參閱 the section called “API 提供者：使用 共用您的私有自訂網域名稱 AWS RAM”。

API 取用者的任務

API 取用者會為其 VPC 端點與網域名稱 ARN 建立關聯，如此才能調用私有自訂網域名稱。API 取用者不需要建立 API Gateway API。

如果您是 API 取用者，請執行下列操作：

1. 在 Amazon VPC 中建立具有私有 DNS 的 VPC 端點。
2. （選用 - AWS RAM 如果使用）在資源共用的 AWS RAM 12 小時內接受 中的私有自訂網域資源共用。如果您和 API 提供者屬於同一個組織，便會自動接受資源分享。
3. 取得私有自訂網域名稱 ARN。由於私有自訂網域名稱 URL 不是唯一，您可以使用私有自訂網域名稱 ARN 來形成 VPC 端點與私有自訂網域名稱之間的存取關聯。您可以使用 AWS RAM 來擷取私有自訂網域名稱 ARN。
4. 建立私有自訂網域 ARN 與 API Gateway 中的 VPC 端點之間的關聯。這會在 VPC 端點和私有自訂網域名稱之間建立安全連線。流量不會離開 Amazon 網路。
5. 等待 API 提供者授予 VPC 端點對私有自訂網域名稱的存取權，以及映射到私有自訂網域名稱的任何私有 API。如果您同時是 API 提供者和 API 取用者，您可以授予自己的 VPC 端點調用存取權。
6. 建立 Route 53 私有託管區域和 Route 53 記錄，以解析 Route 53 中的私有自訂網域名稱。

如需如何在另一個 API 中使用 API 的說明 AWS 帳戶，請參閱 the section called “API 取用者：將 VPC 端點與和您分享的私有自訂網域名稱建立關聯”。

教學：建立和調用私有 API 的自訂網域名稱

在本教學課程中，您會建立一個可以在自己的帳戶中的 VPC 中調用的私有自訂網域名稱。為完成此操作，您既是 API 提供者，也是 API 取用者。您需要現有的私有 API 和 VPC 端點才能完成本教學課程。如果您有用於存取公有自訂網域名稱的 VPC 端點，請勿將其用於本教學課程或建立任何網域名稱存取關聯。

步驟 1：建立私有自訂網域名稱

您可以透過指定網域名稱、ACM 憑證和 `execute-api` 服務的政策來建立私有自訂網域名稱，以控制哪些 VPC 端點可以調用該名稱。

AWS 管理主控台

建立私有自訂網域名稱

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 在主要導覽窗格中，選擇自訂網域名稱。
3. 選擇新增網域名稱。
4. 在 Domain name (網域名稱) 中，輸入網域名稱。

   您的 ACM 憑證必須包含此網域名稱，但網域名稱不必是唯一的。
5. 選取私有。
6. 針對路由模式，選擇僅 API 映射。
7. 針對 ACM 憑證，選取憑證。
8. 選擇新增網域名稱。

API Gateway 會使用 deny 所有資源政策佈建網域名稱。這是 `execute-api` 服務的資源政策。您需要更新此資源政策以授予對 VPC 端點的存取權，如此才能調用您的私有自訂網域名稱。

更新您的資源政策

1. 選擇資源政策標籤，然後選擇編輯資源政策。
2. 在程式碼編輯器中輸入下列資源政策。將 VPC 端點 *`vpce-abcd1234efg`* 取代為您自己的 VPC 端點 ID。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": "*",
            "Action": "execute-api:Invoke",
            "Resource": [
                "execute-api:/*"
            ]
        },
        {
            "Effect": "Deny",
            "Principal": "*",
            "Action": "execute-api:Invoke",
            "Resource": [
                "execute-api:/*"
            ],
            "Condition" : {
                "StringNotEquals": {
                    "aws:SourceVpce": "vpce-abcd1234"
                }
            }
        }
    ]
}
```

3. 選擇儲存變更。

AWS CLI

當您使用 建立私有自訂網域名稱時 AWS CLI，您會為execute-api服務提供資源政策，以授予對 VPC 端點的存取權，以使用 --policy file://policy.json 參數叫用私有自訂網域名稱。之後您可以修改此政策。

在此範例中，您將從檔案載入參數，以附加下列資源政策作為 policy。複製此檔案並儲存為 policy.json。此政策僅允許從 VPC 端點 *vpce-abcd1234efg* 傳入至私有自訂網域名稱的流量：

```
{
```

```
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": "*",
            "Action": "execute-api:Invoke",
            "Resource": [
                "execute-api:/*"
            ]
        },
        {
            "Effect": "Deny",
            "Principal": "*",
            "Action": "execute-api:Invoke",
            "Resource": [
                "execute-api:/*"
            ],
            "Condition" : {
                "StringNotEquals": {
                    "aws:SourceVpce": "vpce-abcd1234"
                }
            }
        }
    ]
}
```

下列 create-domain-name 命令會建立私有自訂網域名稱：

```
aws apigateway create-domain-name \
    --domain-name 'private.example.com' \
    --certificate-arn 'arn:aws:acm:us-west-2:111122223333:certificate/
a1b2c3d4-5678-90ab-cdef' \
    --security-policy 'TLS_1_2' \
    --endpoint-configuration '{"types":["PRIVATE"]}' \
    --policy file://policy.json
```

輸出如下所示：

```
{
    "domainName": "private.example.com",
    "domainNameId": "abcd1234",
```

```
    "domainNameArn": "arn:aws:apigateway:us-west-2:111122223333:/domainnames/
private.example.com+abcd1234",
    "certificateArn": "arn:aws:acm:us-west-2:111122223333:certificate/
a1b2c3d4-5678-90ab-cdef",
    "certificateUploadDate": "2024-09-10T10:31:20-07:00",
    "endpointConfiguration": {
        "types": [
            "PRIVATE"
        ]
    },
    "domainNameStatus": "AVAILABLE",
    "securityPolicy": "TLS_1_2",
    "routingMode" : "API_MAPPING_ONLY",
    "policy": "..."
}
```

步驟 2：建立基本路徑映射，將私有 API 映射至私有自訂網域名稱

建立私有自訂網域名稱後，您可以將私有 API 映射至該名稱。基本路徑映射可以透過私有自訂網域名稱與相關聯基本路徑的組合來存取 API。建議您將單一私有自訂網域用作多個私有 API 的主機名稱。

所有 API 提供者都需要建立基本路徑映射，即使您不打算調用自己的 API。您也需要授予對 VPC 端點的存取權，如此才能調用映射至私有自訂網域名稱的任何私有 API。

AWS 管理主控台

建立基本路徑映射

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 在主要導覽窗格中，選擇自訂網域名稱。
3. 選擇私有自訂網域名稱。
4. 在 API 映射標籤上，選擇設定映射。
5. 選擇 Add new mapping (新增映射)。
6. 輸入 API、Stage (階段)，以及選擇性地輸入 Path (路徑)。
7. 選擇儲存。

AWS CLI

下列 create-base-path-mapping 命令會在私有 API 和私有自訂網域名稱之間建立映射：

```
aws apigateway create-base-path-mapping \
    --domain-name-id abcd1234 \
    --domain-name 'private.example.com' \
    --rest-api-id a1b2c3 \
    --stage prod \
    --base-path v1
```

輸出將如下所示：

```
{
    "basePath": "v1",
    "restApiId": "a1b2c3",
    "stage": "prod"
}
```

如要以更有彈性的方式將流量路由至 API，您可以將路由模式變更為 ROUTING_RULE_ONLY 或 ROUTING_RULE_THEN_API_MAPPING，並建立路由規則。如需詳細資訊，請參閱the section called “在 API Gateway 中透過您的自訂網域名稱將流量傳送至 API”。

> **Note**
>
> 如果您希望其他人 AWS 帳戶 叫用您的私有自訂網域名稱，在完成本教學課程後，請遵循中的步驟the section called “API 提供者：使用 共用您的私有自訂網域名稱 AWS RAM”。

步驟 3：在您的自訂網域名稱和 VPC 端點之間建立網域名稱存取關聯

接著，您可以在私有自訂網域名稱和 VPC 端點之間建立網域名稱存取關聯。您的 VPC 端點會利用網域名稱存取關聯，在與公有網際網路隔離時調用您的私有自訂網域名稱。

AWS 管理主控台

建立網域名稱存取關聯

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 在主要導覽窗格中，選擇自訂網域名稱。
3. 選擇私有自訂網域名稱。
4. 在資源共用標籤中，針對網域名稱存取關聯選擇建立網域名稱存取關聯。

5. 針對網域名稱 ARN，選取您的網域名稱。
6. 針對 VPC 端點 ID，選取您在步驟 1 中提供存取權的 VPC 端點 ID。
7. 選擇網域名稱存取關聯。

您也可以使用主控台的網域名稱存取關聯頁面建立網域名稱存取關聯。

AWS CLI

下列 `create-domain-name-access-association` 命令會在您的私有自訂網域名稱和 VPC 端點之間建立網域名稱存取關聯。

```
aws apigateway create-domain-name-access-association \
    --domain-name-arn arn:aws:apigateway:us-west-2:111122223333:/domainnames/
private.example.com+abcd1234 \
    --access-association-source vpce-abcd1234efg \
    --access-association-source-type VPCE \
    --region us-west-2
```

輸出將如下所示：

```
{
    "domainNameAccessAssociationARN": "arn:aws:apigateway:us-west-2:111122223333:/
domainnameaccessassociations/domainname/private.example.com+abcd1234/vpcesource/
vpce-abcd1234efg",
    "accessAssociationSource": "vpce-abcd1234efg",
    "accessAssociationSourceType": "VPCE",
    "domainNameARN" : "arn:aws:apigateway:us-west-2:111122223333:/domainnames/
private.example.com+abcd1234"
}
```

建立您的網域名稱存取關聯後，大約需要 15 分鐘才能準備就緒。您在等待時，可以繼續執行下列步驟。

步驟 4：建立 Route 53 託管區域

在您更新資源政策，並為私有自訂網域名稱與 VPC 端點建立關聯後，可以在 Route 53 中建立私有託管區域以解析自訂網域名稱。託管區域是一種容器，其中包含的資訊說明您可以如何在一或多個 VPC 中路由某個網域的流量，而不用將資源公開至網際網路。如需詳細資訊，請參閱使用私有託管區域。

AWS 管理主控台

若要使用 AWS 管理主控台，請參閱《Amazon Route 53 開發人員指南》中的建立私有託管區域。

對於名稱，請使用私有自訂網域名稱的名稱。對於 VPC ID，請使用包含您在先前步驟中使用的 VPC 端點的 VPC。

AWS CLI

下列 create-hosted-zone 命令會建立私有託管區域：

```
aws route53 create-hosted-zone --name private.example.com \
    --caller-reference 2014-04-01-18:47 \
    --hosted-zone-config Comment="command-line version",PrivateZone=true \
    --vpc VPCRegion=us-west-2,VPCId=vpc-abcd1234
```

輸出內容包含託管區域 ID。您可以在下列步驟中使用託管區域 ID。

步驟 5：建立 Route 53 DNS 記錄

建立託管區域後，建立記錄來解析私有自訂網域名稱。使用您在上一個步驟中建立的託管區域 ID。在此範例中，您會建立 A 記錄類型。如果您針對 VPC 端點使用 IPv6，請建立 AAAA 記錄類型。如果您為 VPC 端點使用雙堆疊，請同時建立 AAAA 和 A 記錄類型。

AWS 管理主控台

若要使用 AWS 管理主控台，請參閱使用您的網域名稱將流量路由到 Amazon API Gateway API。

使用快速建立並開啟別名。對於端點，請使用 VPC 端點 DNS 名稱。

AWS CLI

若要設定您的 DNS 記錄，將私有自訂網域名稱映射至指定託管區域 ID 的主機名稱，請先建立 JSON 檔案，其中包含用於設定私有網域名稱之 DNS 記錄的組態。

以下 `setup-dns-record.json` 顯示如何建立 DNS A 記錄，將私有自訂網域名稱映射至其私有主機名稱。您提供 VPC DNS ID 的 `DNSName`，以及您在上一個步驟中建立的託管區域 ID。

```
{
  "Changes": [
    {
      "Action": "UPSERT",
      "ResourceRecordSet": {
        "Name": "private.example.com",
```

```
        "Type": "A",
        "AliasTarget": {
          "DNSName": "vpce-abcd1234.execute-api.us-west-2.vpce.amazonaws.com",
          "HostedZoneId": "Z2OJLYMUO9EFXC",
          "EvaluateTargetHealth": false
        }
      }
    }
  ]
}
```

下列 change-resource-record-sets 命令會為您的私有自訂網域名稱建立 DNS 記錄：

```
aws route53 change-resource-record-sets \
    --hosted-zone-id ZABCDEFG1234 \
    --change-batch file://path/to/your/setup-dns-record.json
```

將 hosted-zone-id 取代為您帳戶中所設定之 DNS 記錄的 Route 53 託管區域 ID。change-batch 參數值指向 JSON 檔案。

如果您不打算調用自己的私有自訂網域名稱，可在確認私有自訂網域名稱正常運作後刪除這些資源。

步驟 6：調用您的私有自訂網域名稱

您現在可以在自己的 AWS 帳戶中調用私有自訂網域名稱。在您的 VPC 中，使用以下 curl 命令來存取您的私有自訂網域名稱。

```
curl https://private.example.com/v1
```

如需有關調用私有 API 的其他方式的詳細資訊，請參閱 the section called “使用自訂網域名稱調用私有 API”。

步驟 7：清除

若要避免不必要的成本，請刪除 VPC 端點與私有自訂網域名稱之間的關聯，然後刪除私有自訂網域名稱。

AWS 管理主控台

刪除網域名稱存取關聯

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。

2. 在主要導覽窗格中，選擇網域名稱存取關聯。
3. 選取您的網域名稱存取關聯，然後選擇刪除。
4. 確認您的選擇，然後選擇刪除。

刪除網域名稱存取關聯後，您可以刪除私有自訂網域名稱。

刪除您的私有自訂網域名稱

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 在主要導覽窗格中，選擇自訂網域名稱。
3. 選擇您的自訂網域名稱。
4. 選擇 刪除。
5. 確認您的選擇，然後選擇刪除。

如有必要，您也可以刪除 VPC 端點。如需詳細資訊，請參閱建立介面端點。

AWS CLI

清理方式

1. 下列 `delete-access-association` 命令會刪除網域名稱存取關聯：

```
aws apigateway delete-domain-name-access-association \
    --domain-name-access-association-arn 'arn:aws:apigateway:us-
west-2:111122223333:/domainnameaccessassociations/domainname/private.example.com
+abcd1234/vpcesource/vpce-abcd1234efg' \
    --region us-west-2
```

2. 下列 `delete-domain-name` 命令會刪除您的私有自訂網域名稱。此命令也會移除所有基本路徑映射。

```
aws apigateway delete-domain-name \
    --domain-name test.private.com \
    --domain-name-id abcd1234
```

如有必要，您也可以刪除 VPC 端點。如需詳細資訊，請參閱建立介面端點。

### 最佳實務

我們建議您在建立私有自訂網域名稱時使用下列最佳實務：

- 使用基本路徑映射或路由規則，將流量從一個私有自訂網域名稱傳送至多個私有 API。
- 當 VPC 端點不再需要存取私有自訂網域名稱時，請刪除關聯。此外，從私有自訂網域的 `execute-api` 服務的 `policy` 中移除 VPC 端點。
- 為每個 VPC 端點設定至少兩個可用區域。
- 停用預設端點。建議您停用預設端點，以允許您的 API 取用者只從自訂網域名稱呼叫您的 API。如需詳細資訊，請參閱the section called “停用預設端點”。
- 建議您在設定私有自訂網域名稱時，佈建 Route 53 私有託管區域和 A 型記錄。如果您不打算調用自己的私有自訂網域名稱，您可以稍後刪除這些資源。

## 使用跨帳戶私有自訂網域名稱

本節說明如何使用跨帳戶私有自訂網域名稱。您可以將私有自訂網域名稱提供給另一個 AWS 帳戶 ，並使用另一個 AWS 帳戶 來叫用私有自訂網域名稱。

您可以使用 AWS 帳戶 AWS Resource Access Manager 或 API Gateway 將您的私有自訂網域名稱分享給另一個 。 AWS Resource Access Manager (AWS RAM) 可協助您在組織或組織單位 (OUs) 之間 AWS 帳戶 和內部安全地共用資源。如需詳細資訊，請參閱什麼是 AWS Resource Access Manager 。

如需如何使用 與另一個 共用私有自訂網域名稱的說明 AWS 帳戶 AWS RAM，請參閱 the section called “API 提供者：使用 共用您的私有自訂網域名稱 AWS RAM”。

如需如何使用 AWS 帳戶 API Gateway 與另一個 共用私有自訂網域名稱的說明，請參閱 the section called “API 提供者：使用 API Gateway 共用您的私有自訂網域名稱 AWS CLI”。

如需如何在另一個 中使用私有自訂網域名稱的指示 AWS 帳戶，請參閱 the section called “API 取用者：將 VPC 端點與和您分享的私有自訂網域名稱建立關聯”。

### 使用跨帳戶私有自訂網域名稱的最佳實務

我們建議使用以下跨帳戶私有自訂網域名稱的最佳實務：

- 使用 AWS RAM 來共用私有自訂網域名稱。使用 時 AWS RAM，您可以降低營運開銷，而且不需要`managementPolicy`為 Amazon API Gateway Management 服務建立 。
- 當您列出私有自訂網域名稱或網域名稱存取關聯時，請使用 `resource-owner` 參數。使用 `resource-owner` 參數僅列出您或其他 AWS 帳戶擁有的資源。

下列範例顯示如何取得您擁有的所有網域名稱存取關聯：

```
aws apigateway get-domain-name-access-associations --resource-owner SELF
```

使用 `--resource-owner OTHER_ACCOUNTS` 列出其他帳戶已與您的私有自訂網域名稱形成的所有網域名稱存取關聯。

### API 提供者：使用 共用您的私有自訂網域名稱 AWS RAM

您可以在其他 AWS 帳戶 存取中為 API 取用者提供私有自訂網域名稱。在本節中，您將了解如何使用共用私有自訂網域名稱， AWS RAM 以及如何控制對私有自訂網域名稱的存取。

#### 共用私有自訂網域名稱的考量事項

下列考量可能會影響您使用 AWS RAM提供私有自訂網域名稱之存取權的方式。若要了解如何共用私有自訂網域名稱而不使用 AWS RAM，請參閱 the section called “API 提供者：使用 API Gateway 共用您的私有自訂網域名稱 AWS CLI”。

- 私有自訂網域名稱會在 AWS 區域 層級共用。私有自訂網域名稱和 VPC 端點必須位於相同的 AWS 區域中。
- 您可以與多個主體使用一個資源分享，並在建立資源分享後，為其新增更多主體。建議您盡可能重複使用資源分享。
- 您始終需要授予 API 取用者的 VPC 端點存取權，才能調用您的私有自訂網域名稱，以及任何與其映射的私有 API。
- 如果 API 取用者和 API 提供者位於使用 的相同組織中 AWS Organizations，則會自動接受資源共用。您仍然需要使用 AWS RAM建立資源分享。
- 如果 API 取用者和 API 提供者位於使用 的相同組織中， AWS Organizations 且組織內的資源共用已啟用，則與您共用的組織中的任何主體都會自動獲得資源共用的存取權。不需要邀請函的，可以略過資源分享。
- 如果 API 取用者在 12 小時內不接受資源分享，API 提供者必須再次分享資源。
- 在您建立資源共享後， 會 AWS RAM 更新私有自訂網域名稱`managementPolicy`的 Amazon API Gateway Management 服務的 ，以防止在沒有明確存取的情況下`allow`存取主體。如需詳細資訊，請參閱《IAM 使用者指南》中的判斷帳戶中的請求獲得允許或遭到拒絕。

  更新的 `managementPolicy` 如下所示：

JSON

```
{
    "Version":"2012-10-17",
    "Id": "abcd1234-1234-abcd-abcd-1234abcdefg",
    "Statement": [
        {
            "Sid":
 "APIGatewayPrivateDomainNameManagementPolicyDefaultPermission-org",
            "Effect": "Allow",
            "Principal": "*",
            "Action": "apigateway:CreateAccessAssociation",
            "Resource": "arn:aws:apigateway:us-west-2:111122223333:/
domainnames/private.example.com+abcd1234",
            "Condition": {
                "StringEquals": {
                    "aws:PrincipalOrgID": "o-1234abcd"
                },
                "StringNotEquals": {
                    "aws:PrincipalAccount": "111122223333"
                }
            }
        }
    ]
}
```

AWS RAM 已透過新增下列項目，防止主體沒有明確allow存取來建立與私有自訂網域名稱的存取關聯：

```
"StringNotEquals": {
    "aws:PrincipalAccount": "111122223333"
}
```

您仍然可以在建立私有自訂網域名稱 AWS 帳戶 的 中使用主體來建立網域名稱存取關聯。

### 允許其他帳戶與您的私有自訂網域名稱建立網域名稱存取關聯

首先，您將存取權授予另一個 ， AWS 帳戶 以建立與私有自訂網域名稱的網域名稱存取關聯。

AWS 管理主控台

若要使用 AWS 管理主控台，請參閱AWS RAM 《 使用者指南》中的在 中建立資源共享 AWS RAM。

針對選取資源類型，選擇 API Gateway 私有自訂網域。

AWS CLI

下列 create-resource-share 會為您的私有自訂網域名稱建立資源分享。資源和委託人可能需要幾分鐘的時間才能完成關聯。對於主體，請提供帳戶 ID 或 Organizations ID，例如 `arn:aws:organizations::123456789012:organization/o-1234abcd`。您可以為資源分享提供多個主體。

```
aws ram create-resource-share \
    --region us-west-2 \
    --name privateCustomDomain-resource-share \
    --permission-arns arn:aws:ram::aws:permission/
APIGatewayPrivateDomainNameManagementPolicyDefaultPermission \
    --resource-arns arn:aws:apigateway:us-west-2:111122223333:/domainnames/
private.example.com+abcd1234 \
    --principals 222222222222
```

在您提供對另一個 的存取權之後 AWS 帳戶，該帳戶中的 API 取用者必須在其 VPC 端點與您的私有自訂網域名稱之間建立網域名稱存取關聯。您無法為其建立網域名稱存取關聯。如需詳細資訊，請參閱the section called “為您的 VPC 端點與分享的私有自訂網域名稱建立關聯”。

允許其他帳戶調用您的私有自訂網域名稱

接下來，您會授予 API 取用者之 VPC 端點的存取權，以調用私有自訂網域名稱和任何與其映射的私有 API。

AWS 管理主控台

允許其他帳戶中的 VPC 端點調用您的私有自訂網域名稱

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 在主要導覽窗格中，選擇自訂網域名稱。
3. 選擇您與其他 共用的私有自訂網域名稱 AWS 帳戶。
4. 在資源政策標籤上，選擇編輯資源政策。

5. 將 API 取用者的 VPC 端點 ID 新增至您的資源政策。

   您可以在私有自訂網域名稱的網域詳細資訊頁面上的資源分享標籤的網域名稱存取關聯區段中，找到 API 取用者的 VPC 端點 ID。

6. 選擇儲存變更。

AWS CLI

下列的 `execute-api` 服務 `policy`，允許從 VPC 端點 *vpce-abcd1234efg* 和 *vpce-xyz000abc* 二者傳入私有自訂網域名稱的流量。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": "*",
            "Action": "execute-api:Invoke",
            "Resource": [
                "execute-api:/*"
            ]
        },
        {
            "Effect": "Deny",
            "Principal": "*",
            "Action": "execute-api:Invoke",
            "Resource": [
                "execute-api:/*"
            ],
            "Condition" : {
                "StringNotEquals": {
                    "aws:SourceVpce": [
                    "vpce-abcd1234",
                    "vpce-xyzz0000"
                    ]
                }
            }
        }
    ]
}
```

下列 update-domain-name 命令使用修補程式作業來更新私有自訂網域名稱的 policy：

```
aws apigateway update-domain-name
    --domain-name private.example.com \
    --domain-name-id abcd1234 \
    --patch-operations op=replace,path=/policy,value='"{\"Version\": \"2012-10-17\",
       \"Statement\": [{\"Effect\": \"Allow\",\"Principal\": \"*\",\"Action\":
 \"execute-api:Invoke\",\"Resource\":[\"execute-api:/*\"]},{\"Effect\": \"Deny\",
\"Principal\": \"*\",\"Action\": \"execute-api:Invoke\",\"Resource\":[\"execute-
api:/*\"],\"Condition\":{\"StringNotEquals\":[\"vpce-abcd1234efg\", \"vpce-xyz000abc
\"]}}}]}"
```

API 提供者：使用 停止共用私有自訂網域名稱 AWS RAM

若要停止分享您的私有自訂網域名稱，請先取消資源分享的關聯，以停止 API 取用者建立更多網域名稱存取關聯。然後，您拒絕網域名稱存取關聯，並從 execute-api 服務的 policy 中移除 API 取用者的 VPC 端點。API 取用者接著可以刪除其網域名稱存取關聯。

停止分享您的私有自訂網域名稱

首先，您使用 停止資源共享 AWS RAM。

AWS 管理主控台

若要使用 AWS 管理主控台，請參閱在 中更新資源共用 AWS RAM。

AWS CLI

下列 disassociate-resource-share 會取消您的私有自訂網域名稱的資源分享的關聯。

```
aws ram disassociate-resource-share \
    --region us-west-2 \
    --resource-arns arn:aws:apigateway:us-west-2:111122223333:/domainnames/
private.example.com+abcd1234 \
    --principals 222222222222
```

拒絕網域名稱存取關聯

停止使用 共用資源後 AWS RAM，您會拒絕另一個帳戶中的 VPC 端點與私有自訂網域名稱之間的網域名稱存取關聯。

> **Note**
>
> 您無法拒絕自己帳戶中的網域名稱存取關聯。若要停止資源分享，請刪除網域名稱存取關聯。如需詳細資訊，請參閱刪除網域名稱存取關聯。

當您拒絕網域名稱與 VPC 端點的存取關聯時，如果 API 取用者嘗試呼叫您的私有自訂網域名稱，API Gateway 會拒絕呼叫，並傳回 403 狀態碼。

AWS 管理主控台

拒絕網域名稱存取關聯

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 在主要導覽窗格中，選擇自訂網域名稱。
3. 選擇您與其他 共用的私有自訂網域名稱 AWS 帳戶。
4. 在資源分享上，選擇您要拒絕的網域名稱存取關聯。
5. 選擇拒絕關聯。
6. 確認您的選擇，然後選擇拒絕。

AWS CLI

下列 `reject-domain-name-access-association` 命令會拒絕 VPC 端點與私有自訂網域名稱之間的網域名稱存取關聯：

```
aws apigateway reject-domain-name-access-association \
    --domain-name-access-association-arn arn:aws:apigateway:us-west-2:444455556666:/
domainnameaccessassociations/domainname/private.example.com+abcd1234/vpcesource/
vpce-abcd1234efg \
    --domain-name-arn arn:aws:apigateway:us-west-2:111122223333:/domainnames/
private.example.com+abcd1234
```

拒絕 API 提供者存取，以調用您的私有自訂網域名稱

拒絕網域名稱存取關聯後，您可以從 `execute-api` 服務的 `policy` 中移除 API 取用者 VPC 端點。

AWS 管理主控台

從資源政策中移除 API 取用者的 VPC 端點

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 在主要導覽窗格中，選擇自訂網域名稱。
3. 選擇您與其他 共用的私有自訂網域名稱 AWS 帳戶。
4. 在資源政策標籤上，選擇編輯。
5. 從政策中移除 VPC 端點。
6. 選擇儲存變更。

AWS CLI

下列 update-domain-name 命令使用修補程式作業來更新私有自訂網域名稱之 `execute-api` 服務的 `policy`。此新的 `policy` 會移除 the section called “允許其他帳戶調用您的私有自訂網域名稱” 中新增的其他 VPC 端點 ID：

```
aws apigateway update-domain-name
    --domain-name private.example.com \
    --domain-name-id abcd1234 \
    --patch-operations op=replace,path=/policy,value='"{\"Version\": \"2012-10-17\",
       \"Statement\": [{\"Effect\": \"Allow\",\"Principal\": \"*\",\"Action\":
 \"execute-api:Invoke\",\"Resource\":[\"execute-api:/*\"]},{\"Effect\": \"Deny\",
\"Principal\": \"*\",\"Action\": \"execute-api:Invoke\",\"Resource\":[\"execute-
api:/*\"],\"Condition\":{\"StringNotEquals\":{\"aws:SourceVpce\": \"vpce-abcd1234efg
\"}}}]}"
```

API 取用者接著應該刪除網域名稱存取關聯。您無法為他們刪除存取關聯。如需詳細資訊，請參閱the section called “API 取用者：刪除與私有自訂網域名稱的網域名稱存取關聯”。

API 提供者：使用 API Gateway 共用您的私有自訂網域名稱 AWS CLI

您可以使用 API Gateway 共用私有自訂網域名稱 AWS CLI，但我們建議您使用 AWS RAM 來降低營運開銷。如需如何使用 AWS RAM 來共用私有自訂網域名稱的指示，請參閱 the section called “API 提供者：使用 共用您的私有自訂網域名稱 AWS RAM”。

若要使用 API Gateway 共用私有自訂網域名稱 AWS CLI，您可以授予其他 AWS 帳戶 存取權來建立網域名稱存取關聯，並叫用私有自訂網域名稱。您可以藉由更新 API Gateway Management 服務的

managementPolicy 和私有自訂網域名稱的 execute-api 服務的 policy 來執行此操作。您也需要在資源政策中授予 API 取用者 VPC 端點的存取權，以用於映射到私有自訂網域名稱的任何私有 API。

API 取用者仍需在其 VPC 端點和您的私有自訂網域名稱之間，在其自己的帳戶中建立網域名稱存取關聯。您無法為他們執行此操作。

授予私有自訂網域名稱的存取權

授予私有自訂網域名稱的存取權

1. 若要更新 API Gateway Management 服務的 managementPolicy，您可以建立包含修補程式作業的 JSON 檔案來更新政策。以下內容會將目前的 patch-managementPolicy.json取代managementPolicy為範例政策，授予 AWS 帳戶 111122223333 和 444455556666 的存取權，以建立與私有自訂網域名稱 的網域名稱存取關聯private.example.com。

   ```
   [{
       "op": "replace",
       "path": "/managementPolicy",
       "value": "{\"Version\":\"2012-10-17\",       \"Statement\":[{\"Effect
   \":\"Allow\",\"Principal\":{\"AWS\":[\"arn:aws:iam::111122223333:root
   \", \"arn:aws:iam::444455556666:root\"]},\"Action\":
   \"apigateway:CreateAccessAssociation\",\"Resource\":\"arn:aws:apigateway:us-
   west-2:111122223333:/domainnames/private.example.com+abcd1234\"}]}"
   }]
   ```

   下列 update-domain-name 命令會使用 patch-managementPolicy.json 更新 managementPolicy。

   ```
   aws apigateway update-domain-name \
       --domain-name private.example.com \
       --domain-name-id abcd1234 \
       --patch-operations file://patch-managementPolicy.json
   ```

   授予存取權後，您需要通知 API 取用者，他們可以形成網域名稱存取關聯。如果您使用 AWS RAM， AWS RAM 會為您執行此步驟。

2. 若要更新 execute-api 服務的 policy，您可以建立包含修補程式作業的 JSON 檔案來更新政策。以下 patch-policy.json 會將目前的 policy 取代為範例政策，而此政策會授予兩個 VPC 端點來調用私有自訂網域名稱 private.example.com。

```
[{
    "op": "replace",
    "path": "/policy",
    "value": "{\"Version\": \"2012-10-17\",       \"Statement\": [{\"Effect\":
 \"Allow\",\"Principal\": \"*\",\"Action\": \"execute-api:Invoke\",\"Resource\":
 \"arn:aws:execute-api:us-west-2:111122223333:/domainnames/private.example.com
+abcd1234\"},{\"Effect\": \"Deny\",\"Principal\": \"*\",\"Action\": \"execute-
api:Invoke\",\"Resource\": \"arn:aws:execute-api:us-west-2:111122223333:/
domainnames/private.example.com+abcd1234\",\"Condition\": {\"StringNotEquals\":
 {\"aws:SourceVpce\": [\"vpce-abcd1234\",\"vpce-xyzz0000\"]}}}]}"
}]
```

使用下列 update-domain-name 命令，並使用 patch-policy.json 來更新 policy。

```
aws apigateway update-domain-name \
    --domain-name private.example.com \
    --domain-name-id abcd1234 \
    --patch-operations file://patch-policy.json
```

拒絕對您的私有自訂網域名稱的存取

若要停止分享私有自訂網域名稱，您需要拒絕私有自訂網域名稱與 API 取用者之 VPC 端點之間的網域名稱存取關聯。

拒絕對私有自訂網域名稱的存取

1. 下列 reject-domain-name-access-association 命令會拒絕網域名稱存取關聯。

```
aws apigateway reject-domain-name-access-association \
    --domain-name-access-association-arn arn:aws:apigateway:us-
west-2:444455556666:/domainnameaccessassociations/domainname/private.example.com
+abcd1234/vpcesource/vpce-abcd1234 \
    --domain-name-arn arn:aws:apigateway:us-west-2:111122223333:/domainnames/
private.example.com+abcd1234
```

2. 修改 patch-managementPolicy.json 即可移除 API 提供者帳戶的存取權，以與您的私有自訂網域名稱建立網域名稱存取關聯。下列 patch-managementPolicy.json 會從 managementPolicy 移除一個帳戶：

```
[{
```

```
     "op": "replace",
     "path": "/managementPolicy",
     "value": "{\"Version\":\"2012-10-17\",       \"Statement\":[{\"Effect\":
\"Allow\",\"Principal\":\"*\",\"Action\":\"apigateway:CreateAccessAssociation
\",\"Resource\":\"arn:aws:apigateway:us-west-2:111122223333:/domainnames/
private.example.com+abcd1234\"}]}"
}]
```

下列 update-domain-name 命令使用 patch-managementPolicy.json 更新 managementPolicy。

```
aws apigateway update-domain-name \
    --domain-name private.example.com \
    --domain-name-id abcd1234 \
    --patch-operations file://patch-managementPolicy.json
```

3. 修改 patch-policy.json 以移除用於讓 API 提供者 VPC 端點調用您的私有自訂網域名稱的存取權。以下 patch-policy.json 會從 policy 移除 VPC 端點 ID：

```
[{
    "op": "replace",
    "path": "/policy",
    "value": "{\"Version\":\"2012-10-17\",       \"Statement\":[{\"Effect\":
\"Allow\",\"Principal\":\"*\",\"Action\":\"execute-api:Invoke\",\"Resource\":
\"arn:aws:execute-api:us-west-2:111122223333:/domainnames/private.example.com
+abcd1234\"},{\"Effect\":\"Deny\",\"Principal\":\"*\",\"Action\":\"execute-
api:Invoke\",\"Resource\":\"arn:aws:execute-api:us-west-2:111122223333:/
domainnames/private.example.com+abcd1234\",\"Condition\":{\"StringNotEquals\":
{\"aws:SourceVpce\":\"vpce-abcd1234\"}}}]}"
}]
```

下列 update-domain-name 命令會使用 patch-policy.json 更新 policy。

```
aws apigateway update-domain-name \
    --domain-name private.example.com \
    --domain-name-id abcd1234 \
    --patch-operations file://patch-policy.json
```

此程序中使用的範例政策

下一節說明先前程序中使用的範例政策。

下列範例政策適用於 Amazon API Gateway Management 服務的 managementPolicy。此政策授予 AWS 帳戶 111122223333 和 444455556666 存取權，以建立與私有自訂網域名稱 的網域名稱存取關聯private.example.com。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": [
                    "111122223333",
                    "444455556666"
                ]
            },
            "Action": "apigateway:CreateAccessAssociation",
            "Resource": "arn:aws:apigateway:us-west-2:111122223333:/domainnames/
private.example.com+a1b2c3"
        }
    ]
}
```

下列範例政策是 execute-api 服務的 policy 的政策。此政策會授予 VPC 端點 vpce-abcd1234 和 vpce-xyzz0000 存取權，用以調用私有自訂網域名稱。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": "*",
            "Action": "execute-api:Invoke",
```

```
            "Resource": "arn:aws:execute-api:us-west-2:111122223333:/domainnames/
private.example.com+abcd1234"
        },
        {
            "Effect": "Deny",
            "Principal": "*",
            "Action": "execute-api:Invoke",
            "Resource": "arn:aws:execute-api:us-west-2:111122223333:/domainnames/
private.example.com+abcd1234",
            "Condition": {
                "StringNotEquals": {
                    "aws:SourceVpce": [
                        "vpce-abcd1234",
                        "vpce-xyzz0000"
                    ]
                }
            }
        }
    ]
}
```

API 取用者：將 VPC 端點與和您分享的私有自訂網域名稱建立關聯

下列程序顯示如何在另一個 中使用私有網域名稱 AWS 帳戶。視您與 API 提供者的信任關係而定，AWS RAM 可能會為您完成一些任務。

當您位於 AWS 帳戶 與私有自訂網域名稱不同的 時，您只能將 VPC 端點與私有自訂網域名稱建立關聯並叫用它。您無法檢視私有自訂網域名稱的 `policy` 或其他任何參數。

先決條件

在另一個 中使用私有自訂網域名稱時，需要下列先決條件 AWS 帳戶：

- `execute-api` 服務的 VPC 和 VPC 端點。您的 VPC 必須將 `enableDnsHostnames` 和 `enableDnsSupport` 設為 `true`。
- 建議您為每個 VPC 端點設定至少兩個可用區域。

(選用) 接受私有自訂網域資源分享

如果您的 API 提供者 AWS RAM 用來建立資源共享，您有 12 小時的時間可以接受它。如果您使用 AWS Organizations 與 API 提供者位於相同的組織中，則會自動接受共用。如果您所在的組織已啟用自動分享資源，則會自動與您分享資源。

AWS 管理主控台

若要使用 AWS 管理主控台，請參閱AWS RAM 《 使用者指南》中的接受和拒絕資源共用邀請。

AWS CLI

若要尋找與您分享的所有資源，請使用下列 get-resource-share-invitations 命令：

```
aws ram get-resource-share-invitations \
    --region us-west-2
```

使用產生的資源分享 ARN，接受資源分享邀請。下列 accept-resource-share-invitation 命令接受資源分享。

```
aws ram accept-resource-share-invitation \
    --resource-share-invitation-arn arn:aws:ram:us-west-2:123456789012:resource-
share-invitation/1e3477be-4a95-46b4-bbe0-c4001EXAMPLE \
    --region us-west-2
```

為您的 VPC 端點與分享的私有自訂網域名稱建立關聯

由於私有自訂網域名稱不是唯一的，因此您可以為 VPC 端點與唯一的自訂網域名稱 ARN 建立關聯。建立網域名稱存取關聯後，VPC 端點最多可能需要 15 分鐘才能成功調用您的私有自訂網域名稱。如果您有用於存取公有自訂網域名稱的 VPC 端點，請勿使用該端點建立任何網域名稱存取關聯。

AWS 管理主控台

為您的 VPC 端點與分享的私有自訂網域名稱建立關聯

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 在主要導覽窗格中，選擇網域名稱存取關聯。
3. 選擇建立網域名稱存取關聯。
4. 針對網域名稱 ARN，選取 API 提供者與您共用的網域名稱 ARN。

網域名稱 ARN 可能不會出現在下拉式清單中。您可以使用 AWS RAM 主控台來檢視與您共用的網域名稱，然後複製網域名稱 ARN 並將其輸入此欄位。

5. 針對 VPC 端點 ID，選取您要與之形成網域名稱存取關聯的 VPC 端點 ID。
6. 選擇建立網域名稱存取關聯。

AWS CLI

由於私有自訂網域名稱不是唯一的，因此您可以為 VPC 端點與唯一的自訂網域名稱 ARN 建立關聯。若要尋找網域名稱 ARN，請使用下列其中一個命令。

1. AWS RAM

   下列 list-resources 命令會列出與您分享的資源。API 提供者必須使用 AWS RAM 與您共用其私有自訂網域，才能使用此命令。

   ```
   aws ram list-resources \
       --resource-owner OTHER-ACCOUNTS \
       --region us-west-2
       --resource-type apigateway:Domainnames
   ```

   API Gateway

   下列get-domain-names命令會列出其他 擁有的所有私有自訂網域名稱 AWS 帳戶 ，您可以與其建立網域名稱存取關聯。

   ```
   aws apigateway get-domain-names \
       --resource-owner OTHER_ACCOUNTS \
       --region us-west-2
   ```

2. 擷取 ARN 之後，使用 API Gateway 在您的 VPC 端點與共用私有自訂網域名稱之間建立網域名稱存取關聯。使用下列 create-domain-name-access-association 命令：

   ```
   aws apigateway create-domain-name-access-association \
       --access-association-source-type VPCE \
       --access-association-source 'vpce-1a2b3c4d5e6f1a2b3' \
       --domain-name-arn arn:aws:apigateway:us-west-2:111122223333:/domainnames/
   private.example.com+abcd1234"
   ```

   輸出將如下所示：

```
{
    "domainNameAccessAssociationARN": "arn:aws:apigateway:us-
west-2:444455556666:/domainnameaccessassociations/domainname/private.example.com
+abcd1234/vpcesource/vpce-abcd1234efg",
    "accessAssociationSource": "vpce-1a2b3c4d5e6f1a2b3",
    "accessAssociationSourceType": "VPCE",
    "domainNameARN" : "arn:aws:apigateway:us-west-1:111122223333:/domainnames/
private.example.com+a1b2c3"
}
```

為您的 VPC 端點與私有自訂網域名稱建立關聯後，請確認您的 API 提供者已更新其私有自訂網域名稱的政策，以允許您的 VPC 端點調用其網域名稱。如需詳細資訊，請參閱the section called “允許其他帳戶調用您的私有自訂網域名稱”。

建立 Route 53 託管區域

若要解析私有自訂網域名稱，您需要建立 Route 53 私有託管區域。託管區域是一種容器，其中包含的資訊說明您可以如何在一或多個 VPC 中路由某個網域的流量，而不用將資源公開至網際網路。如需詳細資訊，請參閱使用私有託管區域。

AWS 管理主控台

若要使用 AWS 管理主控台，請參閱《Amazon Route 53 開發人員指南》中的建立私有託管區域。

對於名稱，請使用私有自訂網域名稱的名稱。對於 VPC ID，請使用包含您用於網域名稱存取關聯的 VPC 端點的 VPC。

AWS CLI

下列 create-hosted-zone 命令會建立私有託管區域：

```
aws route53 create-hosted-zone --name private.example.com \
    --caller-reference 2014-04-01-18:47 \
    --hosted-zone-config Comment="command-line version",PrivateZone=true \
    --vpc VPCRegion=us-west-2,VPCId=vpc-abcd1234
```

輸出內容包含託管區域 ID。您可以在下列步驟中使用託管區域 ID。

建立 Route 53 DNS 記錄

建立託管區域後，您可以建立記錄來解析私有自訂網域。在此範例中，您會建立 A 記錄類型。如果您針對 VPC 端點使用 IPv6，請建立 AAAA 記錄類型。如果您為 VPC 端點使用雙堆疊，請同時建立 AAAA 和 A 記錄類型。

AWS 管理主控台

若要使用 AWS 管理主控台，請參閱使用您的網域名稱將流量路由到 Amazon API Gateway API。

使用快速建立並開啟別名。對於端點，請使用 VPC 端點 DNS 名稱。

AWS CLI

若要設定您的 DNS 記錄，將私有自訂網域名稱映射至指定託管區域 ID 的主機名稱，首先請建立 JSON 檔案，其中包含用於設定私有網域名稱之 DNS 記錄的組態。

以下 `setup-dns-record.json` 顯示如何建立 DNS A 記錄，將私有自訂網域名稱映射至其私有主機名稱。您提供 VPC DNS ID 的 `DNSName`，以及您在上一個步驟中建立的託管區域 ID。

```
{
  "Changes": [
    {
      "Action": "UPSERT",
      "ResourceRecordSet": {
        "Name": "private.example.com",
        "Type": "A",
        "AliasTarget": {
          "DNSName": "vpce-abcd1234.execute-api.us-west-2.vpce.amazonaws.com",
          "HostedZoneId": "Z2OJLYMUO9EFXC",
          "EvaluateTargetHealth": false
        }
      }
    }
  ]
}
```

下列 change-resource-record-sets 命令會為私有自訂網域名稱建立 DNS 記錄：

```
aws route53 change-resource-record-sets \
    --hosted-zone-id ZABCDEFG1234 \
    --change-batch file://path/to/your/setup-dns-record.json
```

將 `hosted-zone-id` 取代為您帳戶中所設定之 DNS 記錄的 Route 53 託管區域 ID。`change-batch` 參數值指向 JSON 檔案。

API 取用者的後續步驟

您現在可以在自己的 AWS 帳戶中調用私有 API。在您的 VPC 中，您可以使用下列 curl 命令來存取您的私有自訂網域名稱。

```
curl https://private.example.com/v1
```

如需有關調用私有 API 的其他方式的詳細資訊，請參閱 the section called “使用自訂網域名稱調用私有 API”。

API 取用者：刪除與私有自訂網域名稱的網域名稱存取關聯

如果您是 API 取用者，您可以隨時刪除存取關聯資源。API 提供者無法刪除您的網域名稱存取關聯。

當您不再使用網域名稱存取關聯時，建議您一律刪除該網域名稱存取關聯。

AWS 管理主控台

刪除網域名稱存取關聯

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 在主要導覽窗格中，選擇網域名稱存取關聯。
3. 選取您的網域名稱存取關聯，然後選擇刪除。
4. 確認您的選擇，然後選擇刪除。

AWS CLI

下列 `delete-access-association` 命令會刪除存取關聯：

```
aws apigateway delete-domain-name-access-association \
    --domain-name-access-association-arn 'arn:aws:apigateway:us-
west-2:444455556666:/domainnameaccessassociations/domainname/private.example.com
+abcd1234/vpcesource/vpce-abcd1234efg'
```

使用 建立私有 APIs的自訂網域名稱 CloudFormation

下列範例 CloudFormation 範本會建立私有 API 和私有自訂網域名稱、將私有 API 對應至自訂網域名稱，然後建立網域名稱存取關聯。您需要提供自己的 VPC 端點、網域名稱和憑證 ARN。

下列考量事項可能會影響您使用 CloudFormation 來建立私有自訂網域名稱：

- 您無法使用 拒絕網域名稱存取關聯 CloudFormation。若要拒絕網域名稱存取關聯，請使用 AWS CLI。
- 使用 `AWS::ApiGateway::DomainNameV2` CloudFormation 屬性建立私有自訂網域名稱。
- 使用 `AWS::ApiGateway:BasePathMappingV2` CloudFormation 屬性建立基本路徑映射。

```
AWSTemplateFormatVersion: 2010-09-09
Parameters:
  EndpointID:
    Type: String
    Default: vpce-abcd1234567efg
    Description: A VPC endpoint with enableDnsHostnames and enableDnsSupport set to
 true.
  DomainName:
    Type: String
    Default: private.example.com
    Description: A domain name that you own.
  CertificateArn:
    Type: String
    Default: arn:aws:acm:us-west-2:123456789:certificate/abcd-000-1234-0000-000000abcd
    Description: An ACM certificate that covers the domain name.
Resources:
  PrivateApi:
    Type: 'AWS::ApiGateway::RestApi'
    Properties:
      EndpointConfiguration:
        Types:
          - PRIVATE
        VpcEndpointIds:
          - !Ref EndpointID
      Name: private-api
      Policy:
        Statement:
          - Action: 'execute-api:Invoke'
            Effect: Allow
            Principal: '*'
```

```
            Resource: 'execute-api:/*'
          - Action: 'execute-api:Invoke'
            Condition:
              StringNotEquals:
                'aws:SourceVpce': !Ref EndpointID
            Effect: Deny
            Principal: '*'
            Resource: 'execute-api:/*'
        Version: 2012-10-17
  PrivateApiDeployment:
    Type: 'AWS::ApiGateway::Deployment'
    Properties:
      RestApiId: !Ref PrivateApi
      Description: Private API deployment
    DependsOn:
      - PrivateApiMethod
  PrivateApiStage:
    Type: 'AWS::ApiGateway::Stage'
    Properties:
      RestApiId: !Ref PrivateApi
      DeploymentId: !Ref PrivateApiDeployment
      StageName: prod
  PrivateApiMethod:
    Type: 'AWS::ApiGateway::Method'
    Properties:
      HttpMethod: ANY
      ResourceId: !GetAtt PrivateApi.RootResourceId
      RestApiId: !Ref PrivateApi
      AuthorizationType: NONE
      Integration:
        Type: MOCK
        RequestTemplates:
          application/json: "{\"statusCode\": 200}"
        IntegrationResponses:
          - StatusCode: '200'
      MethodResponses:
        - StatusCode: '200'
  PrivateDomainName:
    Type: AWS::ApiGateway::DomainNameV2
    Properties:
      DomainName: !Ref DomainName
      CertificateArn: !Ref CertificateArn
      EndpointConfiguration:
        Types:
```

```
          - PRIVATE
      SecurityPolicy: TLS_1_2
      Policy:
        Statement:
            - Action: 'execute-api:Invoke'
              Effect: Allow
              Principal: '*'
              Resource: 'execute-api:/*'
            - Action: 'execute-api:Invoke'
              Condition:
                StringNotEquals:
                  'aws:SourceVpce': !Ref EndpointID
              Effect: Deny
              Principal: '*'
              Resource: 'execute-api:/*'
        Version: 2012-10-17
  PrivateBasePathMapping:
    Type: AWS::ApiGateway::BasePathMappingV2
    DependsOn:
      - PrivateApiStage
    Properties:
      BasePath: prod
      DomainNameArn: !GetAtt PrivateDomainName.DomainNameArn
      RestApiId: !Ref PrivateApi
      Stage: prod
  DomainNameAccessAssociation:
    Type: AWS::ApiGateway::DomainNameAccessAssociation
    Properties:
      DomainNameArn: !GetAtt PrivateDomainName.DomainNameArn
      AccessAssociationSource: !Ref EndpointID
      AccessAssociationSourceType: VPCE
```

## 調用私有 API

您只能使用 VPC 端點從 VPC 內部調用私有 API。您的私有 API 必須具有允許特定 VPC 和 VPC 端點的資源政策，才能調用您的 API。

如果您在不使用自訂網域名稱或私有 DNS 名稱的情況下叫用私有 API，且您的 APIs或網域名稱使用開頭為 的安全政策`SecurityPolicy_`，則必須將端點存取模式設定為 `BASIC`。如需詳細資訊，請參閱the section called “端點存取模式”。

使用自訂網域名稱調用私有 API

若要使用自訂網域名稱調用私有 API，您的 VPC 端點需要一個與自訂網域名稱相關聯的網域名稱存取權，而自訂網域名稱需要允許 VPC 端點調用其名稱。如需詳細資訊，請參閱the section called “私有 API 的自訂網域名稱”。

在您自己 AWS 帳戶 或不同的 VPC 中叫用私有自訂網域名稱之間沒有差異 AWS 帳戶。

使用您的自訂網域名稱

在 VPC 內，您可以使用自訂網域名稱調用 API。下列範例是 curl 命令，用於調用您的私有自訂網域名稱：

```
curl https://private.example.com
```

使用端點特定的私有 DNS 主機名稱

您可以使用自訂網域名稱和端點特定的私有 DNS 主機名稱來調用 API。

```
curl https://private-dns-hostname.execute-api.region.vpce.amazonaws.com/basepath -H
 'Host:custom-domain-name'
```

下列範例是 curl 命令，可使用端點特定的私有 DNS 主機名稱調用您的自訂網域名稱：

```
curl https://vpce-123456-abc000.execute-api.us-east-2.vpce.amazonaws.com/test -H
 'Host:private.example.com'
```

未使用自訂網域名稱調用私有 API

若要在未使用自訂網域名稱的情況下調用您的私有 API，您需要識別 API 的 DNS 名稱。下列程序顯示如何尋找您的 DNS 名稱。

AWS 管理主控台

尋找 DNS 名稱

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/vpc/：// 開啟 Amazon VPC 主控台。
2. 在主要導覽窗格中，選擇端點，然後選擇您的 API Gateway 介面 VPC 端點。
3. 在詳細資訊窗格中，您會在DNS 名稱欄位中看到五個值。前三個是您 API 的公有 DNS 名稱。其他兩個是私有 DNS 名稱。

AWS CLI

使用下列 describe-vpc-endpoints 命令列出您的 DNS 值。

```
aws ec2 describe-vpc-endpoints --vpc-endpoint-ids vpce-01234567abcdef012
```

前三個是您 API 的公有 DNS 名稱。其他兩個是私有 DNS 名稱。

使用 Route53 別名調用私有 API

您可以為 VPC 端點和私有 API 建立關聯或取消其關聯。如需詳細資訊，請參閱the section called “(選用) 關聯或取消關聯 VPC 端點與私有 REST API”。

為 VPC 端點與私有 API 建立關聯後，您可以使用下列基本 URL 調用 API：

```
https://{rest-api-id}-{vpce-id}.execute-api.{region}.amazonaws.com/{stage}
```

例如，若為 `test` 階段設定 `GET /pets`方法，且 REST API ID 為 `01234567ab`，VPC 端點 ID 為 `vpce-01234567abcdef012`，區域為 `us-west-2`，則您可以調用 API 做為：

```
curl -v https://01234567ab-vpce-01234567abcdef012.execute-api.us-west-2.amazonaws.com/
test/pets
```

使用私有 DNS 名稱調用私有 API

如果已啟用私有 DNS，您可以使用下列私有 DNS 名稱來存取您的私有 API：

```
{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，若為 `test` 階段設定 `GET /pets` 方法，而您的 REST API ID 是 `01234567ab`，區域是 `us-west-2`，則您可以在瀏覽器中輸入下列 URL 來調用您的私有 API：

```
https://01234567ab.execute-api.us-west-2.amazonaws.com/test/pets
```

或者，您可以使用下列 cURL 命令來調用私有 API：

```
curl -X GET https://01234567ab.execute-api.us-west-2.amazonaws.com/test/pets
```

> **Warning**
>
> 如果為 VPC 端點啟用私有 DNS，則無法存取公有 API 的預設端點。如需詳細資訊，請參閱為什麼我無法從 API Gateway VPC 端點連接到我的公有 API？。

使用 叫用私有 API Direct Connect

您可以使用 Direct Connect 建立從內部部署網路到 Amazon VPC 的專用私有連線，並使用公有 DNS 名稱透過該連線存取您的私有 API 端點。

您也可以透過設定 Amazon Route 53 Resolver 傳入端點並從遠端網路轉送私有 DNS 的所有 DNS 查詢，使用私有 DNS 名稱從內部部署網路存取私有 API。如需詳細資訊，請參閱《Amazon Route 53 開發人員指南》中的將傳入 DNS 查詢轉送到您的 VPC。

使用端點特定的公有 DNS 主機名稱調用私有 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。

產生的基本 URL 格式如下：

```
https://{public-dns-hostname}.execute-api.{region}.vpce.amazonaws.com/{stage}
```

例如，若為 `test` 階段設定 `GET /pets` 方法，且您的 REST API ID 為 `abc1234`，其公有 DNS 主機名稱為 `vpce-def-01234567`，區域為 `us-west-2`，則您可以在 cURL 命令中使用 `Host` 標頭，使用其 VPCe ID 來調用您的私有 API：

```
curl -v https://vpce-def-01234567.execute-api.us-west-2.vpce.amazonaws.com/test/pets -H
 'Host: abc1234.execute-api.us-west-2.amazonaws.com'
```

或者，您可以使用以下格式的 cURL 命令中 `x-apigw-api-id` 標頭，並透過其 API ID 調用您的私有 API：

```
curl -v https://{public-dns-hostname}.execute-api.{region}.vpce.amazonaws.com/{stage} -
H 'x-apigw-api-id:{api-id}'
```

# 在 API Gateway 中監控 REST API

在本節中，您可以了解如何使用 CloudWatch 指標、CloudWatch Logs、Firehose 和 來監控 API AWS X-Ray。藉由結合 CloudWatch 執行日誌和 CloudWatch 指標，您可以記錄錯誤和執行追蹤，並監控 API 的效能。您可能還想要記錄對 Firehose 的 API 呼叫。您也可以使用 透過組成 API 的下游服務 AWS X-Ray 來追蹤呼叫。

> **Note**
>
> 在下列情況下，API Gateway 可能無法產生日誌和指標：
>
> - 413 請求實體過大錯誤
> - 431 請求標頭欄位太大錯誤
> - 過多的 429 請求過多錯誤
> - 400 系列錯誤，來自傳送至沒有 API 映射自訂網域的要求
> - 由於內部失敗造成的 500 系列錯誤
>
> 測試 REST API 方法時，API Gateway 不會產生日誌和指標。系統會模擬 CloudWatch 實體。如需詳細資訊，請參閱the section called “使用主控台測試 REST API 方法。”。

主題


- 使用 Amazon CloudWatch 指標監控 REST API 執行
- 在 API Gateway 中設定 REST API 的 CloudWatch 記錄功能
- 在 API Gateway 中使用 X-Ray 追蹤使用者對 REST API 的請求


## 使用 Amazon CloudWatch 指標監控 REST 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。這些統計資料會記錄 15 個月的時間，以便您存取歷史資訊，並更清楚 Web 應用程式或服務的執行效能。根據預設，API Gateway 指標資料會自動在一分鐘內傳送給 CloudWatch。如需更多詳細資訊，請參閱《Amazon CloudWatch 使用者指南》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。下列清單顯示指標的一些常見用途，這些用途是協助您開始使用的建議：

- 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 維度和指標
- 使用 API Gateway 中的 API 儀表板檢視 CloudWatch 指標
- 在 CloudWatch 主控台中檢視 API Gateway 指標
- 在 CloudWatch 主控台中檢視 API Gateway 日誌事件
- 在 AWS 中監控 API Gateway 的工具


## Amazon API Gateway 維度和指標

下面列出 API Gateway 傳送至 Amazon CloudWatch 的指標和維度。如需更多詳細資訊，請參閱 使用 Amazon CloudWatch 指標監控 REST API 執行。

API Gateway 指標

Amazon API Gateway 每分鐘都會將指標資料傳送到 CloudWatch。

`AWS/ApiGateway` 命名空間包含下列指標。

| 指標 | 描述 |
| --- | --- |
| 4XXError | 在指定期間內擷取的用戶端錯誤數目。<br><br>API Gateway 會將修改後的閘道回應狀態碼計入 4XXError 錯誤。<br><br>`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)。<br><br>Unit: Count |
| 5XXError | 在指定期間內擷取的伺服器端錯誤數目。 |

| 指標 | 描述 |
| --- | --- |
| | 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)。<br><br>Unit: Count |
| CacheHitCount | 在指定期間內從 API 快取提供的要求數目。<br><br>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)。<br><br>Unit: Count |
| CacheMissCount | 啟用 API 快取時，在指定期間內從後端提供的要求數目。<br><br>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)。<br><br>Unit: Count |
| Count | 指定期間內的 API 要求總數。<br><br>SampleCount 統計資訊代表此指標。<br><br>Unit: Count |
| IntegrationLatency | API Gateway 將請求轉送給後端時與收到來自後端的回應時之間的時間。<br><br>Unit: Millisecond |

| 指標 | 描述 |
| --- | --- |
| Latency | API Gateway 收到來自用戶端的請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。<br><br>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。

> **Note**
>
> 傳送指標至 CloudWatch 前，API Gateway 會從 ApiName 維度移除非 ASCII 字元。如果 APIName 不含 ASCII 字元，API ID 則會做為 ApiName 使用。

| 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。<br><br>除非您已明確啟用詳細的 CloudWatch 指標，否則 API Gateway 不會傳送這類指標。在主控台中選擇某個階段，然後針對日誌和追蹤選取編輯。選取詳細指標，然後選擇儲存變更。或者，您可以呼叫 update-stage AWS CLI 命令，將 `metricsEnabled` 屬性更新為 `true`。<br><br>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 Amazon CloudWatch 定價。 |
| ApiName, Stage | 篩選所指定 API 名稱和階段之 API 階段資源的 API Gateway 指標。 |

## 使用 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 儀表板來顯示在 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
- 檢查儀表板中的 API 活動


先決條件

1. 您必須擁有在 API Gateway 中建立的 API。請遵循中的說明進行在 APIs Gateway 中開發 REST API
2. 您必須至少已部署一次 API。請遵循中的說明進行在 API Gateway 中部署 REST API

檢查儀表板中的 API 活動

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 選擇一個 API。
3. 在主導覽窗格中，選擇儀表板。
4. 針對階段，選擇所需的階段。
5. 選擇日期範圍以指定日期的範圍。
6. 重新整理 (如需要) 並檢視個別圖形中顯示的個別指標，圖形的標題包括 API 呼叫、延遲、整合延遲、延遲、4xx 錯誤和 5xx 錯誤。

> **Tip**
>
> 若要檢查方法層級的 CloudWatch 指標，請確定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，請參閱覆寫階段層級設定。

## 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。若要在 API 的方法層級檢視指標，請開啟詳細的指標。如需詳細資訊，請參閱the section called “更新階段設定”。

使用 CloudWatch 主控台檢視 API Gateway 指標

1. 透過 https://console.aws.amazon.com/cloudwatch/ 開啟 CloudWatch 主控台。
2. 如有需要，請變更 AWS 區域。從導覽列中，選取 AWS 資源所在的區域。
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，選取您的 API 和指標名稱。
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，選取您的 API 和指標名稱。

使用 AWS CLI 檢視指標

1. 使用下列 list-metrics 命令來列出指標：

   ```
   aws cloudwatch list-metrics --namespace "AWS/ApiGateway"
   ```

   建立指標後，最多需要 15 分鐘才會顯示指標。若要更快查看指標統計資料，請使用 get-metric-data 或 get-metric-statistics。
2. 使用下列 get-metrics-statistics 命令檢視一段時間內間隔為 5 分鐘的平均值：

   ```
   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

下一節說明必要的先決條件，以及如何在 CloudWatch 主控台檢視 API Gateway 日誌事件。

先決條件

1. 您必須擁有在 API Gateway 中建立的 API。請遵循中的說明進行在 APIs Gateway 中開發 REST API
2. 您必須至少已部署並調用一次 API。請遵循在 API Gateway 中部署 REST API和在 API Gateway 中調用 REST API中的說明進行。
3. 您必須已針對階段啟用 CloudWatch 日誌。請遵循中的說明進行在 API Gateway 中設定 REST API 的 CloudWatch 記錄功能

使用 CloudWatch 主控台檢視記錄的 API 請求與回應

1. 在 https://console.aws.amazon.com/cloudwatch/ 開啟 CloudWatch 主控台。
2. 如有需要，請變更 AWS 區域。從導覽列中，選取 AWS 資源所在的區域。如需詳細資訊，請參閱區域與端點。
3. 在導覽窗格中依序選擇 Logs (日誌)、Log groups (日誌群組)。
4. 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。
5. 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。
6. 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。

## 在 AWS 中監控 API Gateway 的工具

AWS 提供各種工具，您可以用來監控 API Gateway。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。

中的自動化監控工具 AWS

您可以使用下列自動化監控工具來監看 API Gateway，並在發生錯誤時回報：

- 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。此動作是傳送到 Amazon Simple Notification Service (Amazon SNS) 主題或 Amazon EC2 Auto Scaling 政策的通知。CloudWatch 警示不會只因處於特定狀態就調用動作，狀態必須已變更並已維持一段指定的時間。如需詳細資訊，請參閱使用 Amazon CloudWatch 指標監控 REST API 執行。
- Amazon CloudWatch Logs：監控、存放及存取來自 AWS CloudTrail 或其他來源的日誌檔案。如需更多詳細資訊，請參閱《Amazon CloudWatch 使用者指南》中的什麼是 CloudWatch Logs？。

- Amazon EventBridge (以前稱為 Amazon CloudWatch Events)：比對事件，並將事件轉送至一或多個目標函式或串流以進行變更、擷取狀態資訊，以及採取修正動作。如需詳細資訊，請參閱《Amazon EventBridge 使用者指南》中的什麼是 Amazon EventBridge？。
- AWS CloudTrail 日誌監控 – 在帳戶之間共用日誌檔案、透過將日誌檔案傳送到 CloudWatch Logs 來即時監控 CloudTrail 日誌檔案、在 Java 中寫入日誌處理應用程式，以及驗證您的日誌檔案在 CloudTrail 交付後並未變更。如需詳細資訊，請參閱《AWS CloudTrail 使用者指南》中的使用 CloudTrail 日誌檔案。

手動監控工具

監控 API Gateway 的另一個重要部分是手動監控 CloudWatch 警示未涵蓋的項目。API Gateway、CloudWatch 和其他 AWS 主控台儀表板可讓您at-a-glance檢視 AWS 環境的狀態。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
  - 用於疑難排解問題以及探索驅勢的圖形指標資料。
  - 搜尋和瀏覽您的所有 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 警報，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 警示不會只因處於特定狀態就調用動作，狀態必須已變更並已維持一段指定的時間。

# 在 API Gateway 中設定 REST API 的 CloudWatch 記錄功能

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。如需 CloudWatch 的詳細資訊，請參閱 the section called “CloudWatch 指標”。

CloudWatch 中有兩種類型的 API 記錄：

- 執行記錄 – API Gateway 會記錄處理 API 請求所採取的動作，包括錯誤和執行追蹤。您也可以設定 Amazon CloudWatch Logs 交付，將執行日誌路由到您自己的目的地。
- 存取記錄 – 您記錄存取 API 的人員及其方式。您可以建立自己的日誌群組、選擇日誌格式，並指定要包含的`$context`變數。

您可以個別啟用執行記錄和存取記錄。

## CloudWatch 記錄的許可

若要啟用 CloudWatch Logs，您必須授予 API Gateway 為您的帳戶將日誌讀取和寫入 CloudWatch 的許可。AmazonAPIGatewayPushToCloudWatchLogs 受管政策具有所有必要的許可。

> **Note**
>
> API Gateway 呼叫 AWS Security Token Service 以擔任 IAM 角色，因此請務必為 區域啟用 AWS STS 。如需詳細資訊，請參閱在 AWS 區域中管理 AWS STS。

若要將這些許可授予您的帳戶，請建立 `apigateway.amazonaws.com` 作為其信任實體的 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。您必須為要啟用 CloudWatch Logs 的每個區域分別設定 cloudWatchRoleArn 屬性。 AWS CloudWatch

如果您在設定 IAM 角色 ARN 時收到錯誤，請檢查您的 AWS Security Token Service 帳戶設定，以確保 AWS STS 在您所使用的區域中已啟用 。如需啟用 的詳細資訊 AWS STS，請參閱《IAM 使用者指南》中的在 AWS 區域中管理 AWS STS。

主題


- REST APIs執行記錄
- REST APIs存取記錄


## REST APIs執行記錄

執行記錄會記錄 API Gateway 處理 API 請求所採取的動作。記錄的資料包括錯誤、執行追蹤、請求和回應參數值或承載、Lambda 授權方使用的資料、是否需要 API 金鑰，以及是否啟用用量計劃。API Gateway 會從記錄的資料修訂授權標頭、API 金鑰值和類似的敏感請求參數。

若要提升您的安全狀態，建議您在 `ERROR` 或 `INFO` 層級使用執行記錄。您可能需要這樣做，才能符合各種合規架構。如需詳細資訊，請參閱《AWS Security Hub 使用者指南》中的 Amazon API Gateway 控制項。

標準執行記錄

當您在階段上啟用執行記錄時，API Gateway 會自動建立和管理 CloudWatch Logs 日誌群組。日誌群組名為 `API-Gateway-Execution-Logs_{rest-api-id}/{stage_name}`。在每個日誌群組中，日誌會分為依上次事件時間排序的日誌串流。

使用標準執行記錄：

- 日誌事件被截斷為 1 KB。
- API Gateway 會管理日誌群組名稱。您無法選擇日誌群組名稱或將日誌直接傳送到 Amazon S3 或 Firehose。

若要啟用標準執行記錄，請將階段`loggingLevel`上的 設定為 `ERROR`或 `INFO`。無需其他組態。

許可

若要啟用 CloudWatch Logs，您必須授予 API Gateway 為您的帳戶將日誌讀取和寫入 CloudWatch 的許可。AmazonAPIGatewayPushToCloudWatchLogs 受管政策具有所有必要的許可。

使用 建立 IAM 角色`apigateway.amazonaws.com`做為其信任的實體，附加上述政策，並在您帳戶的 cloudWatchRoleArn 屬性上設定 IAM 角色 ARN。您必須針對要啟用 CloudWatch Logs 的每個 AWS 區域分別設定此屬性。

> **Note**
>
> API Gateway 會呼叫 AWS Security Token Service 來擔任 IAM 角色，因此請務必為 區域啟用 AWS STS 。如需詳細資訊，請參閱在 AWS 區域中管理 AWS STS。

啟用執行記錄（主控台）

開始之前，請將 API 部署至階段，並為您的帳戶設定適當的 CloudWatch Logs 角色 ARN。

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 在主導覽窗格中，選擇設定，然後在記錄下選擇編輯。
3. 針對 CloudWatch 日誌角色 ARN，輸入具有適當許可的 IAM 角色 ARN。您需要針對每個使用 APIs建立 API AWS 帳戶 的 執行此操作一次。
4. 在主要導覽窗格中，選擇 APIs、選擇 REST API，然後選擇階段。
5. 選擇階段，然後在日誌和追蹤區段中，選擇編輯。
6. 從 CloudWatch Logs 下拉式功能表中選取記錄層級：
   - 關閉 – API Gateway 不會記錄此階段的請求。
   - 僅限錯誤 – API Gateway 僅記錄錯誤。
   - 錯誤和資訊日誌 – API Gateway 會記錄所有事件。
7. （選用）選取資料追蹤以記錄完整的請求和回應資料。這可能會導致記錄敏感資料。

   > **Note**
   >
   > 建議您不要針對生產 API 使用資料追蹤。

8. (選用) 選取詳細指標以開啟詳細的 CloudWatch 指標。如需詳細資訊，請參閱the section called “CloudWatch 指標”。
9. 選擇 Save changes (儲存變更)。

當您更新階段設定時，不需要重新部署 API。

啟用執行記錄 (CloudFormation)

使用階段資源上的 `MethodSettings` 屬性來啟用執行記錄。

```
TestStage:
  Type: AWS::ApiGateway::Stage
  Properties:
    StageName: test
    RestApiId: !Ref MyAPI
    DeploymentId: !Ref Deployment
    MethodSettings:
      - ResourcePath: "/*"
        HttpMethod: "*"
        LoggingLevel: INFO
```

若要啟用 CloudWatch Logs，您必須授予 API Gateway 為您的帳戶將日誌讀取和寫入 CloudWatch 的許可。若要深入了解，請參閱《AWS CloudFormation 使用者指南》中的將帳戶與 IAM 角色建立關聯。

執行日誌的 Amazon CloudWatch Logs 交付

Amazon CloudWatch Logs 交付提供執行日誌的替代路徑。您可以設定傳送，將執行日誌傳送至您擁有和控制的目的地，而不是寫入 API Gateway 受管日誌群組。

使用 Amazon CloudWatch Logs 交付：

- 日誌事件最多可達 1 MB，可擷取更完整的請求和回應資料。超過 1 MB 的承載仍會遭到截斷。
- 您可以選擇目的地：您自己的 CloudWatch Logs 日誌群組、Amazon S3 儲存貯體或 Firehose 串流。
- 您可以同時將日誌傳送至多個目的地。
- 您可以控制日誌群組組態，包括命名和保留。

> **Important**
>
> CloudWatch Logs 交付會取代標準執行記錄。當您設定交付時，API Gateway 會停止寫入自動受管日誌群組。您不能在同一階段同時使用這兩個路徑。

當您建立交付時，API Gateway 會將執行日誌路由到您設定的目的地，而不是自動管理的日誌群組。當您刪除交付時，執行日誌會自動繼續流向自動管理的日誌群組。除了`loggingLevel`將 設定為 `ERROR`或 的先決條件之外，不需要進行任何 API Gateway 組態變更`INFO`。

> **Note**
>
> 日誌交付是最佳作法。在極少數情況下，可能無法傳遞某些日誌事件。

> **Warning**
>
> 啟用日誌交付之前，請更新參考自動受管日誌群組 () 的任何儀表板、警示或訂閱篩選條件`API-Gateway-Execution-Logs_{rest-api-id}/{stage_name}`。交付處於作用中狀態後，該日誌群組不會再收到新事件。

下表將標準執行記錄與 Amazon CloudWatch Logs 交付進行比較。

| | 標準執行記錄 | Amazon CloudWatch Logs 交付 |
|---|---|---|
| 日誌事件大小上限 | 1 KB | 1 MB |
| 目的地 | CloudWatch Logs （受管日誌群組） | CloudWatch Logs、Amazon S3、Firehose （您自己的資源） |
| 多個目的地 | 否 | 是 |
| 日誌群組名稱 | 由 API Gateway 修正 | 您可以選擇 |
| 設定 | 啟用記錄時自動執行 | 需要建立 Amazon CloudWatch Logs 交付 |
| 先決條件 | `loggingLevel` 設定為 `ERROR`或 `INFO` | `loggingLevel` 設定為 `ERROR`或 `INFO` |

> **Note**
>
> 如果您在設定交付`OFF`之後`loggingLevel`將 設定為 ，則不會產生或交付執行日誌。您的交付目的地會保持設定狀態，但不會接收任何資料。

若要建立交付，請參閱 the section called “建立日誌交付”。

## 建立 REST API 執行日誌的日誌交付

您可以將執行日誌路由到您自己的 CloudWatch Logs 日誌群組、Amazon S3 儲存貯體或 Firehose 串流。若要這樣做，請建立 Amazon CloudWatch Logs 交付。這包括三個步驟：建立交付來源、建立交付目的地，以及將它們與交付連結。

### 先決條件

建立日誌交付之前，請確定您有下列項目：

- 在您的 REST API 階段上啟用執行記錄。將 `loggingLevel` 設為 `ERROR` 或 `INFO`。如果未啟用記錄，則交付來源建立會失敗。
- 目的地資源：您擁有的 CloudWatch Logs 日誌群組、Amazon S3 儲存貯體或 Firehose 串流。
- 呼叫 Amazon CloudWatch Logs 交付 APIs許可。如需詳細資訊，請參閱《Amazon CloudWatch Logs 使用者指南》中的從 AWS 服務啟用記錄。

### Amazon CloudWatch Logs 交付的 API Gateway 特定值

當您為 API Gateway 執行日誌建立交付時，請使用下列值：

資源 ARN

```
arn:aws:apigateway:{region}:{account-id}:/restapis/{rest-api-id}/
stages/{stage-name}
```

日誌類型

```
EXECUTION_LOGS
```

### 建立日誌交付 (AWS CLI)

下列範例會建立交付，將執行日誌從 REST API 傳送至您擁有的 CloudWatch Logs 日誌群組。

1. 建立交付來源

   將您的 REST API 註冊為交付來源。

```
aws logs put-delivery-source \
    --name my-apigw-source \
    --log-type EXECUTION_LOGS \
```

```
    --resource-arn arn:aws:apigateway:{region}:{account-id}:/restapis/{rest-api-id}/stages/{stage-name}
```

2. 建立交付目的地

   將日誌群組註冊為交付目的地。

```
aws logs put-delivery-destination \
    --name my-log-destination \
    --delivery-destination-configuration
 "destinationResourceArn=arn:aws:logs:{region}:{account-id}:log-group:{log-group-name}"
```

3. 建立交付

   將來源連結至目的地。

```
aws logs create-delivery \
    --delivery-source-name my-apigw-source \
    --delivery-destination-arn arn:aws:logs:{region}:{account-id}:delivery-destination:my-log-destination
```

若要將日誌傳送至 Amazon S3 或 Firehose，請在 `put-delivery-destination`命令中指定適當的資源 ARN。如需完整 Amazon CloudWatch Logs 交付工作流程的詳細資訊，請參閱《Amazon CloudWatch Logs 使用者指南》中的從 AWS 服務啟用記錄。

若要將日誌傳送至多個目的地，請使用具有不同目的地的相同交付來源來建立其他交付。

記錄欄位

透過 Amazon CloudWatch Logs 交付，執行日誌包含您可以為每個目的地自訂的結構化欄位。下列欄位可供使用：

- `resource_arn` – API 階段的 ARN
- `event_timestamp` – 日誌事件的時間
- `api_id` – REST API 識別符
- `stage` – 階段名稱
- `resource_path` – 請求的資源路徑
- `http_method` – HTTP 方法

- payload – 執行日誌內容 （與標準執行日誌相同 ）

您可以使用 上的 --record-fields 參數和 上的 參數，選取要包含的欄位create-delivery和每個目的地的輸出格式--output-formatput-delivery-destination。

可用的輸出格式取決於目的地：

- CloudWatch Logs – json、純
- Amazon S3 – json、Plain、w3c、parquet
- Firehose – json、純

> **比對標準執行日誌輸出**
>
> 如果您想要符合標準執行記錄的交付輸出 （僅承載，無其他欄位），請將 --output-format "plain"新增至 put-delivery-destination和 --record-fields "payload" --field-delimiter "" create-delivery。

建立日誌交付 (CloudFormation)

為 CloudFormation 範本組合階段 ARN 時，請使用下列模式：

```
  DeliverySource:
    Type: AWS::Logs::DeliverySource
    Properties:
      Name: my-apigw-source
      LogType: EXECUTION_LOGS
      ResourceArn: !Sub "arn:${AWS::Partition}:apigateway:${AWS::Region}:
${AWS::AccountId}:/restapis/${MyApi}/stages/${MyStageName}"
```

建立日誌交付 (API Gateway 主控台 )

您可以從階段詳細資訊頁面的日誌和追蹤區段建立日誌交付。

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 選擇您的 REST API，然後選擇階段。
3. 選擇一個階段。
4. 在日誌和追蹤區段的日誌交付目的地下，選擇新增目的地。

5. 選擇目的地類型 (CloudWatch Logs、Amazon S3 或 Firehose)，並指定目的地資源。
6. 選擇儲存。

> **Important**
>
> 您必須先在階段上啟用執行記錄，才能新增交付目的地。如果記錄設定為關閉，則交付組態不會產生日誌。

> **Note**
>
> 若要設定跨帳戶交付目的地，請使用 AWS CLI。在來源帳戶中建立交付之前，您必須在目的地帳戶中建立交付目的地政策。如需詳細資訊，請參閱《Amazon CloudWatch Logs 使用者指南》中的跨帳戶交付範例。

驗證執行日誌交付

建立日誌交付後，請確認交付處於作用中狀態，且執行日誌已到達目的地。

若要確認交付處於作用中狀態，請使用 `describe-deliveries`命令：

```
aws logs describe-deliveries
```

然後傳送測試請求並檢查您的目的地：

1. 傳送測試請求到您的 REST API。

   ```
   curl https://{rest-api-id}.execute-api.{region}.amazonaws.com/{stage}/{resource}
   ```

2. 等待 1-2 分鐘讓日誌傳播。
3. 檢查您的目的地是否有日誌事件：

   - CloudWatch Logs – 開啟 CloudWatch Logs 主控台並導覽至您的日誌群組。您應該會看到包含最近事件的日誌串流。
   - Amazon S3 – 開啟 Amazon S3 主控台並導覽至您的儲存貯體。尋找路徑字首包含 REST API ID 的物件。
   - Firehose – 在 Firehose 主控台中檢查交付串流指標，以確認資料正在流動。

### 疑難排解

如果日誌未送達目的地，請檢查下列項目：

- 已啟用記錄 – 確認 `loggingLevel` 設定為階段INFO上的 ERROR或 。如果記錄為 OFF，無論交付組態為何，都不會產生日誌。
- 交付狀態 – 使用 `get-delivery` CLI 命令來檢查交付狀態並確認其處於作用中狀態。

```
aws logs get-delivery --id {delivery-id}
```

- 許可 – 驗證目的地上的資源政策是否允許 Amazon CloudWatch Logs 寫入它。如需詳細資訊，請參閱《Amazon CloudWatch Logs 使用者指南》中的從 AWS 服務啟用記錄。

### 刪除 REST API 執行日誌的日誌交付

當您刪除日誌交付時，API Gateway 會繼續將執行日誌寫入自動受管日誌群組 (`API-Gateway-Execution-Logs_{rest-api-id}/{stage_name}`)，只要階段上仍然啟用執行日誌。

#### 刪除日誌交付 (AWS CLI)

若要刪除交付，您可以刪除交付，然後選擇性地刪除交付來源和目的地。

1. 刪除交付

```
aws logs delete-delivery --id {delivery-id}
```

2. （選用）刪除交付來源

```
aws logs delete-delivery-source --name my-apigw-source
```

3. （選用）刪除交付目的地

```
aws logs delete-delivery-destination --name my-log-destination
```

> **Note**
>
> 刪除交付來源或目的地是選用的。您可以稍後重複使用它們來建立新的交付。

刪除日誌交付 (API Gateway 主控台 )

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 選擇您的 REST API，然後選擇階段。
3. 選擇一個階段。
4. 在日誌和追蹤區段的日誌交付目的地下，選取您要移除的目的地。
5. 選擇 刪除。

維持日誌連續性

> **Tip**
>
> 若要將日誌保留在相同的日誌群組中，並維護現有的保留設定和訂閱篩選條件，請將自動受管日誌群組 (`API-Gateway-Execution-Logs_{rest-api-id}/{stage_name}`) 設定為交付目的地。日誌串流名稱將與標準日誌所建立的名稱不同。

## REST APIs存取記錄

在存取日誌中，您會記錄誰已存取您的 API，以及發起人如何存取 API。您可以建立自己的日誌群組，或選擇現有的日誌群組。若要指定存取權詳細資訊，您可以選取 `$context` 變數、日誌格式及日誌群組目的地。

存取日誌格式

存取日誌格式必須至少包含 `$context.requestId` 或 `$context.extendedRequestId`。最佳實務是將兩者都納入您的日誌格式。

**`$context.requestId`**

這會記錄 `x-amzn-RequestId` 標頭中的值。用戶端可以使用通用唯一識別碼 (UUID) 格式的值來覆寫 `x-amzn-RequestId` 標頭中的值。API Gateway 會傳回 `x-amzn-RequestId` 回應標頭中的此請求 ID。API Gateway 會將您的存取日誌中不是 UUID 格式的已覆寫請求 ID 取代為 *`UUID`*`_REPLACED_INVALID_REQUEST_ID`。

**`$context.extendedRequestId`**

extendedRequestID 是 API Gateway 產生的唯一 ID。API Gateway 會傳回 `x-amz-apigw-id` 回應標頭中的此請求 ID。API 呼叫者無法提供或覆寫此請求 ID。您可能需要將此值提供給 AWS

Support，以協助對 API 進行故障診斷。如需詳細資訊，請參閱the section called “API Gateway 存取記錄的變數”。

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 主控台取得日誌群組 ARN。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
 $context.extendedRequestId
```

- JSON:

```
{ "requestId":"$context.requestId",
 "extendedRequestId":"$context.extendedRequestId","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" }
```

- XML:

```
<request id="$context.requestId"> <extendedRequestId>$context.extendedRequestId</
extendedRequestId> <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>
```

- 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,$context.requestId,$context.extendedRequestId
```

使用 API Gateway 主控台設定存取記錄

設定存取記錄之前，請將 API 部署至階段，並為您的帳戶設定適當的 CloudWatch Logs 角色 ARN。

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 在主導覽窗格中，選擇 API，然後執行下列其中一項操作：
   a. 選擇現有 API，然後選擇階段。
   b. 建立 API，然後將它部署至階段。
3. 在主導覽窗格中，選擇階段。
4. 在日誌和追蹤區段中，選擇編輯。
5. 開啟自訂存取記錄。
6. 在存取日誌目的地 ARN 中輸入日誌群組的 ARN。ARN 格式為 `arn:aws:logs:{region}:{account-id}:log-group:log-group-name`。
7. 針對日誌格式，輸入日誌格式。您可以選擇 CLF、JSON、XML 或 CSV。若要進一步了解範例日誌格式，請參閱 the section called “存取日誌格式”。
8. 選擇儲存變更。

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。

使用 設定存取記錄 CloudFormation

使用下列範例 CloudFormation 範本來建立 Amazon CloudWatch Logs 日誌群組，並設定階段的存取記錄。若要啟用 CloudWatch Logs，您必須授予 API Gateway 為您的帳戶將日誌讀取和寫入 CloudWatch 的許可。若要深入了解，請參閱《AWS CloudFormation 使用者指南》中的將帳戶與 IAM 角色建立關聯。

```
  TestStage:
    Type: AWS::ApiGateway::Stage
    Properties:
      StageName: test
      RestApiId: !Ref MyAPI
      DeploymentId: !Ref Deployment
      AccessLogSetting:
        DestinationArn: !GetAtt MyLogGroup.Arn
        Format: $context.extendedRequestId $context.identity.sourceIp
$context.identity.caller $context.identity.user [$context.requestTime]
```

```
"$context.httpMethod $context.resourcePath $context.protocol" $context.status
$context.responseLength $context.requestId
 MyLogGroup:
   Type: AWS::Logs::LogGroup
   Properties:
     LogGroupName: !Join
       - '-'
       - - !Ref MyAPI
         - access-logs
```

在 API Gateway 中記錄對 Amazon Data Firehose 的 REST API 呼叫

為了協助偵錯 API 用戶端存取相關問題，您可以記錄對 Amazon Data Firehose 的 API 呼叫。如需 Firehose 的詳細資訊，請參閱什麼是 Amazon Data Firehose？。

對於存取記錄，您只能啟用 CloudWatch 或 Firehose，無法同時啟用兩者。不過，您可以讓 CloudWatch 能夠進行執行記錄，以及讓 Firehose 能夠進行存取記錄。

主題

- 適用於 API Gateway 的 Firehose 日誌格式
- Firehose 記錄功能的許可權
- 使用 API Gateway 主控台設定 Firehose 存取記錄功能

適用於 API Gateway 的 Firehose 日誌格式

Firehose 記錄功能使用的格式與 CloudWatch 記錄功能相同。

Firehose 記錄功能的許可權

在階段上啟用 Firehose 存取記錄功能時，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**
>
> Firehose 串流的名稱必須是 amazon-apigateway-*{your-stream-name}*。

使用 API Gateway 主控台設定 Firehose 存取記錄功能

若要設定 API 記錄，您必須已將 API 部署至階段。您也必須已建立 Firehose 串流。

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 執行以下任意一項：
   a. 選擇現有 API，然後選擇階段。
   b. 建立 API，並將它部署至階段。
3. 在主導覽窗格中，選擇階段。
4. 在日誌和追蹤區段中，選擇編輯。
5. 若要啟用 Firehose 串流的存取記錄：
   a. 開啟自訂存取記錄。
   b. 針對存取日誌目的地 ARN，輸入 Firehose 串流的 ARN。ARN 格式為 `arn:aws:firehose:{region}:{account-id}:deliverystream/amazon-apigateway-{your-stream-name}`。

      > **Note**
      >
      > Firehose 串流的名稱必須是 `amazon-apigateway-{your-stream-name}`。

   c. 針對日誌格式，輸入日誌格式。您可以選擇 CLF、JSON、XML 或 CSV。若要進一步了解範例日誌格式，請參閱 the section called “存取日誌格式”。
6. 選擇儲存變更。

API Gateway 現在已準備好將對 API 的請求記錄至 Firehose。當您更新階段設定、日誌或階段變數時，不需要重新部署 API。

API Gateway 存取記錄的變數

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，或選擇由 API Gateway 管理的現有日誌群組。若要指定存取詳細資訊，您可以使用下列區分大小寫的 `$context` 變數。

如需資料轉換的參考變數清單，請參閱 the section called “用於資料轉換的變數”。

| 參數 | 說明 |
| --- | --- |
| `$context.accountId` | API 擁有者 AWS 的帳戶 ID。 |

| 參數 | 說明 |
| --- | --- |
| `$context.apiId` | API Gateway 指派給您 API 的識別碼。 |
| `$context.authorize.error` | 授權錯誤訊息。 |
| `$context.authorize.latency` | 授權延遲 (以毫秒為單位)。 |
| `$context.authorize.status` | 從授權嘗試傳回的狀態碼。 |
| `$context.authorizer.claims.`*`property` | 成功驗證方法發起人之後，從 Amazon Cognito 使用者集區傳回之宣告的屬性。如需詳細資訊，請參閱 the section called “針對 REST API 使用 Amazon Cognito 使用者集區做為授權方”。<br>**Note**<br>呼叫 `$context.authorizer.claims` 會傳回 null。 |
| `$context.authorizer.error` | 從授權方傳回的錯誤訊息。 |
| `$context.authorizer.integrationLatency` | 授權方整合延遲 (以毫秒為單位)。 |
| `$context.authorizer.integrationStatus` | 從 Lambda 授權方傳回的狀態碼。 |
| `$context.authorizer.latency` | 授權方延遲 (以毫秒為單位)。 |
| `$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 the section called “使用 Lambda 授權方”。 |

| 參數 | 說明 |
| --- | --- |
| $context.authorizer. *property* | API Gateway Lambda 授權方函數所傳回 context 映射之指定索引鍵/值組的字串化值。例如，如果授權方傳回下列 context 映射：<br><br>`"context" : { "key": "value", "numKey": 1, "boolKey": true }`<br><br>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。<br><br>對於##，唯一支援的特殊字元是底線 (_) 字元。<br><br>如需詳細資訊，請參閱the section called “使用 Lambda 授權方”。 |
| $context.authorizer.requestId | AWS 端點的請求 ID。 |
| $context.authorizer.status | 從授權方傳回的狀態碼。 |
| $context.authenticate.error | 從驗證嘗試傳回的錯誤訊息。 |
| $context.authenticate.latency | 驗證延遲 (以毫秒為單位)。 |
| $context.authenticate.status | 從驗證嘗試傳回的狀態碼。 |
| $context.awsEndpointRequestId | AWS 端點的請求 ID。 |
| $context.cipherSuite | 在用戶端和 API Gateway 之間的 TLS 交握期間交涉的 IANA 格式密碼。 |

| 參數 | 說明 |
| --- | --- |
| $context.customDomain.basePathMatched | 傳入請求相符的 API 映射路徑。適用於用戶端使用自訂網域名稱來存取 API 時。例如，如果用戶端向 https://api.example.com/v1/orders/1234 傳送請求，並且請求讓 API 映射與路徑 v1/orders 相符，則該值為 v1/orders 。如需進一步了解，請參閱the section called “API 映射”。 |
| $context.customDomain.routingRuleIdMatched | 傳入請求與其相符的路由規則。適用於用戶端使用自訂網域名稱來存取 API 時。如需詳細資訊，請參閱 the section called “路由規則”。 |
| $context.deploymentId | API 部署的 ID。 |
| $context.domainName | 用來調用 API 的完整網域名稱。這應該與傳入的 Host 標頭相同。 |
| $context.domainPrefix | $context.domainName 的第一個標籤。 |
| $context.endpointType | API 的端點類型。 |
| $context.error.message | 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 GatewayResponse 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the section called “指標”及the section called “設定閘道回應以自訂錯誤回應”。 |
| $context.error.messageString | $context.error.message 的引用值，即 "$context.error.message" 。 |

| 參數 | 說明 |
| --- | --- |
| $context.error.responseType | GatewayResponse 的類型。此變數只能用於 Velocity 範本語言引擎無法處理的 GatewayResponse 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the section called “指標”及the section called “設定閘道回應以自訂錯誤回應”。 |
| $context.error.validationErrorString | 字串，其中包含詳細的驗證錯誤訊息。 |
| $context.extendedRequestId | API Gateway 產生和指派給 API 請求的延伸 ID。延伸請求 ID 包含有助於偵錯和疑難排解的資訊。 |
| $context.httpMethod | 使用的 HTTP 方法。有效值包含：DELETE、GET、HEAD、OPTIONS、PATCH、POST 和 PUT。 |
| $context.identity.accountId | 與請求相關聯的 AWS 帳戶 ID。 |
| $context.identity.apiKey | 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 the section called “用量計劃”。 |
| $context.identity.apiKeyId | 與需要 API 金鑰之 API 請求相關聯的 API 金鑰 ID。 |
| $context.identity.caller | 已簽署請求之發起人的主體識別符。支援使用 IAM 授權的資源。 |

| 參數 | 說明 |
| --- | --- |
| $context.identity.cognitoAuthenticationProvider | 提出請求的發起人所使用的所有 Amazon Cognito 驗證提供者清單 (以逗號分隔)。僅在使用 Amazon Cognito 登入資料簽署請求時才可使用。<br><br>例如，適用於 Amazon Cognito 使用者集區的身分，cognito-idp.*region*.amazonaws.com/ *user_pool_id* ,cognito-idp. *region*.amazonaws.com/ *user_pool_id* :CognitoSignIn: *token subject claim*<br><br>如需有關 Amazon Cognito 驗證提供者的詳細資訊，請參閱《Amazon Cognito 開發人員指南》中的使用聯合身分。 |
| $context.identity.cognitoAuthenticationType | 提出請求的發起人的 Amazon Cognito 驗證類型。僅在使用 Amazon Cognito 登入資料簽署請求時才可使用。可能的值包括用於已驗證身分的 authenticated 和未經驗證身分的 unauthenticated 。 |
| $context.identity.cognitoIdentityId | 提出請求的發起人的 Amazon Cognito 身分 ID。僅在使用 Amazon Cognito 登入資料簽署請求時才可使用。 |
| $context.identity.cognitoIdentityPoolId | 提出請求的發起人的 Amazon Cognito 身分集區 ID。僅在使用 Amazon Cognito 登入資料簽署請求時才可使用。 |
| $context.identity.principalOrgId | AWS 組織 ID。 |
| $context.identity.sourceIp | 對 API Gateway 端點提出請求之即時 TCP 連線的來源 IP 位址。 |

| 參數 | 說明 |
| --- | --- |
| `$context.identity.clientCert.clientCertPem` | 用戶端在交互 TLS 驗證期間所呈現的 PEM 編碼用戶端憑證。當用戶端使用已啟用交互 TLS 的自訂網域名稱存取 API 時會顯示。 |
| `$context.identity.clientCert.subjectDN` | 用戶端提供之憑證主體的辨別名稱。當用戶端使用已啟用交互 TLS 的自訂網域名稱存取 API 時會顯示。 |
| `$context.identity.clientCert.issuerDN` | 用戶端提供之憑證發行者的辨別名稱。當用戶端使用已啟用交互 TLS 的自訂網域名稱存取 API 時會顯示。 |
| `$context.identity.clientCert.serialNumber` | 憑證的序號。當用戶端使用已啟用交互 TLS 的自訂網域名稱存取 API 時會顯示。 |
| `$context.identity.clientCert.validity.notBefore` | 憑證無效之前的日期。當用戶端使用已啟用交互 TLS 的自訂網域名稱存取 API 時會顯示。 |
| `$context.identity.clientCert.validity.notAfter` | 憑證無效之後的日期。當用戶端使用已啟用交互 TLS 的自訂網域名稱存取 API 時會顯示。 |
| `$context.identity.vpcId` | 對 API Gateway 端點提出請求之 VPC 的 VPC ID。 |
| `$context.identity.vpceId` | 對 API Gateway 端點提出請求之 VPC 端點的 VPC 端點 ID。只會在您擁有私有 API 時顯示。 |
| `$context.identity.user` | 將針對資源存取授權之使用者的主體識別符。支援使用 IAM 授權的資源。 |
| `$context.identity.userAgent` | API 發起人的 `User-Agent` 標頭。 |
| `$context.identity.userArn` | 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https://docs.aws.amazon.com/IAM/latest/UserGuide/id_users.html。 |

| 參數 | 說明 |
| --- | --- |
| $context.integration.error | 從整合傳回的錯誤訊息。 |
| $context.integration.integrationStatus | 對於 Lambda 代理整合，從 傳回的狀態碼 AWS Lambda，而不是從後端 Lambda 函數程式碼傳回的狀態碼。 |
| $context.integration.latency | 整合延遲 (以毫秒為單位)。等同於 $context.integrationLatency 。 |
| $context.integration.requestId | AWS 端點的請求 ID。等同於 $context.awsEndpointRequestId 。 |
| $context.integration.responseTransferMode | 整合的回應傳輸模式。可以是 BUFFERED 或 STREAMED。 |
| $context.integration.status | 從整合傳回的狀態碼。對於 Lambda 代理整合而言，這是您的 Lambda 函數程式碼傳回的狀態碼。 |
| $context.integration.timeToAllHeaders | 從用戶端接收所有整合回應標頭時，API Gateway 建立與 的整合連線之間的時間。 |
| $context.integration.timeToFirstContent | API Gateway 在收到第一個內容位元組時建立與的整合連線之間的時間。 |
| $context.integrationLatency | 整合延遲 (以毫秒為單位)。 |
| $context.integrationStatus | 對於 Lambda 代理整合，此參數代表從 傳回的狀態碼 AWS Lambda，而不是從後端 Lambda 函數程式碼傳回的狀態碼。 |
| $context.isCanaryRequest | 如果請求導向 Canary，則傳回 true；如果請求未導向 Canary，則傳回 false。只會在啟用 Canary 時顯示。 |

| 參數 | 說明 |
| --- | --- |
| `$context.path` | 請求路徑。例如，對於 `https://{rest-api-id}.execute-api.{region}.amazonaws.com/{stage}/root/child` 的非代理請求 URL，`$context.path` 值是 `/{stage}/root/child` 。 |
| `$context.protocol` | 請求通訊協定，例如 `HTTP/1.1`。<br><br>**Note**<br>API Gateway API 可以接受 HTTP/2 請求，但 API Gateway 會使用 HTTP/1.1 將請求傳送至後端整合。因此，即使用戶端傳送使用 HTTP/2 的請求，請求通訊協定也會記錄為 HTTP/1.1。 |
| `$context.requestId` | 請求的 ID。用戶端可以覆寫此請求 ID。使用 `$context.extendedRequestId` 即可取得 API Gateway 產生的唯一請求 ID。 |
| `$context.requestOverride.header.` *`header_name`* | 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 覆寫 API Gateway 中 REST API 的 API 請求和回應參數及狀態碼。 |
| `$context.requestOverride.path.` *`path_name`* | 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 覆寫 API Gateway 中 REST API 的 API 請求和回應參數及狀態碼。 |

| 參數 | 說明 |
| --- | --- |
| `$context.requestOverride.querystring.` *querystring_name* | 請求查詢字串會覆寫。如果此參數已經定義，它包含要使用的請求查詢字串 (而不是在 Integration Request (整合請求) 窗格中定義的 URL Query String (URL 查詢字串))。如需詳細資訊，請參閱覆寫 API Gateway 中 REST API 的 API 請求和回應參數及狀態碼。 |
| `$context.responseLatency` | 回應延遲 (以毫秒為單位)。 |
| `$context.responseLength` | 回應承載長度 (以位元組為單位)。 |
| `$context.responseOverride.header.` *header_name* | 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 覆寫 API Gateway 中 REST API 的 API 請求和回應參數及狀態碼。 |
| `$context.responseOverride.status` | 回應狀態碼會覆寫。如果此參數已經定義，它包含要傳回的狀態碼 (而不是在 Integration Response (整合回應) 窗格中定義為 Default mapping (預設映射) 的 Method response status (方法回應狀態))。如需詳細資訊，請參閱 覆寫 API Gateway 中 REST API 的 API 請求和回應參數及狀態碼。 |
| `$context.requestTime` | CLF 格式化請求時間 (`dd/MMM/yyyy:HH:mm:ss +-hhmm` )。 |
| `$context.requestTimeEpoch` | Epoch 格式化的要求時間，以毫秒為單位。 |
| `$context.resourceId` | API Gateway 指派給您的資源的識別碼。 |

| 參數 | 說明 |
| --- | --- |
| $context.resourcePath | 您資源的路徑。例如，對於非代理請求 URI https://{rest-api-id}.execute-api.{region}.amazonaws.com/{stage}/root/child ，$context.resourcePath 值是 /root/child 。如需詳細資訊，請參閱 教學：建立具有 HTTP 非代理整合的 REST API。 |
| $context.stage | API 請求的部署階段 (例如，Beta 或 Prod)。 |
| $context.status | 方法回應狀態。 |
| $context.tlsVersion | 在用戶端和 API Gateway 之間的 TLS 交握期間交涉的 TLS 版本。 |
| $context.waf.error | 從 傳回的錯誤訊息 AWS WAF。 |
| $context.waf.latency | 以毫秒為單位的 AWS WAF 延遲。 |
| $context.waf.status | 從 傳回的狀態碼 AWS WAF。 |
| $context.xrayTraceId | X-Ray 追蹤的追蹤 ID。如需詳細資訊，請參閱the section called “設定 AWS X-Ray”。 |
| $context.wafResponseCode | 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 the section called “AWS WAF”。 |
| $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the section called “AWS WAF”。 |

# 在 API Gateway 中使用 X-Ray 追蹤使用者對 REST API 的請求

您可以使用 AWS X-Ray，在使用者請求通過 Amazon API Gateway REST API 進入基礎服務時追蹤及分析這些請求。API Gateway 支援所有 API Gateway REST API 端點類型的 X-Ray 追蹤：區域性、邊緣最佳化和私有。您可以在適用 X-Ray 的所有 AWS 區域中將 X-Ray 與 Amazon API Gateway 搭配使用。

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。您也可以設定取樣規則，以根據您指定的條件告知 X-Ray 要記錄那些請求，以及使用何種取樣率。

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 主控台或使用 API Gateway API 或 CLI，為 API 階段啟用 X-Ray。

主題

- AWS X-Ray 使用 API Gateway REST APIs設定
- 搭配 API Gateway 使用 AWS X-Ray 服務映射和追蹤檢視
- 設定 API Gateway APIs的 AWS X-Ray 抽樣規則
- AWS X-Ray Amazon API Gateway APIs追蹤

## AWS X-Ray 使用 API Gateway REST APIs設定

本節提供如何使用 API Gateway REST API 設定 AWS X-Ray 的詳細資訊。

主題

- API Gateway 的 API Gateway 追蹤模式
- X-Ray 追蹤的許可
- 在 API Gateway 主控台中啟用 X-Ray 追蹤
- 使用 API Gateway CLI 啟用 AWS X-Ray 追蹤

### API Gateway 的 API Gateway 追蹤模式

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。

API Gateway API 有兩種追蹤模式：

- 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 SDK 來變更 API 的追蹤模式。

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 主控台中啟用 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. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 選擇您的 API，然後在主導覽窗格中選擇階段。
3. 在階段窗格中，選擇一個階段。
4. 在日誌和追蹤區段中，選擇編輯。
5. 若要啟用主動 X-Ray 追蹤，選取 X-Ray 追蹤以開啟 X-Ray 追蹤。
6. 選擇儲存變更。

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

以下 create-stage 命令會建立具有作用中 X-Ray 追蹤的階段：

```
aws apigateway create-stage \
    --rest-api-id rest-api-id \
    --stage-name stage-name \
    --deployment-id deployment-id \
    --region region \
    --tracing-enabled=true
```

輸出將如下所示：

```
{
    "tracingEnabled": true,
    "stageName": stage-name,
    "cacheClusterEnabled": false,
    "cacheClusterStatus": "NOT_AVAILABLE",
    "deploymentId": deployment-id,
    "lastUpdatedDate": 1533849811,
    "createdDate": 1533849811,
    "methodSettings": {}
}
```

以下 create-stage 命令會建立不具有作用中 X-Ray 追蹤的階段：

```
aws apigateway create-stage \
    --rest-api-id rest-api-id \
    --stage-name stage-name \
    --deployment-id deployment-id \
    --region region \
    --tracing-enabled=false
```

輸出將如下所示：

```
{
    "tracingEnabled": false,
    "stageName": stage-name,
    "cacheClusterEnabled": false,
    "cacheClusterStatus": "NOT_AVAILABLE",
    "deploymentId": deployment-id,
```

```
    "lastUpdatedDate": 1533849811,
    "createdDate": 1533849811,
    "methodSettings": {}
}
```

以下 update-stage 命令會為部署的 API 開啟作用中 X-Ray 追蹤：

```
aws apigateway update-stage \
    --rest-api-id rest-api-id \
    --stage-name stage-name \
    --patch-operations op=replace,path=/tracingEnabled,value=true
```

以下 update-stage 命令會將部署的 API 的作用中 X-Ray 追蹤關閉：

```
aws apigateway update-stage \
    --rest-api-id rest-api-id \
    --stage-name stage-name \
    --region region \
    --patch-operations op=replace,path=/tracingEnabled,value=false
```

輸出將如下所示：

```
{
    "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，請參閱搭配 CLI AWS 使用 X-Ray API。

## 搭配 API Gateway 使用 AWS X-Ray 服務映射和追蹤檢視

本節提供如何搭配 API Gateway 使用 AWS X-Ray 服務對應和追蹤檢視的詳細資訊。

**主題**

- 範例 X-Ray 服務地圖
- 範例 X-Ray 追蹤檢視

## 範例 X-Ray 服務地圖

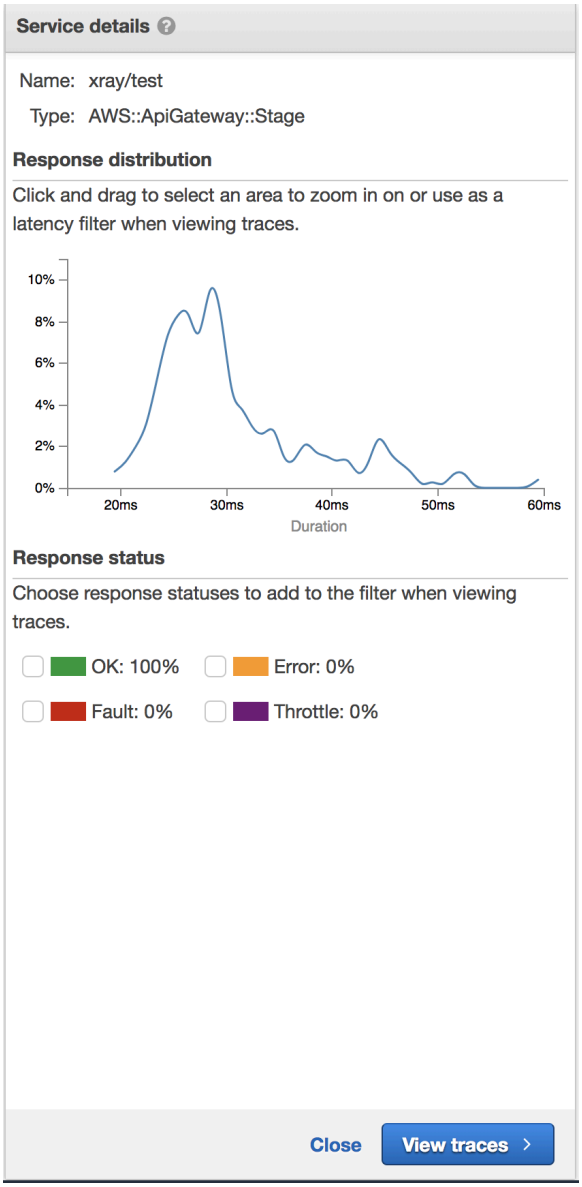
AWS X-Ray 服務映射會顯示 API 及其所有下游服務的相關資訊。在 API Gateway 中為 API 階段啟用 X-Ray 後，您會在服務對應中看到節點，其中包含 API Gateway 服務中所花費整體時間的相關資訊。您可以取得所選時間範圍內 API 回應時間之回應狀態和長條圖的詳細資訊。對於與 AWS Lambda 和 Amazon DynamoDB 等 AWS 服務整合APIs，您會看到更多節點提供與這些服務相關的效能指標。每個 API 階段都會有一個服務對應。

以下範例顯示名為 `test` 之 API `xray` 階段的服務對應。此 API 有兩個 Lambda 整合。這些節點代表 API Gateway 服務和兩個 Lambda 函式。

如需服務圖結構的詳細說明，請參閱使用 X-Ray 追蹤圖。

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 AWS Lambda 和 AWS X-Ray 。

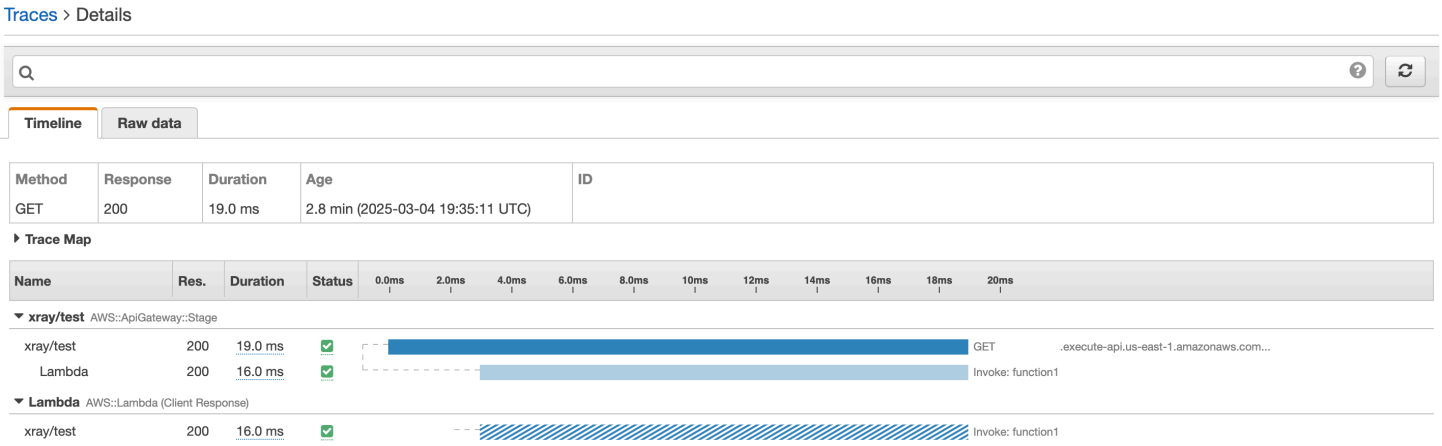
如果您在 X-Ray 服務對應上選擇節點或邊緣，X-Ray 主控台會顯示延遲分佈長條圖。您可以使用延遲長條圖來查看服務需要多久時間才能完成其請求。以下是先前服務對應中 API Gateway 階段的長條圖，名為 `xray/test`。如需延遲分佈長條圖的詳細說明，請參閱使用延遲長條圖。




範例 X-Ray 追蹤檢視

下圖顯示為上述範例 API 產生的追蹤檢視，搭配 Lambda 後端函式。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，請參閱檢視追蹤和追蹤詳細資訊。




## 設定 API Gateway APIs的 AWS X-Ray 抽樣規則

您可以使用 AWS X-Ray 主控台或 SDK 來設定 Amazon API Gateway API 的抽樣規則。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：

- 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
- X-Ray 取樣規則範例

### API Gateway API 的 X-Ray 取樣規則選項值

以下 X-Ra 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，包括如何使用儲槽和比率設定的詳細說明，請設定取樣規則。

- Rule name (規則名稱) (字串) — 規則的唯一名稱。
- Priority (優先順序) (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 (儲槽) (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) (介於 0 到 100 之間的數字) — 在儲槽用盡之後，要檢測的相符請求百分比。
- 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 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 主機標頭的主機名稱。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 ，例如 **`arn:aws:apigateway:`*`region`*`::/restapis/`*`api-id`*`/stages/`*`stage-name`***。

  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 一般參考。
- HTTP method (HTTP 方法) (字串) — 取樣的方法，例如 **`GET`**。
- URL path (URL 路徑) (字串) — 請求的 URL 路徑。
- (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 GET API 在 `testxray` 階段的所有 `test` 請求。

- 規則名稱 — **`test-sampling`**
- 優先順序 — **`17`**
- 儲槽大小 — **`10`**
- 固定頻率 — **`10`**
- 服務名稱 — **`testxray/test`**
- 服務類型 — **`AWS::ApiGateway::Stage`**
- HTTP 方法 — **`GET`**
- 資源 ARN — **`*`**
- 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。

- 規則名稱 — **`prod-sampling`**
- 優先順序 — **`478`**
- 儲槽大小 — **`1`**
- 固定頻率 — **`60`**
- 服務名稱 — **`testxray/prod`**
- 服務類型 — **`AWS::ApiGateway::Stage`**
- HTTP 方法 — **`*`**
- 資源 ARN — **`*`**
- 主機 — **`*`**
- 屬性 — **`{}`**

## AWS X-Ray Amazon API Gateway APIs追蹤

本節討論 Amazon API Gateway APIs 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：

- 使用 AWS 管理主控台
- X-Ray 區段文件
- 概念

主題

- 為 API Gateway API 追蹤物件的範例
- 了解追蹤

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"
}
```

如需註釋的詳細資訊，請參閱《X-Ray 區段文件》，然後選擇 X-Ray 區段文件、註釋。

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。如需取樣規則的深入解說，請參閱設定取樣規則。

```
"aws": {
    "xray": {
        "sampling_rule_name": "Default"
    },
    "api_gateway": {
        "account_id": "123412341234",
        "rest_api_id": "a1b2c3d4e5",
        "stage": "dev",
        "request_id": "a1b2c3d4-a1b2-a1b2-a1b2-a1b2c3d4e5f6"
    }
}
```

如需有關 aws 物件的詳細資訊，請參閱《X-Ray 區段文件》，然後選擇 X-Ray 區段文件、AWS 資源資料。

了解追蹤

以下是 API Gateway 階段的追蹤區段。如需有關組成追蹤區段之欄位的詳細解說，請參閱《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": "abcdefgh12345678",
                        "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"
        }
```

# API Gateway 入口網站

您可以使用 API Gateway 入口網站，與網頁中的 API 消費者分享您的 APIs 和 API 文件。入口網站包含一組入口網站產品。每個入口網站產品都是 REST APIs 的邏輯分組，其中包含您為 API 取用者建立和發佈的文件。入口網站內的產品頁面包含入口網站產品層級的自訂文件。產品 REST 端點頁面包含每個 REST APIs的文件，以及 REST API 的路徑、方法和階段。產品頁面和產品 REST 端點頁面的組合提供完整的文件，讓 API 消費者了解如何開始使用 REST APIs。

使用入口網站執行下列動作：

- 將 API Gateway 用於整個 API 生命週期，從建立 APIs到記錄和分發它們。
- 跨 AWS 帳戶共用入口網站產品。
- 自訂和共用入口網站產品的中央位置，並提供產品頁面。

## 考量事項

下列考量事項可能會影響您使用 API Gateway 入口網站：

- 您只能建立入口網站來共用 REST APIs。
- 入口網站是在 AWS 區域層級建立。您的入口網站只能包含位於您建立入口網站之相同區域中的 REST APIs。
- 您只能使用 AWS 管理主控台 AWS CLI、 或 AWS SDKs 來建立和共用您的入口網站。
- 您只能使用 Amazon Cognito 使用者集區控制對入口網站的存取。您的入口網站消費者可以直接透過使用者集區登入，也可以透過第三方身分提供者 (IdP) 聯合。如果您使用 Amazon Cognito 使用者集區，則無法為您的使用者集區設定用戶端秘密。如需詳細資訊，請參閱 Amazon Cognito 使用者集區。

## API Gateway 中的入口網站產品

入口網站產品代表您要共用的服務或功能。您的入口網站產品是產品 REST 端點和產品頁面的集合。產品 REST 端點是入口網站產品的存取點，由 REST API 的路徑和方法及其部署的階段組成。產品頁面是您用來說明 API 消費者如何使用您產品端點的文件。入口網站產品可以包含整個`Prod`階段做為產品 REST 端點的集合，或僅包含部署到`Prod`階段`GET /pets`的資源做為單一產品 REST 端點。

您的入口網站產品是可自訂的。您可以新增自訂文件、重新命名產品 REST 端點、重組顯示順序、新增區段，以及跨 AWS 帳戶共用產品。若要讓入口網站產品的任何變更生效，您必須重新發佈使用入口網站產品的任何入口網站。

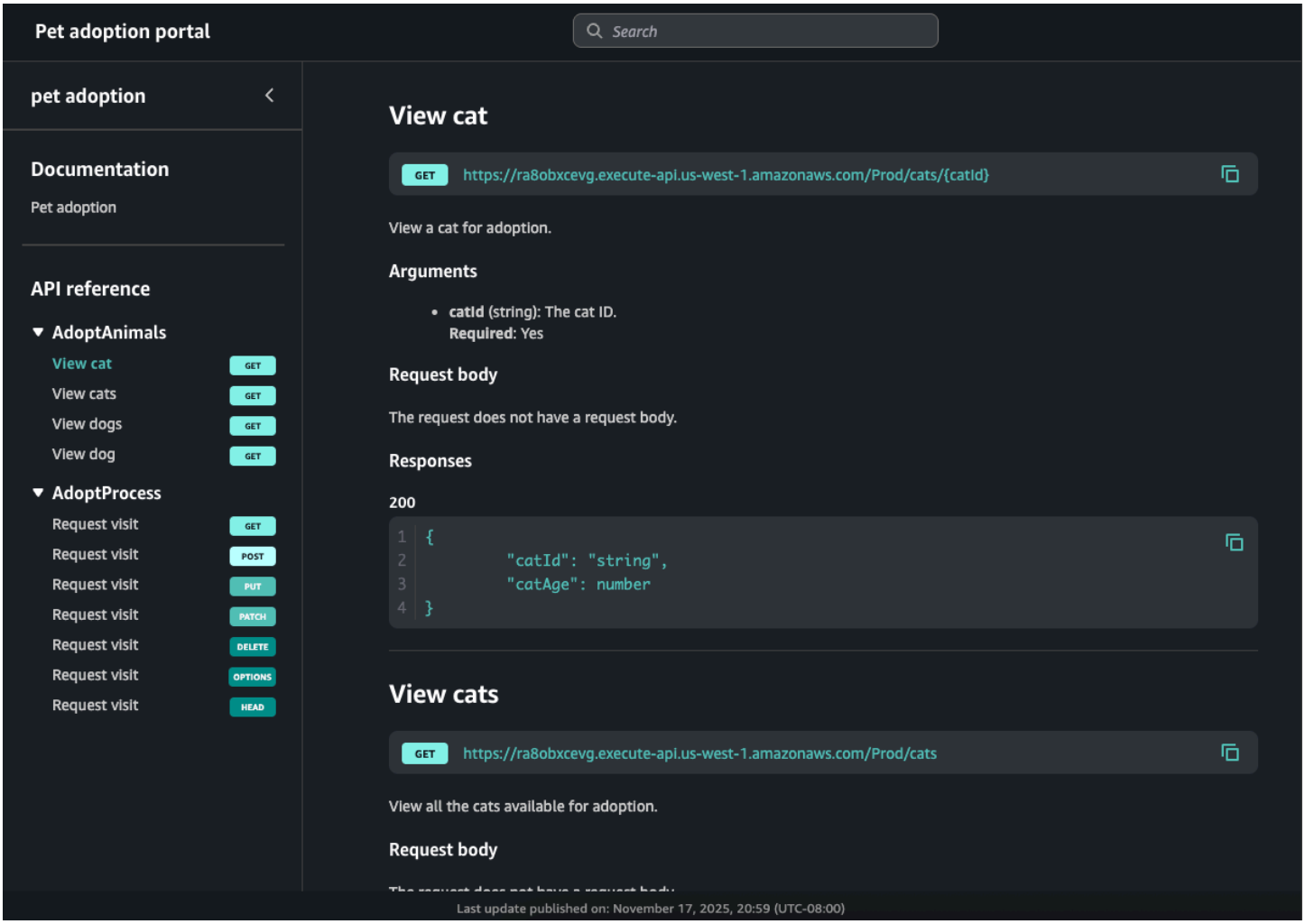
## 寵物採用入口網站產品範例

例如，您可以擁有多個代表寵物採用服務的 REST APIs。您可以使用 API Gateway 來建立pet adoption入口網站產品。此入口網站產品將協助客戶探索他們應該使用哪些 APIs 來滿足和採用寵物。此入口網站產品使用您已建立的 REST APIs，但可讓您重新分組和組織它們。您也可以提供文件，說明使用寵物採用入口網站產品的條款與條件，並讓客戶試用您的 APIs。此資訊全部存放在您的入口網站產品中。

下表顯示代表寵物採用入口網站產品的三個 APIs，及其對應的產品 REST 端點操作名稱和頁面區段名稱。

| REST API ID | REST API 路徑和方法 | REST API 階段 | 操作名稱 | 頁面區段 |
|---|---|---|---|---|
| kf5387miad | GET /dogs | 生產 | View dogs | AdoptAnimals |
| kf5387miad | GET /dogs/{dogId} | 生產 | View dog | AdoptAnimals |
| ra8obxcevg | GET /cats | 生產 | View cats | AdoptAnimals |
| ra8obxcevg | GET /cats/{catId} | 生產 | View cat | AdoptAnimals |
| h0rpx9cm62 | ANY /user/{userId}/{petId+} | 試用版 | Request visit | AdoptProcess |

在此範例中， AdoptAnimals區段中會將兩個 REST APIs kf5387miadra8obxcevg和 分組在一起。此排序導覽的結果看起來與入口網站中的以下內容類似：




由於 ANY方法是全部截獲方法，入口網站會顯示所有支援的 HTTP 方法。此入口網站也包含入口網站產品擁有者所建立的文件。

## 後續步驟

若要開始使用入口網站產品，您可以執行下列動作：

- 若要建立入口網站產品，請參閱 the section called “建立入口網站產品”。
- 若要了解 try it 功能，請參閱 the section called “針對入口網站中的產品 REST 端點啟用試用”。
- 若要了解產品頁面，請參閱 the section called “建立產品頁面”。
- 若要了解如何共用入口網站產品，請參閱 the section called “共用入口網站產品”。

建立入口網站產品之後，您可以將其發佈至入口網站。如需詳細資訊，請參閱the section called “建立入口網站”。

# 在 API Gateway 中建立入口網站產品

下列程序說明如何建立入口網站產品。入口網站是入口網站產品的集合。建立入口網站產品後，您可以建立產品 REST 端點和產品頁面。若要了解入口網站產品，請參閱 the section called “入口網站產品”。

## 考量事項

下列考量可能會影響您建立入口網站產品的方式：

- 您的入口網站產品可以同時包含私有和公有 REST APIs。試用功能不支援私有 APIs，因此在入口網站中具有視覺化差異。身為入口網站擁有者，您可能需要提供說明的文件。
- 如果您使用 AWS CLI AWS SDKs建立入口網站產品，您的入口網站將不會有任何產品端點或產品頁面。您需要使用 AWS CLI 或 主控台新增這些資源。若要了解如何建立產品 REST 端點，請參閱 the section called “建立產品 REST 端點”。若要了解如何建立產品頁面，請參閱 the section called “建立產品頁面”。

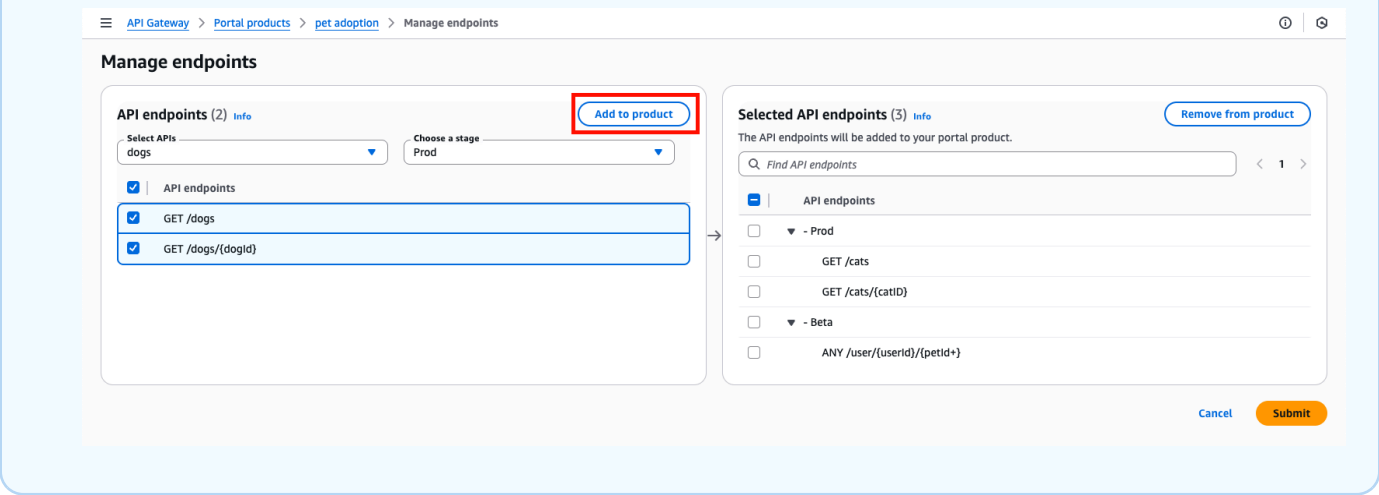
## 建立入口網站產品

下列程序說明如何建立入口網站產品。

建立入口網站產品

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 在主導覽窗格中，選擇入口網站產品。
3. 選擇建立產品。
4. 在產品名稱中，輸入入口網站產品的名稱。
5. 針對產品描述，輸入描述。
6. 選擇下一步。
7. 若要選取您的產品 REST 端點，請在 API 端點下選擇 API，然後選擇階段。
8. 若要將端點新增至您的產品 REST 端點，請選取 API 端點，然後選擇新增至產品。

> **Note**
>
> 在未先選擇新增至產品的情況下，請勿選擇下一步。




API 端點會出現在選取的 API 端點清單中。

9. 選擇下一步。
10. 檢閱您的選擇，然後選擇建立產品。

使用主控台建立入口網站產品後，所有產品頁面和產品 REST 端點頁面都是草稿，不會出現在入口網站中。若要讓消費者看到您的產品頁面和產品 REST 端點頁面，您需要將草稿新增至區段。如果您使用 AWS CLI 或 AWS SDKs 建立入口網站產品，請將草稿新增至 AWS CLI 命令中的 區段。無論您如何將草稿新增至區段，都必須發佈使用入口網站產品的入口網站，消費者才能看見它。

將草稿新增至頁面區段

1. 您的草稿會列在文件索引標籤中。您的產品頁面有草稿文件頁面，產品 REST 端點頁面有草稿 API 參考頁面。選擇草稿 API 參考頁面。
2. 選擇草稿 API 參考頁面。

   如果您沒有任何產品 REST 端點，則不會有任何 API 參考頁面草稿。若要了解如何建立產品 REST 端點，請參閱 the section called “建立產品 REST 端點”。
3. 選擇編輯頁面。
4. 在此頁面上，您可以覆寫任何現有的 API 文件組件，或使用 API Gateway 文件。若要允許消費者看到產品 REST 端點頁面的內容，請在區段名稱下輸入名稱。如果這是`/pets-GET`端點，則頁面名稱可能是 `Pets`。
5. 選擇儲存變更。
6. 您建立的新頁面名稱會顯示在 API 參考頁面區段下方。

若要允許消費者看見新頁面，您仍然需要重新發佈入口網站。如需詳細資訊，請參閱the section called “發佈入口網站”。

# 在 API Gateway 中建立產品 REST 端點

產品 REST 端點是您入口網站產品的存取點。每個端點都包含 REST API 的路徑和方法，以及其部署所在的階段。端點包含您為 REST API 定義的任何文件，也可以使用自訂文件進行更新。當您建立產品 REST 端點時，您可以設定下列設定：

- 產品 REST 端點的操作名稱。如果您的 REST API 具有路徑和方法 `GET /pets`，在入口網站中，名稱為 `/pets`。您可以使用 操作名稱，將產品 REST 端點名稱變更為 `Pet summaries`。這不會影響您現有的 API。
- 適用於您產品端點的 try it 功能。這可讓 API 消費者在您的APIs。如果您為私有 API 建立產品端點，則 try it 功能會設為 `UNAVAILABLE`。如需詳細資訊，請參閱the section called “針對入口網站中的產品 REST 端點啟用試用”。
- 如果您目前有為 REST API 定義的文件組件，您可以匯入文件或覆寫現有的文件，並使用 GitHub Flavored Markdown 建立新的產品層級文件。任何文件覆寫都不會傳播回您的 API Gateway API 文件組件。如果您匯入目前的文件組件，API Gateway 會跨 API 同步變更至您的產品 REST 端點。

## 建立產品 REST 端點

下列程序說明如何將產品 REST 端點新增至現有的入口網站產品。若要了解如何同時建立入口網站產品和產品 REST 端點，請參閱 the section called “建立入口網站產品”。

為產品入口網站建立產品 REST 端點

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 在主導覽窗格中，選擇入口網站產品。
3. 選擇入口網站產品。
4. 選擇關聯的端點索引標籤，然後選擇新增端點。
5. 若要選取您的產品 REST 端點，請在 API 端點下選擇 API，然後選擇階段。
6. 若要將端點新增至您的產品 REST 端點，請選取 API 端點，然後選擇新增至產品。

   API 端點會出現在選取的 API 端點清單中。
7. 選擇提交。

8. 您的產品 REST 端點頁面現在是草稿。選擇草稿 API 參考頁面，然後選擇您的產品 REST 端點頁面，以完成頁面的內容。
9. 選擇編輯頁面。
10. 若要覆寫現有的 API Gateway 文件，請針對文件來源開啟覆寫文件並執行下列動作：
    a. 針對顯示內容，選擇建立覆寫。
    b. 針對操作名稱，輸入新的操作名稱。
    c. 在頁面內文中，輸入您的自訂文件。
    d. （選用）針對端點，輸入入口網站中顯示的自訂端點。
11. 若要讓客戶在您的入口網站中叫用您的 API，請選取試用功能。如需詳細資訊，請參閱the section called “針對入口網站中的產品 REST 端點啟用試用”。
12. 針對區段名稱，輸入名稱或選擇現有區段。當您將產品 REST 端點新增至區段時，消費者可以在已發佈的入口網站上檢視您的內容。
13. 選擇儲存變更。

若要允許消費者看到新頁面，您仍然需要重新發佈入口網站。如需詳細資訊，請參閱the section called “發佈入口網站”。

## 在 API Gateway 中建立產品頁面

產品頁面可協助您的產品消費者了解和使用您的產品。文件位於產品層級，因此包含您產品的任何入口網站都有此文件。API Gateway 支援兩種以 GitHub Flavored Markdown 撰寫的產品頁面類型：

概觀頁面

當您建立產品時，API Gateway 會根據您在建立產品時提供的資訊，自動為您的產品產生概觀頁面。此頁面具有無法變更的其他格式。您可以修改文字或從入口網站產品中移除頁面。

自訂頁面

自訂頁面是以 Markdown 撰寫的頁面。您可以針對條款與條件或如何使用產品的教學使用本節。您可以將任何自訂頁面變更為概觀頁面。當您將自訂頁面變更為概觀頁面時，頁面內容會格式化為符合概觀頁面。

### 考量事項

下列考量事項可能會影響您使用產品頁面：

- 本文件不在 API 層級，且不包含任何方法、資源或任何請求參數。若要了解如何修改產品 API 層級的文件，請參閱 the section called “更新產品 REST 端點”。
- 您無法將.md檔案直接上傳至您的產品。
- 您可以修改入口網站產品的顯示順序，來修改產品頁面的順序。如需詳細資訊，請參閱the section called “更新入口網站產品”。
- 您需要重新發佈包含您產品的任何入口網站，文件變更才會生效。
- 如果您有多個入口網站使用相同的入口網站產品，則所有入口網站的產品頁面都相同。若要擁有相同的產品 REST 端點，但擁有不同的產品頁面，請建立多個產品。

## 建立產品頁面

下列程序說明如何建立產品頁面。

建立產品頁面

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 在主導覽窗格中，選擇入口網站產品。
3. 選擇產品。
4. 在文件下，選擇建立自訂頁面。
5. 針對頁面標題，輸入頁面標題。
6. 針對內文，使用 GitHub Flavored Markdown 輸入您的補充文件。

   預覽區段顯示您的內容在入口網站中的顯示方式。最終視覺效果樣式可能會根據您的入口網站設定而變更。
7. 選擇建立產品頁面。
8. 您的產品頁面現在是草稿。選擇草稿文件頁面，然後選擇您的產品頁面以完成頁面的內容。
9. 選擇編輯頁面。
10. 針對區段名稱，輸入名稱或選擇現有區段。當您將產品頁面新增至區段時，消費者可以在已發佈的入口網站上檢視您的內容。
11. 選擇儲存變更。

若要允許消費者看見新頁面，您仍然需要重新發佈入口網站。如需詳細資訊，請參閱the section called “發佈入口網站”。

# 在 API Gateway 中更新入口網站產品

當您更新入口網站產品時，您可以變更產品 REST 端點和產品頁面顯示方式的名稱、描述或顯示順序。若要修改顯示順序，您可以修改產品頁面的頁面清單，以及產品端點的區段名稱。如果您使用 AWS CLI 或 SDKs 執行此操作，您可以指定產品頁面 ARN 或產品 REST 端點 ARN。您需要為 API 消費者重新發佈任何使用您的 產品的入口網站，以查看入口網站中反映的變更。

## 更新入口網站產品

下列程序說明如何變更區段清單以重新排序產品 REST 端點，因為它們將出現在入口網站中。在此程序中，我們假設您在入口網站產品中至少有兩個產品 REST 端點。

更新入口網站產品

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 在主導覽窗格中，選擇入口網站產品。
3. 選擇產品。
4. 在文件索引標籤中，針對產品頁面，選擇管理。
5. 拖放產品 REST 端點以重新排序。您也可以將產品 REST 端點和產品頁面移至草稿頁面區段。草稿頁面中的頁面不會出現在您的入口網站中。
6. （選用）重新命名產品 REST 端點名稱或新增新的 API 參考區段。這些變更不會影響您APIs。
7. 選擇儲存變更。

# 在 API Gateway 中更新產品 REST 端點

當您更新產品 REST 端點時，您可以變更產品端點的所有設定，但 REST API 的路徑和方法及其部署的階段除外。產品端點的任何變更都會自動更新，但您需要重新發佈使用您的產品的任何入口網站，以查看入口網站中反映的變更。

## 更新產品 REST 端點

下列程序說明如何更新產品 REST 端點，以使用自訂 API 參考文件覆寫目前的 API 文件。

更新產品 REST 端點

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 在主導覽窗格中，選擇入口網站產品。
3. 選擇產品。

4. 在文件索引標籤的 API 參考頁面下，選擇產品 REST 端點的名稱，例如 /dogs - GET。
5. 針對預覽，選擇編輯頁面。
6. 針對文件來源，開啟覆寫文件。
7. 針對顯示內容，選擇覆寫現有內容。

   如果您選擇移除所有內容，則會從頁面中移除內容，但您可以選擇覆寫現有內容以再次存取文件。
8. 在頁面內文中，使用 GitHub Flavored Markdown 輸入自訂 API 參考文件。

   本文件不會傳播回 API Gateway。
9. 選擇儲存變更。
10. 您必須重新發佈任何使用此入口網站產品的入口網站，變更才會生效。

## 在 API Gateway 中更新產品頁面

當您更新產品頁面時，您可以變更產品頁面的所有設定。產品頁面的任何變更都會自動更新，但您需要重新發佈使用您的產品的任何入口網站，以查看入口網站中反映的變更。

## 在 API Gateway 中共用入口網站產品

您可以跨 AWS 帳戶 使用 共用入口網站產品 AWS RAM。當您共用入口網站產品時，另一個帳戶可以在自己的入口網站中使用您的入口網站產品。使用共用入口網站產品，您可以建立組織的 APIs 的單一目錄，並在整個 API 生態系統中強制執行控管標準。同時，共享入口網站產品可讓 API 供應商靈活地在其自己的帳戶中開發、測試和維護 APIs。

### 考量事項

下列考量可能會影響您共用入口網站資源的方式：

- 當您與其他帳戶共用產品時，該帳戶無法修改 REST API 的任何屬性。這包括整合端點、授權策略或階段組態。
- 當您將另一個帳戶的入口網站產品新增至入口網站時，入口網站產品擁有者無法檢視或控制入口網站的任何其他屬性。入口網站產品擁有者只會知道入口網站正在使用該產品。
- API Gateway 入口網站產品會在 AWS 區域 層級共用。
- 您可以與多個主體使用一個資源分享，並在建立資源分享後，為其新增更多主體。建議您盡可能重複使用資源分享。
- 如果兩個帳戶都在使用 的相同組織中 AWS Organizations，則會自動接受資源共用。您仍然需要使用 AWS RAM建立資源分享。

- 如果兩個帳戶都在使用 的相同組織中， AWS Organizations 且組織內的資源共用已啟用，則與您共用的組織中的任何主體都會自動獲得資源共用的存取權。不需要邀請函的，可以略過資源分享。
- 如果您共用產品的帳戶在 12 小時內不接受資源共用，您必須再次共用資源。
- 在您建立資源共享之後， 會 AWS RAM 更新產品的產品共享政策，以防止在沒有明確存取的情況下allow存取委託人。如需詳細資訊，請參閱《IAM 使用者指南》中的判斷帳戶中的請求獲得允許或遭到拒絕。

  更新的資源政策如下所示：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "apigateway:GetProduct",
                "apigateway:ListEndpoints",
                "apigateway:ListPages",
                "apigateway:GetEndpoint",
                "apigateway:GetPage"
            ],
            "Resource": [
                "arn:aws:apigateway:us-east-1:111122223333:/portalproducts/product-id",
                "arn:aws:apigateway:us-east-1:111122223333:/portalproducts/product-id/*"
            ]
        }
    ]
}
```

  AWS RAM 已透過新增下列項目，阻止主體沒有明確允許將產品新增至其入口網站的存取權：

```
"StringNotEquals": { "aws:PrincipalAccount": "555555555555" }
```

若要了解如何共用產品，請參閱 the section called “與入口網站擁有者共用您的入口網站產品”。若要了解如何將與您共用的產品新增至入口網站，請參閱 the section called “將共用入口網站產品新增至您的入口網站”。

# 在 API Gateway 中與入口網站擁有者共用您的入口網站產品

身為產品擁有者，您可以與其他 AWS 帳戶 使用 的入口網站擁有者共用您的產品 AWS RAM。您可以維持對所有產品頁面和產品 REST 端點頁面的完全控制。

## 考量事項

下列考量可能會影響您共用入口網站產品的方式：

- 您可以隨時共用或取消共用您的入口網站產品。如果您在發佈的入口網站中使用入口網站產品時取消共用入口網站產品，入口網站擁有者將無法檢視入口網站產品、檢視對其所做的任何更新，或重新發佈或預覽入口網站，直到移除入口網站產品為止。
- 您可以檢視已與其他 帳戶共用的入口網站產品。
- 除非您自行建立入口網站，否則您無法修改任何包含入口網站產品的入口網站。

## 分享您的入口網站產品

下列程序說明如何建立資源共享。

### AWS 管理主控台

若要使用 AWS 管理主控台，請參閱AWS RAM 《 使用者指南》中的在 中建立資源共享 AWS RAM。

針對選取資源類型，選擇 API Gateway 產品。

### AWS CLI

下列 create-resource-share 會為您的私有自訂網域名稱建立資源分享。資源和委託人可能需要幾分鐘的時間才能完成關聯。對於主體，請提供帳戶 ID 或 Organizations ID，例如 `arn:aws:organizations::123456789012:organization/o-1234abcd`。您可以為資源分享提供多個主體。

```
aws ram create-resource-share \
 --region us-west-2 \
 --name portal-product-resource-share \
 --permission-arns arn:aws:ram::aws:permission/
AWSRAMDefaultPermissionAPIGatewayDeveloperPortalProduct \
 --resource-arns arn:aws:apigateway:us-west-2:111122223333:/portalproducts/
p000000000 \
 --principals 222222222222
```

若要取消共用入口網站產品，請使用 AWS RAM 刪除資源共用。

您可以隨時修改產品共用政策，以修改哪些主體可以在其入口網站中使用您的入口網站產品。

更新產品共用政策

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 在主導覽窗格中，選擇入口網站產品。
3. 選擇產品。
4. 在產品共用索引標籤中，針對產品共用政策，選擇管理產品共用政策。
5. 如果您之前從未共用過您的產品，您的產品共用政策將拒絕對入口網站產品的其他帳戶的所有存取。您需要更新政策，以允許特定帳戶存取您的入口網站產品。
6. 更新產品共用政策後，請選擇儲存變更。

## 在 API Gateway 中將共用入口網站產品新增至您的入口網站

身為入口網站擁有者，您可以使用入口網站中其他帳戶與您共用的入口網站產品。您可以維護入口網站的完整控制權。

考量事項

下列考量可能會影響您共用入口網站資源的方式：

- 您必須接受產品擁有者的產品資源共享，才能在入口網站中使用其產品。如果您和產品擁有者位於相同的 Organizations， AWS RAM 可以為您完成一些共用步驟。
- 如果產品擁有者在您發佈的入口網站中取消共用產品，該產品仍會顯示在入口網站上，但在您移除此產品之前，您將無法再次發佈入口網站。身為入口網站擁有者，您有責任從入口網站移除產品，因為產品擁有者無法為您執行此操作。
- 您可以檢視入口網站產品、產品頁面和產品 REST 端點頁面，但您無法修改任何這些資源。
- 如果產品與您共用，您就無法與其他帳戶共用。

（選用）接受資源共用

產品擁有者建立資源共享後，您有 12 小時的時間可以接受它。如果您使用 AWS Organizations 與產品擁有者位於相同的組織中，則系統會自動接受共用。如果您所在的組織已啟用自動分享資源，則會自動與您分享資源。

AWS 管理主控台

若要使用 AWS 管理主控台，請參閱AWS RAM 《 使用者指南》中的接受和拒絕資源共用邀請。

AWS CLI

若要尋找與您共用的所有資源，請使用下列 get-resource-share-invitations 命令：

```
aws ram get-resource-share-invitations \
    --region us-west-2
```

使用產生的資源分享 ARN，接受資源分享邀請。下列 accept-resource-share-invitation 命令接受資源分享。

```
aws ram accept-resource-share-invitation \
    --resource-share-invitation-arn arn:aws:ram:us-west-2:123456789012:resource-
share-invitation/1e3477be-4a95-46b4-bbe0-c4001EXAMPLE \
    --region us-west-2
```

將共用產品新增至您的入口網站

接受資源共享後，您可以將產品新增至入口網站。

AWS 管理主控台

將共用產品新增至入口網站

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 在主導覽窗格中，選擇入口網站。
3. 選擇入口網站。
4. 在產品索引標籤中，針對入口網站產品，選擇新增產品。
5. 將產品新增至您的入口網站。共用產品會在產品清單中顯示為 shared 。
6. 選擇儲存變更。

AWS CLI

若要尋找與您共用的所有入口網站產品，請使用下列get-portal-products命令：

```
aws apigatewayv2 get-portal-products \
```

```
    --resource-owner OTHER_ACCOUNTS \
    --region us-west-2
```

若要將共用入口網站產品新增至您的入口網站，請使用下列update-portal命令：

```
aws apigateway update-portal \
    --included-portal-product-arns arn:aws:apigateway:us-west-2:111122223333:/
portalproducts/p000000000 \
    --region us-west-2
```

## 針對入口網站中的 API Gateway 產品 REST 端點啟用試用

使用嘗試讓 API 取用者從您的入口網站叫用您的產品端點。當 API 取用者使用 試用時，他們會輸入方法請求參數並叫用您的產品端點。然後，API Gateway 會透過公有網際網路叫用您的 API，並在 try it 視窗中傳回回應。您可以使用自訂產品頁面，提供有關存取 API 所需參數的任何其他資訊。

API Gateway 使用以下限制來保護您的 APIs：

- API Gateway 每秒只允許 3 個請求到您的 API。
- API Gateway 使用內建逾時限制 29000 毫秒。您的實際 API 可能會有較高的逾時限制，但當客戶使用 API Gateway 時，不會套用此逾時。
- API Gateway 會將回應承載限制為 6MB。

### 考量事項

下列考量可能會影響您使用它的方式：

- 當您預覽入口網站時，不支援嘗試。
- 具有下列功能的 REST APIs 不支援嘗試：
  - 私有 API
  - 使用交互 TLS APIs
  - 使用私有或自我簽署 SSL 憑證APIs

  身為入口網站擁有者，您必須負責與 API 消費者通訊，因為不支援的任何 REST APIs 沒有為什麼要嘗試按鈕。API Gateway 不會為您解釋這一點。

### 針對產品 REST 端點啟用試用

下列程序說明如何為產品端點啟用試用。

若要了解如何在入口網站中使用試用，請參閱 the section called “使用入口網站”。

針對產品 REST 端點啟用試用

1. 在主導覽窗格中，選擇入口網站產品。
2. 選擇產品。
3. 在文件索引標籤的 API 參考頁面下，選擇產品 REST 端點的名稱，例如 /dogs - GET。
4. 選擇編輯頁面。
5. 若要讓客戶在您的入口網站中叫用您的 API，請選取試用功能。
6. 選擇儲存變更。
7. 您必須重新發佈任何使用此入口網站產品的入口網站，變更才會生效。

## 在 API Gateway 中刪除入口網站產品

當您刪除入口網站產品時，無法復原。

### 刪除入口網站產品

下列程序說明如何刪除入口網站產品。

刪除入口網站產品

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 在主導覽窗格中，選擇入口網站產品。
3. 選擇入口網站產品。
4. 選擇 刪除。
5. 確認您的選擇，然後選擇刪除。
6. 您必須重新發佈任何使用此入口網站產品的入口網站，變更才會生效。

# 在 API Gateway 中建立入口網站

入口網站是 產品的集合。您可以控制入口網站的授權、品牌和發佈。當您建立入口網站時，其發佈狀態為 Disabled。這表示您的入口網站無法在 Web 上探索。若要讓消費者存取您的入口網站，您需要發佈它。在發佈入口網站之前，您也可以預覽入口網站。

API Gateway 支援下列瀏覽器上的入口網站：

- Firefox
- Google Chrome
- Microsoft Edge
- Safari

## 選擇網域名稱

根據預設，您的入口網站 URL 託管於下列 Amazon 擁有的網域名稱：

```
https://p-portalId.apigw-portal.us-east-1.on.aws
```

若要自訂網域名稱，您可以提供您擁有的網域名稱和自訂網域的 SSL 憑證。憑證必須在美國東部 AWS Certificate Manager（維吉尼亞北部）使用 (ACM) 進行管理，並由 Amazon CloudFront 支援。如需詳細資訊，請參閱搭配 CloudFront 使用 SSL/TLS 憑證的要求。

當您使用自訂網域名稱時，API Gateway 會建立 CloudFront 分佈，以 ACM 憑證保護傳輸中。然後，API Gateway 會提供別名記錄，您將其新增至 DNS 組態。別名記錄是 Amazon 擁有的網域名稱。您必須新增此別名記錄，才能將流量導向自訂網域的 CloudFront 分佈。您提供的網域名稱必須是唯一的，且尚未由現有 CloudFront 分佈上的別名清單使用。

如果您使用自己的網域名稱，API Gateway 會使用TLS_1_2安全政策來保護您的入口網站。如需詳細資訊，請參閱檢視器之間支援的通訊協定和密碼。您無法變更此值。

## 建立入口網站

您可以使用 API Gateway 主控台來建立和設定入口網站。下列程序會引導您完成所有必要的步驟。您也可以選擇跳至檢閱，並在第一個程序結束時建立 ，以建立空白入口網站。

下列程序說明如何建立入口網站。首先，定義您的入口網站詳細資訊。

定義您的入口網站詳細資訊

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 在主導覽窗格中，選擇入口網站。
3. 選擇 Create portal (建立入口網站)。
4. 在入口網站詳細資訊下，針對入口網站名稱輸入名稱。
5. 對於入口網站描述，輸入描述。
6. 針對入口網站授權，選取如何授權您的入口網站。
   - 如果這是您第一次建立入口網站，建議您選擇無。如果您打算將任何生產 APIs 新增至入口網站，請使用需要授權。
   - 若要要求入口網站消費者驗證以使用您的入口網站，請選擇需要授權，然後執行下列動作：
     a. 針對 Amazon Cognito 使用者集區，選取您的使用者集區。

        使用者集區必須與入口網站位於相同的區域。
     b. 針對應用程式用戶端 ID，選擇應用程式用戶端 ID。

        在您的應用程式用戶端中，您必須將回呼 URL 設定為入口網站的預設 URL。
     c. 對於使用者集區網域，輸入使用者集區網域。如需詳細資訊，請參閱設定使用者集區網域。
7. 針對網域組態，選取入口網站的網域名稱。
   - 如果這是您第一次建立入口網站，我們建議您選擇預設網域。發佈入口網站後，即可在 取得`https://p-portalId.apigw-portal.us-east-1.on.aws`。
   - 若要提供您擁有的自訂網域名稱，請選取自訂網域名稱並執行下列動作：
     a. 針對網域名稱，輸入完整網域名稱。
     b. 針對 ACM 憑證，選擇涵蓋網域名稱的 ACM 憑證。

        您需要將 API Gateway 佈建的別名記錄新增至 DNS 記錄，以便從入口網站提供流量。
8. 若要將 CloudWatch RUM 應用程式監視器新增至入口網站，請在 RUM 應用程式監視器中選擇應用程式監視器。如需詳細資訊，請參閱 CloudWatch RUM。
9. 選擇下一步。

定義入口網站詳細資訊後，您可以新增入口網站產品。您可以隨時變更入口網站中的入口網站產品，但您需要重新發佈入口網站，變更才會生效。

選取您的入口網站產品

1. 從資源清單中，選取要新增至入口網站的產品。
2. 選擇下一步。

選取產品後，您可以自訂入口網站設計。若要上傳入口網站標誌，您必須將映像上傳至 Amazon S3。您可以隨時變更這些值，但您需要重新發佈入口網站，變更才會生效。

（選用）自訂您的入口網站

1. 對於入口網站標誌，請提供 檔案的 S3 URI。

   僅.jpg支援 .png、 .svg或 。
2. 若要自訂入口網站的設計，您有兩個選項：

   a. 您可以選擇入口網站佈景主題，在入口網站上套用預先製作的顏色佈景主題。

   b. 您可以使用顏色選擇器來選取所有入口網站設計選項。
3. 選擇下一步。

選取入口網站設計後，請檢閱入口網站的資訊。

檢閱您的入口網站

- 在檢閱和建立頁面上檢閱您的設定。當您滿意設定時，請選擇 Create (建立)。

如果您使用 Amazon Cognito 使用者集區來控制對入口網站的存取，則必須在應用程式用戶端中將回呼 URL 設定為入口網站的預設 URL。如需詳細資訊，請參閱應用程式用戶端的應用程式特定設定。

在您建立入口網站之後，消費者將無法使用該入口網站。您必須發佈入口網站，才能在網際網路上存取。如需詳細資訊，請參閱the section called “發佈入口網站”。

# 在 API Gateway 中更新入口網站

您可以在入口網站中更新佈景主題、標誌或入口網站產品。當您更新入口網站時，在您重新發佈入口網站之前，變更不會生效。

## 更新入口網站

下列程序說明如何更新入口網站的設計。

更新入口網站設計

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 在主導覽窗格中，選擇入口網站。
3. 選擇入口網站。
4. 在入口網站設計索引標籤中，選擇編輯。
5. 若要更新入口網站標誌，請選擇更新入口網站標誌。您可以移除現有的標誌或使用新的標誌。
6. 對您的入口網站設計進行任何佈景主題或背景更新。
7. 選擇儲存變更。
8. （選用）選擇產生入口網站預覽查看您的變更。

   在您確認變更之後，請發佈您的入口網站。
9. 選擇發佈入口網站。
10. 輸入入口網站更新的描述，然後選擇發佈。

# 在 API Gateway 中預覽入口網站

您可以在發佈入口網站之前預覽入口網站。API Gateway 會建立短期 URL，可用來預覽客戶在發佈入口網站後將會看到的入口網站版本。

## 考量事項

下列考量可能會影響您預覽入口網站的方式：

- 預覽入口網站未啟用嘗試。如需詳細資訊，請參閱the section called “針對入口網站中的產品 REST 端點啟用試用”。

## 預覽入口網站

下列程序說明如何預覽入口網站。

預覽入口網站

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 在主導覽窗格中，選擇入口網站。
3. 選擇入口網站。
4. 選擇產生入口網站預覽。

   請勿重新整理您的頁面。
5. API Gateway 產生預覽後，請選擇開啟入口網站預覽。

   您的入口網站預覽將在新索引標籤中開啟。

# 在 API Gateway 中發佈入口網站

若要讓 API 取用者存取您的入口網站，您必須發佈它。網際網路上的任何人都可以探索入口網站 URL。建議您在發佈入口網站之前，先預覽並保護其安全。

## 考量事項

發佈您的入口網站可能需要幾分鐘的時間。您可以在 主控台中監控發佈狀態。

## 發佈入口網站

下列程序說明如何發佈入口網站。

發佈入口網站

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 在主導覽窗格中，選擇入口網站。
3. 選擇入口網站。
4. 選擇發佈入口網站。
5. （選用）針對變更描述，輸入變更的描述。

   當您發佈入口網站時，我們建議您一律提供變更的簡短說明。
6. 選擇 Publish (發布)。

   API Gateway 需要幾分鐘的時間才能完成發佈您的入口網站。API Gateway 會在入口網站可用時提供其連結。

若要刪除入口網站，您必須先將其停用。如需詳細資訊，請參閱the section called “停用入口網站”。

# 使用 API Gateway 入口網站

發佈入口網站之後，即可在 Web 上使用。若要使用您的入口網站，請複製入口網站預設網域名稱或網域名稱，然後在瀏覽器中輸入。如果您的入口網站使用網域名稱，則預設網域名稱無法用於存取您的入口網站。

## 使用入口網站

下列程序說明如何存取入口網站，並針對產品 REST 端點使用試用入口網站。

使用入口網站

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 在主導覽窗格中，選擇入口網站。
3. 選擇入口網站。
4. 複製入口網站預設網域名稱或網域名稱，並在瀏覽器中輸入。
5. 在入口網站首頁上，您可以搜尋入口網站產品和探索產品頁面。使用入口網站頂端的搜尋來搜尋入口網站中的所有內容。使用產品下的搜尋列來搜尋入口網站產品。
6. 選擇入口網站產品。
7. 選擇產品 REST 端點。
8. 在嘗試它視窗中，輸入參數以嘗試呼叫您的 API。
9. 選擇試用。API Gateway 將調用您的 API，並在視窗中顯示結果。

## 對入口網站進行故障診斷

以下提供使用入口網站時可能遇到的錯誤和問題的疑難排解建議。

### 無法存取入口網站網頁網站

如果您收到無法連線入口網站網頁網站的錯誤訊息，請確定您已發佈入口網站。如果您使用網域名稱，請確定您已新增 API Gateway 提供的別名記錄，以將流量導向自訂網域的 CloudFront 分佈。

### 嘗試失敗

如果您無法針對產品 REST 端點使用 try it，請確定 try it 狀態為 `ACTIVE`。此外，請確定您輸入正確的參數來叫用 API。

### 頁面不會出現在您的入口網站中

如果您建立了產品 REST 端點頁面或產品頁面，但入口網站中看不到它，請確定您的頁面不是草稿。若要驗證，請執行下列動作：

驗證您的頁面不在草稿中

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 在主導覽窗格中，選擇入口網站產品。
3. 選擇您的入口網站產品。
4. 在文件索引標籤中，確認您的頁面不在草稿文件頁面或草稿 API 參考頁面區段中，但在自訂文件頁面或 API 參考頁面中。

## 在 API Gateway 中停用入口網站

當您停用入口網站時，您的入口網站不再可在 Web 上使用。停用入口網站之後，您可以刪除入口網站，或視需要重新發佈。

### 停用入口網站

下列程序說明如何停用入口網站。

停用入口網站

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 在主導覽窗格中，選擇入口網站。
3. 選擇入口網站。
4. 選擇動作，然後選擇停用入口網站。
5. 確認您的選擇，然後選擇停用入口網站。

   發佈狀態將變更為已停用。

若要移除入口網站的停用狀態，請再次發佈入口網站。

# 在 API Gateway 中刪除入口網站

當您刪除入口網站時，該入口網站無法復原，您必須先將其停用。如果您想要從 Web 移除入口網站，您可以停用它。這可讓您修改入口網站，並在稍後重新發佈。

## 刪除入口網站

下列程序說明如何刪除入口網站。

刪除入口網站

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 在主導覽窗格中，選擇入口網站。
3. 選擇入口網站。
4. 如果您的入口網站未停用，請選擇動作、停用入口網站來停用它。

   停用入口網站需要幾分鐘的時間。您可以監控發佈狀態，以查看入口網站發佈狀態的停用時間。
5. 取消發佈入口網站後，您可以將其刪除。若要刪除入口網站，請選擇動作、刪除入口網站。確認您的選擇，然後選擇刪除。

# API Gateway HTTP API

REST API 和 HTTP API 都是 RESTful API 產品。REST API 支援比 HTTP API 更多的功能，而 HTTP API 的設計具有最少功能，因此它們能以較低的價格提供。如需更多詳細資訊，請參閱 the section called “在 REST API 與 HTTP API 之間選擇 ”。

您可以使用 HTTP API 將請求傳送到 AWS Lambda 函數或任何可路由的 HTTP 端點。例如，您可以在後端建立與 Lambda 函數整合的 HTTP API。當用戶端呼叫您的 API 時，API Gateway 會將請求傳送到該 Lambda 函數並傳回該函數的回應給用戶端。

HTTP API 支援 OpenID Connect 和 OAuth 2.0 授權。它們提供跨來源資源共享 (CORS) 和自動部署的內建支援。

您可以使用 AWS 管理主控台、AWS CLI、API、CloudFormation 或軟體開發套件來建立 HTTP API。

主題

- 在 API Gateway 中開發 HTTP API
- 發布 HTTP API 以供客戶調用
- 在 API Gateway 中保護您的 HTTP API
- 在 API Gateway 中監控 HTTP API
- 疑難排解 API Gateway 中的 HTTP API 問題

## 在 API Gateway 中開發 HTTP API

本節提供開發 API Gateway API 時所需的 API Gateway 功能的詳細資料。

在開發 API Gateway API 時，您可以決定 API 的許多特性。這些特性取決於 API 的使用案例。例如，您可能只允許特定用戶端呼叫您的 API，或是您可能希望讓所有人都可以使用它。您可能需要 API 呼叫來執行 Lambda 函數、進行資料庫查詢或呼叫應用程式。

主題

- 建立 HTTP API
- 在 API Gateway 中建立 HTTP API 的路由
- API Gateway 中 HTTP API 的 IP 位址類型
- 在 API Gateway 中控制和管理對 HTTP API 的存取
- 在 API Gateway 中建立 HTTP API 的整合

- 在 API Gateway 中設定 HTTP API 的 CORS
- 在 API Gateway 中轉換 HTTP API 的 API 請求和回應
- 在 API Gateway 中使用 HTTP API 的 OpenAPI 定義

# 建立 HTTP API

若要建立可操作的 API，您必須至少有一個路由、整合、階段和部署。

下列範例示範如何建立具有 AWS Lambda 或 HTTP 整合的 API、路由，以及設定為自動部署變更的預設階段。

本指南假設您已經熟悉 API Gateway 和 Lambda。如需更詳細的清單，請參閱開始使用。

主題

- 使用 建立 HTTP API AWS 管理主控台
- 使用 CLI AWS 建立 HTTP API

## 使用 建立 HTTP API AWS 管理主控台

1. 開啟 API Gateway 主控台。
2. 選擇 Create API (建立 API)。
3. 在 HTTP API 下，選擇 Build (組建)。
4. 選擇 Add integration (新增整合)，然後選擇 AWS Lambda 函數或輸入 HTTP 端點。
5. 在 Name (名稱) 中，輸入 API 的名稱。
6. 選擇 Review and create (檢閱和建立)。
7. 選擇 Create (建立)。

現在您的 API 已準備好叫用。您可以透過在瀏覽器中輸入其調用 URL 或使用 Curl 來測試您的 API。

```
curl https://api-id.execute-api.us-east-2.amazonaws.com
```

## 使用 CLI AWS 建立 HTTP API

您可以使用快速建立來建立具有 Lambda 或 HTTP 整合的 API、預設的全部捕獲路由，以及設定為自動部署變更的預設階段。以下 create-api 命令會使用快速建立來建立 API，以便與後端的 Lambda 函式整合。

> **Note**
>
> 若要叫用 Lambda 整合，API Gateway 必須具有必要的許可。您可以使用以資源為基礎的政策或 IAM 角色來授與 API Gateway 叫用 Lambda 函數的許可。若要進一步了解，請參閱《AWS Lambda 開發人員指南》中的AWS Lambda 許可。

Example

```
aws apigatewayv2 create-api --name my-api --protocol-type HTTP --target
 arn:aws:lambda:us-east-2:123456789012:function:function-name
```

現在您的 API 已準備好叫用。您可以透過在瀏覽器中輸入其調用 URL 或使用 Curl 來測試您的 API。

```
curl https://api-id.execute-api.us-east-2.amazonaws.com
```

## 在 API Gateway 中建立 HTTP API 的路由

將直接傳入的 API 請求路由到後端資源。路由由兩部分組成：HTTP 方法和資源路徑，例如 `GET /pets`。您可以為您的路由定義特定的 HTTP 方法。或者，您可以使用 `ANY` 方法來比對您尚未為資源定義的所有方法。您可以建立一個 `$default` 路由，充當不與任何其他路由搭配的請求的全部捕獲。

> **Note**
>
> API Gateway 會先解碼 URL 編碼的請求參數，然後再將其傳遞至後端整合。

### 使用路徑變數

您可以在 HTTP API 路由中使用路徑變數。

例如，`GET /pets/{petID}` 路由會擷取用戶端提交給 `https://api-id.execute-api.us-east-2.amazonaws.com/pets/6` 的 `GET` 請求。

Greedy 路徑變數會擷取路由的所有子資源。若要建立 Greedy 路徑變數，請將 `+` 新增至變數名稱，例如 `{proxy+}`。Greedy 路徑變數必須位於資源路徑結尾。

## 使用查詢字串參數

根據預設，API Gateway 會將查詢字串參數傳送至您的後端整合 (如果它們包含在對 HTTP API 的請求中)。

例如，當用戶端傳送要求給 `https://api-id.execute-api.us-east-2.amazonaws.com/pets?id=4&type=dog` 時，查詢字串參數 `?id=4&type=dog` 會傳送至您的整合。

## 使用 **$default** 路由

`$default` 路由會擷取未明確與 API 中其他路由相符的請求。

當 `$default` 路由收到請求時，API Gateway 會將完整的請求路徑傳送到整合。例如，您可以建立僅使用 `$default` 路由的 API，並將其與 `https://petstore-demo-endpoint.execute-api.com` HTTP 端點整合在 ANY 方法上。當您將請求傳送給 `https://api-id.execute-api.us-east-2.amazonaws.com/store/checkout` 時，API Gateway 會將請求傳送給 `https://petstore-demo-endpoint.execute-api.com/store/checkout`。

若要進一步了解 HTTP 整合，請參閱為 HTTP API 建立 HTTP 代理整合。

## 路由傳送 API 請求

當用戶端傳送 API 請求時，API Gateway 會先決定要將請求路由傳送到哪個階段。如果請求明確符合階段，API Gateway 會將請求傳送至該階段。如果沒有任何階段完全符合請求，API Gateway 會將請求傳送到 `$default` 階段。如果沒有 `$default` 階段，則 API 會傳回 `{"message":"Not Found"}` 且不會產生 CloudWatch 日誌。

選擇階段後，API Gateway 就會選擇路由。API Gateway 會使用下列優先順序來選取具有最特定相符項目的路由：

1. 完全相符的路由和方法。
2. 使用貪婪路徑變量 (`{proxy+}`) 配對路由和方法。
3. `$default` 路由。

如果沒有路由與請求相符，API Gateway 會將 `{"message":"Not Found"}` 傳回用戶端。

例如，假設具有 `$default` 階段的 API，以及下列範例路由：

1. `GET /pets/dog/1`

2. GET /pets/dog/{id}
3. GET /pets/{proxy+}
4. ANY /{proxy+}
5. $default

下表摘要說明 API Gateway 路由傳送請求到範例路由的方式。

| 要求 | 選取的路由 | 說明 |
| --- | --- | --- |
| GET https://*api-id*.execute-api.*region*.amazonaws.com/pets/dog/1 | GET /pets/dog/1 | 請求完全匹配此靜態路由。 |
| GET https://*api-id*.execute-api.*region*.amazonaws.com/pets/dog/2 | GET /pets/dog/{id} | 請求完全與此路由相符。 |
| GET https://*api-id*.execute-api.*region*.amazonaws.com/pets/cat/1 | GET /pets/{proxy+} | 請求不完全與路由相符。具備 GET 方法和貪婪路徑變量的路由會擷取這個請求。 |
| POST https://*api-id*.execute-api.*region*.amazonaws.com/test/5 | ANY /{proxy+} | ANY 方法與您尚未為路由定義的所有方法相符。貪婪路徑變量的路由具有比 $default 路由更高的優先順序。 |

## API Gateway 中 HTTP API 的 IP 位址類型

建立 API 時，您可以指定可調用 API 的 IP 位址類型。您可以選擇 IPv4 來解析 IPv4 位址以調用 API，也可以選擇雙堆疊，以同時允許 IPv4 和 IPv6 位址調用您的 API。您可能會希望將 IP 位址類型設定為雙堆疊，以緩解 IP 空間耗盡或用於安全狀態。如需雙堆疊 IP 位址類型優點的詳細資訊，請參閱 AWS 上的 IPv6。

## IP 位址類型的考量事項

下列考量事項可能會影響您使用 IP 位址類型：

- HTTP API 的預設 IP 位址類型為 IPv4。
- 如果您將現有 API 的 IP 位址類型從 IPv4 變更為雙堆疊，請確認控制存取 API 的任何政策都已更新，以便進行 IPv6 呼叫。當您變更 IP 位址類型時，變更會立即生效。
- 您的 API 可以對應至與您的 API 具有不同 IP 位址類型的自訂網域名稱。如果您停用預設 API 端點，則可能會影響呼叫方調用 API 的方式。

## 變更 HTTP API 的 IP 位址類型

您可以藉由更新 API 的組態來變更 IP 位址類型。您可以使用 AWS 管理主控台、AWS CLI、CloudFormation 或 AWS SDK 來更新 API 的組態。如果您變更 API 的 IP 位址類型，您不需重新部署 API 讓變更生效。

AWS 管理主控台

若要變更 HTTP API 的 IP 位址類型

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 選擇一個 HTTP API。
3. 針對 API 設定，選擇編輯。
4. 針對 IP 位址類型，選取 IPv4 或雙堆疊。
5. 選擇儲存。

   對 API 組態的變更會立即生效。

AWS CLI

以下 update-api 命令會將 API 更新為雙堆疊 IP 位址類型：

```
aws apigatewayv2 update-api \
    --api-id abcd1234 \
    --ip-address-type dualstack
```

輸出將如下所示：

```
{
    "ApiEndpoint": "https://abcd1234.execute-api.us-east-1.amazonaws.com",
    "ApiId": "abcd1234",
    "ApiKeySelectionExpression": "$request.header.x-api-key",
    "CreatedDate": "2025-02-04T22:20:20+00:00",
    "DisableExecuteApiEndpoint": false,
    "Name": "My-HTTP-API",
    "ProtocolType": "HTTP",
    "RouteSelectionExpression": "$request.method $request.path",
    "Tags": {},
    "NotificationUris": [],
    "IpAddressType": "dualstack"
}
```


# 在 API Gateway 中控制和管理對 HTTP API 的存取

API Gateway 支援多種機制來控制和管理 HTTP API 的存取：

- Lambda 授權方 會使用 Lambda 函數來控制 API 存取權。如需詳細資訊，請參閱使用 AWS Lambda 授權方控制對 HTTP APIs存取。
- JWT 授權方會使用 JSON 網路字符來控制對 API 的存取。如需詳細資訊，請參閱使用 API Gateway 中的 JWT 授權方來控制對 HTTP API 的存取。
- 標準 AWS IAM 角色和政策提供靈活且強大的存取控制。您可以使用 IAM 角色和政策來控制誰可以建立和管理您的 API，以及誰可以叫用它們。如需詳細資訊，請參閱使用 API Gateway 中的 IAM 授權控制對 HTTP API 的存取。

為了改善您的安全狀態，建議您為 HTTP API 上的所有路由設定授權方。您可能需要這樣做，才能符合各種合規架構。如需詳細資訊，請參閱《AWS Security Hub 使用者指南》中的 Amazon API Gateway 控制項。

## 使用 AWS Lambda 授權方控制對 HTTP APIs存取

您可以使用 Lambda 授權方來使用 Lambda 函數，以控制對 HTTP API 的存取權。然後，當用戶端呼叫您的 API 時，API Gateway 會叫用您的 Lambda 函數。API Gateway 會使用 Lambda 函數的回應來判斷用戶端是否可以存取您的 API。

### 裝載格式版本

授權方承載格式版本會指定 API Gateway 傳送到 Lambda 授權方的資料格式，以及 API Gateway 如何解釋 Lambda 的回應。如果您未指定承載格式版本， 預設 AWS 管理主控台 會使用最新版本。如果您使用 AWS CLI CloudFormation或 SDK 建立 Lambda 授權方，則必須指定 `authorizerPayloadFormatVersion`。支援的值為 `1.0` 和 `2.0`。

如果您需要與 REST API 相容，請使用 `1.0` 版。

下列範例顯示每個裝載格式版本的結構。

2.0

```
{
  "version": "2.0",
  "type": "REQUEST",
  "routeArn": "arn:aws:execute-api:us-east-1:123456789012:abcdef123/test/GET/
request",
  "identitySource": ["user1", "123"],
  "routeKey": "$default",
  "rawPath": "/my/path",
  "rawQueryString": "parameter1=value1&parameter1=value2&parameter2=value",
  "cookies": ["cookie1", "cookie2"],
  "headers": {
    "header1": "value1",
    "header2": "value2"
  },
  "queryStringParameters": {
    "parameter1": "value1,value2",
    "parameter2": "value"
  },
  "requestContext": {
    "accountId": "123456789012",
    "apiId": "api-id",
    "authentication": {
      "clientCert": {
        "clientCertPem": "CERT_CONTENT",
        "subjectDN": "www.example.com",
        "issuerDN": "Example issuer",
        "serialNumber": "1",
        "validity": {
          "notBefore": "May 28 12:30:02 2019 GMT",
          "notAfter": "Aug  5 09:36:04 2021 GMT"
```

```
        }
      }
    },
    "domainName": "id.execute-api.us-east-1.amazonaws.com",
    "domainPrefix": "id",
    "http": {
      "method": "POST",
      "path": "/my/path",
      "protocol": "HTTP/1.1",
      "sourceIp": "IP",
      "userAgent": "agent"
    },
    "requestId": "id",
    "routeKey": "$default",
    "stage": "$default",
    "time": "12/Mar/2020:19:03:58 +0000",
    "timeEpoch": 1583348638390
  },
  "pathParameters": { "parameter1": "value1" },
  "stageVariables": { "stageVariable1": "value1", "stageVariable2": "value2" }
}
```

1.0

```
{
  "version": "1.0",
  "type": "REQUEST",
  "methodArn": "arn:aws:execute-api:us-east-1:123456789012:abcdef123/test/GET/
request",
  "identitySource": "user1,123",
  "authorizationToken": "user1,123",
  "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": "..."
```

```
  },
  "queryStringParameters": {
    "QueryString1": "queryValue1"
  },
  "pathParameters": {},
  "stageVariables": {
    "StageVar1": "stageValue1"
  },
  "requestContext": {
    "path": "/request",
    "accountId": "123456789012",
    "resourceId": "05c7jb",
    "stage": "test",
    "requestId": "...",
    "identity": {
      "apiKey": "...",
      "sourceIp": "...",
      "clientCert": {
        "clientCertPem": "CERT_CONTENT",
        "subjectDN": "www.example.com",
        "issuerDN": "Example issuer",
        "serialNumber": "a1:a1:a1:a1:a1:a1:a1:a1:a1:a1:a1:a1:a1:a1:a1:a1",
        "validity": {
          "notBefore": "May 28 12:30:02 2019 GMT",
          "notAfter": "Aug  5 09:36:04 2021 GMT"
        }
      }
    },
    "resourcePath": "/request",
    "httpMethod": "GET",
    "apiId": "abcdef123"
  }
}
```

### Lambda 授權方回應格式

承載格式版本會決定 Lambda 函數必須傳回的回應結構。

#### 格式 1.0 的 Lambda 函數回應

如果您選擇 `1.0` 格式版本，Lambda 授權方必須傳回允許或拒絕存取 API 路由的 IAM 政策。您可以在政策中使用標準 IAM 政策語法。如需 IAM 政策範例，請參閱the section called “控制對 API 的呼叫存取權”。您可以使用 `$context.authorizer.`*`property`* 將內容屬性傳遞給 Lambda 整合或存取日

誌。context 物件為選擇性，claims是保留的預留位置，不能用作內容物件。如需詳細資訊，請參閱 the section called “記錄變數”。

Example

JSON

```
{
  "principalId": "abcdef",
  "policyDocument": {
    "Version":"2012-10-17",
    "Statement": [
      {
        "Action": "execute-api:Invoke",
        "Effect": "Allow|Deny",
        "Resource": "arn:aws:execute-api:{regionId}:{accountId}:{apiId}/{stage}/
{httpVerb}/[{resource}/[{child-resources}]]"
      }
    ]
  },
  "context": {
    "exampleKey": "exampleValue"
  }
}
```

格式 2.0 的 Lambda 函數回應

如果您選擇 2.0 格式版本，則可以從 Lambda 函數傳回布林值或使用標準 IAM 政策語法的 IAM 政策。要傳回一個布林值，請啟用授權方的簡易回應。下列範例會示範您必須編碼 Lambda 函數才能傳回的格式。該 context 物件是選用的物件。您可以使用 $context.authorizer.*property* 將內容屬性傳遞給 Lambda 整合或存取日誌。如需進一步了解，請參閱the section called “記錄變數”。

Simple response

```
{
  "isAuthorized": true/false,
  "context": {
    "exampleKey": "exampleValue"
  }
}
```

IAM policy

```
{
  "principalId": "abcdef",
  "policyDocument": {
    "Version":"2012-10-17",
    "Statement": [
      {
        "Action": "execute-api:Invoke",
        "Effect": "Allow|Deny",
        "Resource": "arn:aws:execute-api:{regionId}:{accountId}:{apiId}/{stage}/
{httpVerb}/[{resource}/[{child-resources}]]"
      }
    ]
  },
  "context": {
    "exampleKey": "exampleValue"
  }
}
```

範例 Lambda 授權方函數

下列範例 Node.js Lambda 函數會示範您需要從 Lambda 函數傳回的 `2.0` 承載格式版本的必要回應格式。

Simple response - Node.js

```
export const handler = async(event) => {
    let response = {
        "isAuthorized": false,
        "context": {
            "stringKey": "value",
            "numberKey": 1,
            "booleanKey": true,
            "arrayKey": ["value1", "value2"],
            "mapKey": {"value1": "value2"}
        }
    };

    if (event.headers.authorization === "secretToken") {
        console.log("allowed");
```

```
        response = {
            "isAuthorized": true,
            "context": {
                "stringKey": "value",
                "numberKey": 1,
                "booleanKey": true,
                "arrayKey": ["value1", "value2"],
                "mapKey": {"value1": "value2"}
            }
        };
    }

    return response;

};
```

### Simple response - Python

```
import json


def lambda_handler(event, context):
    response = {
        "isAuthorized": False,
        "context": {
            "stringKey": "value",
            "numberKey": 1,
            "booleanKey": True,
            "arrayKey": ["value1", "value2"],
            "mapKey": {"value1": "value2"}
        }
    }

    try:
        if (event["headers"]["authorization"] == "secretToken"):
            response = {
                "isAuthorized": True,
                "context": {
                    "stringKey": "value",
                    "numberKey": 1,
                    "booleanKey": True,
                    "arrayKey": ["value1", "value2"],
                    "mapKey": {"value1": "value2"}
```

```
                }
            }
            print('allowed')
            return response
        else:
            print('denied')
            return response
    except BaseException:
        print('denied')
        return response
```

### IAM policy - Node.js

```
export const handler = async(event) => {
  if (event.headers.authorization == "secretToken") {
    console.log("allowed");
    return {
      "principalId": "abcdef", // The principal user identification associated with
 the token sent by the client.
      "policyDocument": {
        "Version": "2012-10-17",
        "Statement": [{
          "Action": "execute-api:Invoke",
          "Effect": "Allow",
          "Resource": event.routeArn
        }]
      },
      "context": {
        "stringKey": "value",
        "numberKey": 1,
        "booleanKey": true,
        "arrayKey": ["value1", "value2"],
        "mapKey": { "value1": "value2" }
      }
    };
  }
  else {
    console.log("denied");
    return {
      "principalId": "abcdef", // The principal user identification associated with
 the token sent by the client.
      "policyDocument": {
        "Version": "2012-10-17",
```

```
        "Statement": [{
          "Action": "execute-api:Invoke",
          "Effect": "Deny",
          "Resource": event.routeArn
        }]
      },
      "context": {
        "stringKey": "value",
        "numberKey": 1,
        "booleanKey": true,
        "arrayKey": ["value1", "value2"],
        "mapKey": { "value1": "value2" }
      }
    };
  }
};
```

### IAM policy - Python

```
import json


def lambda_handler(event, context):
    response = {
        # The principal user identification associated with the token sent by
        # the client.
        "principalId": "abcdef",
        "policyDocument": {
            "Version": "2012-10-17",
            "Statement": [{
                "Action": "execute-api:Invoke",
                "Effect": "Deny",
                "Resource": event["routeArn"]
            }]
        },
        "context": {
            "stringKey": "value",
            "numberKey": 1,
            "booleanKey": True,
            "arrayKey": ["value1", "value2"],
            "mapKey": {"value1": "value2"}
        }
    }
```

```
    try:
        if (event["headers"]["authorization"] == "secretToken"):
            response = {
                # The principal user identification associated with the token
                # sent by the client.
                "principalId": "abcdef",
                "policyDocument": {
                    "Version": "2012-10-17",
                    "Statement": [{
                        "Action": "execute-api:Invoke",
                        "Effect": "Allow",
                        "Resource": event["routeArn"]
                    }]
                },
                "context": {
                    "stringKey": "value",
                    "numberKey": 1,
                    "booleanKey": True,
                    "arrayKey": ["value1", "value2"],
                    "mapKey": {"value1": "value2"}
                }
            }
            print('allowed')
            return response
        else:
            print('denied')
            return response
    except BaseException:
        print('denied')
        return response
```

身分識別來源

您可以選擇性地指定 Lambda 授權方的身分來源。身分來源會指定授權請求所需的資料位置。例如，您可以指定標頭或查詢字串值做為身分來源。如果您指定身分來源，用戶端必須將其包含在請求中。如果用戶端的請求不包含身分來源，則 API Gateway 不會叫用您的 Lambda 授權方，且用戶端會收到 401 錯誤。

下表說明適用於 Lambda 授權方的支援身分來源。

| 類型 | 範例 | 備註 |
| --- | --- | --- |
| 標頭值 | $request.header.*name* | 網域名稱需區分大小寫。 |
| 查詢字串值 | $request.querystring.*name* | 查詢字串名稱區分大小寫。 |
| 環境變數 | $context.*variableName* | 支援的內容變數值。 |
| 階段變數 | $stageVariables.*variableName* | 階段變數的值。 |

您也可以直接從 Lambda 函式傳回 `{"errorMessage" : "Unauthorized"}`，以將 401 錯誤傳回至用戶端。如果您直接從 Lambda 函式將 401 錯誤傳回至用戶端，則請勿在建立 Lambda 授權方時指定任何身分來源。

快取授權方回應

您可藉由指定 authorizerResultTtlInSeconds 來啟用 Lambda 授權方的快取。啟用授權方的快取時，API Gateway 會使用授權方的身分來源做為快取金鑰。如果用戶端在設定的 TTL 內的身分來源中指定相同的參數，則 API Gateway 會使用快取的授權方結果，而不是叫用您的 Lambda 函數。

若要啟用快取，您的授權方必須至少有一個身分來源。

如果您為授權方啟用簡單回應，授權方的回應會完全允許或拒絕符合快取身分來源值的所有 API 請求。如需更精細的許可，請停用簡易回應並傳回 IAM 政策。根據您的授權方而定，您的 IAM 政策可能需要控制多方的存取權。

預設情況下，API Gateway 會使用快取授權回應 API 的所有路由使用授權。若要快取每個路由的回應，請新增 `$context.routeKey` 至授權方的身分來源。

建立 Lambda 授權方

當您建立 Lambda 授權方時，您會指定可供 API Gateway 使用的 Lambda 函數。您必須授予 API Gateway 使用 Lambda 函數的資源政策或 IAM 角色叫用該函數的許可。以下 create-authorizer 命令會建立 Lambda 授權方：

```
aws apigatewayv2 create-authorizer \
    --api-id abcdef123 \
    --authorizer-type REQUEST \
```

```
    --identity-source '$request.header.Authorization' \
    --name lambda-authorizer \
    --authorizer-uri 'arn:aws:apigateway:us-west-2:lambda:path/2015-03-31/
functions/arn:aws:lambda:us-west-2:123456789012:function:my-function/invocations' \
    --authorizer-payload-format-version '2.0' \
    --enable-simple-responses
```

以下 add-permission 命令會更新 Lambda 函式的資源政策，以授予 API Gateway 調用函式的許可。如果 API Gateway 沒有叫用函數的許可，用戶端會收到 `500 Internal Server Error`。

```
aws lambda add-permission \
    --function-name my-authorizer-function \
    --statement-id apigateway-invoke-permissions-abc123 \
    --action lambda:InvokeFunction \
    --principal apigateway.amazonaws.com \
    --source-arn "arn:aws:execute-api:us-west-2:123456789012:api-
id/authorizers/authorizer-id"
```

在您建立了一個授權方並授予 API Gateway 叫用它的許可後，請更新您的路由以使用授權方。以下 update-route 命令會將 Lambda 授權方新增至路由。如果您的 Lambda 授權方使用授權快取，請務必更新政策以控制其他路由的存取。

```
aws apigatewayv2 update-route \
    --api-id abcdef123 \
    --route-id abc123 \
    --authorization-type CUSTOM \
    --authorizer-id def123
```

Lambda 授權方疑難排解

如果 API Gateway 無法叫用您的 Lambda 授權方，或者您的 Lambda 授權方傳回無效格式的回應，則用戶端會收到 `500 Internal Server Error`。

若要故障排除錯誤，請啟用 API 階段的存取記錄。在 `$context.authorizer.error` 記錄格式中包含日誌變數。

如果記錄表示 API Gateway 沒有叫用函數的許可，請更新函數的資源政策或提供 IAM 角色，以授予 API Gateway 叫用授權方的許可。

如果日誌指出您的 Lambda 函數傳回無效的回應，請確認 Lambda 函數傳回所需格式的回應。

# 使用 API Gateway 中的 JWT 授權方來控制對 HTTP API 的存取

您可以使用 JSON Web Tokens (JWT) 做為 OpenID Connect (OIDC) 和 OAuth 2.0 框架的一部分來限制用戶端存取您的 API。

如果您為 API 的路由設定 JWT 授權方，API Gateway 會驗證用戶端使用 API 請求提交的 JWT。API Gateway 會根據字符驗證，並選擇性地根據字符中的範圍，來允許或拒絕請求。如果您設定某個路由的範圍，字符至少必須包含其中一個路由的範圍。

您可以為 API 的每個路由設定不同的授權方，或為多個路由使用相同的授權方。

> **Note**
>
> 將 JWT 存取字符與其他類型的 JWT (如 OpenID Connect ID 字符) 加以區分並無標準機制。除非您需要 API 授權的 ID 字符，否則建議您將路由設定為需要授權範圍。您也可以將 JWT 授權方設定為僅在發行 JWT 存取字符時要求身分提供者使用的發行者或對象。

## 使用 JWT 授權方來授予 API 請求權限

API Gateway 使用下列一般工作流程來授權請求對設定為使用 JWT 授權方的路由。

1. 檢查權杖的 `identitySource`。`identitySource` 只能包含字符，或字首加上 `Bearer` 的字符。
2. 解碼字符
3. 使用從發行者的 `jwks_uri` 獲取的公開金鑰檢查字符的演算法和簽章。目前只支援以 RSA 為基礎的演算法。API Gateway 會盡力快取公有金鑰最多兩個小時。在快取遺漏時，API Gateway 會透過網路從發行者擷取公有金鑰，這會將延遲新增至觸發它的請求。低請求率或間歇性請求率的 APIs 更可能遇到這些擷取。由於公有金鑰最多可保持快取兩小時，因此輪換金鑰時請允許寬限期，在此期間，舊金鑰和新金鑰都有效。
4. 驗證宣告。API Gateway 會評估下列權杖宣告：
   - `kid` – 權杖必須有符合簽署權杖之 `jwks_uri` 中金鑰的標頭宣告。
   - `iss` – 必須符合為授權方設定的 `issuer`。
   - `aud` 或 `client_id` – 必須符合為授權方設定的其中一個 `audience` 項目。API Gateway 僅於 `aud` 不存在時驗證 `client_id`。當 `aud` 和 `client_id` 同時存在時，API Gateway 會評估 `aud`。
   - `exp` – 必須在目前時間 (以 UTC 表示) 之後。
   - `nbf` – 必須在目前時間 (以 UTC 表示) 之前。

- `iat` – 必須在目前時間 (以 UTC 表示) 之前。
- `scope` 或 `scp` – 權杖至少必須包含路由的 `authorizationScopes` 的其中一個範圍。

如果上述任何步驟失敗，API Gateway 會拒絕 API 請求。

驗證 JWT 後，API Gateway 會將字符中的宣告傳遞給 API 路由的整合。後端資源 (例如 Lambda 函數) 可以存取 JWT 宣告。例如，如果 JWT 包含身分宣告 `emailID`，則可在 `$event.requestContext.authorizer.jwt.claims.emailID` 中供 Lambda 整合使用。如需 API Gateway 傳送至 Lambda 整合之承載的詳細資訊，請參閱the section called “AWS Lambda 整合”。

建立一個 JWT 授權器

建立 JWT 授權方之前，您必須向身分提供者註冊用戶端應用程式。您還必須建立一個 HTTP API。如需建立 HTTP API 的範例，請參閱建立 HTTP API。

使用主控台建立 JWT 授權方

下列步驟顯示如何使用主控台建立 JWT 授權方。

使用主控台建立 JWT 授權方

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 選擇一個 HTTP API。
3. 在主要導覽窗格中，選擇授權。
4. 選擇管理授權方標籤。
5. 選擇建立。
6. 對於授權方類型，選擇 JWT。
7. 設定您的 JWT 授權方，並指定定義字符來源的身分來源。
8. 選擇建立。

使用 建立 JWT 授權方 AWS CLI

以下 create-authorizer 命令會建立 JWT 授權方。對於 `jwt-configuration`，為您的身分提供者指定 `Audience` 和 `Issuer`。如果您使用 Amazon Cognito 做為身分提供者，則 `IssuerUrl` 為 `https://cognito-idp.us-east-2.amazonaws.com/userPoolID`。

```
aws apigatewayv2 create-authorizer \
```

```
    --name authorizer-name \
    --api-id api-id \
    --authorizer-type JWT \
    --identity-source '$request.header.Authorization' \
    --jwt-configuration Audience=audience,Issuer=IssuerUrl
```

使用 建立 JWT 授權方 AWS CloudFormation

下列 CloudFormation 範本會使用 Amazon Cognito 做為身分提供者的 JWT 授權方來建立 HTTP API。

CloudFormation 範本的輸出是 Amazon Cognito 託管 UI 的 URL，用戶端可以在其中註冊和登入以接收 JWT。用戶端登入後，會使用 URL 中的存取字符重新導向至您的 HTTP API。若要使用存取字符調用 API，請將 URL 中的 # 變更為 ？，以使用字符做為查詢字串參數。

範本範例 CloudFormation

```
AWSTemplateFormatVersion: '2010-09-09'
Description: |
  Example HTTP API with a JWT authorizer. This template includes an Amazon Cognito user
 pool as the issuer for the JWT authorizer
  and an Amazon Cognito app client as the audience for the authorizer. The outputs
 include a URL for an Amazon Cognito hosted UI where clients can
  sign up and sign in to receive a JWT. After a client signs in, the client is
 redirected to your HTTP API with an access token
  in the URL. To invoke the API with the access token, change the '#' in the URL to a
 '?' to use the token as a query string parameter.

Resources:
  MyAPI:
    Type: AWS::ApiGatewayV2::Api
    Properties:
      Description: Example HTTP API
      Name: api-with-auth
      ProtocolType: HTTP
      Target: !GetAtt MyLambdaFunction.Arn
  DefaultRouteOverrides:
    Type: AWS::ApiGatewayV2::ApiGatewayManagedOverrides
    Properties:
      ApiId: !Ref MyAPI
      Route:
        AuthorizationType: JWT
        AuthorizerId: !Ref JWTAuthorizer
  JWTAuthorizer:
```

```
    Type: AWS::ApiGatewayV2::Authorizer
    Properties:
      ApiId: !Ref MyAPI
      AuthorizerType: JWT
      IdentitySource:
        - '$request.querystring.access_token'
      JwtConfiguration:
        Audience:
        - !Ref AppClient
        Issuer: !Sub https://cognito-idp.${AWS::Region}.amazonaws.com/${UserPool}
      Name: test-jwt-authorizer
  MyLambdaFunction:
    Type: AWS::Lambda::Function
    Properties:
      Runtime: nodejs18.x
      Role: !GetAtt FunctionExecutionRole.Arn
      Handler: index.handler
      Code:
        ZipFile: |
          exports.handler = async (event) => {
              const response = {
                  statusCode: 200,
                  body: JSON.stringify('Hello from the ' + event.routeKey + ' route!'),
              };
              return response;
          };
  APIInvokeLambdaPermission:
    Type: AWS::Lambda::Permission
    Properties:
      FunctionName: !Ref MyLambdaFunction
      Action: lambda:InvokeFunction
      Principal: apigateway.amazonaws.com
      SourceArn: !Sub arn:${AWS::Partition}:execute-api:${AWS::Region}:
${AWS::AccountId}:${MyAPI}/$default/$default
  FunctionExecutionRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Version: '2012-10-17       '
        Statement:
          - Effect: Allow
            Principal:
              Service:
              - lambda.amazonaws.com
```

```
            Action:
              - 'sts:AssumeRole'
      ManagedPolicyArns:
        - arn:aws:iam::aws:policy/service-role/AWSLambdaBasicExecutionRole
  UserPool:
    Type: AWS::Cognito::UserPool
    Properties:
      UserPoolName: http-api-user-pool
      AutoVerifiedAttributes:
        - email
      Schema:
        - Name: name
          AttributeDataType: String
          Mutable: true
          Required: true
        - Name: email
          AttributeDataType: String
          Mutable: false
          Required: true
  AppClient:
    Type: AWS::Cognito::UserPoolClient
    Properties:
      AllowedOAuthFlows:
        - implicit
      AllowedOAuthScopes:
        - aws.cognito.signin.user.admin
        - email
        - openid
        - profile
      AllowedOAuthFlowsUserPoolClient: true
      ClientName: api-app-client
      CallbackURLs:
        - !Sub https://${MyAPI}.execute-api.${AWS::Region}.amazonaws.com
      ExplicitAuthFlows:
        - ALLOW_USER_PASSWORD_AUTH
        - ALLOW_REFRESH_TOKEN_AUTH
      UserPoolId: !Ref UserPool
      SupportedIdentityProviders:
        - COGNITO
  HostedUI:
    Type: AWS::Cognito::UserPoolDomain
    Properties:
      Domain: !Join
        - '-'
```

```
        - - !Ref MyAPI
          - !Ref AppClient
      UserPoolId: !Ref UserPool
Outputs:
  SignupURL:
    Value: !Sub https://${HostedUI}.auth.${AWS::Region}.amazoncognito.com/login?
client_id=${AppClient}&response_type=token&scope=email+profile&redirect_uri=https://
${MyAPI}.execute-api.${AWS::Region}.amazonaws.com
```

更新路由以使用 JWT 授權方

您可以使用 主控台 AWS CLI、 或 AWS 開發套件來更新路由，以使用 JWT 授權方。

使用主控台更新路由以使用 JWT 授權方

下列步驟顯示如何使用主控台更新路由，以使用 JWT 授權方。

使用主控台建立 JWT 授權方

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 選擇一個 HTTP API。
3. 在主要導覽窗格中，選擇授權。
4. 選擇一種方法，然後從下拉式選單中選取您的授權方，然後選擇連接授權方。

使用 更新路由以使用 JWT 授權方 AWS CLI

以下 update-route 命令會將路由更新為使用 JWT 授權方：

```
aws apigatewayv2 update-route \
   --api-id api-id  \
   --route-id route-id  \
   --authorization-type JWT \
   --authorizer-id authorizer-id \
   --authorization-scopes user.email
```

## 使用 API Gateway 中的 IAM 授權控制對 HTTP API 的存取

您可以為 HTTP API 路由啟用 IAM 授權。啟用 IAM 授權時，用戶端必須使用 Signature 第 4 版 (SigV4) 以 AWS 登入資料簽署其請求。只有在用戶端具有路由的 `execute-api` 許可時，API Gateway 才會叫用您的 API 路由。

HTTP API 的 IAM 授權與 REST API 的 IAM 授權類似。

> **Note**
>
> HTTP API 目前不支援資源原則。

如需授予用戶端叫用 API 許可的 IAM 政策範例，請參閱the section called “控制對 API 的呼叫存取權”。

啟用路由的 IAM 授權

以下 update-route 命令會為 HTTP API 路由啟用 IAM 授權：

```
aws apigatewayv2 update-route \
    --api-id abc123 \
    --route-id abcdef \
    --authorization-type AWS_IAM
```

# 在 API Gateway 中建立 HTTP API 的整合

「整合」會將路由連接到後端資源。HTTP API 支援 Lambda 代理、AWS 服務和 HTTP 代理整合。例如，您可以設定 API `/signup` 路由的 `POST` 請求，以便與可處理客戶註冊的 Lambda 函數整合。

主題

- 在 APIs AWS Lambda 代理整合
- 為 HTTP API 建立 HTTP 代理整合
- 在 APIs Gateway 中建立 HTTP API AWS 的服務整合
- 在 API Gateway 中建立 HTTP API 的私有整合

## 在 APIs AWS Lambda 代理整合

Lambda 代理整合可讓您將 API 路由與 Lambda 函數整合。當用戶端呼叫您的 API 時，API Gateway 會將請求傳送到該 Lambda 函數並傳回該函數的回應給用戶端。如需建立 HTTP API 的範例，請參閱建立 HTTP API。

### 裝載格式版本

承載格式版本會指定 API Gateway 傳送到 Lambda 整合的事件格式，以及 API Gateway 如何解釋 Lambda 的回應。如果您未指定承載格式版本， 預設 AWS 管理主控台 會使用最新版本。如果您使用 AWS CLI CloudFormation或 SDK 建立 Lambda 整合，則必須指定 `payloadFormatVersion`。支援的值為 `1.0` 和 `2.0`。

如需有關如何設定 `payloadFormatVersion` 的詳細資訊，請參閱 create-integration。如需有關如何判斷現有整合之 `payloadFormatVersion` 的詳細資訊，請參閱 get-integration

### 承載格式差異

下列清單顯示 `1.0` 和 `2.0` 承載格式版本之間的差異：

- 格式 `2.0` 沒有 `multiValueHeaders` 或 `multiValueQueryStringParameters` 欄位。重複的標題與逗號相結合，並包含在 `headers` 欄位中。重複的查詢字串與逗號相結合，並包含在 `queryStringParameters` 欄位中。
- 格式 `2.0` 具有 `rawPath`。如果您使用 API 映射將您的階段連接到自訂網域名稱，`rawPath` 不會提供 API 映射值。使用格式 `1.0` 和 `path` 存取自訂網域名稱的 API 映射。
- 格式 `2.0` 包括新的 `cookies` 欄位。請求中的所有 Cookie 標頭都與逗號相結合並新增到該 `cookies` 欄位中。在對用戶端地回應中，每個 Cookie 都會變成 `set-cookie` 標題。

### 承載格式結構

下列範例顯示每個裝載格式版本的結構。所有標頭名稱均為小寫。

2.0

```
{
  "version": "2.0",
  "routeKey": "$default",
  "rawPath": "/my/path",
  "rawQueryString": "parameter1=value1&parameter1=value2&parameter2=value",
  "cookies": [
    "cookie1",
    "cookie2"
  ],
  "headers": {
    "header1": "value1",
    "header2": "value1,value2"
  },
```

```
  "queryStringParameters": {
    "parameter1": "value1,value2",
    "parameter2": "value"
  },
  "requestContext": {
    "accountId": "123456789012",
    "apiId": "api-id",
    "authentication": {
      "clientCert": {
        "clientCertPem": "CERT_CONTENT",
        "subjectDN": "www.example.com",
        "issuerDN": "Example issuer",
        "serialNumber": "a1:a1:a1:a1:a1:a1:a1:a1:a1:a1:a1:a1:a1:a1:a1:a1",
        "validity": {
          "notBefore": "May 28 12:30:02 2019 GMT",
          "notAfter": "Aug  5 09:36:04 2021 GMT"
        }
      }
    },
    "authorizer": {
      "jwt": {
        "claims": {
          "claim1": "value1",
          "claim2": "value2"
        },
        "scopes": [
          "scope1",
          "scope2"
        ]
      }
    },
    "domainName": "id.execute-api.us-east-1.amazonaws.com",
    "domainPrefix": "id",
    "http": {
      "method": "POST",
      "path": "/my/path",
      "protocol": "HTTP/1.1",
      "sourceIp": "192.0.2.1",
      "userAgent": "agent"
    },
    "requestId": "id",
    "routeKey": "$default",
    "stage": "$default",
    "time": "12/Mar/2020:19:03:58 +0000",
```

```
    "timeEpoch": 1583348638390
  },
  "body": "Hello from Lambda",
  "pathParameters": {
    "parameter1": "value1"
  },
  "isBase64Encoded": false,
  "stageVariables": {
    "stageVariable1": "value1",
    "stageVariable2": "value2"
  }
}
```

1.0

```
{
  "version": "1.0",
  "resource": "/my/path",
  "path": "/my/path",
  "httpMethod": "GET",
  "headers": {
    "header1": "value1",
    "header2": "value2"
  },
  "multiValueHeaders": {
    "header1": [
      "value1"
    ],
    "header2": [
      "value1",
      "value2"
    ]
  },
  "queryStringParameters": {
    "parameter1": "value1",
    "parameter2": "value"
  },
  "multiValueQueryStringParameters": {
    "parameter1": [
      "value1",
      "value2"
    ],
    "parameter2": [
```

```
      "value"
    ]
  },
  "requestContext": {
    "accountId": "123456789012",
    "apiId": "id",
    "authorizer": {
      "claims": null,
      "scopes": null
    },
    "domainName": "id.execute-api.us-east-1.amazonaws.com",
    "domainPrefix": "id",
    "extendedRequestId": "request-id",
    "httpMethod": "GET",
    "identity": {
      "accessKey": null,
      "accountId": null,
      "caller": null,
      "cognitoAuthenticationProvider": null,
      "cognitoAuthenticationType": null,
      "cognitoIdentityId": null,
      "cognitoIdentityPoolId": null,
      "principalOrgId": null,
      "sourceIp": "192.0.2.1",
      "user": null,
      "userAgent": "user-agent",
      "userArn": null,
      "clientCert": {
        "clientCertPem": "CERT_CONTENT",
        "subjectDN": "www.example.com",
        "issuerDN": "Example issuer",
        "serialNumber": "a1:a1:a1:a1:a1:a1:a1:a1:a1:a1:a1:a1:a1:a1:a1:a1",
        "validity": {
          "notBefore": "May 28 12:30:02 2019 GMT",
          "notAfter": "Aug  5 09:36:04 2021 GMT"
        }
      }
    },
    "path": "/my/path",
    "protocol": "HTTP/1.1",
    "requestId": "id=",
    "requestTime": "04/Mar/2020:19:15:17 +0000",
    "requestTimeEpoch": 1583349317135,
    "resourceId": null,
```

```
    "resourcePath": "/my/path",
    "stage": "$default"
  },
  "pathParameters": null,
  "stageVariables": null,
  "body": "Hello from Lambda!",
  "isBase64Encoded": false
}
```

Lambda 函數回應格式

承載格式版本會決定 Lambda 函數必須傳回的回應結構。

格式 1.0 的 Lambda 函數回應

有了 `1.0` 格式版本，Lambda 整合就必須以下列 JSON 格式傳回回應：

Example

```
{
    "isBase64Encoded": true|false,
    "statusCode": httpStatusCode,
    "headers": { "headername": "headervalue", ... },
    "multiValueHeaders": { "headername": ["headervalue", "headervalue2", ...], ... },
    "body": "..."
}
```

格式 2.0 的 Lambda 函數回應

有了 `2.0` 格式版本，API Gateway 可以為您推斷回應格式。如果您的 Lambda 函數傳回有效的 JSON 但未傳回 `statusCode`，則 API Gateway 會進行下列假設：

- `isBase64Encoded` 是 `false`。
- `statusCode` 是 `200`。
- `content-type` 是 `application/json`。
- `body` 是函數的回應。

下列範例顯示 Lambda 函數和 API Gateway 解譯的輸出。

| Lambda 函數輸出 | API Gateway 解譯 |
| --- | --- |
| `"Hello from Lambda!"` | `{ "isBase64Encoded": false, "statusCode": 200, "body": "Hello from Lambda!", "headers": { "content-type": "application/json" } }` |
| `{ "message": "Hello from Lambda!" }` | `{ "isBase64Encoded": false, "statusCode": 200, "body": "{ \"message\": \"Hello from Lambda!\" }", "headers": { "content-type": "application/json" } }` |

若要自定義回應，您的 Lambda 函數應該會傳回下列格式的回應。

```
{
    "cookies" : ["cookie1", "cookie2"],
    "isBase64Encoded": true|false,
    "statusCode": httpStatusCode,
    "headers": { "headername": "headervalue", ... },
    "body": "Hello from Lambda!"
}
```

## 為 HTTP API 建立 HTTP 代理整合

HTTP 代理整合可讓您將 API 路由連線至公開可路由的 HTTP 端點。有了這種整合類型，API Gateway 會在前端和後端之間傳遞整個請求和回應。

若要建立 HTTP 代理整合，請提供公開可路由 HTTP 端點的 URL。

HTTP 代理與路徑變數整合

您可以在 HTTP API 路由中使用路徑變數。

例如，路由 `/pets/{petID}` 捕獲請求 `/pets/6`。您可以參考整合 URI 中的路徑變數，將變數的內容傳送至整合。例如，`/pets/extendedpath/{petID}`。

您可以使用 Greedy 路徑變量來捕獲路由的所有子資源。若要建立 Greedy 路徑變數，請將 + 新增至變數名稱，例如 `{proxy+}`。

若要設定包含 HTTP 代理整合以捕獲所有請求的路由，請使用 Greedy 路徑變數 (例如，`/parent/{proxy+}`) 建立 API 路由。將路由與 ANY 方法上的 HTTP 端點 (例如，`https://petstore-demo-endpoint.execute-api.com/petstore/{proxy}`) 整合。Greedy 路徑變數必須位於資源路徑結尾。

## 在 APIs Gateway 中建立 HTTP API AWS 的服務整合

您可以使用一級整合，將 HTTP API 與 AWS 服務整合。 一級整合會將 HTTP API 路由連接到 AWS 服務 API。當用戶端叫用由一級整合支援的路由時，API Gateway 會為您叫用 AWS 服務 API。例如，您可以使用一級整合將訊息傳送至 Amazon Simple Queue Service 佇列，或啟動 AWS Step Functions 狀態機器。如需支援的服務動作，請參閱the section called “AWS 服務整合參考”。

映射請求參數

一級整合具有必要和選用的參數。您必須設定所有必要的參數，才能建立整合。您可以使用在執行階段動態評估的靜態值或映射參數。如需支援整合與參數的完整清單，請參閱the section called “AWS 服務整合參考”。

下表說明受支援的映射請求參數。

| Type | 範例 | 備註 |
| --- | --- | --- |
| 標頭值 | $request.header.*name* | 網域名稱需區分大小寫。API Gateway 會將多個標頭值與逗號結合起來，例如 `"header1" : "value1,value2"` 。 |
| 查詢字串值 | $request.querystring.*name* | 查詢字串名稱區分大小寫。API Gateway 會將多個值與逗號結合起來，例如 `"querystr` |

| Type | 範例 | 備註 |
| --- | --- | --- |
| | | `ing1": "Value1,V alue2"` 。 |
| 路徑參數 | $request.path.*name* | 請求中的路徑參數值。例如，如果路由是 `/pets/{petId}` ，您可以透過 *`$request.path.petId`* 從請求映射 `petId` 參數。 |
| 請求內文傳遞 | $request.body | API Gateway 會傳遞整個請求內文。 |
| 請求內文 | $request.body.*name* | JSON 路徑表達式。不支援遞迴下降 (`$request.body..` *`name`*) 和篩選表達式 (`?(`*`expression`*` )`)。<br><br>**Note**<br>當您指定 JSON 路徑時，API Gateway 會在 100 KB 時截斷請求主體，然後套用選擇表達式。若要傳送大於 100 KB 的承載，請指定 `$request.body` 。 |
| 環境變數 | $context.*`variableName`* | 支援的內容變數值。 |
| 階段變數 | $stageVariables.*`variableName`* | 階段變數的值。 |
| 靜態值 | *`string`* | 常數值。 |

建立一級整合

建立一級整合之前，您必須建立授予 API Gateway 許可的 IAM 角色，以叫用您要整合 AWS 的服務動作。如需進一步了解，請參閱為 AWS 服務建立角色。

若要建立一級整合，請選擇支援的 AWS 服務動作，例如 SQS-SendMessage、設定請求參數，並提供授予 API Gateway 呼叫整合 AWS 服務 API 許可的角色。根據整合子類型，需要不同的請求參數。如需進一步了解，請參閱the section called “AWS 服務整合參考”。

以下 create-integration 命令會建立可傳送 Amazon SQS 訊息的整合：

```
aws apigatewayv2 create-integration \
    --api-id abcdef123 \
    --integration-subtype SQS-SendMessage \
    --integration-type AWS_PROXY \
    --payload-format-version 1.0 \
    --credentials-arn arn:aws:iam::123456789012:role/apigateway-sqs \
    --request-parameters '{"QueueUrl": "$request.header.queueUrl", "MessageBody":
 "$request.body.message"}'
```

使用 建立一級整合 CloudFormation

下列範例顯示 程式碼片段，該程式碼片段會建立與 CloudFormation Amazon EventBridge 進行一級整合的/{source}/{detailType}路由。

Source 參數會對應至 {source} 路徑參數、DetailType 對應至 {DetailType} 路徑參數，且 Detail 參數會對應至要求主體。

程式碼片段不會顯示授與 API Gateway 許可來調用 PutEvents 動作的事件匯流排或 IAM 角色。

```
Route:
    Type: AWS::ApiGatewayV2::Route
    Properties:
      ApiId: !Ref HttpApi
      AuthorizationType: None
      RouteKey: 'POST /{source}/{detailType}'
      Target: !Join
        - /
        - - integrations
          - !Ref Integration
  Integration:
    Type: AWS::ApiGatewayV2::Integration
```

```
    Properties:
      ApiId: !Ref HttpApi
      IntegrationType: AWS_PROXY
      IntegrationSubtype: EventBridge-PutEvents
      CredentialsArn: !GetAtt EventBridgeRole.Arn
      RequestParameters:
        Source: $request.path.source
        DetailType: $request.path.detailType
        Detail: $request.body
        EventBusName: !GetAtt EventBus.Arn
      PayloadFormatVersion: "1.0"
```

整合子類型參照

HTTP API 支援下列整合子類型。

整合子類型

- EventBridge-PutEvents 1.0
- SQS-SendMessage 1.0
- SQS-ReceiveMessage 1.0
- SQS-DeleteMessage 1.0
- SQS-PurgeQueue 1.0
- AppConfig-GetConfiguration 1.0
- Kinesis-PutRecord 1.0
- StepFunctions-StartExecution 1.0
- StepFunctions-StartSyncExecution 1.0
- StepFunctions-StopExecution 1.0

EventBridge-PutEvents 1.0

將自訂事件傳送到 Amazon EventBridge 以將這些事件與規則比對。

| 參數 | 必要 |
| --- | --- |
| Detail | True |
| DetailType | True |

| 參數 | 必要 |
| --- | --- |
| Source | True |
| Time | False |
| EventBusName | False |
| Resources | False |
| Region | False |
| TraceHeader | False |

若要進一步了解，請參閱《Amazon EventBridge API 參考》中的 PutEvents。

SQS-SendMessage 1.0

將訊息傳遞到指定的佇列。

| 參數 | 必要 |
| --- | --- |
| QueueUrl | True |
| MessageBody | True |
| DelaySeconds | False |
| MessageAttributes | False |
| MessageDeduplicationId | False |
| MessageGroupId | False |
| MessageSystemAttributes | False |
| Region | False |

若要進一步了解，請參閱《Amazon Simple Queue Service API 參考》中的 SendMessage。

SQS-ReceiveMessage 1.0

您從指定的佇列擷取一個或多個訊息 (最多可達 10 個)。

| 參數 | 必要 |
| --- | --- |
| QueueUrl | True |
| AttributeNames | False |
| MaxNumberOfMessages | False |
| MessageAttributeNames | False |
| ReceiveRequestAttemptId | False |
| VisibilityTimeout | False |
| WaitTimeSeconds | False |
| Region | False |

若要進一步了解，請參閱《Amazon Simple Queue Service API 參考》中的 ReceiveMessage。

SQS-DeleteMessage 1.0

從指定的佇列刪除指定的訊息。

| 參數 | 必要 |
| --- | --- |
| ReceiptHandle | True |
| QueueUrl | True |
| Region | False |

若要進一步了解，請參閱《Amazon Simple Queue Service API 參考》中的 DeleteMessage。

SQS-PurgeQueue 1.0

刪除指定佇列中的所有訊息。

| 參數 | 必要 |
| --- | --- |
| QueueUrl | True |
| Region | False |

若要進一步了解，請參閱《Amazon Simple Queue Service API 參考》中的 PurgeQueue。

AppConfig-GetConfiguration 1.0

接收組態的相關資訊。

| 參數 | 必要 |
| --- | --- |
| Application | True |
| Environment | True |
| Configuration | True |
| ClientId | True |
| ClientConfigurationVersion | False |
| Region | False |

如需進一步了解，請參閱《AWS AppConfig API 參考》中的 GetConfiguration。

Kinesis-PutRecord 1.0

將單一資料記錄寫入 Amazon Kinesis 資料串流。

| 參數 | 必要 |
| --- | --- |
| StreamName | True |
| Data | True |
| PartitionKey | True |

| 參數 | 必要 |
| --- | --- |
| SequenceNumberForOrdering | False |
| ExplicitHashKey | False |
| Region | False |

若要進一步了解，請參閱《Amazon Kinesis Data Streams API 參考》中的 PutRecord。

StepFunctions-StartExecution 1.0

啟動狀態機器的執行。

| 參數 | 必要 |
| --- | --- |
| StateMachineArn | True |
| Name | False |
| Input | False |
| Region | False |

如需進一步了解，請參閱《AWS Step Functions API 參考》中的 StartExecution。

StepFunctions-StartSyncExecution 1.0

啟動同步狀態機器執行。

| 參數 | 必要 |
| --- | --- |
| StateMachineArn | True |
| Name | False |
| Input | False |
| Region | False |

| 參數 | 必要 |
| --- | --- |
| TraceHeader | False |

如需進一步了解，請參閱《AWS Step Functions API 參考》中的 StartSyncExecution。

StepFunctions-StopExecution 1.0

停止執行。

| 參數 | 必要 |
| --- | --- |
| ExecutionArn | True |
| Cause | False |
| Error | False |
| Region | False |

如需進一步了解，請參閱《AWS Step Functions API 參考》中的 StopExecution。

## 在 API Gateway 中建立 HTTP API 的私有整合

私有整合可讓您在 VPC 中建立具有私有資源的 API 整合，例如 Application Load Balancer 或 Amazon ECS 容器型應用程式。

您可以使用私有整合來公開 VPC 中的資源，供 VPC 以外的用戶端存取。您可以使用 API Gateway 支援的任何授權方法來控制對您的 API 的存取。

> **Note**
>
> 若要建立私有整合，您必須先建立 VPC 連結。HTTP 和 REST APIs 現在都支援 VPC 連結 V2。若要進一步了解 VPC 連結 V2，請參閱 the section called “VPC 連結 V2”。

建立 VPC 連結 V2 之後，您可以設定連線至 Application Load Balancer、Network Load Balancer 或向 AWS Cloud Map 服務註冊的資源的私有整合。

考量事項

下列考量事項可能會影響您使用私有整合：

- 所有資源都必須由相同的 擁有 AWS 帳戶。這包括負載平衡器 AWS Cloud Map 或服務、VPC 連結和 HTTP API。
- 依預設，私有整合流量會使用 HTTP 通訊協定。若要使用 HTTPS，請指定 `tlsConfig`。若要使用執行此操作 AWS 管理主控台，當您建立私有整合時，請選擇進階設定，然後輸入安全伺服器名稱。
- 對於私有整合，API Gateway 會在對後端資源的請求中包含 API 端點的階段部分。例如，對 API `test` 階段的請求會包含 `test/route-path` 在私人整合的請求中。若要移除從請求至後端資源的階段名稱，請使用參數映射將請求路徑覆寫為 `$request.path`。

使用 Application Load Balancer 或 Network Load Balancer 建立私有整合

建立私有整合之前，您必須建立 VPC 連結 V2。若要進一步了解 VPC 連結 V2，請參閱 the section called “VPC 連結 V2”。

若要使用 Application Load Balancer 或 Network Load Balancer 建立私有整合，請建立 HTTP 代理整合、指定要使用的 VPC 連結，以及提供負載平衡器的接聽程式 ARN。

以下 create-integration 命令會建立私有整合，以使用 VPC 連結連線至負載平衡器：

```
aws apigatewayv2 create-integration --api-id api-id --integration-type HTTP_PROXY \
    --integration-method GET --connection-type VPC_LINK \
    --connection-id VPC-link-ID \
    --integration-uri arn:aws:elasticloadbalancing:us-east-2:123456789012:listener/app/
my-load-balancer/50dc6c495c0c9188/0467ef3c8400ae65
    --payload-format-version 1.0
```

使用 AWS Cloud Map 服務探索建立私有整合

建立私有整合之前，您必須建立 VPC 連結 V2。若要深入了解 VPC 連結，請參閱the section called “VPC 連結 V2”。

對於與 的整合 AWS Cloud Map，API Gateway 會使用 `DiscoverInstances` 來識別資源。您可以使用查詢參數來鎖定特定資源。已註冊資源的屬性必須包含 IP 位址和連接埠。API Gateway 會將請求分散於 `DiscoverInstances` 傳回的狀態良好資源。若要進一步了解，請參閱 AWS Cloud Map API 參考中的 DiscoverInstances。

> **Note**
>
> 如果您使用 Amazon ECS 在 中填入項目 AWS Cloud Map，則必須將 Amazon ECS 任務設定為搭配 Amazon ECS Service Discovery 使用 SRV 記錄，或開啟 Amazon ECS Service Connect。如需詳細資訊，請參閱《Amazon Elastic Container Service 開發人員指南》中的服務探索。

若要與 建立私有整合 AWS Cloud Map，請建立 HTTP 代理整合、指定要使用的 VPC 連結，並提供 AWS Cloud Map 服務的 ARN。

下列 create-integration 命令會建立使用 AWS Cloud Map 服務探索來識別資源的私有整合：

```
aws apigatewayv2 create-integration --api-id api-id --integration-type HTTP_PROXY  \
    --integration-method GET --connection-type VPC_LINK \
    --connection-id VPC-link-ID \
    --integration-uri arn:aws:servicediscovery:us-east-2:123456789012:service/srv-id?
stage=prod&deployment=green_deployment
    --payload-format-version 1.0
```

## 在 API Gateway 中設定 HTTP API 的 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。如果您無法存取 API 並收到包含 `Cross-Origin Request Blocked` 的錯誤訊息，您可能需要啟用 CORS。如需詳細資訊，請參閱什麼是 CORS？。

建立 Web 應用程式來存取託管在不同網域或來源上的 API，通常需要 CORS。您可以啟用 CORS 以允許來自不同網域上託管的 Web 應用程式對 API 的請求。例如，如果您的 API 託管在 `https://{api_id}.execute-api.{region}.amazonaws.com/` 上，並且您想要從 `example.com` 上託管的 Web 應用程式呼叫 API，則您的 API 必須支援 CORS。

如果您為 API 設定 CORS，即使沒有為您的 API 設定 OPTIONS 路由，API Gateway 也會自動傳送回應給預檢 OPTIONS 請求。對於 CORS 請求，API Gateway 會將設定的 CORS 標頭加入至整合的回應。

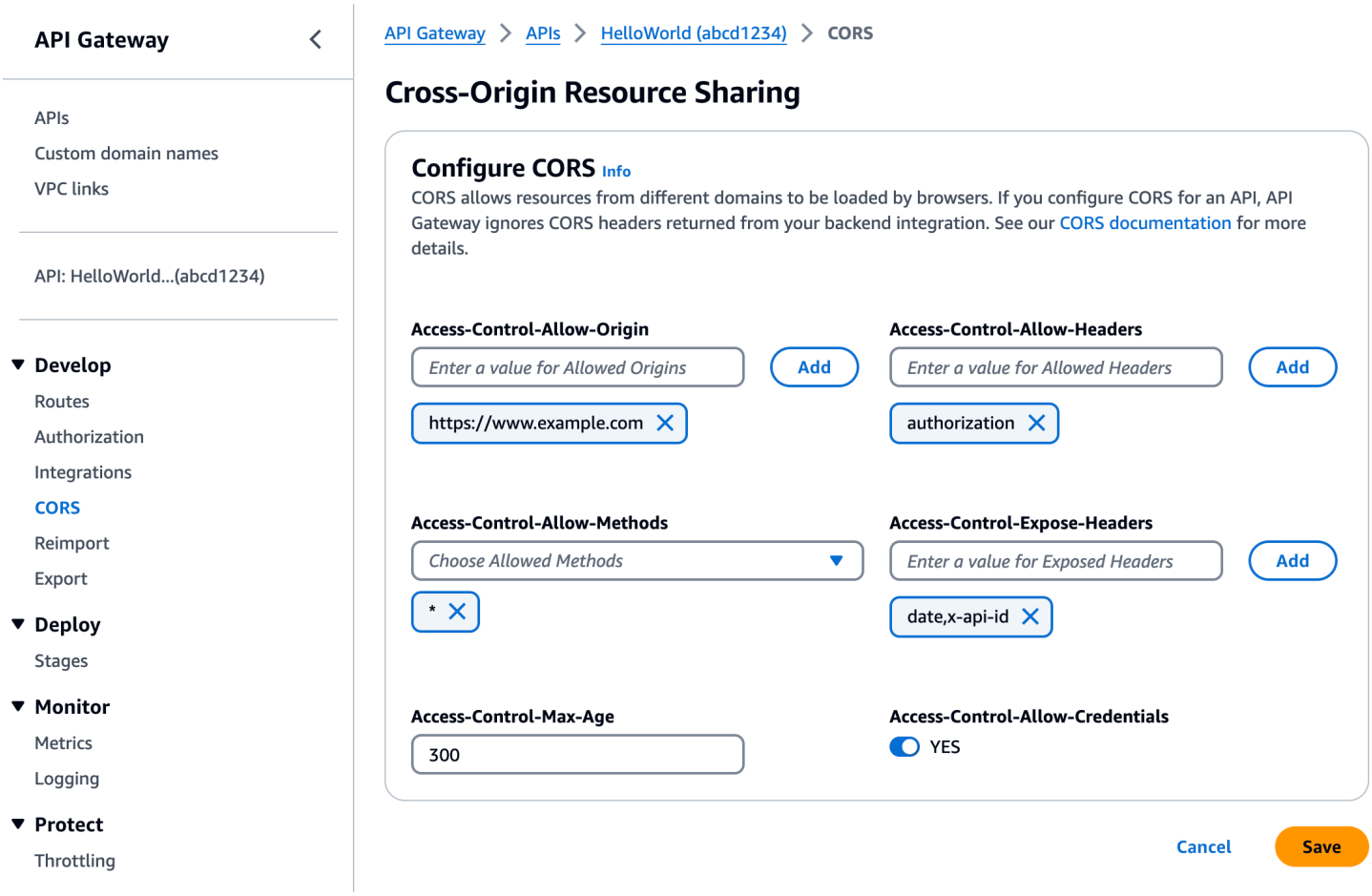
> **Note**
>
> 如果您為 API 設定 CORS，則 API Gateway 會忽略從後端整合傳回的 CORS 標頭。

您可以在 CORS 組態中指定下列參數。若要使用 API Gateway HTTP API 主控台新增這些參數，請在輸入值之後選擇新增。

| CORS 標頭 | CORS 組態屬性 | 範例值 |
| --- | --- | --- |
| Access-Control-Allow-Origin | allowOrigins | • `https://www.example.com`<br>• * (允許所有來源)<br>• `https://*` (允許以 https:// 開頭的任何來源)<br>• `http://*` (允許以 http:// 開頭的任何來源) |
| Access-Control-Allow-Credentials | allowCredentials | true |
| Access-Control-Expose-Headers | exposeHeaders | date、x-api-id、* |
| Access-Control-Max-Age | maxAge | 300 |
| Access-Control-Allow-Methods | allowMethods | GET、POST、DELETE、* |
| Access-Control-Allow-Headers | allowHeaders | authorization、* |

若要傳回 CORS 標頭，您的請求必須包含 `origin` 標頭。對於 `OPTIONS` 方法，您的請求必須包含 `origin` 標頭和 `Access-Control-Request-Method` 標頭。

您的 CORS 組態可能看起來如下：




## 為具有 **$default** 路由和授權方的 HTTP API 設定 CORS

您可以啟用 CORS 並設定任何 HTTP API 的任何路由授權。當您為 $default 路由 啟用 CORS 和授權時，有些特殊事項需要考量。$default 路由會擷取您未明確定義之所有方法和路由的請求，包括 OPTIONS 請求。為了支援未授權的 OPTIONS 請求，請將 OPTIONS /{proxy+} 路由新增到您不需要授權的 API，並將整合連接到路由。該 OPTIONS /{proxy+} 路由具有比 $default 路由更高的優先順序。因此，它會允許用戶端在未經授權的情況下向您的 API 提交 OPTIONS 請求。如需有關路由傳送優先順序的詳細資訊，請參閱路由傳送 API 請求。

## 使用 CLI 設定 HTTP API AWS 的 CORS

以下 update-api 命令會啟用來自 https://www.example.com 的 CORS 請求：

### Example

```
aws apigatewayv2 update-api --api-id api-id --cors-configuration AllowOrigins="https://
www.example.com"
```

如需詳細資訊，請參閱《Amazon API Gateway 第 2 版 API 參考》中的 CORS。

# 在 API Gateway 中轉換 HTTP API 的 API 請求和回應

您可以在用戶端到達後端整合之前修改 API 請求。您也可以在 API Gateway 將回應傳回給用戶端之前，變更來自整合變的回應。您可以使用參數映射，來修改 HTTP API 的 API 請求和回應。若要使用參數映射，請指定要修改的 API 請求或回應參數，並指定如何修改這些參數。

## 轉換 API 請求

您可以使用請求參數，在請求到達後端整合之前變更請求。您可以修改標頭、查詢字串或請求路徑。

請求參數為金鑰-值映射。金鑰可確定要變更的請求參數的位置，以及變更方式。該值指定參數的新資料。

下表顯示支援的金鑰。

| Type | 語法 |
| --- | --- |
| 標頭 | `append\|overwrite\|remove:header.` *`headername`* |
| 查詢字串 | `append\|overwrite\|remove:querystring.` *`querystring-name`* |
| 路徑 | `overwrite:path` |

下表顯示可映射至參數的支援值。

| Type | 語法 | 備註 |
| --- | --- | --- |
| 標頭值 | $request.header.*name* 或 ${request.header.*name*} | 網域名稱需區分大小寫。API Gateway 會將多個標頭值與逗號結合起來，例如 `"header1" : "value1,value2"` 。有些標頭為預留。如需進一步了解，請參閱 the section called “預留的標頭”。 |

| Type | 語法 | 備註 |
| --- | --- | --- |
| 查詢字串值 | $request.querystring.*name* 或 ${request.querystring.*name*} | 查詢字串名稱區分大小寫。API Gateway 會將多個值與逗號結合起來，例如 `"querystring1" "Value1,Value2"` 。 |
| 請求內文 | $request.body.*name* 或 ${request.body.*name*} | JSON 路徑表達式。不支援遞迴下降 (`$request.body..name`) 和篩選表達式 (`?(expression)`)。<br><br>**Note**<br>當您指定 JSON 路徑時，API Gateway 會在 100 KB 時截斷請求主體，然後套用選擇表達式。若要傳送大於 100 KB 的承載，請指定 `$request.body` 。 |
| 請求路徑。 | $request.path 或 ${request.path} | 請求路徑，不含階段名稱。 |
| 路徑參數 | $request.path.*name* 或 ${request.path.*name*} | 請求中的路徑參數值。例如，如果路由是 `/pets/{petId}` ，您可以透過 `$request.path.petId` 從請求映射 `petId` 參數。 |

| Type | 語法 | 備註 |
| --- | --- | --- |
| 環境變數 | $context.*variableName* 或 ${context.*variableName* } | 內容變數的值。<br>**Note**<br>僅支援特殊字元 . 和 _。 |
| 階段變數 | $stageVariables.*variableName* 或 ${stageVariables.*variableName* } | 階段變數的值。 |
| 靜態值 | *string* | 常數值。 |

**Note**

要在選擇表達式中使用多個變數，請將變數括在括號內。例如 `${request.path.name}` `${request.path.id}`。

## 轉換 API 回應

您可以使用回應參數，在將回應傳回給用戶端之前，從後端整合轉換 HTTP 回應。您可以在 API Gateway 將回應傳回給用戶端之前，修改標題或回應的狀態碼。

您可以為整合傳回的每個狀態碼設定回應參數。回應參數為金鑰-值映射。金鑰可確定要變更的請求參數的位置，以及變更方式。該值指定參數的新資料。

下表顯示支援的金鑰。

| Type | 語法 |
| --- | --- |
| 標頭 | `append\|overwrite\|remove:header.` *`headername`* |
| 狀態碼 | `overwrite:statuscode` |

下表顯示可映射至參數的支援值。

| Type | 語法 | 備註 |
| --- | --- | --- |
| 標頭值 | $response.header.*name* 或 ${response.header.*name*} | 網域名稱需區分大小寫。API Gateway 會將多個標頭值與逗號結合起來，例如 `"header1" : "value1,value2"` 。有些標頭為預留。如需進一步了解，請參閱 the section called "預留的標頭"。 |
| 回應內文 | $response.body.*name* 或 ${response.body.*name*} | JSON 路徑表達式。不支援遞迴下降 (`$response.body..name` ) 和篩選表達式 (`?(expression)` )。<br>**Note**<br>當您指定 JSON 路徑時，API Gateway 會在 100 KB 時截斷回應主體，然後套用選擇表達式。若要傳送大於 100 KB 的承載，請指定 `$response.body` 。 |
| 環境變數 | $context.*variableName* 或 ${context.*variableName* } | 支援的內容變數值。 |
| 階段變數 | $stageVariables.*variableName* 或 ${stageVariables.*variableName* } | 階段變數的值。 |
| 靜態值 | *string* | 常數值。 |

> **Note**
>
> 要在選擇表達式中使用多個變數，請將變數括在括號內。例如 `${request.path.name}` `${request.path.id}`。

## 預留的標頭

以下標題為預留。您無法設定這些標頭的請求或回應映射。

- access-control-*
- apigw-*
- Authorization
- Connection
- Content-Encoding
- Content-Length
- Content-Location
- Forwarded
- Keep-Alive
- Origin
- Proxy-Authenticate
- Proxy-Authorization
- TE
- Trailers
- Transfer-Encoding
- Upgrade
- x-amz-*
- x-amzn-*
- X-Forwarded-For
- X-Forwarded-Host
- X-Forwarded-Proto

- Via

## 範例

下列 AWS CLI 範例會設定參數映射。如需 CloudFormation 範本範例，請參閱 GitHub。

### 將標頭新增至 API 請求

以下 create-integration 命令會在 API 請求到達您的後端整合之前，為其建立名為 header1 的標頭。API Gateway 會填充請求 ID 的標頭。

```
aws apigatewayv2 create-integration \
    --api-id abcdef123 \
    --integration-type HTTP_PROXY \
    --payload-format-version 1.0 \
    --integration-uri 'https://api.example.com' \
    --integration-method ANY \
    --request-parameters '{ "append:header.header1": "$context.requestId" }'
```

### 重新命名請求標頭

以下 create-integration 命令會將請求標頭從 header1 重新命名 header2：

```
aws apigatewayv2 create-integration \
    --api-id abcdef123 \
    --integration-type HTTP_PROXY \
    --payload-format-version 1.0 \
    --integration-uri 'https://api.example.com' \
    --integration-method ANY \
    --request-parameters '{ "append:header.header2": "$request.header.header1",
 "remove:header.header1": "''"}'
```

### 變更整合的回應

以下 create-integration 命令會設定整合的回應參數。若整合傳回 500 狀態碼，API Gateway 會將狀態碼變更為 403，並在回應中新增 header11。若整合傳回 404 狀態碼時，API Gateway 會將 error 標頭新增至回應。

```
aws apigatewayv2 create-integration \
    --api-id abcdef123 \
    --integration-type HTTP_PROXY \
```

```
    --payload-format-version 1.0 \
    --integration-uri 'https://api.example.com' \
    --integration-method ANY \
    --response-parameters '{"500" : {"append:header.header1": "$context.requestId",
"overwrite:statuscode" : "403"}, "404" : {"append:header.error" :
"$stageVariables.environmentId"}  }'
```

移除已設定的參數映射

以下 update-integration 命令會移除先前為 append:header.header1 設定的請求參數。還會移除 200 狀態碼之前設定的回應參數。

```
aws apigatewayv2 update-integration \
    --api-id abcdef123 \
    --integration-id hijk456 \
    --request-parameters '{"append:header.header1" : ""}' \
    --response-parameters '{"200" : {}}'
```

# 在 API Gateway 中使用 HTTP API 的 OpenAPI 定義

您可以使用 OpenAPI 3.0 定義檔來定義 HTTP API。之後，您就能將定義匯入 API Gateway，以建立 API。若要進一步了解 OpenAPI 的 API Gateway 延伸，請參閱OpenAPI 延伸。

## 匯入 HTTP API

您可以透過匯入 OpenAPI 3.0 定義檔案來建立 HTTP API。

若要從 REST API 遷移至 HTTP API，您可以將 REST API 匯出為 OpenAPI 3.0 定義檔案。然後將 API 定義匯入為 HTTP API。若要進一步了解如何匯出 REST API，請參閱從 API Gateway 匯出 REST API。

> **Note**
>
> HTTP APIs支援與 REST APIs相同的 AWS 變數。如需進一步了解，請參閱AWS OpenAPI 匯入的 變數。

### 匯入驗證資訊

匯入 API 時，API Gateway 會提供三種類別的驗證資訊。

Info

根據 OpenAPI 規格，屬性有效，但該屬性不支援 HTTP API。

例如，下列 OpenAPI 3.0 程式碼片段會產生匯入資訊，因為 HTTP API 不支援請求驗證。API Gateway 會忽略 `requestBody` 和 `schema` 欄位。

```
"paths": {
  "/": {
    "get": {
      "x-amazon-apigateway-integration": {
        "type": "AWS_PROXY",
        "httpMethod": "POST",
        "uri": "arn:aws:lambda:us-east-2:123456789012:function:HelloWorld",
        "payloadFormatVersion": "1.0"
      },
      "requestBody": {
        "content": {
          "application/json": {
            "schema": {
              "$ref": "#/components/schemas/Body"
            }
          }
        }
      }
    }
  }
  ...
},
"components": {
  "schemas": {
    "Body": {
      "type": "object",
      "properties": {
        "key": {
          "type": "string"
        }
      }
    }
    ...
  }
  ...
}
```

警告

根據 OpenAPI 規範，屬性或結構無效，但它不會封鎖 API 建立。您可以指定 API Gateway 是否應忽略這些警告並繼續建立 API，或在警告時停止建立 API。

下列 OpenAPI 3.0 文件會在匯入時產生警告，因為 HTTP API 僅支援 Lambda 代理和 HTTP 代理整合。

```
"x-amazon-apigateway-integration": {
  "type": "AWS",
  "httpMethod": "POST",
  "uri": "arn:aws:lambda:us-east-2:123456789012:function:HelloWorld",
  "payloadFormatVersion": "1.0"
}
```

錯誤

OpenAPI 規格無效或格式錯誤。API Gateway 無法從格式錯誤的文件建立任何資源。您必須修正錯誤，然後再試一次。

下列 API 會定義在匯入時產生錯誤，因為 HTTP API 僅支援 OpenAPI 3.0 規格。

```
{
  "swagger": "2.0.0",
  "info": {
    "title": "My API",
    "description": "An Example OpenAPI definition for Errors/Warnings/ImportInfo",
    "version": "1.0"
  }
  ...
}
```

作為另一個範例，雖然 OpenAPI 允許使用者定義可將多個安全性需求附加至特定操作的 API，但 API Gateway 不支援該操作。每項作業只能有一個 IAM 授權、Lambda 授權方或 JWT 授權方。嘗試建立多個安全性需求的模型會導致錯誤。

使用 匯入 API AWS CLI

以下 import-api 命令會將 OpenAPI 3.0 定義檔 `api-definition.json` 匯入為 HTTP API：

Example

```
aws apigatewayv2 import-api --body file://api-definition.json
```

Example

您可以匯入下列範例 OpenAPI 3.0 定義來建立 HTTP API。

```
{
  "openapi": "3.0.1",
  "info": {
    "title": "Example Pet Store",
    "description": "A Pet Store API.",
    "version": "1.0"
  },
  "paths": {
    "/pets": {
      "get": {
        "operationId": "GET HTTP",
        "parameters": [
          {
            "name": "type",
            "in": "query",
            "schema": {
              "type": "string"
            }
          },
          {
            "name": "page",
            "in": "query",
            "schema": {
              "type": "string"
            }
          }
        ],
        "responses": {
          "200": {
            "description": "200 response",
            "headers": {
              "Access-Control-Allow-Origin": {
                "schema": {
                  "type": "string"
                }
```

```
            }
          },
          "content": {
            "application/json": {
              "schema": {
                "$ref": "#/components/schemas/Pets"
              }
            }
          }
        }
      },
      "x-amazon-apigateway-integration": {
        "type": "HTTP_PROXY",
        "httpMethod": "GET",
        "uri": "http://petstore.execute-api.us-west-1.amazonaws.com/petstore/pets",
        "payloadFormatVersion": 1.0
      }
    },
    "post": {
      "operationId": "Create Pet",
      "requestBody": {
        "content": {
          "application/json": {
            "schema": {
              "$ref": "#/components/schemas/NewPet"
            }
          }
        },
        "required": true
      },
      "responses": {
        "200": {
          "description": "200 response",
          "headers": {
            "Access-Control-Allow-Origin": {
              "schema": {
                "type": "string"
              }
            }
          },
          "content": {
            "application/json": {
              "schema": {
                "$ref": "#/components/schemas/NewPetResponse"
```

```
              }
            }
          }
        }
      },
      "x-amazon-apigateway-integration": {
        "type": "HTTP_PROXY",
        "httpMethod": "POST",
        "uri": "http://petstore.execute-api.us-west-1.amazonaws.com/petstore/pets",
        "payloadFormatVersion": 1.0
      }
    }
  },
  "/pets/{petId}": {
    "get": {
      "operationId": "Get Pet",
      "parameters": [
        {
          "name": "petId",
          "in": "path",
          "required": true,
          "schema": {
            "type": "string"
          }
        }
      ],
      "responses": {
        "200": {
          "description": "200 response",
          "headers": {
            "Access-Control-Allow-Origin": {
              "schema": {
                "type": "string"
              }
            }
          },
          "content": {
            "application/json": {
              "schema": {
                "$ref": "#/components/schemas/Pet"
              }
            }
          }
        }
```

```
        },
        "x-amazon-apigateway-integration": {
          "type": "HTTP_PROXY",
          "httpMethod": "GET",
          "uri": "http://petstore.execute-api.us-west-1.amazonaws.com/petstore/pets/
{petId}",
          "payloadFormatVersion": 1.0
        }
      }
    }
  },
  "x-amazon-apigateway-cors": {
    "allowOrigins": [
      "*"
    ],
    "allowMethods": [
      "GET",
      "OPTIONS",
      "POST"
    ],
    "allowHeaders": [
      "x-amzm-header",
      "x-apigateway-header",
      "x-api-key",
      "authorization",
      "x-amz-date",
      "content-type"
    ]
  },
  "components": {
    "schemas": {
      "Pets": {
        "type": "array",
        "items": {
          "$ref": "#/components/schemas/Pet"
        }
      },
      "Empty": {
        "type": "object"
      },
      "NewPetResponse": {
        "type": "object",
        "properties": {
          "pet": {
```

```
            "$ref": "#/components/schemas/Pet"
          },
          "message": {
            "type": "string"
          }
        }
      },
      "Pet": {
        "type": "object",
        "properties": {
          "id": {
            "type": "string"
          },
          "type": {
            "type": "string"
          },
          "price": {
            "type": "number"
          }
        }
      },
      "NewPet": {
        "type": "object",
        "properties": {
          "type": {
            "$ref": "#/components/schemas/PetType"
          },
          "price": {
            "type": "number"
          }
        }
      },
      "PetType": {
        "type": "string",
        "enum": [
          "dog",
          "cat",
          "fish",
          "bird",
          "gecko"
        ]
      }
    }
  }
```

```
}
```

## 從 APIs Gateway 匯出 HTTP API

建立 HTTP API 之後，您可以從 API Gateway 匯出您的 API 的 OpenAPI 3.0 定義。您可以選擇要匯出的階段，也可以匯出 API 的最新組態。您也可以將匯出的 API 定義匯入到 API Gateway，以建立另一個相同的 API。若要進一步了解如何匯入 API 定義，請參閱匯入 HTTP API。

使用 CLI 匯出階段的 OpenAPI 3.0 AWS 定義

以下 export-api 命令會將名為 `prod` 的 API 階段的 OpenAPI 定義，匯出至名為 `stage-definition.yaml` 的 YAML 檔案。匯出的定義檔案預設會包含 API Gateway 延伸。

```
aws apigatewayv2 export-api \
    --api-id api-id  \
    --output-type YAML  \
    --specification OAS30 \
    --stage-name prod \
    stage-definition.yaml
```

使用 CLI 匯出 API 最新變更的 OpenAPI 3.0 AWS 定義

以下 export-api 命令會將 HTTP API 的 OpenAPI 定義匯出至名為 `latest-api-definition.json` 的 JSON 檔案。由於命令未指定階段，因此無論 API 是否已部署至階段，API Gateway 都會匯出 API 的最新組態。匯出的定義檔案不會包含 API Gateway 延伸。

```
aws apigatewayv2 export-api \
    --api-id api-id  \
    --output-type JSON  \
    --specification OAS30 \
    --no-include-extensions \
    latest-api-definition.json
```

如需詳細資訊，請參閱《Amazon API Gateway 第 2 版 API 參考》中的 ExportAPI。

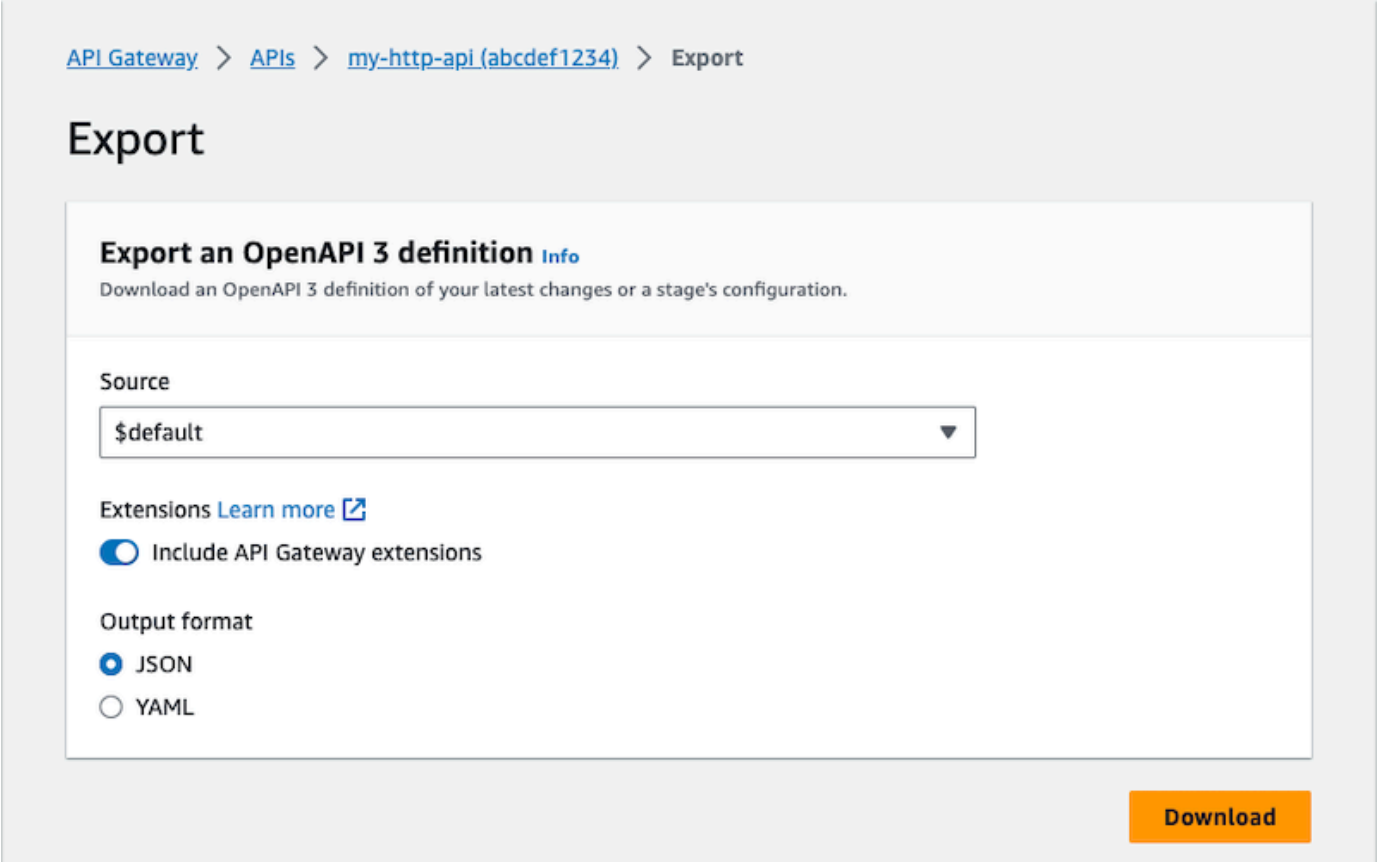
使用 API Gateway 主控台匯出 OpenAPI 3.0 定義

下列程序說明如何匯出 HTTP API 的 OpenAPI 定義。

若要使用 API Gateway 主控台匯出 OpenAPI 3.0 定義

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。

2. 選擇一個 HTTP API。
3. 在主導覽窗格的開發底下，選擇匯出。
4. 從下方選取匯出 API 的選項：




   a. 針對來源，選取 OpenAPI 3.0 定義的來源。您可以選擇要匯出的階段，也可以匯出 API 的最新組態。
   b. 開啟包含 API Gateway 延伸模組以包含 API Gateway 延伸模組。
   c. 針對輸出格式，選取輸出格式。
5. 選擇 Download (下載)。

# 發布 HTTP API 以供客戶調用

您可以使用階段和自訂網域名稱來發布 API，以供用戶端呼叫。

API 階段是 API 生命週期狀態的邏輯參考 (例如，`dev`、`prod`、`beta` 或 `v2`)。每個階段都是 API 部署的具名參考，且可供用戶端應用程式呼叫。您可以為 API 的每個階段設定不同的整合和設定。

您可以使用自訂網域名稱來提供更簡單、更直覺的 URL，讓用戶端呼叫您的 API，而非預設的 URL `https://api-id.execute-api.region.amazonaws.com/stage`。

> **Note**
>
> 為了增強 API Gateway API 的安全性，`execute-api.{region}.amazonaws.com` 網域會在公用尾碼清單 (PSL) 中註冊。為了加強安全性，如果您需要在 API Gateway API 的預設網域名稱中設定敏感性 Cookie，我們建議您使用具 `__Host-` 前置詞的 Cookie。此做法將有助於保護您的網域免受跨站請求偽造 (CSRF) 攻擊。如需更多資訊，請參閱 Mozilla 開發人員網路中的設定 Cookie 頁面。

主題


- API Gateway 中 HTTP API 的階段
- API Gateway 中 HTTP API 的安全政策
- API Gateway 中 HTTP API 的自訂網域名稱


## API Gateway 中 HTTP API 的階段

API 階段是 API 生命週期狀態的邏輯參考 (例如，`dev`、`prod`、`beta` 或 `v2`)。API 階段是由其 API ID 及階段名稱來識別，而且它們會包含在您用來呼叫 API 的 URL 中。每個階段都是 API 部署的具名參考，且可供用戶端應用程式呼叫。

您可以建立從 API URL 基礎提供的 `$default` 階段，例如 `https://{api_id}.execute-api.{region}.amazonaws.com/`。您可以使用此 URL 來呼叫 API 階段。

部署是 API 組態的快照。將 API 部署到階段之後，用戶端就可以叫用它。您必須部署 API 才能讓變更生效。如果您啟用自動部署，則會自動為您發行 API 的變更。

### 在 API Gateway 中使用 HTTP API 的階段變數

階段變數是您可為 HTTP API 定義階段的索引鍵/值對。它們的作用如同環境變數，而且可用於 API 設定。

階段變數並非用於敏感資料，例如登入資料。若要將敏感資料傳遞至整合，請使用 AWS Lambda 授權方。您可以將敏感資料傳遞至 Lambda 授權方輸出中的整合。如需進一步了解，請參閱the section called “Lambda 授權方回應格式”。

範例 – 使用階段變數自訂 HTTP 整合端點

例如，您可以定義階段變數，然後將其值設定為 HTTP Proxy 整合的 HTTP 端點。稍後，您可以使用相關聯的階段變數名稱來參照端點。如此，您可以在每個階段使用不同的端點來使用相同的 API 設定。同樣地，您可以使用階段變數，為 API 的每個階段指定不同的 AWS Lambda 函數整合。

若要使用階段變數來自訂 HTTP 整合端點，您必須先將階段變數 (例如，`url`) 的名稱和值設定為 `example.com`。之後，設定 HTTP 代理整合。您可以告訴 API Gateway 使用階段變數值 **`http://${stageVariables.url}`**，而不需要輸入端點的 URL。此值會指示 API Gateway 在執行時間替換您的階段變數 `${}`，視您 API 的階段而定。

您可以使用類似的方式來參考階段變數，以指定 Lambda 函數名稱或 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。以下 add-permission 命令會設定 Lambda 函式的許可：

```
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
```

## API Gateway 中適用於 HTTP API 的 API Gateway 階段變數參考

在下列情況下，您可以針對 HTTP API 使用 API Gateway 階段變數。

HTTP 整合 URI

您可以使用階段變數作為 HTTP 整合 URI 的一部分，如下例範例所示。

- 不含通訊協定的完整 URI – `http://${stageVariables.<variable_name>}`
- 完整的網域 – `http://${stageVariables.<variable_name>}/resource/operation`
- 子網域 – `http://${stageVariables.<variable_name>}.example.com/resource/operation`
- 路徑 – `http://example.com/${stageVariables.<variable_name>}/bar`
- 查詢字串 – `http://example.com/foo?q=${stageVariables.<variable_name>}`

Lambda 函式

您可以使用階段變數取代 Lambda 函數整合名稱或別名，如下列範例所示。

- `arn:aws:apigateway:<region>:lambda:path/2015-03-31/functions/arn:aws:lambda:<region>:<account_id>:function:${stageVariables.<function_variable_name>}/invocations`
- `arn:aws:apigateway:<region>:lambda:path/2015-03-31/functions/arn:aws:lambda:<region>:<account_id>:function:<function_name>:${stageVariables.<version_variable_name>}/invocations`

> **Note**
>
> 若要使用 Lambda 函數的階段變數，函數必須與 API 位於相同的帳戶中。階段變數不支援跨帳戶 Lambda 函數。

AWS 整合憑證

您可以使用階段變數做為 AWS 使用者或角色登入資料 ARN 的一部分，如下列範例所示。

- `arn:aws:iam::<account_id>:${stageVariables.<variable_name>}`

## API Gateway 中 HTTP API 的安全政策

API Gateway 會針對所有 HTTP API 端點強制執行 TLS_1_2 的安全政策。

安全政策是由 Amazon API Gateway 提供的最低 TLS 版本與密碼套件的預先定義組合。TLS 通訊協定可解決用戶端和伺服器間的竄改與竊聽等網路安全問題。當用戶端透過自訂網域建立 API 的 TLS 信號交換時，安全政策會強制執行用戶端可選擇使用的 TLS 版本和密碼套件。此安全政策接受 TLS 1.2 和 TLS 1.3 流量，並拒絕 TLS 1.0 流量。

### HTTP API 支援的 TLS 通訊協定和密碼

下表說明 HTTP API 支援的 TLS 通訊協定。

| TLS 通訊協定 | TLS_1_2 安全政策 |
| --- | --- |
| TLSv1.3 | 是 |
| TLSv1.2 | 是 |

下表說明適用於 HTTP API 的 TLS 1_2 安全政策的 TLS 加密。

| TLS 加密 | TLS_1_2 安全政策 |
| --- | --- |
| TLS-AES-128-GCM-SHA256 | 是 |
| TLS-AES-256-GCM-SHA384 | 是 |
| TLS-CHACHA20-POLY1305-SHA256 | 是 |
| ECDHE-ECDSA-AES128-GCM-SHA256 | 是 |
| ECDHE-RSA-AES128-GCM-SHA256 | 是 |
| ECDHE-ECDSA-AES128-SHA256 | 是 |
| ECDHE-RSA-AES128-SHA256 | 是 |
| ECDHE-ECDSA-AES256-GCM-SHA384 | 是 |
| ECDHE-RSA-AES256-GCM-SHA384 | 是 |
| ECDHE-ECDSA-AES256-SHA384 | 是 |
| ECDHE-RSA-AES256-SHA384 | 是 |
| AES128-GCM-SHA256 | 是 |
| AES128-SHA256 | 是 |
| AES256-GCM-SHA384 | 是 |
| AES256-SHA256 | 是 |

## OpenSSL 和 RFC 密碼名稱

OpenSSL 和 IETF RFC 5246 使用相同密碼的不同名稱。如需密碼名稱的清單，請參閱 the section called “OpenSSL 和 RFC 密碼名稱”。

### REST API 和 WebSocket API 的相關資訊

如需 REST API 和 WebSocket API 的詳細資訊，請參閱 the section called “選擇安全政策” 和 the section called “API Gateway 中適用於 WebSocket API 的安全政策”。

## API Gateway 中 HTTP API 的自訂網域名稱

自訂網域名稱是更簡單且更直觀的 URL，可提供給 API 使用者。

部署 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* 即為 (AWS 區域)，而 *stage* 是您在部署 API 時所指定。

URL 的主機名稱部分 (即 *api-id*.execute-api.*region*.amazonaws.com) 指的是 API 端點。預設 API 端點是隨機產生的，不容易取回，而且不方便使用。

使用自訂網域名稱，您就能設定 API 的主機名稱，並選擇基本路徑 (例如，myservice) 將替代 URL 映射至您的 API。例如，更方便使用者使用的 API 基本 URL 可以成為：

```
https://api.example.com/myservice
```

### 考量事項

下列考量事項可能會影響您使用自訂網域名稱。

- 區域性自訂網域名稱可與 REST API 和 HTTP API 相關聯。您可以使用 API Gateway 第 2 版 API 來建立及管理 REST API 的區域性自訂網域名稱。
- 對於最低 TLS 版本，僅 TLS 1.2 受支援。
- 您必須建立或更新 DNS 提供者的資源記錄，以映射至您的 API 端點。如果沒有這種映射，送往自訂網域名稱的 API 請求無法到達 API Gateway。
- 您可以使用萬用字元憑證來支援幾乎不限數量的網域名稱，而不會超過預設配額。如需更多詳細資訊，請參閱 the section called “萬用字元自訂網域名稱”。

### 先決條件

以下是建立自訂網域名稱的先決條件。

註冊網域名稱

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。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。您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。

您的網域名稱必須遵循 RFC 1035 規格，每個標籤最多可有 63 個八位元組，總共有 255 個八位元組。

自訂網域名稱的憑證

必須為 API 設定自訂網域名稱前，您必須先在 ACM 中備妥 SSL/TLS 憑證。如果您建立自訂網域名稱的 AWS 區域中無法使用 ACM，則必須將憑證匯入該區域中的 API Gateway。

若要匯入 SSL/TLS 憑證，您必須提供 PEM 格式化 SSL/TLS 憑證內文、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。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。若要使用網域名稱的 AWS 受管憑證，您只需要參考其 ARN。

如果您的應用程式是使用憑證關聯 (有時稱為 SSL 關聯) 來關聯 ACM 憑證，則 AWS 續約憑證後，應用程式可能會無法連線到您的網域。如需詳細資訊，請參閱 AWS Certificate Manager 使用者指南中的憑證關聯問題。

## 萬用字元自訂網域名稱

使用萬用字元自訂網域名稱，您可以支援幾乎無限數目的網域名稱，而不會超過預設配額。例如，您可以為每個客戶提供其自己的網域名稱，*customername*.api.example.com。

若要建立萬用字元自訂網域名稱，可以指定萬用字元 (*) 作為自訂網域的第一個子網域，藉以表示根網域所有可能的子網域。

例如，萬用字元自訂網域名稱 *.example.com 會產生如 a.example.com、b.example.com 和 c.example.com 等子網域，而這些子網域全都路由至相同的網域。

萬用字元自訂網域名稱支援來自 API Gateway 標準自訂網域名稱的相異組態。例如，您可以在單一 AWS 帳戶中，設定 *.example.com 和 a.example.com 採取不同的行為。

若要建立萬用字元自訂網域名稱，您必須提供由 ACM 發行並已經使用 DNS 或電子郵件驗證方法驗證的憑證。

> **Note**
>
> 如果不同的 AWS 帳戶已建立與萬用字元自訂網域名稱衝突的自訂網域名稱，則您無法建立萬用字元自訂網域名稱。例如，如果帳戶 A 已建立 `a.example.com`，則帳戶 B 無法建立萬用字元自訂網域名稱 `*.example.com`。
> 如果帳戶 A 和帳戶 B 共用擁有者，您可以聯絡 AWS 支援中心，以請求例外狀況。

## 自訂網域名稱的後續步驟

若要設定 HTTP API 的自訂網域名稱，請參閱《API Gateway 開發人員指南》中的 REST API 相關章節文件。

首先，指定自訂網域名稱所用的憑證。如需更多詳細資訊，請參閱 the section called “在 中備妥憑證 AWS Certificate Manager”。接著，建立區域性自訂網域名稱。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”。

## 將 API 階段映射至 HTTP API 的自訂網域名稱

您可以使用 API 映射將 API 階段連線至自訂網域名稱。建立網域名稱並設定 DNS 記錄之後，您可以使用 API 映射，透過自訂網域名稱將流量傳送至您的 API。

API 映射指定一個 API，一個階段，以及選擇性用於映射的路徑。例如，您可以將 API 的 `production` 階段映射至 `https://api.example.com/orders`。

您可以將 HTTP 和 REST API 階段映射至相同的自訂網域名稱。

建立 API 映射之前，您必須先擁有 API、階段和自訂網域名稱。如需進一步了解如何建立自訂網域名稱，請參閱the section called “設定區域性自訂網域名稱”。

路由傳送 API 請求

您可以設定具有多個層級的 API 映射，例如 `orders/v1/items` 和 `orders/v2/items`。

針對有多個層級的 API 映射，API Gateway 會將請求路由傳送至具有最長相符路徑的 API 映射。API Gateway 只會考慮為 API 映射而非 API 路由設定的路徑，以選取要調用的 API。如果沒有路徑與請求相符，則 API Gateway 會將請求傳送到您映射到空白路徑（none）的 API。

針對使用有多個層級的 API 映射的自訂網域名稱，API Gateway 會將請求路由傳送至具有最長相符字首的 API 映射。

例如，假設具有下列 API 映射的自訂網域名稱 `https://api.example.com`：

1. (none) 已映射到 API 1。
2. orders 已映射到 API 2。
3. orders/v1/items 已映射到 API 3。
4. orders/v2/items 已映射到 API 4。
5. orders/v2/items/categories 已映射到 API 5。

| 要求 | 選取的 API | 說明 |
| --- | --- | --- |
| https://api.example.com/orders | API 2 | 請求完全符合此 API 映射。 |
| https://api.example.com/orders/v1/items | API 3 | 請求完全符合此 API 映射。 |
| https://api.example.com/orders/v2/items | API 4 | 請求完全符合此 API 映射。 |
| https://api.example.com/orders/v1/items/123 | API 3 | API Gateway 會選擇具有最長相符路徑的映射。請求結束時的 123 不會影響選擇。 |
| https://api.example.com/orders/v2/items/categories/5 | API 5 | API Gateway 會選擇具有最長相符路徑的映射。 |
| https://api.example.com/customers | API 1 | API Gateway 使用空白映射作為全部擷取。 |
| https://api.example.com/ordersandmore | API 2 | API Gateway 會選擇具有最長相符字首的映射。<br><br>針對設定使用單一層級映射的自訂網域名稱，例如只有 https://api.example.com/ |

| 要求 | 選取的 API | 說明 |
| --- | --- | --- |
| | | `orders` 和 `https://api.example.com/` ，API Gateway 會選擇 `API 1`，因為沒有與 `ordersandmore` 相符的路徑。 |

限制

- 在 API 映射中，自訂網域名稱和映射的 API 必須位於同一個 AWS 帳戶中。
- API 映射只能包含字母、數字和下列字元：`$-_.+!*'()/`。
- API 映射中路徑的最大長度為 300 個字元。
- 您可以為每個域名設定具有 200 個具多個層級的 API 映射。此限制不包括具有單一層級的 API 映射，例如 `/prod`。
- 您只能將 HTTP API 映射至具有 TLS 1.2 安全政策的區域自訂網域名稱。
- 您無法將 WebSocket API 映射至與 HTTP API 或 REST API 相同的自訂網域名稱。
- 如果您建立具有多層的 API 映射，API Gateway 會將所有標頭名稱轉換為小寫。

建立 API 映射

若要建立 API 映射，您必須先建立自訂網域名稱、API 和階段。如需建立自訂網域名稱的資訊，請參閱the section called “設定區域性自訂網域名稱”。

有關建立所有資源的 AWS Serverless Application Model 範本範例，請參閱 GitHub 上的使用 SAM 的工作階段。

AWS 管理主控台

建立 API 映射

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 選擇 Custom domain names (自訂網域名稱)。
3. 選取您已建立的自訂網域名稱。
4. 選擇 API mappings (API 映射)。
5. 選擇 Configure API mappings (設定 API 映射)。

6. 選擇 Add new mapping (新增映射)。
7. 輸入 API、Stage (階段)，以及選擇性地輸入 Path (路徑)。
8. 選擇儲存。

AWS CLI

以下 create-api-mapping 命令會建立 API 映射。在此範例中，API Gateway 會將請求傳送至 `api.example.com/v1/orders`，到指定的 API 和階段。

```
aws apigatewayv2 create-api-mapping \
    --domain-name api.example.com \
    --api-mapping-key v1/orders \
    --api-id a1b2c3d4 \
    --stage test
```

CloudFormation

以下 CloudFormation 範例會建立 API 映射。

```
MyApiMapping:
  Type: 'AWS::ApiGatewayV2::ApiMapping'
  Properties:
    DomainName: api.example.com
    ApiMappingKey: 'orders/v2/items'
    ApiId: !Ref MyApi
    Stage: !Ref MyStage
```

## 停用 HTTP API 的預設端點

預設情況下，用戶端可以使用 API Gateway 為 API 產生的 `execute-api` 端點來調用 API。若要確保用戶端只能使用自訂網域名稱來存取您的 API，請停用預設 `execute-api` 端點。當您停用預設端點時，它會影響 API 的所有階段。

下列程序顯示如何停用 HTTP API 的預設端點。

AWS 管理主控台

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。

2. 選擇一個 HTTP API。
3. 選擇 API 的 ID 以開啟 API 詳細資訊頁面。
4. 在 API 詳細資訊上，選擇編輯。
5. 對於預設端點，選取停用。
6. 選擇儲存。

   如果您為階段開啟自動部署，則不需要為了讓變更生效而重新部署 API。否則，您必須重新部署 API。
7. (選用) 選擇部署，然後重新部署您的 API 或建立新的階段，以使變更生效。

AWS CLI

以下 update-domain-name 命令會停用 HTTP API 的預設端點：

```
aws apigatewayv2 update-api \
    --api-id abcdef123 \
    --disable-execute-api-endpoint
```

停用預設端點後，您必須部署 API 才能讓變更生效，除非已啟用自動部署。

下列 create-deployment 命令會建立部署：

```
aws apigatewayv2 create-deployment \
    --api-id abcdef123 \
    --stage-name dev
```

## HTTP API 自訂網域名稱的 IP 位址類型

建立 API 時，您可以指定可調用網域的 IP 位址類型。您可以選擇 IPv4 來解析 IPv4 位址以調用網域，也可以選擇雙堆疊，以同時允許 IPv4 和 IPv6 位址調用您的網域。我們建議您將 IP 位址類型設定為雙堆疊，以緩解 IP 空間耗盡或用於安全狀態。如需雙堆疊 IP 位址類型優點的詳細資訊，請參閱 AWS 上的 IPv6。

IP 位址類型的考量事項

下列考量事項可能會影響您使用 IP 位址類型。

- API Gateway 自訂網域名稱的預設 IP 位址類型為 IPv4。

- 您的自訂網域名稱不需要針對所有與其對應的 API 使用相同 IP 位址類型。如果您停用預設 API 端點，則可能會影響呼叫方調用 API 的方式。

### 變更自訂網域名稱的 IP 位址類型

您可以藉由更新網域的端點組態來變更 IP 位址類型。您可以使用 AWS 管理主控台、AWS CLI、CloudFormation 或 AWS SDK 來更新網域的端點組態。

#### AWS 管理主控台

若要變更自訂網域名稱的 IP 位址類型

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 選擇公有自訂網域名稱。
3. 選擇端點組態。
4. 針對 IP 位址類型，選取 IPv4 或雙堆疊。
5. 選擇儲存。

#### AWS CLI

以下 update-domain-name 命令會將 API 更新為雙堆疊 IP 位址類型：

```
aws apigatewayv2 update-domain-name \
   --domain-name dualstack.example.com \
   --domain-name-configurations CertificateArn=arn:aws:acm:us-
east-1:111122223333:certificate/abcd1234-5678-abc,IpAddressType=dualstack
```

輸出將如下所示：

```
{
    "ApiMappingSelectionExpression": "$request.basepath",
    "DomainName": "dualstack.example.com",
    "DomainNameConfigurations": [
        {
            "ApiGatewayDomainName": "d-abcd1234.execute-api.us-
east-1.amazonaws.com",
            "CertificateArn": "arn:aws:acm:us-east-1:111122223333:certificate/
abcd1234-5678-abc",
            "DomainNameStatus": "AVAILABLE",
```

```
            "EndpointType": "REGIONAL",
            "HostedZoneId": "Z3LQWSYCGH4ADY",
            "SecurityPolicy": "TLS_1_2",
            "IpAddressType": "dualstack"
        }
    ],
    "Tags": {}
}
```

# 在 API Gateway 中保護您的 HTTP API

API Gateway 提供多種方法來保護 API 免於遭受特定威脅，例如惡意使用者或流量高峰。您可以使用設定調節目標和啟用交互 TLS 等策略來保護您的 API。您可以在本節中了解如何使用 API Gateway 啟用這些功能。

主題

- 調節對 HTTP API 的請求，以提高 API Gateway 中的輸送量
- 如何在 API Gateway 中為 HTTP API 開啟交互 TLS 驗證

## 調節對 HTTP API 的請求，以提高 API Gateway 中的輸送量

您可以為您的 API 設定調節，以防止 API 接收過多請求。調節會依最佳作法來套用，它們都應該視為目標，而非確定的請求上限。

API Gateway 會使用字符儲存貯體演算法將字符計算為請求，進而調節傳送給 API 的請求量。具體而言，API Gateway 會按區域檢查帳戶中所有 API 的速率和爆量請求次數。在字符儲存貯體演算法中，爆量可以實現預先定義的超限，但在某些情況下，其他因素也可能導致超限。

提交的請求量超出穩定狀態請求率和爆量限制時，API Gateway 會開始調節請求量。此時用戶端可能會收到 `429 Too Many Requests` 的錯誤回應。發現這類例外狀況時，用戶端可以採用限制速率的方式來重新提交失敗的請求。

身為 API 開發人員，您可以設定個別 API 階段或路由的目標限制，來改善您帳戶中所有 API 的整體效能。

### 每個區域的帳戶層級調節

預設情況下，API Gateway 會按區域限制 AWS 帳戶內所有 API 的每秒穩定狀態請求量 (RPS)。它還會按區域限制 AWS 帳戶內所有 API 的爆量 (即儲存貯體大小上限)。在 API Gateway 中，爆量限制代

表 API Gateway 傳回 429 Too Many Requests 錯誤回應前可實現的並行請求提交數上限。如需有關調節配額的詳細資訊，請參閱 配額和重要說明。

帳戶型限制會套用至指定區域內某帳戶的所有 API。帳戶層級速率限制可按請求提高。使用較短逾時值和較小酬載的 API 可擁有更高的上限。如需請求提高區域內帳戶層級的調節限制，請與 AWS 支援中心聯絡。如需詳細資訊，請參閱配額和重要說明。請注意，這些限制不能高於限 AWS 流限制。

### 路由層級調節

您可以設定路由層級調節，來覆寫特定階段或 API 中個別路由的帳戶層級請求調節限制。預設路由調節限制不能超出帳戶層級速率限制。

您可以使用 AWS CLI設定路由層級的節流設定。以下 update-stage 命令會針對指定的 API 階段和路由設定自訂限流：

```
aws apigatewayv2 update-stage \
    --api-id a1b2c3d4 \
    --stage-name dev \
    --route-settings '{"GET /pets":
{"ThrottlingBurstLimit":100,"ThrottlingRateLimit":2000}}'
```

## 如何在 API Gateway 中為 HTTP API 開啟交互 TLS 驗證

交互 TLS 驗證需要用戶端與伺服器之間的雙向驗證。使用交互 TLS 時，用戶端必須出示 X.509 憑證，以驗證其身分才能存取您的 API。交互 TLS 是物聯網 (IoT) 和企業對企業應用程式的常見需求。

交互 TLS 可與 API Gateway 支援的其他授權和身分驗證作業一起使用。API Gateway 會將用戶端提供的憑證轉送給 Lambda 授權方和後端整合系統。

> **Important**
>
> 預設情況下，用戶端可以使用 API Gateway 為 API 產生的 execute-api 端點來叫用 API。若要確保用戶端只能透過使用具有交互 TLS 的自訂網域名稱來存取您的 API，請停用預設 execute-api 端點。如需進一步了解，請參閱the section called “停用預設端點”。

### 交互 TLS 的先決條件

若要設定交互 TLS，您需要：

- 自訂網域名稱

- 在 中 AWS Certificate Manager 為您的自訂網域名稱設定至少一個憑證
- 已設定並上傳至 Amazon S3 的信任庫

### 自訂網域名稱

若要啟用 HTTP API 的交互 TLS，必須設定 API 的自訂網域名稱。您可以為自訂網域名稱啟用交互 TLS，然後將自訂網域名稱提供給用戶端。若要使用已啟用交互 TLS 的自訂網域名稱來存取 API，用戶端必須提供您在 API 要求中信任的憑證。您可以在 the section called “自訂網域名稱” 中尋找詳細資訊。

### 使用 AWS Certificate Manager 發行的憑證

您可以直接從 ACM 請求公開信任的憑證，或匯入公開或自行簽署的憑證。若要在 ACM 中設定憑證，請前往 ACM。如果想要匯入憑證，請繼續閱讀下一節。

### 使用匯入的 或 AWS 私有憑證授權單位 憑證

若要使用匯入 ACM 的憑證或從 的憑證 AWS 私有憑證授權單位 搭配交互 TLS，API Gateway 需要 ACM `ownershipVerificationCertificate`發行的 。此擁有權憑證僅用於確認您具有使用網域名稱的許可，不會用於 TLS 信號交換。如果您尚未擁有 `ownershipVerificationCertificate`，請前往 https://console.aws.amazon.com/acm/ 以設定一個。

您需要讓此憑證在網域名稱的有效生命週期內保持有效狀態。如果憑證過期且自動續約失敗，則網域名稱的所有更新都會遭到鎖定。您需要使用有效的 `ownershipVerificationCertificate` 更新 `ownershipVerificationCertificateArn`，才能進行任何其他變更。所以，`ownershipVerificationCertificate` 不可作為 API Gateway 中另一個交互 TLS 網域的伺服器憑證。如果直接將憑證重新匯入至 ACM，發行者必須保持不變。

### 設定您的信任庫

信任庫是副檔名為 `.pem` 的文字檔案。它們是來自憑證授權機構的受信任憑證清單。若要使用交互 TLS，請建立信任的 X.509 憑證信任存放區，以存取您的 API。

您必須在信任庫中包含從發行的憑證授權機構憑證到根憑證授權機構憑證完整的信任鏈。API Gateway 接受信任鏈中存在的任何憑證授權機構所發行的用戶端憑證。憑證可以來自公開或私有憑證授權單位。憑證的鏈接長度上限為四。您也可以提供自我簽署憑證。信任庫支援下列雜湊演算法：

- SHA-256 或更強的憑證
- RSA-2048 或更強的憑證
- ECDSA-256 或更強的憑證

API Gateway 會驗證許多憑證屬性。當用戶端叫用 API 時，您可以透過 Lambda 授權方執行其他檢查，包括檢查憑證是否已遭撤銷。如此一來，API Gateway 就會驗證下列屬性：

| 驗證 | 描述 |
| --- | --- |
| X.509 語法 | 憑證必須符合 X.509 語法需求。 |
| 完整性 | 憑證的內容不得從信任庫的憑證授權單位所簽署的內容進行變更。 |
| Validity | 憑證的有效期必須是最新的。 |
| 名稱鏈接/金鑰鏈接 | 憑證的名稱和主體必須形成一個完整的鏈接。憑證的鏈接長度上限為四。 |

請將信任庫以單一檔案的形式上傳到 Amazon S3 儲存貯體中

Example certificates.pem

```
-----BEGIN CERTIFICATE-----
<Certificate contents>
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
<Certificate contents>
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
<Certificate contents>
-----END CERTIFICATE-----
...
```

下列 cp AWS CLI 命令`certificates.pem`會上傳至您的 Amazon S3 儲存貯體：

```
aws s3 cp certificates.pem s3://bucket-name
```

## 設定自訂網域名稱的交互 TLS

若要設定 HTTP API 的交互 TLS，您必須使用 API 的區域性自訂網域名稱，且要求的最低 TLS 版本為 1.2。如需深入瞭解如何建立和設定自訂網域名稱，請參閱the section called “設定區域性自訂網域名稱”。

> **Note**
>
> 私有 API 不支援交互 TLS。

將信任庫上傳到 Amazon S3 後，您可以將自訂網域名稱設定為使用交互 TLS。以下 create-domain-name 命令會建立具有交互 TLS 的自訂網域名稱：

```
aws apigatewayv2 create-domain-name \
    --domain-name api.example.com \
    --domain-name-configurations CertificateArn=arn:aws:acm:us-
west-2:123456789012:certificate/123456789012-1234-1234-1234-12345678 \
    --mutual-tls-authentication TruststoreUri=s3://bucket-name/key-name
```

建立網域名稱之後，您必須設定 API 作業的 DNS 記錄和基本路徑對應。如需進一步了解，請參閱在 API Gateway 中設定區域性自訂網域名稱。

## 使用需要交互 TLS 的自訂網域名稱叫用 API

若要調用啟用交互 TLS 的 API，用戶端必須在 API 要求中提供受信任的憑證。當用戶端嘗試叫用您的 API 時，API Gateway 會在您的信任庫中尋找用戶端憑證的發行者。若要讓 API Gateway 繼續執行要求，憑證發行者和完整信任鏈 (一路深入到根憑證授權機構憑證) 必須位於您的信任庫中。

下列範例 `curl` 命令會將要求傳送至 `api.example.com`，要求 `my-cert.pem` 中包含的要求。`my-key.key` 是憑證的私密金鑰。

```
curl -v --key ./my-key.key --cert ./my-cert.pem api.example.com
```

只有在您的信任庫信任憑證時，才會調用您的 API。下列情況會導致 API Gateway 的 TLS 信號交換失敗，並拒絕狀態碼為 403 的請求。如果您的憑證：

- 不受信任
- 已過期
- 未使用支援的演算法

> **Note**
>
> API Gateway 並不會驗證憑證是否已遭撤銷。

## 更新您的信任庫

若要更新信任庫中的憑證，請將新的憑證套件上傳至 Amazon S3。之後，您可以更新自訂網域名稱，以使用更新的憑證。

您可以使用 Amazon S3 版本控制來維護多個信任庫版本。在更新自訂網域名稱以使用新的信任庫版本時，如果憑證無效，則 API Gateway 會傳回警告。

API Gateway 只會在您更新網域名稱時產生憑證警告。如果先前上傳的憑證過期，API Gateway 並不會通知您。

以下 update-domain-name 命令會更新自訂網域名稱，以使用新的信任庫版本：

```
aws apigatewayv2 update-domain-name \
    --domain-name api.example.com \
    --domain-name-configurations CertificateArn=arn:aws:acm:us-
west-2:123456789012:certificate/123456789012-1234-1234-1234-12345678 \
    --mutual-tls-authentication TruststoreVersion='abcdef123'
```

## 停用交互 TLS

若要停用自訂網域名稱的交互 TLS，請從自訂網域名稱移除信任庫，如下列命令所示。

以下 update-domain-name 命令會更新自訂網域名稱，以從您的自訂網域名稱中移除信任庫：

```
aws apigatewayv2 update-domain-name \
    --domain-name api.example.com \
    --domain-name-configurations CertificateArn=arn:aws:acm:us-
west-2:123456789012:certificate/123456789012-1234-1234-1234-12345678 \
    --mutual-tls-authentication TruststoreUri=''
```

## 針對 HTTP API 的交互 TLS 進行疑難排解

以下針對當您開啟交互 TLS 時可能遇到錯誤和問題，所提供的疑難排解建言。

故障診斷憑證警告

建立帶有交互 TLS 的自訂網域名稱時，如果信任庫中的憑證無效，則 API Gateway 會傳回警告。在更新自訂網域名稱以使用新的信任庫時，也會發生這種情況。警告指出憑證和產生警告之憑證主體的問題。您的 API 仍然啟用交互 TLS，但有些用戶端可能無法存取您的 API。

若要識別產生警告的憑證，請解碼信任庫中的憑證。您可以使用諸如 openssl 解碼憑證及識別其主體之類的工具。

下列命令會顯示憑證的內容，包括其主體：

```
openssl x509 -in certificate.crt -text -noout
```

請更新或移除產生警告的憑證，然後將新的信任庫上傳至 Amazon S3。上傳新的信任庫之後，請更新您的自訂網域名稱以使用新的信任庫。

疑難排解網域名稱衝突

錯誤 "The certificate subject <certSubject> conflicts with an existing certificate from a different issuer." 表示多個憑證授權單位已發行此網域的憑證。對於憑證中的每個主體，交互 TLS 網域的 API Gateway 中只能有一個發行者。您需要透過單一發行者取得該主體的所有憑證。如果問題是您無法控制的憑證，但您可以證明您擁有網域名稱，請聯絡 支援 以開啟票證。

疑難排解網域名稱狀態訊息

PENDING_CERTIFICATE_REIMPORT：這表示您已將憑證重新匯入至 ACM，而且由於新憑證具有 SAN (主體別名)，而且這個 SAN 不屬於 ownershipVerificationCertificate 範圍或憑證中的主體或 SAN 不包含網域名稱，導致憑證驗證失敗。某些項目可能設定不正確或匯入了無效的憑證。您需要將有效憑證重新匯入 ACM。如需有關驗證的詳細資訊，請參閱驗證網域擁有權。

PENDING_OWNERSHIP_VERIFICATION：這表示您先前驗證的憑證已過期，ACM 無法對其進行自動續約。您將需要為憑證續約或申請新憑證。關於憑證續約的詳細資訊，請參閱 ACM 的疑難排解受管憑證續約指南。

# 在 API Gateway 中監控 HTTP API

您可以使用 CloudWatch 指標和 CloudWatch Logs 來監控 HTTP API。藉由結合日誌和指標，您可以記錄錯誤並監控 API 的效能。

> **Note**
>
> 在下列情況下，API Gateway 可能無法產生日誌和指標：
>
> - 413 請求實體過大錯誤

- 過多的 429 請求過多錯誤
- 400 系列錯誤，來自傳送至沒有 API 映射自訂網域的要求
- 由於內部失敗造成的 500 系列錯誤

主題


- 在 API Gateway 中監控 HTTP API 的 CloudWatch 指標
- 在 API Gateway 中設定 HTTP API 的日誌


## 在 API Gateway 中監控 HTTP API 的 CloudWatch 指標

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。這些統計資料會記錄 15 個月的時間，以便您存取歷史資訊，並更清楚 Web 應用程式或服務的執行效能。根據預設，API Gateway 指標資料會自動在一分鐘內傳送給 CloudWatch。若要監控您的指標，請為您的 API 建立 CloudWatch 儀表板。如需如何建立 CloudWatch 儀表板的詳細資訊，請參閱《Amazon CloudWatch 使用者指南》中的建立 CloudWatch 儀表板。如需更多詳細資訊，請參閱《Amazon CloudWatch 使用者指南》中的什麼是 Amazon CloudWatch？。

HTTP API 支援下列指標。您也可以啟用詳細指標，將路由層級的指標寫入 Amazon CloudWatch。

| 指標 | 描述 |
| --- | --- |
| 4xx | 在指定期間內擷取的用戶端錯誤數目。 |
| 5xx | 在指定期間內擷取的伺服器端錯誤數目。 |
| Count | 指定期間內的 API 要求總數。 |
| IntegrationLatency | API Gateway 將請求轉送給後端時與收到來自後端的回應時之間的時間。 |
| Latency | API Gateway 收到來自用戶端的請求時與它將回應傳回給用 |

| 指標 | 描述 |
| --- | --- |
| | 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。 |
| DataProcessed | 處理的資料量 (以位元組為單位)。 |

您可以使用下表中的維度來篩選 API Gateway 指標。

| 維度 | 描述 |
| --- | --- |
| ApiId | 篩選具指定 API ID 之 API 的 API Gateway 指標。 |
| ApiId、階段 | 篩選具指定 API ID 與階段 ID 之 API 階段的 API Gateway 指標。 |
| ApiId、方法、資源、階段 | 篩選具指定 API ID、階段 ID、資源路徑和路由 ID 之 API 方法的 API Gateway 指標。<br><br>除非您已明確啟用詳細的 CloudWatch 指標，否則 API Gateway 不會傳送這類指標。您可以透過呼叫 API Gateway V2 REST API 的 UpdateStage 動作，將 `detailedMetricsEnabled` 屬性更新為 `true`。或者，您可以呼叫更新階段 AWS CLI 命令，將 `DetailedMetricsEnabled` 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 |

| 維度 | 描述 |
| --- | --- |
| | 參閱 Amazon CloudWatch 定價。 |

## 在 API Gateway 中設定 HTTP API 的日誌

您可以開啟記錄功能，將日誌寫入 CloudWatch Logs。您可以使用記錄變數來自訂日誌的內容。

為了改善您的安全狀態，建議您將 HTTP API 所有階段的日誌寫入 CloudWatch Logs。您可能需要這樣做，才能符合各種合規架構。如需詳細資訊，請參閱《AWS Security Hub 使用者指南》中的 Amazon API Gateway 控制項。

若要開啟 HTTP API 的記錄功能，您必須執行下列動作。

1. 確定您的使用者具有啟用記錄所需的許可。
2. 建立 CloudWatch Logs 日誌群組。
3. 為 API 的某個階段提供 CloudWatch Logs 日誌群組的 ARN。

### 用以啟用記錄的許可。

若要開啟 API 的記錄功能，您的使用者必須具有下列許可。

Example

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogGroups",
                "logs:DescribeLogStreams",
                "logs:GetLogEvents",
                "logs:FilterLogEvents"
            ],
```

```
            "Resource": "arn:aws:logs:us-east-2:123456789012:log-group:*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogDelivery",
                "logs:PutResourcePolicy",
                "logs:UpdateLogDelivery",
                "logs:DeleteLogDelivery",
                "logs:CreateLogGroup",
                "logs:DescribeResourcePolicies",
                "logs:GetLogDelivery",
                "logs:ListLogDeliveries"
            ],
            "Resource": "*"
        }
    ]
}
```

## 建立日誌群組並啟動 HTTP API 的記錄

您可以使用 AWS 管理主控台 或 建立日誌群組並啟用存取記錄 AWS CLI。

AWS 管理主控台

1. 建立 日誌群組

   如要了解如何使用主控台建立日誌群組，請參閱《Amazon CloudWatch Logs 使用者指南》中的建立日誌群組

2. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
3. 選擇一個 HTTP API。
4. 在主導覽面板中的 Monitor (監視器) 標籤下，選擇 Logging (記錄)。
5. 選取要啟動記錄的階段，然後選擇 Select (選取)。
6. 選擇 Edit (編輯) 以啟動存取記錄。
7. 開啟 Access logging (存取記錄功能)，輸入 CloudWatch Logs，然後選取日誌格式。
8. 選擇儲存。

AWS CLI

以下 create-log-group 命令會建立日誌群組：

```
aws logs create-log-group --log-group-name my-log-group
```

您需要日誌群組的 Amazon Resource Name (ARN) 才能啟用記錄。ARN 格式為 arn:aws:logs:*region*:*account-id*:log-group:*log-group-name*。

下列 update-stage 命令會啟動 HTTP API 的 `$default` 階段的記錄功能：

```
aws apigatewayv2 update-stage --api-id abcdef \
    --stage-name '$default' \
    --access-log-settings '{"DestinationArn": "arn:aws:logs:region:account-
id:log-group:log-group-name", "Format": "$context.identity.sourceIp - -
 [$context.requestTime] \"$context.httpMethod $context.routeKey $context.protocol\"
 $context.status $context.responseLength $context.requestId"}'
```

## 日誌格式範例

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.requestTime] "$context.httpMethod
 $context.routeKey $context.protocol" $context.status $context.responseLength
 $context.requestId $context.extendedRequestId
```

- JSON:

```
{ "requestId":"$context.requestId", "ip": "$context.identity.sourceIp",
 "requestTime":"$context.requestTime",
 "httpMethod":"$context.httpMethod","routeKey":"$context.routeKey",
 "status":"$context.status","protocol":"$context.protocol",
 "responseLength":"$context.responseLength", "extendedRequestId":
 "$context.extendedRequestId" }
```

- XML:

```
<request id="$context.requestId"> <ip>$context.identity.sourceIp</ip> <requestTime>
$context.requestTime</requestTime> <httpMethod>$context.httpMethod</httpMethod>
```

```
 <routeKey>$context.routeKey</routeKey> <status>$context.status</status> <protocol>
$context.protocol</protocol> <responseLength>$context.responseLength</responseLength>
 <extendedRequestId>$context.extendedRequestId</extendedRequestId> </request>
```

- CSV (逗號分隔值)：

```
$context.identity.sourceIp,$context.requestTime,$context.httpMethod,
$context.routeKey,$context.protocol,$context.status,$context.responseLength,
$context.requestId,$context.extendedRequestId
```

## 自訂 HTTP API 存取日誌

您可以使用下列變數來自訂 HTTP API 存取日誌。要進一步了解 HTTP API 的存取日誌，請參閱在 API Gateway 中設定 HTTP API 的日誌。

| 參數 | Description |
| --- | --- |
| `$context.accountId` | API 擁有者 AWS 的帳戶 ID。 |
| `$context.apiId` | API Gateway 指派給您 API 的識別碼。 |
| `$context.authorizer.claims.`*`property`* | 成功驗證方法發起人 (例如 `$context.authorizer.claims.username`) 之後，從 JSON Web Token (JWT) 傳回之宣告的屬性。如需詳細資訊，請參閱使用 API Gateway 中的 JWT 授權方來控制對 HTTP API 的存取。<br>**Note**<br>呼叫 `$context.authorizer.claims` 會傳回 null。 |
| `$context.authorizer.error` | 從授權方傳回的錯誤訊息。 |
| `$context.authorizer.`*`property`* | API Gateway Lambda 授權方函數所傳回 `context` 對應之指定索引鍵/值對的值。例如，如果授權方傳回下列 `context` 映射： |

| 參數 | Description |
| --- | --- |
| | <pre>"context" : {<br>  "key": "value",<br>  "numKey": 1,<br>  "boolKey": true<br>}</pre><br>呼叫 $context.authorizer.key 會傳回 "value" 字串，呼叫 $context.authorizer.numKey 會傳回 1，而呼叫 $context.authorizer.boolKey 會傳回 true。 |
| $context.awsEndpointRequestId | 來自 x-amz-request-id 或 x-amzn-requestId 標頭的 AWS 端點請求 ID。 |
| $context.awsEndpointRequestId2 | 來自 x-amz-id-2 標頭的 AWS 端點請求 ID。 |
| $context.customDomain.basePathMatched | 傳入請求相符的 API 映射路徑。適用於用戶端使用自訂網域名稱來存取 API 時。例如，如果用戶端向 https://api.example.com/v1/orders/1234 傳送請求，並且請求讓 API 映射與路徑 v1/orders 相符，則該值為 v1/orders 。如需進一步了解，請參閱the section called “API 映射”。 |
| $context.dataProcessed | 處理的資料量 (以位元組為單位)。 |
| $context.domainName | 用來叫用 API 的完整網域名稱。這應該與傳入的 Host 標頭相同。 |
| $context.domainPrefix | $context.domainName 的第一個標籤。 |
| $context.error.message | 包含 API Gateway 錯誤訊息的字串。 |
| $context.error.messageString | $context.error.message 的引用值，即 "$context.error.message" 。 |

| 參數 | Description |
| --- | --- |
| $context.error.responseType | GatewayResponse 的類型。如需詳細資訊，請參閱 the section called “指標” 和 the section called “設定閘道回應以自訂錯誤回應”。 |
| $context.extendedRequestId | 等同於 $context.requestId 。 |
| $context.httpMethod | 使用的 HTTP 方法。有效值包含：DELETE、GET、HEAD、OPTIONS、PATCH、POST 和 PUT。 |
| $context.identity.accountId | 與請求相關聯的 AWS 帳戶 ID。支援使用 IAM 授權的路由。 |
| $context.identity.caller | 已簽署請求之發起人的主體識別符。支援使用 IAM 授權的路由。 |
| $context.identity.cognitoAuthenticationProvider | 提出請求的發起人所使用的所有 Amazon Cognito 驗證提供者清單 (以逗號分隔)。僅在使用 Amazon Cognito 登入資料簽署請求時才可使用。<br><br>例如，適用於 Amazon Cognito 使用者集區的身分，cognito-idp.*region*.amazonaws.com/ *user_pool_id* ,cognito-idp. *region*.amazonaws.com/ *user_pool_id* :CognitoSignIn: *token subject claim*<br><br>如需有關 Amazon Cognito 驗證提供者的詳細資訊，請參閱《Amazon Cognito 開發人員指南》中的使用聯合身分。 |

| 參數 | Description |
| --- | --- |
| $context.identity.cognitoAuthenticationType | 提出請求的發起人的 Amazon Cognito 驗證類型。僅在使用 Amazon Cognito 登入資料簽署請求時才可使用。可能的值包括用於已驗證身分的 authenticated 和未經驗證身分的 unauthenticated 。 |
| $context.identity.cognitoIdentityId | 提出請求的發起人的 Amazon Cognito 身分 ID。僅在使用 Amazon Cognito 登入資料簽署請求時才可使用。 |
| $context.identity.cognitoIdentityPoolId | 提出請求的發起人的 Amazon Cognito 身分集區 ID。僅在使用 Amazon Cognito 登入資料簽署請求時才可使用。 |
| $context.identity.principalOrgId | AWS 組織 ID。支援使用 IAM 授權的路由。 |
| $context.identity.clientCert.clientCertPem | 用戶端在交互 TLS 驗證期間所呈現的 PEM 編碼用戶端憑證。當用戶端使用已啟用交互 TLS 的自訂網域名稱存取 API 時會顯示。 |
| $context.identity.clientCert.subjectDN | 用戶端提供之憑證主體的辨別名稱。當用戶端使用已啟用交互 TLS 的自訂網域名稱存取 API 時會顯示。 |
| $context.identity.clientCert.issuerDN | 用戶端提供之憑證發行者的辨別名稱。當用戶端使用已啟用交互 TLS 的自訂網域名稱存取 API 時會顯示。 |
| $context.identity.clientCert.serialNumber | 憑證的序號。當用戶端使用已啟用交互 TLS 的自訂網域名稱存取 API 時會顯示。 |
| $context.identity.clientCert.validity.notBefore | 憑證無效之前的日期。當用戶端使用已啟用交互 TLS 的自訂網域名稱存取 API 時會顯示。 |
| $context.identity.clientCert.validity.notAfter | 憑證無效之後的日期。當用戶端使用已啟用交互 TLS 的自訂網域名稱存取 API 時會顯示。 |

| 參數 | Description |
| --- | --- |
| `$context.identity.sourceIp` | 對 API Gateway 提出請求之即時 TCP 連線的來源 IP 地址。 |
| `$context.identity.user` | 將針對資源存取授權之使用者的主體識別符。支援使用 IAM 授權的路由。 |
| `$context.identity.userAgent` | API 發起人的 `User-Agent` 標頭。 |
| `$context.identity.userArn` | 身分驗證之後識別之有效使用者的 Amazon Resource Name (ARN)。支援使用 IAM 授權的路由。如需更多詳細資訊，請參閱 https://docs.aws.amazon.com/IAM/latest/UserGuide/id_users.html。 |
| `$context.integration.error` | 從整合傳回的錯誤訊息。等同於 `$context.integrationErrorMessage`。 |
| `$context.integration.integrationStatus` | 對於 Lambda 代理整合，從 傳回的狀態碼 AWS Lambda，而不是從後端 Lambda 函數程式碼傳回的狀態碼。 |
| `$context.integration.latency` | 整合延遲 (以毫秒為單位)。等同於 `$context.integrationLatency`。 |
| `$context.integration.requestId` | AWS 端點的請求 ID。等同於 `$context.awsEndpointRequestId`。 |
| `$context.integration.status` | 從整合傳回的狀態碼。對於 Lambda 代理整合而言，這是您的 Lambda 函數程式碼傳回的狀態碼。 |
| `$context.integrationErrorMessage` | 包含整合錯誤訊息的字串。 |
| `$context.integrationLatency` | 整合延遲 (以毫秒為單位)。 |

| 參數 | Description |
| --- | --- |
| $context.integrationStatus | 對於 Lambda 代理整合，此參數代表從 傳回的狀態碼 AWS Lambda，而不是從後端 Lambda 函數傳回的狀態碼。 |
| $context.path | 請求路徑。例如，/{stage}/root/child 。 |
| $context.protocol | 請求通訊協定，例如 HTTP/1.1。<br><br>**Note**<br>API Gateway API 可以接受 HTTP/2 請求，但 API Gateway 會使用 HTTP/1.1 將請求傳送至後端整合。因此，即使用戶端傳送使用 HTTP/2 的請求，請求通訊協定也會記錄為 HTTP/1.1。 |
| $context.requestId | API Gateway 指派給 API 請求的 ID。 |
| $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.responseLatency | 回應延遲 (以毫秒為單位)。 |
| $context.responseLength | 回應承載長度 (以位元組為單位)。 |
| $context.routeKey | API 請求的路由金鑰，例如 /pets。 |
| $context.stage | API 請求的部署階段 (例如，beta 或 prod)。 |
| $context.status | 方法回應狀態。 |

# 疑難排解 API Gateway 中的 HTTP API 問題

下列主題說明當您在使用 HTTP API 時，可如何疑難排解可能遭遇到的錯誤和問題的建議。

主題


- 對 HTTP API Lambda 整合的問題進行疑難排解
- 對 HTTP API JWT 授權方的問題進行疑難排解


## 對 HTTP API Lambda 整合的問題進行疑難排解

以下建議說明在您將 AWS Lambda 整合 與 HTTP API 搭配使用時，如何疑難排解可能會遇到的錯誤和問題。

### 問題：具有 Lambda 整合的 API 會傳回 `{"message":"Internal Server Error"}`

若要對內部伺服器錯誤進行疑難排解，請將 `$context.integrationErrorMessage` 記錄變數新增至您的日誌格式，然後檢視您 HTTP API 的日誌。若要完成此動作，請執行下列操作：

使用 建立日誌群組 AWS 管理主控台

1. 在 https://console.aws.amazon.com/cloudwatch/ 開啟 CloudWatch 主控台。
2. 選擇 Log groups (日誌群組)。
3. 選擇 Create log group (建立日誌群組)。
4. 輸入日誌群組名稱，然後選擇 Create (建立)。
5. 記下日誌群組的 Amazon Resource Name (ARN)。ARN 格式為 arn:aws:logs:*region*:*account-id*:log-group:*log-group-name*。您需要日誌群組 ARN 才能存取 HTTP API 的記錄功能。

新增 **`$context.integrationErrorMessage`** 記錄變數

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 選擇 HTTP API。
3. 在 Monitor (監控) 下，選擇 Logging (記錄)。
4. 選取 API 的階段。

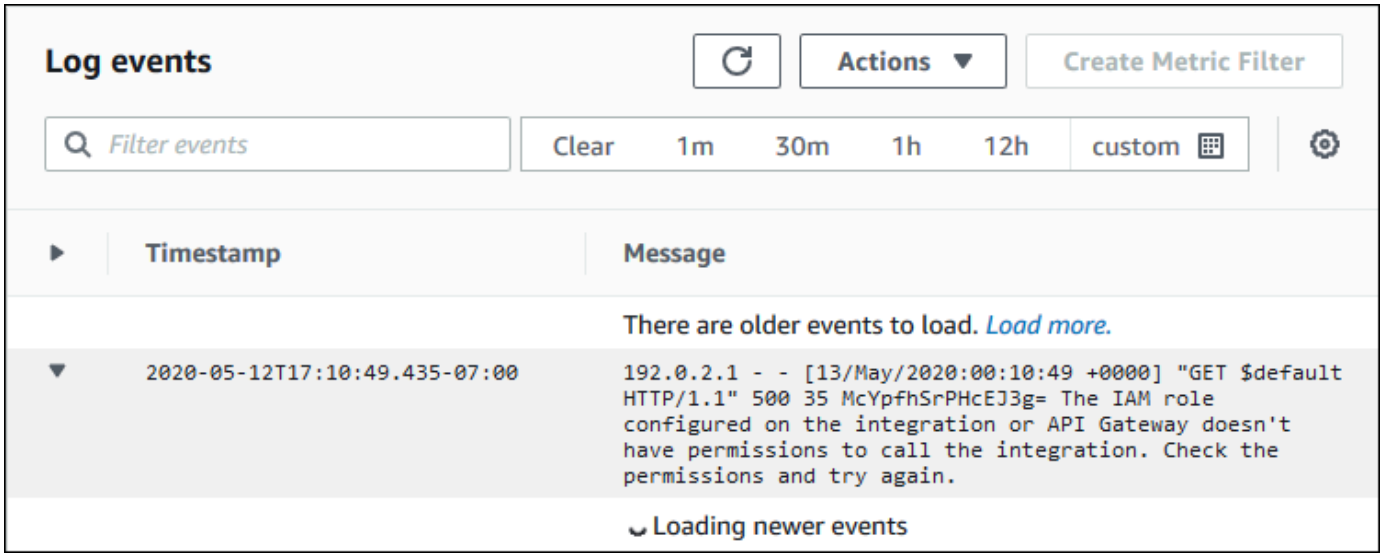
5. 選擇 Edit (編輯)，然後存取記錄。
6. 針對 Log destination (日誌目的地)，輸入您在上一個步驟中建立之日誌群組的 ARN。
7. 在 Log format (日誌格式) 中，選擇 CLF。API Gateway 會建立範例日誌格式。
8. 將 `$context.integrationErrorMessage` 新增至日誌格式的結尾。
9. 選擇 Save (儲存)。

檢視 API 的日誌

1. 產生日誌。使用瀏覽器或 `curl` 來叫用您的 API。

```
$curl https://api-id.execute-api.us-west-2.amazonaws.com/route
```

2. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
3. 選擇 HTTP API。
4. 在 Monitor (監控) 下，選擇 Logging (記錄)。
5. 選取您啟用記錄功能的 API 階段。
6. 選擇 View logs in CloudWatch (檢視 CloudWatch 中的日誌)。
7. 選擇最新的日誌串流來檢視 HTTP API 的日誌。
8. 您的日誌項目看起來應與下列類似：




因為我們已將 `$context.integrationErrorMessage` 新增至日誌格式，所以我們會在摘要問題的日誌中看到錯誤訊息。

您的日誌可能包含不同的錯誤訊息，其中會指出您的 Lambda 函數程式碼發生問題。在這種情況下，請檢查您的 Lambda 函數程式碼，並確認您的 Lambda 函數以所需的格式傳回回應。如果您的日誌未包含錯誤訊息，請將 `$context.error.message` 和 `$context.error.responseType` 新增至您的日誌格式，以取得更多有助於疑難排解的資訊。

在這種情況下，日誌會顯示 API Gateway 沒有叫用 Lambda 函式所需的許可。

當您在 API Gateway 主控台中建立 Lambda 整合時，API Gateway 會自動設定叫用 Lambda 函數的許可。當您使用 AWS CLI CloudFormation或 SDK 建立 Lambda 整合時，您必須授予 API Gateway 調用函數的許可。以下 add-permission 命令會對不同的 HTTP API 路由授予許可，以調用 Lambda 函式。

Example範例 – 適用於 HTTP API 的 $default 階段和 $default 路由

```
aws lambda add-permission \
    --function-name my-function \
    --statement-id apigateway-invoke-permissions \
    --action lambda:InvokeFunction \
    --principal apigateway.amazonaws.com \
    --source-arn "arn:aws:execute-api:us-west-2:123456789012:api-id/\$default/\
$default"
```

Example範例 – 適用於 HTTP API 的 prod 階段和 test 路由

```
aws lambda add-permission \
    --function-name my-function \
    --statement-id apigateway-invoke-permissions \
    --action lambda:InvokeFunction \
    --principal apigateway.amazonaws.com \
    --source-arn "arn:aws:execute-api:us-west-2:123456789012:api-id/prod/*/test"
```

在 Lambda 主控台的 Permissions (許可) 索引標籤中，確認函數政策。

嘗試再次叫用您的 API。您應會看見 Lambda 函數的回應。

## 對 HTTP API JWT 授權方的問題進行疑難排解

以下建議說明當您在搭配 HTTP API 使用 JSON Web Token (JWT) 授權方時，如何對可能遇到的錯誤和問題進行疑難排解。

### 問題：我的 API 傳回 `401 {"message":"Unauthorized"}`

檢查 API 回應中的 `www-authenticate` 標頭。

下列命令會用 curl 來將請求傳送至 API，並使用 JWT 授權方 $request.header.Authorization 做為其身分識別來源。

```
$curl -v -H "Authorization: token" https://api-id.execute-api.us-
west-2.amazonaws.com/route
```

來自 API 的回應包括一個 www-authenticate 標頭。

```
...
< HTTP/1.1 401 Unauthorized
< Date: Wed, 13 May 2020 04:07:30 GMT
< Content-Length: 26
< Connection: keep-alive
< www-authenticate: Bearer scope="" error="invalid_token" error_description="the token
 does not have a valid audience"
< apigw-requestid: Mc7UVioPPHcEKPA=
<
* Connection #0 to host api-id.execute-api.us-west-2.amazonaws.com left intact
{"message":"Unauthorized"}}
```

在這種情況下，www-authenticate 標頭會顯示字符並未發給有效的對象。若 API Gateway 要授權請求，JWT 的 aud or client_id 要求必須符合為授權方設定的其中一個對象項目。API Gateway 僅於 aud 不存在時驗證 client_id。當 aud 和 client_id 同時存在時，API Gateway 會評估 aud。

您也可以解碼 JWT，並確認它符合您的 API 所需的發行者、對象和範圍。該網站 jwt.io 可以在瀏覽器中對 JWT 偵錯。OpenID Foundation 會提供可搭配 JWT 使用的程式庫清單。

若要進一步了解 JWT 授權方，請參閱使用 API Gateway 中的 JWT 授權方來控制對 HTTP API 的存取。

# API Gateway WebSocket API

API Gateway 中的 WebSocket API 是與後端 HTTP 端點、Lambda 函數或其他 AWS 服務整合的 WebSocket 路由集合。您可以使用 API Gateway 功能來協助您處理 API 生命週期的各個層面，從建立到監控生產 API。

API Gateway WebSocket API 是雙向的。用戶端可以將訊息傳送至服務，而且服務可以獨立地將訊息傳送給用戶端。這種雙向行為可提供更豐富的用戶端/服務互動，因為服務可以將資料推送至用戶端，而不需要用戶端提出明確請求。WebSocket 經常用於聊天應用程式、協作平台、多玩家遊戲和金融平台等即時應用程式。

如需開始使用的應用程式範例，請參閱 教學：使用 WebSocket API、Lambda 和 DynamoDB 建立 WebSocket 聊天應用程式。

在本節中，您可以學習如何開發、發布、保護以及使用 API Gateway 監控您的 WebSocket API。

主題


- API Gateway 中的 WebSocket API 概觀
- 在 API Gateway 中開發的 WebSocket API
- 發布 WebSocket API 以供客戶調用
- 在 API Gateway 中保護您的 WebSocket API
- 在 API Gateway 中監控的 WebSocket API


## API Gateway 中的 WebSocket API 概觀

在 API Gateway 中，您可以建立 WebSocket API 做為 AWS 服務（例如 Lambda 或 DynamoDB) 或 HTTP 端點的狀態前端。WebSocket API 會根據自用戶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)，會支援用戶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戶端。

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 “”。

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。

## 使用 WebSocket API 的路由

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 會在用戶端和 WebSocket API 之間的持續連線進行起始化時呼叫 `$connect` 路由。
- 當用戶端或伺服器中斷與 API 的連線時，API Gateway 會呼叫 `$disconnect` 路由。
- 若發現相符的路由，針對該訊息評估路由選擇運算式後，API Gateway 即會呼叫自訂路由；該比對會判斷要叫用哪些整合。
- 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 “管理連線的使用者和用戶端應用程式”。

如需 `$default` 路由和自訂路由的相關資訊，請參閱the section called “調用後端整合”。

## 將資料傳送至連線的用戶端應用程式

後端服務可將資料傳送到連線的用戶端應用程式。您可以藉由執行以下動作來傳送電子郵件：

- 使用整合以傳送回應，該回應會由您已定義的路由傳回到用戶端。
- 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”。

## WebSocket API 狀態碼

API Gateway WebSocket API 使用以下狀態碼作為從伺服器到用戶端的通訊方式，如 WebSocket 關閉代碼編號登錄檔中所述：

1001

當用戶端閒置 10 分鐘或達到最長 2 小時連線生命週期時，API Gateway 會傳回此狀態碼。

1003

當端點收到二進位媒體類型時，API Gateway 會傳回此狀態碼。WebSocket API 不支援二進位媒體類型。

1005

如果用戶端傳送的關閉框架沒有關閉碼，API Gateway 就會傳回此狀態碼。

1006

如果連線意外關閉，例如，TCP 連線在沒有 WebSocket 關閉框架的情況下關閉，API Gateway 就會傳回此狀態碼。

1008

當端點收到來自特定用戶端的過多請求時，API Gateway 會傳回此狀態碼。

1009

當端點收到的訊息太大而無法處理時，API Gateway 會傳回此狀態碼。

1011

發生內部伺服器錯誤時，API Gateway 會傳回此狀態碼。

1012

如果服務重新啟動，API Gateway 會傳回此狀態碼。

# 管理連線的使用者和用戶端應用程式：**$connect** 和 **$disconnect** 路由

下一節說明如何使用 WebSocket API 的 `$connect` 和 `$disconnect` 路由。

主題

- $connect 路由
- 從 $connect 路由傳遞連線資訊
- $disconnect 路由

## **$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。您應在以下情況考慮設定 $connect 整合：

- 您想要讓用戶端使用 Sec-WebSocket-Protocol 欄位來指定子通訊協定。如需範例程式碼，請參閱設定需要 WebSocket 子協定的 $connect 路由。
- 您希望在用戶端連線時收到通知。
- 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。
- 您想要將連線 IDs、網域名稱或其他連線詳細資訊存放在資料庫中 （ 例如 Amazon DynamoDB)。連線 ID 是可變長度字串，其長度和格式可能會變更。

## 從 $connect 路由傳遞連線資訊

您可以同時使用代理和非代理整合，將資訊從 $connect 路由傳遞至資料庫或其他 AWS 服務。

使用代理整合傳遞連線資訊

您可以從事件中的 Lambda 代理整合存取連線資訊。使用另一個 AWS 服務 或 AWS Lambda 函數來發佈至連線。

下列 Lambda 函數顯示如何使用 requestContext 物件來記錄連線 ID、網域名稱、階段名稱和查詢字串。

Node.js

```
export const handler = async(event, context) => {
   const connectId = event["requestContext"]["connectionId"]
```

```
    const domainName = event["requestContext"]["domainName"]
    const stageName = event["requestContext"]["stage"]
    const qs = event['queryStringParameters']
    console.log('Connection ID: ', connectId, 'Domain Name: ', domainName, 'Stage
 Name: ', stageName, 'Query Strings: ', qs )
    return {"statusCode" : 200}
};
```

Python

```
import json
import logging
logger = logging.getLogger()
logger.setLevel("INFO")


def lambda_handler(event, context):
    connectId = event["requestContext"]["connectionId"]
    domainName = event["requestContext"]["domainName"]
    stageName = event["requestContext"]["stage"]
    qs = event['queryStringParameters']
    connectionInfo = {
        'Connection ID': connectId,
        'Domain Name': domainName,
        'Stage Name': stageName,
        'Query Strings': qs}
    logging.info(connectionInfo)
    return {"statusCode": 200}
```

使用非代理整合傳遞連線資訊

- 您可以透過非代理整合存取連線資訊。設定整合要求並提供 WebSocket API 要求範本。下列 Velocity 範本語言 (VTL) 對應範本以提供整合要求。此要求會將下列詳細資訊傳送至非 Proxy 整合：
  - 連線 ID
  - 網域名稱
  - 階段名稱
  - 路徑
  - 標頭

- 查詢字串

此要求會將連線 ID、網域名稱、階段名稱、路徑、標頭和查詢字串傳送至非 Proxy 整合。

```
{
    "connectionId": "$context.connectionId",
    "domain": "$context.domainName",
    "stage": "$context.stage",
    "params": "$input.params()"
}
```

如需設定資料轉換的詳細資訊，請參閱 the section called “資料轉換”。

如要完成整合請求，請設定整合回應的 StatusCode: 200。若要深入瞭解如何設定整合回應，請參閱 使用 API Gateway 主控台設定整合回應。

### $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。由於連接在執行時已經關閉，$disconnect 是一個最佳努力的事件。API Gateway 會盡力將 $disconnect 事件傳遞給整合，但無法保證傳遞成功。

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”。

## 在 API Gateway 中調用與 $default Route 和自訂路由的後端整合

下一節說明如何使用 WebSocket API 的$default 路由或自訂路由來調用您的後端整合。

主題

- 使用路由來處理訊息
- $default 路由
- 自訂路由
- 使用 API Gateway WebSocket API 整合，以連接到商業邏輯
- WebSocket API 和 REST API 之間的重要差異

## 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 “二進位媒體類型”。

可透過 API Gateway 中的 WebSocket API，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。您可以使用 AWS CLI或使用 AWS SDK，在 API Gateway 主控台中設定 WebSocket API 的路由請求。

> **Note**
>
> 在 AWS CLI 和 AWS SDKs中，您可以在建立整合之前或之後建立路由。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 中發布驗證結果：

```
{"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結果相關的資訊傳送到用戶端。

### `$default` 路由

每個 API Gateway WebSocket API 都可以具備 `$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 “WebSocket API 的概觀”」。)

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 “整合”。

## 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。在 AWS CLI 和 AWS SDKs中，您可以透過將路由的目標設定為 的值來重複使用整合`"integrations/{integration-id}"`，其中 `{integration-id}`"是與路由相關聯的整合的唯一 ID。

- API Gateway 提供您可以在路由和整合使用的多種選擇表達式。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`、`$integ` 和靜態值。

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。

# WebSocket 選擇表達式

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 "/2\d\d/" 狀態碼的 2*XX*。

主題


- 路由回應選擇表達式
- API 金鑰選擇表達式
- API 映射選擇表達式
- WebSocket 選擇表達式總結


## 路由回應選擇表達式

路由回應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 訊息時應將回應回傳至用戶端。

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`。

## 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 |
| --- | --- | --- | --- |
| `Api.RouteSelectionExpression` | `Route.RouteKey` | 所支援的 `$defaul` 為全部截獲路由。 | 依據用戶端請求內容路由 WebSocket 訊息。 |
| `Route.ModelSelectionExpression` | `Route.RequestModels` 的金鑰 | 選用。<br><br>若提供給非代理整合，將出現模型驗證。<br><br>所支援的 `$defaul` 為全部截獲。 | 在相同路由內動態執行請求驗證。 |
| `Integration.TemplateSelecti` | `Integration.RequestTemplates` 的金鑰 | 選用。<br><br>可供非代理整 | 依據請求的動態屬性操控發 |

| 選擇表達式 | 判斷值為下列的金鑰 | 備註 | 範例使用案例 |
| --- | --- | --- | --- |
| onExpression | | 合使用，藉此操控傳入承載。<br><br>支援 ${reque .body.j nPath} 和靜態值。<br><br>所支援的 $defau 為全部截獲。 | 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 |
| --- | --- | --- | --- |
| `Integration.IntegrationResponseSelectionExpression` | `IntegrationResponse.IntegrationResponseKey` | 選用。可供非代理整合使用。<br><br>做為錯誤訊息 (來自 Lambda) 或狀態碼 (來自 HTTP 整合) 的模式比對。<br><br>非代理整合必須有 `$default` 來全部截獲成功回應。 | 從後端操控回應。<br><br>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)。 |

| 選擇表達式 | 判斷值為下列的金鑰 | 備註 | 範例使用案例 |
| --- | --- | --- | --- |
| IntegrationResponse.TemplateSelectionExpression | IntegrationResponse.ResponseTemplates 的金鑰 | 選用。可供非代理整合使用。<br><br>支援 $default。 | 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。<br><br>支援 ${request.body.jsonPath} 、${integration.response.statuscode} 、${integration.response.header.headerName} 、${integration.response.multivalueheader.headerName} 和靜態值。<br><br>所支援的 $default |

| 選擇表達式 | 判斷值為下列的金鑰 | 備註 | 範例使用案例 |
| --- | --- | --- | --- |
| | | | 為全部截獲。 |
| Route.RouteResponseSelectionExpression | RouteResponse.RouteResponseKey | 應提供，藉以啟動 WebSoc 路由的雙向通訊。<br><br>目前此值限制為 $defau | |
| RouteResponse.ModelSelectionExpression | RouteResponse.RequestModels 的金鑰 | 目前不支援。 | |

# 在 API Gateway 中開發的 WebSocket API

本節提供開發 API Gateway API 時所需的 API Gateway 功能的詳細資料。

在開發 API Gateway API 時，您可以決定 API 的許多特性。這些特性取決於 API 的使用案例。例如，您可能只允許特定用戶端呼叫您的 API，或是您可能希望讓所有人都可以使用它。您可能需要 API 呼叫來執行 Lambda 函數、進行資料庫查詢或呼叫應用程式。

主題

- 在 API Gateway 中建立 WebSocket API
- API Gateway 中 WebSocket API 的 IP 位址類型
- 在 API Gateway 中建立 WebSocket API 的路由

- 在 API Gateway 中控制和管理對 WebSocket API 的存取
- API Gateway 中 WebSocket API 的整合
- 請求驗證 API Gateway 中的 WebSocket API
- API Gateway 中用於 WebSocket API 的資料轉換
- API Gateway 中 WebSocket API 的二進位媒體類型
- 調用 WebSocket API

## 在 API Gateway 中建立 WebSocket API

您可以在 API Gateway 主控台中使用 create-api 命令， AWS CLI 或在 AWS SDK 中使用 `CreateApi` 命令來建立 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 和 TLS 1.3。不支援舊版 TLS。

### 使用 AWS CLI 命令建立 WebSocket API

以下 create-api 命令會建立具有 `$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'
```

輸出看起來如下：

```
{
    "ApiKeySelectionExpression": "$request.header.x-api-key",
    "Name": "myWebSocketApi3",
    "CreatedDate": "2018-11-15T06:23:51Z",
    "ProtocolType": "WEBSOCKET",
    "RouteSelectionExpression": "'$request.body.action'",
    "ApiId": "aabbccddee"
}
```

### 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 WebSocket API 下，選擇 Build (建置)。僅支援區域端點。
3. 針對 API 名稱，輸入您 API 的名稱。
4. 針對路由選擇表達式輸入值。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 “”。
5. 執行以下任意一項：
   - 選擇建立空白 API 以建立沒有路由的 API。
   - 選擇下一步將路由附加到您的 API。

   您可以在建立 API 之後附加路由。

## API Gateway 中 WebSocket API 的 IP 位址類型

建立 API 時，您可以指定可調用 API 的 IP 位址類型。您可以選擇 IPv4 來解析 IPv4 位址以調用 API，也可以選擇雙堆疊，以同時允許 IPv4 和 IPv6 位址調用您的 API。我們建議您將 IP 位址類型設定為雙堆疊，以緩解 IP 空間耗盡或用於安全狀態。如需雙堆疊 IP 位址類型優點的詳細資訊，請參閱 AWS 上的 IPv6。

### IP 位址類型的考量事項

下列考量事項可能會影響您使用 IP 位址類型：

- 所有 WebSocket API 的預設 IP 位址類型都是 IPv4。
- 如果您將現有 API 的 IP 位址類型從 IPv4 變更為雙堆疊，請確認控制存取 API 的任何政策都已更新，以便進行 IPv6 呼叫。當您變更 IP 位址類型時，變更會立即生效。
- 您的 API 可以對應至與您的 API 具有不同 IP 位址類型的自訂網域名稱。如果您停用預設 API 端點，則可能會影響呼叫方調用 API 的方式。

## 變更 WebSocket API 的 IP 位址類型

您可以藉由更新 API 的組態來變更 IP 位址類型。您可以使用 AWS 管理主控台、AWS CLI、CloudFormation 或 AWS SDK 來更新 API 的組態。如果您變更 API 的 IP 位址類型，您不需重新部署 API 讓變更生效。

AWS 管理主控台

若要變更 WebSocket API 的 IP 位址類型

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 選擇 WebSocket API。
3. 選擇 API 設定，然後選擇編輯。
4. 針對 IP 位址類型，選取 IPv4 或雙堆疊。
5. 選擇儲存。

   對 API 組態的變更會立即生效。

AWS CLI

以下 update-api 命令會將 API 更新為雙堆疊 IP 位址類型：

```
aws apigatewayv2 update-api \
    --api-id abcd1234 \
    --ip-address-type dualstack
```

輸出將如下所示：

```
{
    "ApiEndpoint": "https://abcd1234.execute-api.us-east-1.amazonaws.com",
    "ApiId": "abcd1234",
    "ApiKeySelectionExpression": "$request.header.x-api-key",
    "CreatedDate": "2025-02-04T22:20:20+00:00",
    "DisableExecuteApiEndpoint": false,
    "Name": "My-WebSocket-API",
    "ProtocolType": "WEBSOCKET",
    "RouteSelectionExpression": "$request.method $request.path",
    "Tags": {},
    "NotificationUris": [],
    "IpAddressType": "dualstack"
```

```
}
```

# 在 API Gateway 中建立 WebSocket API 的路由

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 “”。

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 會在用戶端和 WebSocket API 之間的持續連線進行起始化時呼叫 `$connect` 路由。
- 當用戶端或伺服器中斷與 API 的連線時，API Gateway 會呼叫 `$disconnect` 路由。
- 若發現相符的路由，針對該訊息評估路由選擇運算式後，API Gateway 即會呼叫自訂路由；該比對會判斷要叫用哪些整合。
- 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。

## 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 | 描述 |
| --- | --- | --- |
| $request.body.action | join | 未包裝的變數 |
| ${request.body.action} | join | 已包裝的變數 |
| ${request.body.service}/${request.body.action} | chat/join | 具備靜態值的多個變數 |
| ${request.body.action}-${request.body.invalidPath} | join- | 若未找到 JSONPath，則變數將解析為 ""。 |
| action | action | 靜態值 |

| 表達式 | 評估結果 | 描述 |
| --- | --- | --- |
| \$default | $default | 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 會嘗試尋找 $default 路由 (如有)。若未定義 $default 路由，則 API Gateway 會傳回錯誤。

## 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 建立它們 AWS CLI。如有需要，您可以建立自訂路由。如需更多詳細資訊，請參閱 the section called “WebSocket API 的概觀”。

> **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。

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. 選擇建立路由。
3. 針對路由金鑰，輸入路由金鑰名稱。您可以建立預先定義的路由 ($connect、$disconnect 和 $default)，或是自訂路由。

   > **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。

4. 選取並設定路由的整合類型。如需詳細資訊，請參閱the section called “使用 API Gateway 主控台設定 WebSocket API 整合請求”。

使用 建立路由 AWS CLI

以下 create-route 命令會建立路由：

```
aws apigatewayv2 --region us-east-1 create-route --api-id aabbccddee --route-key
 $default
```

輸出將如下所示：

```
{
    "ApiKeyRequired": false,
    "AuthorizationType": "NONE",
    "RouteKey": "$default",
    "RouteId": "1122334"
}
```

指定 **$connect** 的路由請求設定

設定 API 的 $connect 路由時，可使用下列選用設定，以啟用您 API 的授權。如需更多詳細資訊，請參閱 the section called “$connect 路由”。

- Authorization (授權)：若無須授權，可指定為 NONE。否則，您可指定：
  - AWS_IAM 使用標準 AWS IAM 政策來控制對 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。授權方可以位於您自己的 AWS 帳戶或其他 AWS 帳戶中。如需 Lambda 授權方的詳細資訊，請參閱 使用 API Gateway Lambda 授權方。

    > **Note**
    >
    > 在 API Gateway 主控台中，只有在您設定授權方函數 (如 CUSTOM 所述) 後，才會看到 the section called “設定 Lambda 授權方 (主控台)” 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。如需更多詳細資訊，請參閱 the section called “用量計劃”。

使用 API Gateway 主控台設定 **$connect** 路由請求

若要使用 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. 在路由下，選擇 $connect，或依照 the section called “使用 API Gateway 主控台建立路由” 建立 $connect 路由。
3. 在路由請求設定區段中，選擇編輯。
4. 針對授權，選取授權類型。
5. 若要針對 $connect 路由要求 API，請選取需要 API 金鑰。
6. 選擇儲存變更。

## 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。API Gateway 不會將後端回應傳遞至路由回應，除非您有設定路由回應。

> **Note**
>
> 您只能定義 WebSocket API 的 $default 路由回應。您可以使用整合回應來操作後端服務的回應。如需詳細資訊，請參閱the section called “整合回應概觀”。

您可以使用 API Gateway 主控台或 AWS CLI 或 AWS SDK 來設定路由回應和回應選取表達式。

如需路由回應選擇表達式的詳細資訊，請參閱 the section called “”。

主題

- 使用 API Gateway 主控台設定路由回應
- 使用 設定路由回應 AWS CLI

使用 API Gateway 主控台設定路由回應

建立 WebSocket API 並將代理 Lambda 函數附加至預設路由之後，您可以使用 API Gateway 主控台設定路由回應：

1. 登入 API Gateway 主控台，在 $default 路由上選擇具備代理 Lambda 函數整合的 WebSocket API。

2. 在 Routes (路由) 下選擇 $default 路由。
3. 選擇啟用雙向通訊。
4. 選擇部署 API。
5. 將您的 API 部署至階段。

使用下列 wscat 命令來連線到您的 API。如需 wscat 的相關資訊，請參閱 the section called “使用 wscat 以連接到 WebSocket API 和將訊息傳送到其中”。

```
wscat -c wss://api-id.execute-api.us-east-2.amazonaws.com/test
```

按下 Enter 按鈕以呼叫預設路由。您 Lambda 函數的主體應該會傳回。

使用 設定路由回應 AWS CLI

以下 create-route-response 命令會建立 $default 路由的路由回應。您可以使用 get-apis 和 get-routes 命令來識別 API ID 和路由 ID。

```
aws apigatewayv2 create-route-response \
    --api-id aabbccddee \
    --route-id 1122334  \
    --route-response-key '$default'
```

輸出將如下所示：

```
{
    "RouteResponseId": "abcdef",
    "RouteResponseKey": "$default"
}
```

## 設定需要 WebSocket 子協定的 **$connect** 路由

用戶端可以使用 Sec-WebSocket-Protocol 欄位在連線到您的 WebSocket API 期間請求一個 WebSocket 子通訊協定。您可以設定 $connect 路由整合，以允許只有在用戶端請求您的 API 支援的子通訊協定時才允許連線。

下列範例 Lambda 函數會將 Sec-WebSocket-Protocol 標題傳回用戶端。只有在用戶端指定 myprotocol 子協議時，該函數才會建立到您的 API 的連線。

如需建立此範例 API 和 Lambda 代理整合的 CloudFormation 範本，請參閱 ws-subprotocol.yaml。

```
export const handler = async (event) => {
    if (event.headers != undefined) {
        const headers = toLowerCaseProperties(event.headers);

        if (headers['sec-websocket-protocol'] != undefined) {
            const subprotocolHeader = headers['sec-websocket-protocol'];
            const subprotocols = subprotocolHeader.split(',');

            if (subprotocols.indexOf('myprotocol') >= 0) {
                const response = {
                    statusCode: 200,
                    headers: {
                        "Sec-WebSocket-Protocol" : "myprotocol"
                    }
                };
                return response;
            }
        }
    }

    const response = {
        statusCode: 400
    };

    return response;
};

function toLowerCaseProperties(obj) {
    var wrapper = {};
    for (var key in obj) {
        wrapper[key.toLowerCase()] = obj[key];
    }
    return wrapper;
}
```

您可以使用 wscat 來測試 API 是否允許連線，但只有在用戶端要求您的 API 支援的子通訊協定時才可以。下列命令會使用 -s 旗標來指定連線期間的子通訊協定。

下列命令會嘗試使用不受支援的子通訊協定進行連線。因為用戶端指定 chat1 子通訊協定，Lambda 整合會傳回 400 錯誤訊息，而且連線失敗。

```
wscat -c wss://api-id.execute-api.region.amazonaws.com/beta -s chat1
error: Unexpected server response: 400
```

下列命令在連線要求中包含支援的子通訊協定。Lambda 整合允許連線。

```
wscat -c wss://api-id.execute-api.region.amazonaws.com/beta -s chat1,myprotocol
connected (press CTRL+C to quit)
```

若要進一步了解如何叫用 WebSocket API，請參閱調用 WebSocket API。

## 在 API Gateway 中控制和管理對 WebSocket API 的存取

API Gateway 支援多種機制來控制和管理對 WebSocket API 的存取。

您可以使用下列機制進行身分驗證和授權：

- 標準 AWS IAM 角色和政策提供靈活且強大的存取控制。您可以使用 IAM 角色和原則來控制誰可以建立和管理您的 API，以及誰可以叫用它們。如需更多詳細資訊，請參閱 使用 IAM 授權控制對 WebSocket API 的存取。
- IAM 標籤可搭配 IAM 政策一起用來控制存取。如需更多詳細資訊，請參閱 使用標籤來控制對 API Gateway REST API 資源的存取。
- Lambda 授權方 是控制 API 存取權的 Lambda 函數。如需詳細資訊，請參閱使用 AWS Lambda REQUEST 授權方控制對 WebSocket APIs存取。

為了改善您的安全狀態，建議您為所有 WebSocket API 上的 $connect 路由設定授權方。您可能需要這樣做，才能符合各種合規架構。如需詳細資訊，請參閱《AWS Security Hub 使用者指南》中的 Amazon API Gateway 控制項。

主題

- 使用 IAM 授權控制對 WebSocket API 的存取
- 使用 AWS Lambda REQUEST 授權方控制對 WebSocket APIs存取

### 使用 IAM 授權控制對 WebSocket API 的存取

WebSocket API 及 REST API 具備類似的 IAM 授權，除了以下例外：

- 除了現有動作 (execute-api、ManageConnections)，Invoke 動作也支援 InvalidateCache。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
```

> **Important**
>
> 使用 IAM 授權時，您必須使用第 4 版簽署程序 (SigV4) 來簽署請求。

例如，您可針對用戶端設定下列政策。除了 Invoke 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 (prod)，並防止所有人在任何階段回傳訊息給已連線的用戶端 (ManageConnections)。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "execute-api:Invoke"
            ],
            "Resource": [
                "arn:aws:execute-api:us-east-1:111122223333:api-id/prod/*"
            ]
        },
        {
            "Effect": "Deny",
            "Action": [
                "execute-api:Invoke"
            ],
            "Resource": [
```

```
                "arn:aws:execute-api:us-east-1:111122223333:api-id/prod/secret"
            ]
        },
        {
            "Effect": "Deny",
            "Action": [
                "execute-api:ManageConnections"
            ],
            "Resource": [
                "arn:aws:execute-api:us-east-1:111122223333:api-id/*"
            ]
        }
    ]
}
```

## 使用 AWS Lambda REQUEST 授權方控制對 WebSocket APIs存取

WebSocket API 及 REST API 具備類似的 Lambda 授權方函數，除了以下例外：

- 您只能使用 `$connect` 路由的 Lambda 授權方函數。
- 您不能使用路徑變數 (`event.pathParameters`)，因為路徑已固定。
- `event.methodArn` 與同等的 REST API 不同，因其沒有 HTTP 方法。若是 `$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。

下列範例顯示對 WebSocket API 的 `REQUEST` 授權方的輸入：

```
{
    "type": "REQUEST",
    "methodArn": "arn:aws:execute-api:us-east-1:123456789012:abcdef123/default/
$connect",
    "headers": {
        "Connection": "upgrade",
        "content-length": "0",
        "HeaderAuth1": "headerValue1",
        "Host": "abcdef123.execute-api.us-east-1.amazonaws.com",
        "Sec-WebSocket-Extensions": "permessage-deflate; client_max_window_bits",
```

```
        "Sec-WebSocket-Key": "...",
        "Sec-WebSocket-Version": "13",
        "Upgrade": "websocket",
        "X-Amzn-Trace-Id": "...",
        "X-Forwarded-For": "...",
        "X-Forwarded-Port": "443",
        "X-Forwarded-Proto": "https"
    },
    "multiValueHeaders": {
        "Connection": [
            "upgrade"
        ],
        "content-length": [
            "0"
        ],
        "HeaderAuth1": [
            "headerValue1"
        ],
        "Host": [
            "abcdef123.execute-api.us-east-1.amazonaws.com"
        ],
        "Sec-WebSocket-Extensions": [
            "permessage-deflate; client_max_window_bits"
        ],
        "Sec-WebSocket-Key": [
            "..."
        ],
        "Sec-WebSocket-Version": [
            "13"
        ],
        "Upgrade": [
            "websocket"
        ],
        "X-Amzn-Trace-Id": [
            "..."
        ],
        "X-Forwarded-For": [
            "..."
        ],
        "X-Forwarded-Port": [
            "443"
        ],
        "X-Forwarded-Proto": [
            "https"
```

```
        ]
    },
    "queryStringParameters": {
        "QueryString1": "queryValue1"
    },
    "multiValueQueryStringParameters": {
        "QueryString1": [
            "queryValue1"
        ]
    },
    "stageVariables": {},
    "requestContext": {
        "routeKey": "$connect",
        "eventType": "CONNECT",
        "extendedRequestId": "...",
        "requestTime": "19/Jan/2023:21:13:26 +0000",
        "messageDirection": "IN",
        "stage": "default",
        "connectedAt": 1674162806344,
        "requestTimeEpoch": 1674162806345,
        "identity": {
            "sourceIp": "..."
        },
        "requestId": "...",
        "domainName": "abcdef123.execute-api.us-east-1.amazonaws.com",
        "connectionId": "...",
        "apiId": "abcdef123"
    }
}
```

下列範例 Lambda 授權方函數為 the section called “Lambda 授權方函數的其他範例” 內 REST API 之 Lambda 授權方函數的 WebSocket 版本：

Node.js

```
  // 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 and QueryString1 query
parameter
  // in the request context match the specified values of
  // of 'headerValue1' and 'queryValue1' respectively.
          export const handler = function(event, context, callback) {
   console.log('Received event:', JSON.stringify(event, null, 2));
```

```
    // 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 ApiId = 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 = {};
     condition.IpAddress = {};

    if (headers.HeaderAuth1 === "headerValue1"
        && queryStringParameters.QueryString1 === "queryValue1") {
         callback(null, generateAllow('me', event.methodArn));
     }  else {
         callback(null, generateDeny('me', event.methodArn));
     }
}

// Helper 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);
}
```

## Python

```
# 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 and QueryString1 query
 parameter
# in the request context match the specified values of
# of 'headerValue1' and 'queryValue1' respectively.

import json


def lambda_handler(event, context):
    print(event)

    # Retrieve request parameters from the Lambda function input:
    headers = event['headers']
    queryStringParameters = event['queryStringParameters']
    stageVariables = event['stageVariables']
    requestContext = event['requestContext']

    # Parse the input for the parameter values
    tmp = event['methodArn'].split(':')
    apiGatewayArnTmp = tmp[5].split('/')
```

```
    awsAccountId = tmp[4]
    region = tmp[3]
    ApiId = apiGatewayArnTmp[0]
    stage = apiGatewayArnTmp[1]
    route = apiGatewayArnTmp[2]

    # Perform authorization to return the Allow policy for correct parameters
    # and the 'Unauthorized' error, otherwise.

    authResponse = {}
    condition = {}
    condition['IpAddress'] = {}

    if (headers['HeaderAuth1'] ==
            "headerValue1" and queryStringParameters["QueryString1"] ==
 "queryValue1"):
        response = generateAllow('me', event['methodArn'])
        print('authorized')
        return json.loads(response)
    else:
        response = generateDeny('me', event['methodArn'])
        print('unauthorized')
        return json.loads(response)

    # Help function to generate IAM policy


def generatePolicy(principalId, effect, resource):
    authResponse = {}
    authResponse['principalId'] = principalId
    if (effect and resource):
        policyDocument = {}
        policyDocument['Version'] = '2012-10-17       '
        policyDocument['Statement'] = []
        statementOne = {}
        statementOne['Action'] = 'execute-api:Invoke'
        statementOne['Effect'] = effect
        statementOne['Resource'] = resource
        policyDocument['Statement'] = [statementOne]
        authResponse['policyDocument'] = policyDocument

    authResponse['context'] = {
        "stringKey": "stringval",
        "numberKey": 123,
```

```
        "booleanKey": True
    }

    authResponse_JSON = json.dumps(authResponse)

    return authResponse_JSON


def generateAllow(principalId, resource):
    return generatePolicy(principalId, 'Allow', resource)


def generateDeny(principalId, resource):
    return generatePolicy(principalId, 'Deny', resource)
```

若要將前述 Lambda 函數設定為 WebSocket API 的 REQUEST 授權方函數，請遵循 REST API 內的相同程序。

若要將 $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。在路由請求設定區段中，選擇編輯。在授權下拉式選單中選取您的授權方，然後選擇儲存變更。

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?
QueryString1=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
```

## API Gateway 中 WebSocket API 的整合

設定 API 路由後，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 端點或服務 AWS 動作。API 整合有整合請求和整合回應。

在本節中，您可以了解如何為您的 WebSocket API 設定整合請求和整合回應。

主題


- 在 API Gateway 中設定一個 WebSocket API 整合請求
- 在 API Gateway 中設定 WebSocket API 整合回應


## 在 API Gateway 中設定一個 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。
- 指定要調用的後端端點。WebSocket API 支援下列整合類型：
  - AWS_PROXY
  - AWS
  - HTTP_PROXY
  - HTTP
  - MOCK

  如需整合類型的詳細資訊，請參閱《API Gateway V2 REST API》中的 IntegrationType。
- 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. 選擇整合請求索引標籤，然後在整合請求設定區段中，選擇編輯。
4. 針對整合類型，選擇下列其中一項：

   - 只有當您的 API 將與您在此帳戶或其他帳戶中建立的 函數整合時，才選擇 Lambda AWS Lambda 函數。

     若要在 中建立新的 Lambda 函數 AWS Lambda、設定 Lambda 函數的資源許可，或執行任何其他 Lambda 服務動作，請改為選擇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 API Gateway 中 REST API 的 HTTP 整合。

- 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API Gateway 中 REST API 的模擬整合。
- 如果您的 API 將與 AWS 服務整合，請選擇 AWS 服務。
- 如果您的 API 將使用 VpcLink 做為私有整合端點，請選擇 VPC 連結。如需詳細資訊，請參閱設定私有整合。

5. 如果您選擇 Lambda 函數，請執行下列動作：

   a. 若您希望使用 Lambda 代理整合或跨帳戶 Lambda 代理整合，請勾選使用 Lambda 代理整合核取方塊。

   b. 針對 Lambda 函數，以下列方式之一指定函數：

      - 若您的 Lambda 函數位於同一個帳戶，請輸入函數名稱，然後從下拉式清單中選取函數。

        > **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。

   c. 若要使用 29 秒的預設逾時值，請將預設逾時保持開啟。若要設定自訂逾時，請選擇預設逾時，然後輸入介於 `50` 和 `29000` 毫秒之間的逾時值。

6. 若您選擇 HTTP (HTTP)，請依照 the section called “ 使用主控台設定整合請求” 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 服務，請依照 the section called “ 使用主控台設定整合請求” 的步驟 6 的指示進行。
9. 若您選擇 VPC 連結，請執行下列動作：

   a. 若您希望將請求代理到 `VPCLink` 端點，請勾選 VPC 代理整合核取方塊。

   b. 針對 HTTP method (HTTP 方法)，選擇最符合 HTTP 後端中方法的 HTTP 方法類型。

   c. 從 VPC 連結下拉式清單中，選取 VPC 連結。您可以在清單下方的文字方塊中選取 `[Use Stage Variables]` 並輸入 **`${stageVariables.vpcLinkId}`**。

      將 API 部署到階段後，您就可以定義 `vpcLinkId` 階段變數，並將其值設定為 `VpcLink` 的 ID。

   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 29 秒的預設逾時值，請將預設逾時保持開啟。若要設定自訂逾時，請選擇預設逾時，然後輸入介於 50 和 29000 毫秒之間的逾時值。

10. 選擇儲存變更。
11. 在請求範本下，執行下列動作：

a. 在請求範本下選擇編輯，以輸入範本選擇表達式。

b. 輸入範本選擇表達式。使用 API Gateway 在訊息承載中尋找的表達式。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。您會在下一個步驟中建立資料映射範本。選擇編輯，儲存您的變更。

c. 選擇建立範本，以建立資料映射範本。針對範本金鑰輸入範本金鑰值，該值將用來選取要對訊息承載中的資料套用的資料映射範本。接著，輸入映射範本。選擇建立範本。

如需範本選擇表達式的相關資訊，請參閱 the section called “範本選擇表達式”。

使用 設定整合請求 AWS CLI

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. 使用下列 create-integration 命令來建立模擬整合。

```
aws apigatewayv2 --region us-east-1 create-integration --api-id aabbccddee --cli-
input-json file://integration-params.json
```

輸出將如下所示：

```
{
    "PassthroughBehavior": "WHEN_NO_MATCH",
    "TimeoutInMillis": 29000,
    "ConnectionType": "INTERNET",
    "IntegrationResponseSelectionExpression": "${response.statuscode}",
    "RequestTemplates": {
        "application/json": "{\"statusCode\":200}"
    },
```

```
    "IntegrationId": "0abcdef",
    "IntegrationType": "MOCK"
}
```

或者，您可以使用 設定代理整合的整合請求 AWS CLI。

1. 在 Lambda 主控台中建立 Lambda 函數，並賦予基本的 Lambda 執行角色。
2. 使用下列 create-integration 命令來建立整合。

```
aws apigatewayv2 create-integration --api-id aabbccddee --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
```

輸出將如下所示：

```
{
    "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"
}
```

WebSocket API 之代理整合的 Lambda 函數的輸入格式

在 Lambda 代理整合中，API Gateway 會將整個用戶端請求映射至後端 Lambda 函數的輸入 `event` 參數。下列範例顯示來自 `$connect` 路由的輸入事件結構，以及來自 API Gateway 傳送至 Lambda 代理整合之 `$disconnect` 路由的輸入事件結構。

Input from the $connect route

```
{
    headers: {
      Host: 'abcd123.execute-api.us-east-1.amazonaws.com',
      'Sec-WebSocket-Extensions': 'permessage-deflate; client_max_window_bits',
      'Sec-WebSocket-Key': '...',
```

```
      'Sec-WebSocket-Version': '13',
      'X-Amzn-Trace-Id': '...',
      'X-Forwarded-For': '192.0.2.1',
      'X-Forwarded-Port': '443',
      'X-Forwarded-Proto': 'https'
    },
    multiValueHeaders: {
      Host: [ 'abcd123.execute-api.us-east-1.amazonaws.com' ],
      'Sec-WebSocket-Extensions': [ 'permessage-deflate; client_max_window_bits' ],
      'Sec-WebSocket-Key': [ '...' ],
      'Sec-WebSocket-Version': [ '13' ],
      'X-Amzn-Trace-Id': [ '...' ],
      'X-Forwarded-For': [ '192.0.2.1' ],
      'X-Forwarded-Port': [ '443' ],
      'X-Forwarded-Proto': [ 'https' ]
    },
    requestContext: {
      routeKey: '$connect',
      eventType: 'CONNECT',
      extendedRequestId: 'ABCD1234=',
      requestTime: '09/Feb/2024:18:11:43 +0000',
      messageDirection: 'IN',
      stage: 'prod',
      connectedAt: 1707502303419,
      requestTimeEpoch: 1707502303420,
      identity: { sourceIp: '192.0.2.1' },
      requestId: 'ABCD1234=',
      domainName: 'abcd1234.execute-api.us-east-1.amazonaws.com',
      connectionId: 'AAAA1234=',
      apiId: 'abcd1234'
    },
    isBase64Encoded: false
  }
```

Input from the $disconnect route

```
{
    headers: {
      Host: 'abcd1234.execute-api.us-east-1.amazonaws.com',
      'x-api-key': '',
      'X-Forwarded-For': '',
      'x-restapi': ''
```

```
  },
  multiValueHeaders: {
    Host: [ 'abcd1234.execute-api.us-east-1.amazonaws.com' ],
    'x-api-key': [ '' ],
    'X-Forwarded-For': [ '' ],
    'x-restapi': [ '' ]
  },
  requestContext: {
    routeKey: '$disconnect',
    disconnectStatusCode: 1005,
    eventType: 'DISCONNECT',
    extendedRequestId: 'ABCD1234=',
    requestTime: '09/Feb/2024:18:23:28 +0000',
    messageDirection: 'IN',
    disconnectReason: 'Client-side close frame status not set',
    stage: 'prod',
    connectedAt: 1707503007396,
    requestTimeEpoch: 1707503008941,
    identity: { sourceIp: '192.0.2.1' },
    requestId: 'ABCD1234=',
    domainName: 'abcd1234.execute-api.us-east-1.amazonaws.com',
    connectionId: 'AAAA1234=',
    apiId: 'abcd1234'
  },
  isBase64Encoded: false
}
```

## 在 API Gateway 中設定 WebSocket API 整合回應

下一節提供有關 WebSocket API 整合回應的簡短概觀，以及如何設定 WebSocket API 整合回應。

**主題**

- 整合回應概觀
- 雙向通訊的整合回應
- 使用 API Gateway 主控台設定整合回應
- 使用 設定整合回應 AWS CLI

### 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，但概念上的行為相同。

WebSocket 路由可設定為雙向或單向通訊。

- 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。

API Gateway 不會將後端回應傳遞至路由回應，除非您有設定路由回應。若要了解設定路由回應，請參閱 the section called “設定 WebSocket API 路由回應”。

#### 雙向通訊的整合回應

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，若找不到配對項目，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的更多資訊，請參閱 the section called “範本選擇表達式”。

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：

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 選擇您的 WebSocket API 並選擇您的路由。
3. 選擇整合請求索引標籤，然後在整合回應設定區段中，選擇建立整合回應。
4. 在回應金鑰中輸入值，此值將可在評估回應選擇表達式之後，於傳出訊息的回應金鑰中找到。例如，您可以輸入 **`/4\d\d/`** 來接收和轉換錯誤的請求錯誤，或者輸入 **`$default`** 來接收並轉換符合範本選擇表達式的所有回應。
5. 在範本選擇表達式中，輸入選擇表達式以評估傳出訊息。
6. 選擇建立回應。
7. 您也可以定義對應範本，以設定傳回訊息承載的轉換。選擇建立範本。
8. 輸入金鑰名稱。如果您要選擇預設範本選擇表達式，請輸入 **`\$default`**。
9. 在程式碼編輯器中，針對回應範本輸入您的映射範本。
10. 選擇建立範本。
11. 選擇部署 API 以部署您的 API。

使用下列 wscat 命令來連線到您的 API。如需 `wscat` 的相關資訊，請參閱 the section called “使用 wscat 以連接到 WebSocket API 和將訊息傳送到其中”。

```
wscat -c wss://api-id.execute-api.us-east-2.amazonaws.com/test
```

當您呼叫路由時，傳回的訊息承載應傳回。

使用 設定整合回應 AWS CLI

以下 create-integration-response 命令會建立 `$default` 整合回應：

```
aws apigatewayv2 create-integration-response \
    --api-id vaz7da96z6 \
```

```
    --integration-id a1b2c3 \
    --integration-response-key '$default'
```

# 請求驗證 API Gateway 中的 WebSocket API

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 的請求失敗，而不呼叫後端、傳送與以下用戶端類似的「錯誤的請求內文」閘道回應，以及在 CloudWatch Logs 中發布驗證結果。以這種方式使用驗證減少對 API 後端的不必要呼叫。

## 模型選擇表達式

您可以使用模型選擇表示式來動態驗證相同路由內的請求。如果您為代理或非代理整合提供模型選取表示式，就會發生模型驗證。找不到相符的模型時，您可能需要將 `$default` 模型定義為回復。如果沒有相符的模型，而且沒有定義 `$default`，驗證會失敗。選取表示式看起來類似 `Route.ModelSelectionExpression` 並評估為 `Route.RequestModels` 的索引鍵。

當您定義 WebSocket API 的路由時，您可以選擇指定模型選取表達式。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。此運算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 “REST API 的資料模型”。

## 使用 API Gateway 主控台設定請求驗證

下列範例說明何在路由上設定請求驗證。

首先，建立模型，然後建立路由。接著，在剛建立的路由上設定請求驗證。最後，部署並測試 API。若要完成本教學課程，您需要 WebSocket API 搭配 `$request.body.action` 做為新路由的路由選擇表達式和整合端點。

您還需要 `wscat` 來連線到 API。如需詳細資訊，請參閱the section called “使用 `wscat` 以連接到 WebSocket API 和將訊息傳送到其中”。

建立裝置

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 選擇 WebSocket API。
3. 在主導覽窗格中，選擇模型。

4. 選擇建立模型。
5. 對於名稱，輸入 **emailModel**。
6. 針對內容類型，輸入 **application/json**。
7. 針對模型結構描述，輸入下列模型：

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "type" : "object",
    "required" : [ "address"],
    "properties" : {
        "address": {
            "type": "string"
        }
    }
}
```

   此模型要求該請求應包含電子郵件地址。
8. 選擇儲存。

在此步驟中，為 WebSocket API 建立路由。

建立路由

1. 在主要導覽窗格中，選擇路由。
2. 選擇 Create route (建立路由)。
3. 對於 Route key (路由金鑰)，輸入 **sendMessage**。
4. 選擇整合類型並指定整合端點。如需更多資訊，請參閱the section called “整合”。
5. 選擇 Create route (建立路由)。

在此步驟中，設定 sendMessage 路由的請求驗證。

設定請求驗證

1. 在路由請求標籤的路由請求設定下，選擇編輯。
2. 針對模型選取表達式，輸入 **${request.body.messageType}**。

   API Gateway 使用 messageType 屬性驗證傳入的請求。

3. 選擇新增請求模型。
4. 對於模型金鑰，輸入 **email**。
5. 對於模型，選擇 emailModel。

   針對此模型，API Gateway 會以 `messageType` 屬性設定為 `email` 的方式來驗證傳入的訊息。

   > **Note**
   >
   > 如果 API Gateway 無法將模型選取表達式與模型金鑰相配，便會選取 `$default` 模型。如果沒有 `$default` 模型，則驗證會失敗。對於生產 API，我們建議您建立 `$default` 模型。

6. 選擇儲存變更。

在此步驟中，您會部署和測試 API。

部署和測試 API

1. 選擇部署 API。
2. 從下拉式清單中選擇所需的階段，或輸入新階段的名稱。
3. 選擇部署。
4. 在主導覽窗格中，選擇階段。
5. 複製 API 的 WebSocket URL。URL 看起來應該會像這樣：`wss://abcdef123.execute-api.us-east-2.amazonaws.com/production`。
6. 開啟新的終端機，並使用下列參數執行 wscat 命令。

```
wscat -c wss://abcdef123.execute-api.us-west-2.amazonaws.com/production
```

```
Connected (press CTRL+C to quit)
```

7. 使用下列命令測試 API。

```
{"action": "sendMessage", "messageType": "email"}
```

```
{"message": "Invalid request body", "connectionId":"ABCD1=234",
 "requestId":"EFGH="}
```

API Gateway 將使請求無法過關。

使用下一個命令，將有效的請求傳送至您的 API。

```
{"action": "sendMessage", "messageType": "email", "address":
 "mary_major@example.com"}
```

## API Gateway 中用於 WebSocket API 的資料轉換

在 API Gateway 中，WebSocket 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 可讓您使用映射範本轉換，將承載從方法請求映射至對應的整合請求，以及從整合回應映射至對應的方法回應。您會建立映射範本並指定範本選擇表達式，以決定要使用哪個範本來執行必要的資料轉換。

您可以使用資料映射，將 路由請求 中的資料映射至後端整合。如需進一步了解，請參閱the section called “資料映射”。

### 對應範本和模型

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。如需 API Gateway 對應範本的詳細資訊，請參閱 API Gateway 中 REST API 的映射範本轉換。

根據 JSON 結構描述草稿第 4 版，承載可以有一個資料模型。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。如需 API Gateway 模型的詳細資訊，請參閱 REST API 的資料模型。

### 範本選擇表達式

若要使用對應範本轉換承載，您可以整合請求或整合回應中指定 WebSocket API 範本選取運算式。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.respo` 和靜態值。

## 整合回應選擇表達式

設定 WebSocket API 的整合回應時，您可選擇指定整合回應選擇表達式。此運算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。

## 設定 API Gateway 中 WebSocket API 的資料映射

資料映射可讓您將路由請求中的資料映射至後端整合。

> **Note**
>
> 不支援 WebSocket APIs的資料映射 AWS 管理主控台。您必須使用 AWS CLI AWS CloudFormation、 或 SDK 來設定資料映射。

主題

- 將路由請求資料對應到整合請求參數
- 範例

### 將路由請求資料對應到整合請求參數

整合請求參數可以從任何定義的路由請求參數，請求內文、`context` 或 `stage` 的變量，以及靜態值進行映射。

下表展示整合請求資料映射表達式。在下表中，*`PARAM_NAME`* 是指定參數類型的路由請求參數的名稱。它必須符合規則表達式 `'^[a-zA-Z0-9._$-]+$]'`。*`JSONPath_EXPRESSION`* 是請求內文的 JSON 欄位的 JSONPath 表達式。

| 映射的資料來源 | 對應表達式 |
| --- | --- |
| 請求查詢字串 (僅支援 $connect 路由) | route.request.querystring. *PARAM_NAME* |
| 請求標頭 (僅支援 $connect 路由) | route.request.header. *PARAM_NAME* |
| 多值請求查詢字串 (僅支援 $connect 路由) | route.request.multivaluequerystring. *PARAM_NAME* |
| 多值請求標頭 (僅支援 $connect 路由) | route.request.multivalueheader. *PARAM_NAME* |
| 請求內文 | route.request.body. *JSONPath_EXPRESSION* |
| 階段變數 | stageVariables. *VARIABLE_NAME* |
| 環境變數 | context.*VARIABLE_NAME* 必須是支援的環境變數之一。 |
| 靜態值 | *'STATIC_VALUE'* 。*STATIC_VALUE* 是字串常值，而且必須以一對單引號括住。 |

當您建立資料映射時，使用 AWS CLI 時，請務必遵循在 中使用常值與字串的正確格式 AWS CLI。如需詳細資訊，請參閱《AWS Command Line Interface 使用者指南》中的搭配 AWS CLI中的字串使用引號和常值。

範例

下列 AWS CLI 範例會設定資料映射。如需範例 CloudFormation 範本，請參閱 websocket-data-mapping.yaml。

將用戶端的連線 ID 映射至整合請求中的標頭

以下 update-integration 命令會將用戶端的 connectionId 映射至後端整合請求中的 connectionId 標頭：

```
aws apigatewayv2 update-integration \
```

```
    --integration-id abc123 \
    --api-id a1b2c3d4 \
    --request-parameters
 'integration.request.header.connectionId'='context.connectionId'
```

將查詢字串參數映射至整合請求中的標頭

下列範例會將 authToken 查詢字串參數映射至整合請求中的 authToken 標頭。

1. 使用下列 update-route 命令，將 authToken 查詢字串參數新增至路由的請求參數。

```
aws apigatewayv2 update-route --route-id 0abcdef \
    --api-id a1b2c3d4 \
    --request-parameters '{"route.request.querystring.authToken": {"Required":
 false}}'
```

2. 使用下列 update-integration 命令，將查詢字串參數映射至後端整合請求中的 authToken 標頭。

```
aws apigatewayv2 update-integration \
    --integration-id abc123 \
    --api-id a1b2c3d4 \
    --request-parameters
 'integration.request.header.authToken'='route.request.querystring.authToken'
```

3. (選用) 如有必要，使用下列 delete-route-request-parameter 從路由的請求參數中刪除 authToken 查詢字串參數。

```
aws apigatewayv2 delete-route-request-parameter \
    --route-id 0abcdef \
    --api-id a1b2c3d4 \
    --request-parameter-key 'route.request.querystring.authToken'
```

## 用於 API Gateway 的 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 | 說明 |
| --- | --- |
| $context.connectionId | 連線的唯一 ID，可用來回呼用戶端。連線 ID 是可變長度字串，其長度和格式可能會變更。 |

| 參數 | 說明 |
| --- | --- |
| $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 的識別碼。 |
| $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 函數所傳回之權杖建立關聯的主要使用者識別。 |

| 參數 | 說明 |
| --- | --- |
| `$context.authorizer.` *property* | API Gateway Lambda 授權方函數所傳回 `context` 映射之指定索引鍵/值組的字串化值。例如，如果授權方傳回下列 `context` 映射：<br>`"context" : { "key": "value", "numKey": 1, "boolKey": true }`<br>呼叫 `$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。 |
| `$context.error.messageString` | `$context.error.message` 的引用值，即 `"$context.error.message"`。 |
| `$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。 |

| 參數 | 說明 |
| --- | --- |
| `$context.identity.cognitoAuthenticationProvider` | 提出請求的發起人所使用的所有 Amazon Cognito 驗證提供者清單 (以逗號分隔)。僅在使用 Amazon Cognito 登入資料簽署請求時才可使用。<br><br>例如，適用於 Amazon Cognito 使用者集區的身分，`cognito-idp.`*`region`*`.amazonaws.com/`*`user_pool_id`*`,cognito-idp.`*`region`*`.amazonaws.com/`*`user_pool_id`*`:CognitoSignIn:`*`token subject claim`*<br><br>如需有關 Amazon Cognito 驗證提供者的詳細資訊，請參閱《Amazon Cognito 開發人員指南》中的使用聯合身分。 |
| `$context.identity.cognitoAuthenticationType` | 提出請求的發起人的 Amazon Cognito 驗證類型。僅在使用 Amazon Cognito 登入資料簽署請求時才可使用。可能的值包括用於已驗證身分的 `authenticated` 和未經驗證身分的 `unauthenticated` 。 |
| `$context.identity.cognitoIdentityId` | 提出請求的發起人的 Amazon Cognito 身分 ID。僅在使用 Amazon Cognito 登入資料簽署請求時才可使用。 |
| `$context.identity.cognitoIdentityPoolId` | 提出請求的發起人的 Amazon Cognito 身分集區 ID。僅在使用 Amazon Cognito 登入資料簽署請求時才可使用。 |
| `$context.identity.sourceIp` | 對 API Gateway 提出請求之即時 TCP 連線的來源 IP 地址。 |
| `$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。 |

| 參數 | 說明 |
| --- | --- |
| `$context.identity.userArn` | 身分驗證之後識別之有效使用者的 Amazon Resource Name (ARN)。 |
| `$context.requestTime` | CLF 格式化請求時間 (`dd/MMM/yyyy:HH:mm:ss +-hhmm` )。 |
| `$context.requestTimeEpoch` | Epoch 格式化的要求時間，以毫秒為單位。 |
| `$context.stage` | API 呼叫的部署階段 (例如，Beta 或 Prod)。 |
| `$context.status` | 回應狀態。 |
| `$input.body` | 傳回原始承載作為字串。 |
| `$input.json(x)` | 此函數會評估 JSONPath 表達式，並傳回結果作為 JSON 字串。<br><br>例如，`$input.json('$.pets')` 會傳回代表 pets 結構的 JSON 字串。<br><br>如需 JSONPath 的詳細資訊，請參閱 JSONPath 或適用於 Java 的 JSONPath。 |

| 參數 | 說明 |
| --- | --- |
| `$input.path(x)` | 採用 JSONPath 表達式字串 (x)，並傳回結果的 JSON 物件呈現。這可讓您存取和運用 Apache Velocity 範本語言 (VTL) 中原生承載的元素。<br><br>例如，如果表達式 `$input.path('$.pets')` 傳回如下物件：<br><br>`[ { "id": 1, "type": "dog", "price": 249.99 }, { "id": 2, "type": "cat", "price": 124.99 }, { "id": 3, "type": "fish", "price": 0.99 } ]`<br><br>`$input.path('$.pets').count()` 會傳回 "3"。<br><br>如需 JSONPath 的詳細資訊，請參閱 JSONPath 或適用於 Java 的 JSONPath。 |
| `$stageVariables.` *<variable_name>* | *<variable_name>* 代表階段變數名稱。 |
| `$stageVariables['` *<variable_name>* `']` | *<variable_name>* 代表任何階段變數名稱。 |
| `${stageVariables['` *<variable_name>* `']}` | *<variable_name>* 代表任何階段變數名稱。 |

| 參數 | 說明 |
| --- | --- |
| `$util.escapeJavaScript()` | 使用 JavaScript 字串規則來逸出字串中的字元。<br><br>**Note**<br>此函數會將任何一般單引號 (') 轉換為逸出單引號 (\')。不過，逸出單引號不適用於 JSON。因此，將此函數的輸出用於 JSON 屬性時，您必須將任何逸出單引號 (\') 轉換為一般單引號 (')。下列範例顯示這種情況：<br><br>`$util.escapeJavaScript( data).replaceAll("\\'",'''')` |

| 參數 | 說明 |
| --- | --- |
| $util.parseJson() | 採用「字串化」JSON，並傳回結果的物件呈現。您可以使用此函數的結果，來存取和運用 Apache Velocity 範本語言 (VTL) 中原生承載的元素。例如，如果您有下列承載：<br><br>`{"errorMessage":"{\"key1\":\"var1\", \"key2\":{\"arr\":[1,2,3]}}"}`<br><br>並使用下列映射範本<br><br>`#set ($errorMessageObj = $util.parseJson($input.path('$.errorMessage')))`<br>`{`<br>`  "errorMessageObjKey2ArrVal" : $errorMessageObj.key2.arr[0]`<br>`}`<br><br>您將會收到下列輸出：<br><br>`{`<br>`  "errorMessageObjKey2ArrVal" : 1`<br>`}` |
| $util.urlEncode() | 將字串轉換為 "application/x-www-form-urlencoded" 格式。 |
| $util.urlDecode() | 解碼 "application/x-www-form-urlencoded" 字串。 |
| $util.base64Encode() | 將資料編碼為 base64 編碼字串。 |
| $util.base64Decode() | 解碼 base64 編碼字串中的資料。 |

## API Gateway 中 WebSocket API 的二進位媒體類型

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。

## 調用 WebSocket API

在您部署 WebSocket API 之後，用戶端應用程式可以連接到其中和傳送訊息，且後端服務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”。
- 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”。
- 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. 執行下列命令來安裝 `wscat`：

   ```
   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://aabbccddee.execute-api.us-east-1.amazonaws.com/test/
   ```

您需要使用實際 API ID 取代 *aabbccddee*，該 ID 會顯示在 API Gateway 主控台中或由 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。

4. 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。

5. 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，所以您必須使用 Signature 第 4 版 (SigV4) 來進行簽署。若要執行此作業，您可以使用 API Gateway 管理 API。如需詳細資訊，請參閱 ApiGatewayManagementApi。

在下列命令中，您需要將 取代*{api-id}*為實際的 API ID，該 ID 會顯示在 API Gateway 主控台中，或由 AWS CLI createe-api 命令傳回。使用此命令之前，您必須先建立連線。

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}
```

您需要以 URL 編碼處理命令，如以下範例所示：

```
awscurl --service execute-api -X POST -d "hello world" https://aabbccddee.execute-api.us-east-1.amazonaws.com/prod/%40connections/R0oXAdfD0kwCH6w%3D
```

若要取得用戶端的最新連線狀態，請使用下列命令：

```
GET https://{api-id}.execute-api.us-east-1.amazonaws.com/{stage}/
@connections/{connection_id}
```

若要中斷用戶端連線，請使用下列命令：

```
DELETE https://{api-id}.execute-api.us-east-1.amazonaws.com/{stage}/
@connections/{connection_id}
```

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 代理整合搭配 Node.js Lambda 函數，您可以如下所示建置 URL，並傳送訊息到連線用戶端：

```
import {
  ApiGatewayManagementApiClient,
  PostToConnectionCommand,
} from "@aws-sdk/client-apigatewaymanagementapi";

export const handler = async (event) => {
  const domain = event.requestContext.domainName;
  const stage = event.requestContext.stage;
  const connectionId = event.requestContext.connectionId;
  const callbackUrl = `https://${domain}/${stage}`;
  const client = new ApiGatewayManagementApiClient({ endpoint: callbackUrl });

  const requestParams = {
    ConnectionId: connectionId,
    Data: "Hello!",
  };

  const command = new PostToConnectionCommand(requestParams);
```

```
  try {
    await client.send(command);
  } catch (error) {
    console.log(error);
  }

  return {
    statusCode: 200,
  };
};
```

如果您使用 WebSocket API 的自訂網域名稱，請從函數程式碼中移除 `stage` 變數。

傳送回呼訊息時，您的 Lambda 函數必須具有呼叫 API Gateway 管理 API 的權限。如果在建立連線前或用戶端中斷連線後張貼訊息，您可能會收到含有 `GoneException` 的錯誤訊息。

# 發布 WebSocket API 以供客戶調用

只是建立和開發 API Gateway API，並不代表使用者可以自動呼叫 API。若要使其可供呼叫，您必須將 API 部署到階段。此外，您可能會想要自訂使用者用於存取 API 的 URL。您可以為它提供一個與您品牌一致的網域，或是比 API 預設 URL 更好記的網域。

在本節中，您可以了解如何部署 API，以及如何自訂提供給使用者以存取 API 的 URL。

> **Note**
>
> 為了增強 API Gateway API 的安全性，`execute-api.{region}.amazonaws.com` 網域會在公用尾碼清單 (PSL) 中註冊。為了加強安全性，如果您需要在 API Gateway API 的預設網域名稱中設定敏感性 Cookie，我們建議您使用具 `__Host-` 前置詞的 Cookie。此做法將有助於保護您的網域免受跨站請求偽造 (CSRF) 攻擊。如需更多資訊，請參閱 Mozilla 開發人員網路中的設定 Cookie 頁面。

主題

- 在 API Gateway 中建立 WebSocket API 的各階段
- 在 API Gateway 中部署 WebSocket API
- API Gateway 中適用於 WebSocket API 的安全政策
- API Gateway 中的 WebSocket API 私有 API 的自訂網域名稱

# 在 API Gateway 中建立 WebSocket API 的各階段

API 階段是 API 生命週期狀態的邏輯參考 (例如，`dev`、`prod`、`beta` 或 `v2`)。API 階段是由其 API ID 及階段名稱來識別，而且它們會包含在您用來呼叫 API 的 URL 中。每個階段都是 API 部署的具名參考，且可供用戶端應用程式呼叫。

部署是 API 組態的快照。將 API 部署到階段之後，用戶端就可以叫用它。您必須部署 API 才能讓變更生效。

## 階段變數

階段變數是您可為 WebSocket API 定義階段的索引鍵/值對。它們的作用如同環境變數，而且可用於 API 設定。

例如，您可以定義階段變數，然後將其值設定為 HTTP Proxy 整合的 HTTP 端點。稍後，您可以使用相關聯的階段變數名稱來參照端點。如此，您可以在每個階段使用不同的端點來使用相同的 API 設定。同樣地，您可以使用階段變數，為 API 的每個階段指定不同的 AWS Lambda 函數整合。

> **Note**
>
> 階段變數並非用於敏感資料，例如登入資料。若要將敏感資料傳遞至整合，請使用 AWS Lambda 授權方。您可以將敏感資料傳遞至 Lambda 授權方輸出中的整合。如需進一步了解，請參閱the section called “Lambda 授權方回應格式”。

### 範例

若要使用階段變數來自訂 HTTP 整合端點，您必須先將階段變數 (例如，`url`) 的名稱和值設定為 `example.com`。之後，設定 HTTP 代理整合。您可以告訴 API Gateway 使用階段變數值 **`http://${stageVariables.url}`**，而不需要輸入端點的 URL。此值會指示 API Gateway 在執行時間替換您的階段變數 `${}`，視您 API 的階段而定。

您可以使用類似的方式來參考階段變數，以指定 Lambda 函數名稱或 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。以下 add-permission 命令會新增必要的許可：

```
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
```

## API Gateway 階段變數參考

### HTTP 整合 URI

您可以使用階段變數作為 HTTP 整合 URI 的一部分，如下例範例所示。

- 不含通訊協定的完整 URI – `http://${stageVariables.<variable_name>}`
- 完整的網域 – `http://${stageVariables.<variable_name>}/resource/operation`
- 子網域 – `http://${stageVariables.<variable_name>}.example.com/resource/operation`
- 路徑 – `http://example.com/${stageVariables.<variable_name>}/bar`
- 查詢字串 – `http://example.com/foo?q=${stageVariables.<variable_name>}`

### Lambda 函數

您可以使用階段變數取代 Lambda 函數名稱或別名，如下列範例所示。

- `arn:aws:apigateway:<region>:lambda:path/2015-03-31/functions/arn:aws:lambda:<region>:<account_id>:function:${stageVariables.<function_variable_name>}/invocations`
- `arn:aws:apigateway:<region>:lambda:path/2015-03-31/functions/arn:aws:lambda:<region>:<account_id>:function:<function_name>:${stageVariables.<version_variable_name>}/invocations`

> **Note**
>
> 若要使用 Lambda 函數的階段變數，函數必須與 API 位於相同的帳戶中。階段變數不支援跨帳戶 Lambda 函數。

### AWS 整合憑證

您可以使用階段變數做為 AWS 使用者或角色登入資料 ARN 的一部分，如下列範例所示。

- `arn:aws:iam::<account_id>:${stageVariables.<variable_name>}`

# 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，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**
>
> 每次更新 API 時，都必須將其重新部署。對階段設定以外的任何變更都需要重新部署，對下列資源進行修改時也是如此：
>
> - 路由
> - 整合
> - 授權方
>
> 在預設情況下，每個 API 只能有 10 個階段。建議您針對部署重複使用這些階段。

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。如需更多詳細資訊，請參閱 the section called “自訂網域名稱”。

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。如需階段的詳細資訊，請參閱 the section called “設定階段”。

主題

- 使用 建立 WebSocket API 部署 AWS CLI
- 使用 API Gateway 主控台建立 WebSocket API 部署

## 使用 建立 WebSocket API 部署 AWS CLI

下列 create-deployment 命令會建立部署：

```
aws apigatewayv2 --region us-east-1 create-deployment --api-id aabbccddee
```

輸出將如下所示：

```
{
    "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。

以下 create-stage 命令會建立一個新階段，並將其與部署建立關聯：

```
aws apigatewayv2 --region us-east-1 create-stage --api-id aabbccddee --deployment-id
 fedcba --stage-name test
```

輸出看起來如下：

```
{
    "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": {}
}
```

您也可以使用新建立的部署 ID (*deployment-id*) 來更新階段的 deploymentId 屬性，以重複使用現有階段。以下 update-stage 命令會更新階段的部署 ID：

```
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 主控台並選擇 API。
2. 選擇部署 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。

## API Gateway 中適用於 WebSocket API 的安全政策

API Gateway 會針對所有 WebSocket API 端點強制執行 TLS_1_2 的安全政策。

安全政策是由 Amazon API Gateway 提供的最低 TLS 版本與密碼套件的預先定義組合。TLS 通訊協定可解決用戶端和伺服器間的竄改與竊聽等網路安全問題。當用戶端透過自訂網域建立 API 的 TLS 信號交換時，安全政策會強制執行用戶端可選擇使用的 TLS 版本和密碼套件。此安全政策接受 TLS 1.2 和 TLS 1.3 流量，並拒絕 TLS 1.0 流量。

### WebSocket API 支援的 TLS 通訊協定和密碼

下表說明 WebSocket API 支援的 TLS 通訊協定。

| TLS 通訊協定 | TLS_1_2 安全政策 |
| --- | --- |
| TLSv1.3 | 是 |

| TLS 通訊協定 | TLS_1_2 安全政策 |
| --- | --- |
| TLSv1.2 | 是 |

下表說明適用於 WebSocket API 的 TLS 1_2 安全政策的 TLS 加密。

| TLS 加密 | TLS_1_2 安全政策 |
| --- | --- |
| TLS_AES_128_GCM_SHA256 | 是 |
| TLS_AES_256_GCM_SHA384 | 是 |
| TLS_CHACHA20_POLY1305_SHA256 | 是 |
| ECDHE-ECDSA-AES128-GCM-SHA256 | 是 |
| ECDHE-RSA-AES128-GCM-SHA256 | 是 |
| ECDHE-ECDSA-AES128-SHA256 | 是 |
| ECDHE-RSA-AES128-SHA256 | 是 |
| ECDHE-ECDSA-AES256-GCM-SHA384 | 是 |
| ECDHE-RSA-AES256-GCM-SHA384 | 是 |
| ECDHE-ECDSA-AES256-SHA384 | 是 |
| ECDHE-RSA-AES256-SHA384 | 是 |
| AES128-GCM-SHA256 | 是 |
| AES128-SHA256 | 是 |
| AES256-GCM-SHA384 | 是 |
| AES256-SHA256 | 是 |

### OpenSSL 和 RFC 密碼名稱

OpenSSL 和 IETF RFC 5246 使用相同密碼的不同名稱。如需密碼名稱的清單，請參閱 the section called “OpenSSL 和 RFC 密碼名稱”。

### REST API 和 HTTP API 的相關資訊

如需 REST API 和 HTTP API 的詳細資訊，請參閱 the section called “選擇安全政策” 和 the section called “API Gateway 中 HTTP API 的安全政策”。

## API Gateway 中的 WebSocket API 私有 API 的自訂網域名稱

自訂網域名稱是更簡單且更直觀的 URL，可提供給 API 使用者。

部署 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* 即為 (AWS 區域)，而 *stage* 是您在部署 API 時所指定。

URL 的主機名稱部分 (即 *api-id*.execute-api.*region*.amazonaws.com) 指的是 API 端點。預設 API 端點是隨機產生的，不容易取回，而且不方便使用。

使用自訂網域名稱，您就能設定 API 的主機名稱，並選擇基本路徑 (例如，myservice) 將替代 URL 映射至您的 API。例如，更方便使用者使用的 API 基本 URL 可以成為：

```
https://api.example.com/myservice
```

### 考量事項

下列考量事項可能會影響您使用自訂網域名稱。

- 如果您將自訂網域名稱映射至 WebSocket API，則無法映射至 REST API 或 HTTP API。
- 僅支援區域性自訂網域名稱。
- 對於最低 TLS 版本，僅 TLS 1.2 受支援。
- 您必須建立或更新 DNS 提供者的資源記錄，以映射至您的 API 端點。如果沒有這種映射，送往自訂網域名稱的 API 請求無法到達 API Gateway。
- 您可以使用萬用字元憑證來支援幾乎不限數量的網域名稱，而不會超過預設配額。如需更多詳細資訊，請參閱 the section called “萬用字元自訂網域名稱”。

## 先決條件

以下是自訂網域名稱的先決條件。

註冊網域名稱

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。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。您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。

您的網域名稱必須遵循 RFC 1035 規格，每個標籤最多可有 63 個八位元組，總共有 255 個八位元組。

自訂網域名稱的憑證

必須為 API 設定自訂網域名稱前，您必須先在 ACM 中備妥 SSL/TLS 憑證。如果您建立自訂網域名稱的 AWS 區域中無法使用 ACM，則必須將憑證匯入該區域中的 API Gateway。

若要匯入 SSL/TLS 憑證，您必須提供 PEM 格式化 SSL/TLS 憑證內文、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。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。若要使用網域名稱的 AWS 受管憑證，您只需要參考其 ARN。

如果您的應用程式是使用憑證關聯 (有時稱為 SSL 關聯) 來關聯 ACM 憑證，則 AWS 續約憑證後，應用程式可能會無法連線到您的網域。如需詳細資訊，請參閱 AWS Certificate Manager 使用者指南中的憑證關聯問題。

## 萬用字元自訂網域名稱

使用萬用字元自訂網域名稱，您可以支援幾乎無限數目的網域名稱，而不會超過預設配額。例如，您可以為每個客戶提供其自己的網域名稱，*customername*.api.example.com。

若要建立萬用字元自訂網域名稱，可以指定萬用字元 (*) 作為自訂網域的第一個子網域，藉以表示根網域所有可能的子網域。

例如，萬用字元自訂網域名稱 *.example.com 會產生如 a.example.com、b.example.com 和 c.example.com 等子網域，而這些子網域全都路由至相同的網域。

萬用字元自訂網域名稱支援來自 API Gateway 標準自訂網域名稱的相異組態。例如，您可以在單一 AWS 帳戶中，設定 *.example.com 和 a.example.com 採取不同的行為。

您可以使用 $context.domainName 和 $context.domainPrefix 內容變數來判斷用戶端用來呼叫 API 的網域名稱。若要進一步了解環境變數，請參閱 用於 API Gateway 資料轉換的變數。

若要建立萬用字元自訂網域名稱，您必須提供由 ACM 發行並已經使用 DNS 或電子郵件驗證方法驗證的憑證。

> **Note**
>
> 如果不同的 AWS 帳戶已建立與萬用字元自訂網域名稱衝突的自訂網域名稱，則您無法建立萬用字元自訂網域名稱。例如，如果帳戶 A 已建立 a.example.com，則帳戶 B 無法建立萬用字元自訂網域名稱 *.example.com。
> 如果帳戶 A 和帳戶 B 共用擁有者，您可以聯絡 AWS 支援中心，以請求例外狀況。

## 自訂網域名稱的後續步驟

若要設定 HTTP API 的自訂網域名稱，請參閱《API Gateway 開發人員指南》中的 REST API 相關章節文件。

首先，指定自訂網域名稱所用的憑證。如需更多詳細資訊，請參閱 the section called “在 中備妥憑證 AWS Certificate Manager”。接著，建立區域性自訂網域名稱。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”。

## 將 API 階段映射至 WebSocket API 的自訂網域名稱

您可以使用 API 映射將 API 階段連線至自訂網域名稱。建立網域名稱並設定 DNS 記錄之後，您可以使用 API 映射，透過自訂網域名稱將流量傳送至您的 API。

API 映射指定一個 API，一個階段，以及選擇性用於映射的路徑。例如，您可以將 API 的 production 階段映射至 wss://api.example.com/orders。

建立 API 映射之前，您必須先擁有 API、階段和自訂網域名稱。如需進一步了解如何建立自訂網域名稱，請參閱the section called “設定區域性自訂網域名稱”。

限制

- 在 API 映射中，自訂網域名稱和映射的 API 必須位於同一個 AWS 帳戶中。
- API 映射只能包含字母、數字和下列字元：$-_.+!*'()。
- API 映射中路徑的最大長度為 300 個字元。
- 您無法將 WebSocket API 映射至與 HTTP API 或 REST API 相同的自訂網域名稱。

- 如果您建立具有多層的 API 映射，API Gateway 會將所有標頭名稱轉換為小寫。

### 建立 API 映射

若要建立 API 映射，您必須先建立自訂網域名稱、API 和階段。如需建立自訂網域名稱的資訊，請參閱the section called “設定區域性自訂網域名稱”。

#### AWS 管理主控台

建立 API 映射

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 選擇 Custom domain names (自訂網域名稱)。
3. 選取您已建立的自訂網域名稱。
4. 選擇 API mappings (API 映射)。
5. 選擇 Configure API mappings (設定 API 映射)。
6. 選擇 Add new mapping (新增映射)。
7. 輸入 API、Stage (階段)，以及選擇性地輸入 Path (路徑)。
8. 選擇儲存。

#### AWS CLI

以下 create-api-mapping 命令會建立 API 映射。在此範例中，API Gateway 會將請求傳送至 `api.example.com/v1`，到指定的 API 和階段。

```
aws apigatewayv2 create-api-mapping \
    --domain-name api.example.com \
    --api-mapping-key v1 \
    --api-id a1b2c3d4 \
    --stage test
```

#### CloudFormation

以下 CloudFormation 範例會建立 API 映射。

```
MyApiMapping:
  Type: 'AWS::ApiGatewayV2::ApiMapping'
  Properties:
    DomainName: api.example.com
```

```
    ApiMappingKey: 'v1'
    ApiId: !Ref MyApi
    Stage: !Ref MyStage
```

## WebSocket API 自訂網域名稱的 IP 位址類型

建立自訂網域名稱時，您可以指定可調用網域的 IP 位址類型。您可以選擇 IPv4 來解析 IPv4 位址以調用網域，也可以選擇雙堆疊，以同時允許 IPv4 和 IPv6 位址調用您的網域。我們建議您將 IP 位址類型設定為雙堆疊，以緩解 IP 空間耗盡或用於安全狀態。如需雙堆疊 IP 位址類型優點的詳細資訊，請參閱 AWS 上的 IPv6。

### IP 位址類型的考量事項

下列考量事項可能會影響您使用 IP 位址類型。

- API Gateway 自訂網域名稱的預設 IP 位址類型為 IPv4。
- 您的自訂網域名稱不需要針對所有與其對應的 API 使用相同 IP 位址類型。如果您停用預設 API 端點，則可能會影響呼叫方調用 API 的方式。

### 變更自訂網域名稱的 IP 位址類型

您可以藉由更新網域名稱的端點組態來變更 IP 位址類型。您可以使用 AWS 管理主控台、AWS CLI、CloudFormation 或 AWS SDK 來更新端點組態。

#### AWS 管理主控台

若要變更自訂網域名稱的 IP 位址類型

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 選擇公有自訂網域名稱。
3. 選擇端點組態。
4. 針對 IP 位址類型，選取 IPv4 或雙堆疊。
5. 選擇儲存。

#### AWS CLI

以下 update-domain-name 命令會將 API 更新為雙堆疊 IP 位址類型：

```
aws apigatewayv2 update-domain-name \
   --domain-name dualstack.example.com \
   --domain-name-configurations CertificateArn=arn:aws:acm:us-
east-1:111122223333:certificate/abcd1234-5678-abc,IpAddressType=dualstack
```

輸出將如下所示：

```
{
    "ApiMappingSelectionExpression": "$request.basepath",
    "DomainName": "dualstack.example.com",
    "DomainNameConfigurations": [
        {
            "ApiGatewayDomainName": "d-abcd1234.execute-api.us-
east-1.amazonaws.com",
            "CertificateArn": "arn:aws:acm:us-east-1:111122223333:certificate/
abcd1234-5678-abc",
            "DomainNameStatus": "AVAILABLE",
            "EndpointType": "REGIONAL",
            "HostedZoneId": "Z3LQWSYCGH4ADY",
            "SecurityPolicy": "TLS_1_2",
            "IpAddressType": "dualstack"
        }
    ],
    "Tags": {}
}
```

## 停用 WebSocket API 的預設端點

預設情況下，用戶端可以使用 API Gateway 為 API 產生的 `execute-api` 端點來調用 API。若要確保用戶端只能使用自訂網域名稱來存取您的 API，請停用預設 `execute-api` 端點。當您停用預設端點時，它會影響 API 的所有階段。

下列程序顯示如何停用 WebSocket API 的預設端點。

AWS 管理主控台

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 選擇 WebSocket API。
3. 選擇 API 設定。
4. 在 API 詳細資訊上，選擇編輯。

5. 針對預設端點，選取非作用中。
6. 選擇儲存變更。
7. 在主要導覽窗格中，選擇路由。
8. 選擇部署，然後重新部署您的 API 或建立新的階段，以使變更生效。

AWS CLI

以下 update-api 命令會停用 WebSocket API 的預設端點：

```
aws apigatewayv2 update-api \
    --api-id abcdef123 \
    --disable-execute-api-endpoint
```

停用預設端點之後，您必須部署 API，變更才會生效。

下列 AWS CLI 命令會建立部署。

```
aws apigatewayv2 create-deployment \
    --api-id abcdef123 \
    --stage-name dev
```

# 在 API Gateway 中保護您的 WebSocket API

API Gateway 提供多種方法來保護 API 免於遭受特定威脅，例如惡意使用者或流量高峰。您可以使用產生 SSL 憑證或設定限流目標等策略來保護 API。如需產生 SSL 憑證的詳細資訊，請參閱 the section called “API Gateway 產生的憑證”。本節其餘部分涵蓋設定限流目標。

您可以為您的 API 設定調節，以防止 API 接收過多請求。調節會依最佳作法來套用，它們都應該視為目標，而非確定的請求上限。

API Gateway 會使用字符儲存貯體演算法將字符計算為請求，進而調節傳送給 API 的請求量。具體而言，API Gateway 會按區域檢查帳戶中所有 API 的速率和爆量請求次數。在字符儲存貯體演算法中，爆量可以實現預先定義的超限，但在某些情況下，其他因素也可能導致超限。

提交的請求量超出穩定狀態請求率和爆量限制時，API Gateway 會開始調節請求量。此時用戶端可能會收到 429 Too Many Requests 的錯誤回應。發現這類例外狀況時，用戶端可以採用限制速率的方式來重新提交失敗的請求。

身為 API 開發人員，您可以設定個別 API 階段或路由的目標限制，來改善您帳戶中所有 API 的整體效能。

## 每個區域的帳戶層級調節

預設情況下，API Gateway 會按區域限制 AWS 帳戶內所有 API 的每秒穩定狀態請求量 (RPS)。它還會按區域限制 AWS 帳戶內所有 API 的爆量 (即儲存貯體大小上限)。在 API Gateway 中，爆量限制代表 API Gateway 傳回 `429 Too Many Requests` 錯誤回應前可實現的並行請求提交數上限。如需有關調節配額的詳細資訊，請參閱 配額和重要說明。

帳戶型限制會套用至指定區域內某帳戶的所有 API。帳戶層級速率限制可按請求提高。使用較短逾時值和較小酬載的 API 可擁有更高的上限。如需請求提高區域內帳戶層級的調節限制，請與 AWS 支援中心聯絡。如需詳細資訊，請參閱配額和重要說明。請注意，這些限制不能高於限 AWS 流限制。

## 路由層級調節

您可以設定路由層級調節，來覆寫特定階段或 API 中個別路由的帳戶層級請求調節限制。預設路由調節限制不能超出帳戶層級速率限制。

您可以使用 AWS CLI設定路由層級的節流設定。以下 update-stage 命令會針對指定的 API 階段和路由設定自訂限流：

```
aws apigatewayv2 update-stage \
    --api-id a1b2c3d4 \
    --stage-name dev \
    --route-settings '{"messages":
{"ThrottlingBurstLimit":100,"ThrottlingRateLimit":2000}}'
```

# 在 API Gateway 中監控的 WebSocket API

您可以使用 CloudWatch 指標和 CloudWatch Logs 來監控 WebSocket API。藉由結合日誌和指標，您可以記錄錯誤並監控 API 的效能。

> **Note**
>
> 在下列情況下，API Gateway 可能無法產生日誌和指標：
>
> - 413 請求實體過大錯誤
> - 過多的 429 請求過多錯誤

- 400 系列錯誤，來自傳送至沒有 API 映射自訂網域的要求
- 由於內部失敗造成的 500 系列錯誤

主題


- 使用 CloudWatch 指標監控 WebSocket API 執行
- 在 API Gateway 中設定 WebSocket API 的記錄


## 使用 CloudWatch 指標監控 WebSocket API 執行

您可以使用 Amazon CloudWatch 指標來監控 WebSocket API。該組態與用於 REST API 的組態類似。如需更多詳細資訊，請參閱 使用 Amazon CloudWatch 指標監控 REST API 執行。

WebSocket API 支援以下指標：

| 指標 | 描述 |
| --- | --- |
| 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 | API Gateway 將請求傳送至整合與 API Gateway 從整合接收回應之間的時間差異。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 | 描述 |
| --- | --- |
| ApiId | 篩選具指定 API ID 之 API 的 API Gateway 指標。 |
| ApiId、階段 | 篩選具指定 API ID 與階段 ID 之 API 階段的 API Gateway 指標。 |
| ApiId、方法、資源、階段 | 篩選具指定 API ID、階段 ID、資源路徑和路由 ID 之 API 方法的 API Gateway 指標。<br><br>除非您已明確啟用詳細的 CloudWatch 指標，否則 API Gateway 不會傳送這類指標。您可以透過呼叫 API Gateway V2 REST API 的 UpdateStage 動作，將 `detailedMetricsEnabled` 屬性更新為 `true`。或者，您可以呼叫 update-stage AWS CLI 命令，將 `DetailedMetricsEnabled` 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 Amazon CloudWatch 定價。 |

## 在 API Gateway 中設定 WebSocket API 的記錄

您可以啟用記錄功能，將日誌寫入 CloudWatch Logs。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。

若要提升您的安全狀態，建議您在 ERROR 或 INFO 層級使用執行記錄。您可能需要這樣做，才能符合各種合規架構。如需詳細資訊，請參閱《AWS Security Hub 使用者指南》中的 Amazon API Gateway 控制項。

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，或選擇由 API Gateway 管理的現有日誌群組。若要指定存取權詳細資料，您可以選取 $context 變數 (以所選擇格式表示)，以及選擇日誌群組作為目標。

如需有關如何設定 CloudWatch 記錄功能的指示，請參閱the section called “使用 API Gateway 主控台設定存取記錄”。

當您指定 Log Format (日誌格式)，您可以選擇要記錄哪些環境變數。支援下列變數。

| 參數 | 描述 |
| --- | --- |
| $context.apiId | API Gateway 指派給您 API 的識別碼。 |
| $context.authorize.error | 授權錯誤訊息。 |
| $context.authorize.latency | 授權延遲 (以毫秒為單位)。 |
| $context.authorize.status | 從授權嘗試傳回的狀態碼。 |
| $context.authorizer.error | 從授權方傳回的錯誤訊息。 |
| $context.authorizer.integrationLatency | Lambda 授權方延遲 (以毫秒為單位)。 |
| $context.authorizer.integrationStatus | 從 Lambda 授權方傳回的狀態碼。 |
| $context.authorizer.latency | 授權方延遲 (以毫秒為單位)。 |
| $context.authorizer.requestId | AWS 端點的請求 ID。 |
| $context.authorizer.status | 從授權方傳回的狀態碼。 |
| $context.authorizer.principalId | 與用戶端所傳送並從 API Gateway Lambda 授權方 Lambda 函數所傳回之權杖建立關聯的主要使用者識別。(Lambda 授權方之前稱為自訂授權方。) |

| 參數 | 描述 |
| --- | --- |
| $context.authorizer. *property* | API Gateway Lambda 授權方函數所傳回 context 映射之指定索引鍵/值組的字串化值。例如，如果授權方傳回下列 context 映射：<br>`"context" : { "key": "value", "numKey": 1, "boolKey": true }`<br>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.authenticate.error | 從驗證嘗試傳回的錯誤訊息。 |
| $context.authenticate.latency | 驗證延遲 (以毫秒為單位)。 |
| $context.authenticate.status | 從驗證嘗試傳回的狀態碼。 |
| $context.connectedAt | Epoch 格式化連線時間。 |
| $context.connectionId | 連線的唯一 ID，可用來回呼用戶端。 |
| $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.error.message | 包含 API Gateway 錯誤訊息的字串。 |
| $context.error.messageString | $context.error.message 的引用值，即 "$context.error.message"。 |
| $context.error.responseType | 錯誤回應類型。 |

| 參數 | 描述 |
| --- | --- |
| $context.error.validationErrorString | 包含詳細驗證錯誤訊息的字串。 |
| $context.eventType | 事件類型：CONNECT、MESSAGE 或 DISCONNECT 。 |
| $context.extendedRequestId | 等同於 $context.requestId 。 |
| $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 | 已簽署請求之發起人的主體識別符。支援使用 IAM 授權的路由。 |
| $context.identity.cognitoAuthenticationProvider | 提出請求的發起人所使用的所有 Amazon Cognito 驗證提供者清單 (以逗號分隔)。僅在使用 Amazon Cognito 登入資料簽署請求時才可使用。<br><br>例如，適用於 Amazon Cognito 使用者集區的身分，cognito-idp.*region*.amazonaws.com/ *user_pool_id* ,cognito-idp. *region*.amazonaws.com/ *user_pool_id* :CognitoSignIn: *token subject claim*<br><br>如需有關 Amazon Cognito 驗證提供者的詳細資訊，請參閱《Amazon Cognito 開發人員指南》中的使用聯合身分。 |

| 參數 | 描述 |
| --- | --- |
| `$context.identity.cognitoAuthenticationType` | 提出請求的發起人的 Amazon Cognito 驗證類型。僅在使用 Amazon Cognito 登入資料簽署請求時才可使用。可能的值包括用於已驗證身分的 `authenticated` 和未經驗證身分的 `unauthenticated` 。 |
| `$context.identity.cognitoIdentityId` | 提出請求的發起人的 Amazon Cognito 身分 ID。僅在使用 Amazon Cognito 登入資料簽署請求時才可使用。 |
| `$context.identity.cognitoIdentityPoolId` | 提出請求的發起人的 Amazon Cognito 身分集區 ID。僅在使用 Amazon Cognito 登入資料簽署請求時才可使用。 |
| `$context.identity.principalOrgId` | AWS 組織 ID。支援使用 IAM 授權的路由。 |
| `$context.identity.sourceIp` | 對 API Gateway 提出請求之 TCP 連線的來源 IP 地址。 |
| `$context.identity.user` | 將針對資源存取授權之使用者的主體識別符。支援使用 IAM 授權的路由。 |
| `$context.identity.userAgent` | API 發起人的使用者代理程式。 |
| `$context.identity.userArn` | 身分驗證之後識別之有效使用者的 Amazon Resource Name (ARN)。 |
| `$context.integration.error` | 從整合傳回的錯誤訊息。 |
| `$context.integration.integrationStatus` | 對於 Lambda 代理整合，從 傳回的狀態碼 AWS Lambda，而不是從後端 Lambda 函數程式碼傳回的狀態碼。 |
| `$context.integration.latency` | 整合延遲 (以毫秒為單位)。等同於 `$context.integrationLatency` 。 |

| 參數 | 描述 |
| --- | --- |
| $context.integration.requestId | AWS 端點的請求 ID。等同於 $context.awsEndpointRequestId 。 |
| $context.integration.status | 從整合傳回的狀態碼。對於 Lambda 代理整合而言，這是您的 Lambda 函數程式碼傳回的狀態碼。等同於 $context.integrationStatus 。 |
| $context.integrationLatency | 毫秒的整合延遲僅可用於存取記錄。 |
| $context.messageId | 訊息的伺服器端唯一 ID。僅在 $context.eventType 為 MESSAGE 時可用。 |
| $context.requestId | 與 $context.extendedRequestId 相同。 |
| $context.requestTime | CLF 格式化請求時間 (dd/MMM/yyyy:HH:mm:ss +-hhmm )。 |
| $context.requestTimeEpoch | Epoch 格式化的要求時間，以毫秒為單位。 |
| $context.routeKey | 所選路由金鑰。 |
| $context.stage | API 呼叫的部署階段 (例如，Beta 或 Prod)。 |
| $context.status | 回應狀態。 |
| $context.waf.error | 從 傳回的錯誤訊息 AWS WAF。 |
| $context.waf.latency | 以毫秒為單位的 AWS WAF 延遲。 |
| $context.waf.status | 從 傳回的狀態碼 AWS WAF。 |

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.eventType $context.routeKey
 $context.connectionId" \
$context.status $context.requestId
```

延續字符 (\) 代表作為一種視覺輔助。記錄格式必須是單行。您可以在記錄格式的結尾新增新行字元 (\n)，以便在每個記錄項目的結尾加入新行。

- JSON:

```
{
"requestId":"$context.requestId", \
"ip": "$context.identity.sourceIp", \
"caller":"$context.identity.caller", \
"user":"$context.identity.user", \
"requestTime":"$context.requestTime", \
"eventType":"$context.eventType", \
"routeKey":"$context.routeKey", \
"status":"$context.status", \
"connectionId":"$context.connectionId"
}
```

延續字符 (\) 代表作為一種視覺輔助。記錄格式必須是單行。您可以在記錄格式的結尾新增新行字元 (\n)，以便在每個記錄項目的結尾加入新行。

- XML:

```
<request id="$context.requestId"> \
 <ip>$context.identity.sourceIp</ip> \
 <caller>$context.identity.caller</caller> \
 <user>$context.identity.user</user> \
 <requestTime>$context.requestTime</requestTime> \
 <eventType>$context.eventType</eventType> \
 <routeKey>$context.routeKey</routeKey> \
 <status>$context.status</status> \
 <connectionId>$context.connectionId</connectionId> \
</request>
```

延續字符 (\) 代表作為一種視覺輔助。記錄格式必須是單行。您可以在記錄格式的結尾新增新行字元 (\n)，以便在每個記錄項目的結尾加入新行。

- CSV (逗號分隔值)：

```
$context.identity.sourceIp,$context.identity.caller, \
```

```
$context.identity.user,$context.requestTime,$context.eventType, \
$context.routeKey,$context.connectionId,$context.status, \
$context.requestId
```

延續字符 (\) 代表作為一種視覺輔助。記錄格式必須是單行。您可以在記錄格式的結尾新增新行字元 (\n)，以便在每個記錄項目的結尾加入新行。

# API Gateway Amazon Resource Name (ARN) 參考資料

下表列出 API Gateway 資源的 Amazon Resource Name (ARN)。若要進一步了解如何在 AWS Identity and Access Management 政策中使用 ARNs，請參閱 Amazon API Gateway 與 IAM 搭配運作的方式和 使用 IAM 許可權控制 REST API 的存取。

## HTTP API 和 WebSocket API 資源

| 資源 | ARN |
| --- | --- |
| AccessLogSettings | arn:*partition* :apigateway: *region*::/apis/*api-id*/stages/*stage-name* /accesslogsettings |
| Api | arn:*partition* :apigateway: *region*::/apis/*api-id* |
| Apis | arn:*partition* :apigateway: *region*::/apis |
| 授權方 | arn:*partition* :apigateway: *region*::/apis/*api-id*/authorizers/ *id* |
| 授權方 | arn:*partition* :apigateway: *region*::/apis/*api-id*/authorizers |
| Cors | arn:*partition* :apigateway: *region*::/apis/*api-id*/cors |
| 部署 | arn:*partition* :apigateway: *region*::/apis/*api-id*/deployments/ *id* |

| 資源 | ARN |
| --- | --- |
| 部署 | arn:*partition* :apigateway: *region*::/apis/*api-id*/deployments |
| ExportedAPI | arn:*partition* :apigateway: *region*::/apis/*api-id*/exports/ *specification* |
| 整合 | arn:*partition* :apigateway: *region*::/apis/*api-id*/integrations/ *integration-id* |
| 整合 | arn:*partition* :apigateway: *region*::/apis/*api-id*/integrations |
| IntegrationResponse | arn:*partition* :apigateway: *region*::/apis/*api-id*/integrationresponses/ *integration-response* |
| IntegrationResponses | arn:*partition* :apigateway: *region*::/apis/*api-id*/integrationresponses |
| 模型 | arn:*partition* :apigateway: *region*::/apis/*api-id*/models/*id* |
| 模型 | arn:*partition* :apigateway: *region*::/apis/*api-id*/models |
| ModelTemplate | arn:*partition* :apigateway: *region*::/apis/*api-id*/models/*id*/template |
| 路由 | arn:*partition* :apigateway: *region*::/apis/*api-id*/routes/*id* |

| 資源 | ARN |
| --- | --- |
| 路由 | arn:*partition* :apigatew ay: *region*::/apis/*api-id*/routes |
| RouteRequestParameter | arn:*partition* :apigatew ay: *region*::/apis/*api-id*/routes/*id*/ requestparameters/ *key* |
| RouteResponse | arn:*partition* :apigatew ay: *region*::/apis/*api-id*/routes/*id*/ routeresponses/ *id* |
| RouteResponses | arn:*partition* :apigatew ay: *region*::/apis/*api-id*/routes/*id*/ routeresponses |
| RouteSettings | arn:*partition* :apigatew ay: *region*::/apis/*api-id*/ stages/*stage-name* /routeset tings/ *route-key* |
| 階段 | arn:*partition* :apigatew ay: *region*::/apis/*api-id*/ stages/*stage-name* |
| 階段 | arn:*partition* :apigatew ay: *region*::/apis/*api-id*/stages |
| VpcLink | arn:*partition* :apigatew ay: *region*::/vpclinks/ *vpclink-id* |
| VpcLinks | arn:*partition* :apigatew ay: *region*::/vpclinks |

# REST API 資源

| 資源 | ARN |
| --- | --- |
| 帳戶 | arn:*partition* :apigatew ay: *region*::/account |
| ApiKey | arn:*partition* :apigatew ay: *region*::/apikeys/ *id* |
| ApiKeys | arn:*partition* :apigatew ay: *region*::/apikeys |
| 授權方 | arn:*partition* :apigatew ay: *region*::/restapis/ *api-id*/ authorizers/ *id* |
| 授權方 | arn:*partition* :apigatew ay: *region*::/restapis/ *api-id*/ authorizers |
| ClientCertificate | arn:*partition* :apigatew ay: *region*::/clientcertifica tes/ *id* |
| ClientCertificates | arn:*partition* :apigatew ay: *region*::/clientcertificates |
| 部署 | arn:*partition* :apigatew ay: *region*::/restapis/ *api-id*/ deployments/ *id* |
| 部署 | arn:*partition* :apigatew ay: *region*::/restapis/ *api-id*/ deployments |

| 資源 | ARN |
| --- | --- |
| DocumentationPart | arn:*partition* :apigatew ay: *region*::/restapis/ *api-id*/ documentation/parts/ *id* |
| DocumentationParts | arn:*partition* :apigatew ay: *region*::/restapis/ *api-id*/ documentation/parts |
| DocumentationVersion | arn:*partition* :apigatew ay: *region*::/restapis/ *api-id*/ documentation/versions/ *version* |
| DocumentationVersions | arn:*partition* :apigatew ay: *region*::/restapis/ *api-id*/ documentation/versions |
| GatewayResponse | arn:*partition* :apigatew ay: *region*::/restapis/ *api-id*/ gatewayresponses/ *response-type* |
| GatewayResponses | arn:*partition* :apigatew ay: *region*::/restapis/ *api-id*/ gatewayresponses |
| 整合 | arn:*partition* :apigatew ay: *region*::/restapis/ *api-id*/ resources/ *resource-id* /methods/ *http-method* /integration |
| IntegrationResponse | arn:*partition* :apigatew ay: *region*::/restapis/ *api-id*/ resources/ *resource-id* /methods/ *http-method* /integration/respo nses/ *status-code* |

| 資源 | ARN |
| --- | --- |
| Method | arn:*partition* :apigatew ay: *region*::/restapis/ *api-id*/ resources/ *resource-id* /methods/ *http-method* |
| MethodResponse | arn:*partition* :apigatew ay: *region*::/restapis/ *api-id*/ resources/ *resource-id* /methods/ *http-method* /responses/ *status-co de* |
| 模型 | arn:*partition* :apigatew ay: *region*::/restapis/ *api-id*/ models/*model-name* |
| 模型 | arn:*partition* :apigatew ay: *region*::/restapis/ *api-id*/ models |
| RequestValidator | arn:*partition* :apigatew ay: *region*::/restapis/ *api-id*/ requestvalidators/ *id* |
| RequestValidators | arn:*partition* :apigatew ay: *region*::/restapis/ *api-id*/ requestvalidators |
| 資源 | arn:*partition* :apigatew ay: *region*::/restapis/ *api-id*/ resources/ *id* |
| Resources | arn:*partition* :apigatew ay: *region*::/restapis/ *api-id*/ resources |

| 資源 | ARN |
| --- | --- |
| RestApi | arn:*partition* :apigateway: *region*::/restapis/ *api-id* |
| RestApis | arn:*partition* :apigateway: *region*::/restapis |
| 階段 | arn:*partition* :apigateway: *region*::/restapis/ *api-id*/stages/*stage-name* |
| 階段 | arn:*partition* :apigateway: *region*::/restapis/ *api-id*/stages |
| Tags (標籤) | arn:*partition* :apigateway: *region*::/tags/*url-encoded-resource-arn* |
| 範本 | arn:*partition* :apigateway: *region*::/restapis/models/ *model-name* /template |
| UsagePlan | arn:*partition* :apigateway: *region*::/usageplans/ *usageplan-id* |
| UsagePlans | arn:*partition* :apigateway: *region*::/usageplans |
| UsagePlanKey | arn:*partition* :apigateway: *region*::/usageplans/ *usageplan-id* /keys/*id* |
| UsagePlanKeys | arn:*partition* :apigateway: *region*::/usageplans/ *usageplan-id* /keys |

| 資源 | ARN |
| --- | --- |
| VpcLink | arn:*partition* :apigatew ay: *region*::/vpclinks/ *vpclink-id* |
| VpcLinks | arn:*partition* :apigatew ay: *region*::/vpclinks |

## 網域名稱資源

| 資源 | ARN |
| --- | --- |
| ApiMapping | arn:*partition* :apigatew ay: *region*::/domainnames/ *domain-na me* /apimappings/ *id* |
| ApiMappings | arn:*partition* :apigatew ay: *region*::/domainnames/ *domain-na me* /apimappings |
| BasePathMapping | arn:*partition* :apigatew ay: *region*::/domainnames/ *domain-na me* /basepathmappings/ *basepath* |
| BasePathMappings | arn:*partition* :apigatew ay: *region*::/domainnames/ *domain-na me* /basepathmappings |
| DomainName | arn:*partition* :apigatew ay: *region*::/domainnames/ *domain-na me* |
| DomainNameAccessAssociation | arn:*partition* :apigatew ay: *region*:*account-id* :/domainn ameaccessassociations/domai nname/ *domain-name* /vpcesour ce/ *vpce-source-id* |

| 資源 | ARN |
| --- | --- |
| DomainNameAccessAssociations | arn:*partition* :apigatew ay: *region*:*account-id* :/domainn ameaccessassociations |
| DomainNames | arn:*partition* :apigatew ay: *region*::/domainnames |
| RoutingRule | arn:*partition* :apigatew ay: *region*:*account-id* :/domainn ames/ *domain-name* /routingr ules/ *routing-rule-id* |

## **execute-api** (HTTP API、WebSocket API 和 REST API)

| 資源 | ARN |
| --- | --- |
| WebSocket API 端點 | arn:*partition* :execute-api: *region*:*account-id* :*api-id*/*stage*/*route-key* |
| HTTP API 和 REST API 端點 * | arn:*partition* :execute-api: *region*:*account-id* :*api-id*/*stage*/*http-method* /*resource-path* |
| Lambda 授權方 ** | arn:*partition* :execute-api: *region*:*account-id* :*api-id*/ authorizers/ *authorizer-id* |

* HTTP API 的 $default 路由端點的 ARN 為 arn:*partition*:execute-api:*region*:*account-id*:*api-id*/*/$default。

** 此 ARN 僅適用於在資源政策中設定 Lambda 授權方功能的 SourceArn 條件。如需範例，請參閱the section called “建立 Lambda 授權方”。

# API Gateway 的 OpenAPI 擴充功能

API Gateway 延伸支援 REST API 和 HTTP API AWS的特定授權和 APIs Gateway 特定的 APIs整合。在本節中，我們會說明 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 REST API 或 HTTP API，再將它匯出到 OpenAPI 定義檔。如需如何匯出 API 的詳細資訊，請參閱從 API Gateway 匯出 REST API 和從 APIs Gateway 匯出 HTTP API。

主題


- x-amazon-apigateway-any-method 物件
- x-amazon-apigateway-cors 物件
- x-amazon-apigateway-api-key-source 屬性
- x-amazon-apigateway-auth 物件
- x-amazon-apigateway-authorizer 物件
- x-amazon-apigateway-authtype 屬性
- x-amazon-apigateway-binary-media-types 屬性
- x-amazon-apigateway-documentation 物件
- x-amazon-apigateway-endpoint-access-mode
- x-amazon-apigateway-endpoint-configuration 物件
- x-amazon-apigateway-gateway-responses 物件
- x-amazon-apigateway-gateway-responses.gatewayResponse 物件
- x-amazon-apigateway-gateway-responses.responseParameters 物件
- x-amazon-apigateway-gateway-responses.responseTemplates 物件
- x-amazon-apigateway-importexport-version
- x-amazon-apigateway-integration 物件
- x-amazon-apigateway-integrations 物件
- x-amazon-apigateway-integration.requestTemplates 物件

- x-amazon-apigateway-integration.requestParameters 物件
- x-amazon-apigateway-integration.responses 物件
- x-amazon-apigateway-integration.response 物件
- x-amazon-apigateway-integration.responseTemplates 物件
- x-amazon-apigateway-integration.responseParameters 物件
- x-amazon-apigateway-integration.tlsConfig object
- x-amazon-apigateway-minimum-compression-size
- x-amazon-apigateway-policy
- x-amazon-apigateway-request-validator 屬性
- x-amazon-apigateway-request-validators 物件
- x-amazon-apigateway-request-validators.requestValidator 物件
- x-amazon-apigateway-security-policy
- x-amazon-apigateway-tag-value 屬性

## 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。

| 屬性名稱 | 類型 | 描述 |
| --- | --- | --- |
| `isDefaultRoute` | `Boolean` | 指定路由是否為 `$default` 路由。僅支援 HTTP API。如需進一步了解，請參閱在 API Gateway 中建立 HTTP API 的路由。 |
| `x-amazon-apigateway-integration` | x-amazon-apigateway-integration 物件 | 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` |

| 屬性名稱 | 類型 | 描述 |
| --- | --- | --- |
| | | 、HTTP、HTTP_PROXY 或 MOCK。 |

## x-amazon-apigateway-any-method 範例

下列範例會整合代理資源 ANY 之 {proxy+} 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",
          "httpMethod": "POST",
          "type": "aws_proxy"
        }
```

下列範例會為 HTTP API 建立與 Lambda 函數 $default 整合的 HelloWorld 路由。

```
"/$default": {
    "x-amazon-apigateway-any-method": {
      "isDefaultRoute": true,
      "x-amazon-apigateway-integration": {
        "type": "AWS_PROXY",
        "httpMethod": "POST",
        "uri": "arn:aws:apigateway:us-east-1:lambda:path/2015-03-31/functions/
arn:aws:lambda:us-east-1:123456789012:function:HelloWorld/invocations",
        "timeoutInMillis": 1000,
```

```
        "connectionType": "INTERNET",
        "payloadFormatVersion": 1.0
      }
   }
}
```

## x-amazon-apigateway-cors 物件

指定 HTTP API 的跨來源資源共用 (CORS) 組態。該擴展適用於根層級 OpenAPI 結構。如需進一步了解，請參閱在 API Gateway 中設定 HTTP API 的 CORS。

| 屬性名稱 | 類型 | 描述 |
| --- | --- | --- |
| allowOrigins | Array | 指定允許的原始伺服器。 |
| allowCredentials | Boolean | 指定 CORS 請求中是否包含登入資料。 |
| exposeHeaders | Array | 指定公開的標頭。 |
| maxAge | Integer | 指定瀏覽器應快取預檢請求結果的秒數。 |
| allowMethods | Array | 指定允許的 HTTP 方法。 |
| allowHeaders | Array | 指定允許的標頭。 |

## x-amazon-apigateway-cors 範例

以下是 HTTP API 的 CORS 組態範例。

```
"x-amazon-apigateway-cors": {
    "allowOrigins": [
      "https://www.example.com"
    ],
    "allowCredentials": true,
    "exposeHeaders": [
      "x-apigateway-header",
      "x-amz-date",
```

```
      "content-type"
    ],
    "maxAge": 3600,
    "allowMethods": [
      "GET",
      "OPTIONS",
      "POST"
    ],
    "allowHeaders": [
      "x-apigateway-header",
      "x-amz-date",
      "content-type"
    ]
}
```

## 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。如需有關設定方法以要求 API 金鑰的詳細資訊，請參閱 the section called “設定使用具有 OpenAPI 定義的 API 金鑰方法”。

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",
   .
   .
   .
}
```

OpenAPI 3.0.1

```
{
  "openapi" : "3.0.1",
  "info" : {
    "title" : "Test1"
  },
  "servers" : [ {
    "url" : "/{basePath}",
    "variables" : {
      "basePath" : {
        "default" : "import"
      }
    }
  } ],
  "x-amazon-apigateway-api-key-source" : "HEADER",
   .
   .
   .
}
```

# x-amazon-apigateway-auth 物件

在 API Gateway 中定義方法叫用授權要套用的授權類型。

| 屬性名稱 | 類型 | 描述 |
| --- | --- | --- |
| type | string | 指定授權類型。指定 "NONE" 以開放存取；指定 "AWS_IAM" 以使用 IAM 許可。值不區分大小寫。 |

## x-amazon-apigateway-auth 範例

下列範例會設定 REST API 方法的授權類型。

OpenAPI 3.0.1

```
{
  "openapi": "3.0.1",
  "info": {
    "title": "openapi3",
    "version": "1.0"
  },
  "paths": {
    "/protected-by-iam": {
      "get": {
        "x-amazon-apigateway-auth": {
          "type": "AWS_IAM"
        }
      }
    }
  }
}
```

## x-amazon-apigateway-authorizer 物件

定義要套用的 Lambda 授權方、Amazon Cognito 使用者集區或 JWT 授權方，以授權在 API Gateway 中的方法調用。此延伸適用於 OpenAPI 2 中的安全性定義和 OpenAPI 3 中的安全性結構描述。

| 屬性名稱 | 類型 | 描述 |
| --- | --- | --- |
| `type` | `string` | 授權方的類型。這是必要屬性。<br><br>對於 REST API，指定 `token` 為授權方，並在授權權杖中嵌入發起人身分。為發起人身分包含在請求參數的授權方指定 `request`。針對使用 Amazon Cognito 使用者集區 |

| 屬性名稱 | 類型 | 描述 |
| --- | --- | --- |
| | | 進行 API 存取控制的授權方指定 `cognito_user_pools`。<br><br>對於 HTTP API，指定 `request` 為 Lambda 授權方，並在授權參數中包含發起人身分。為 JWT 授權方指定 `jwt`。 |
| `authorizerUri` | `string` | 授權方 Lambda 函數的統一資源識別符 (URI)。語法如下：<br><br>`"arn:aws:apigateway:us-east-1:lambda:path/2015-03-31/functions/arn:aws:lambda:us-east-1:`*account-id*`:function:`*auth_function_name*`/invocations"` |
| `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::*account-id*`:`*IAM_role*"。 |
| `authorizerPayloadFormatVersion` | `string` | 對於 HTTP API，指定 API Gateway 傳送到 Lambda 授權方的資料格式，以及 API Gateway 如何解釋 Lambda 的回應。如需進一步了解，請參閱the section called “裝載格式版本”。 |

| 屬性名稱 | 類型 | 描述 |
| --- | --- | --- |
| enableSimpleResponses | Boolean | 對於 HTTP API，指定 request 授權方是否傳回布林值或 IAM 政策。僅支援具有 2.0 的 authorizerPayloadFormatVersion 的授權者。如果啟用，Lambda 授權方函數會傳回一個布林值。如需進一步了解，請參閱the section called “格式 2.0 的 Lambda 函數回應”。 |
| 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 和 jwt 類型的授權方。 |
| jwtConfiguration | Object | 指定 JWT 授權方的發行者和對象。若要進一步了解，請參閱《API Gateway 第 2 版 API 參考》中的 JWTConfiguration。僅支援 HTTP API。 |
| identityValidationExpression | string | 驗證做為傳入身分之字符的一般表達式。例如，"^x-[a-z]+"。僅支援 REST API 的 TOKEN 授權方。 |
| authorizerResultTtlInSeconds | string | 快取授權方結果的秒數。 |
| providerARNs | string 陣列 | COGNITO_USER_POOLS 的 Amazon Cognito 使用者集區 ARN 清單。 |

# REST API 的 x-amazon-apigateway-authorizer 範例

下列 OpenAPI 安全定義範例會指定類型為「權杖」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" : "custom", // 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/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 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/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 安全定義範例會將 Amazon Cognito 使用者集區指定為授權方。

```
"securityDefinitions": {
   "cognito-pool": {
     "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:us-east-1:123456789012:userpool/us-east-1_ABC123"
       ]
     }
   }
```

下列 OpenAPI 操作物件程式碼片段會將 `GET /http` 設定為使用之前的 Amazon Cognito 使用者集區做為授權方，且沒有自訂範圍。

```
  "/http" : {
     "get" : {
       "responses" : { },
       "security" : [ {
         "cognito-pool" : [ ]
       } ],
       "x-amazon-apigateway-integration" : {
         "type" : "http",
         "responses" : {
           "default" : {
             "statusCode" : "200"
           }
         },
         "httpMethod" : "GET",
         "uri" : "http://api.example.com"
       }
     }
   }
```

## HTTP API 的 x-amazon-apigateway-authorizer 範例

以下 OpenAPI 3.0 範例為 HTTP API 建立 JWT 授權方，它會使用 Amazon Cognito 做為身分提供者，並以 `Authorization` 標頭做為身分來源。

```
"securitySchemes": {
  "jwt-authorizer-oauth": {
    "type": "oauth2",
     "x-amazon-apigateway-authorizer": {
       "type": "jwt",
       "jwtConfiguration": {
          "issuer": "https://cognito-idp.region.amazonaws.com/userPoolId",
          "audience": [
            "audience1",
            "audience2"
          ]
        },
        "identitySource": "$request.header.Authorization"
    }
  }
}
```

以下 OpenAPI 3.0 範例會產生與前例相同的 JWT 授權方。不過，此範例使用 OpenAPI 的 `openIdConnectUrl` 屬性來自動偵測發行者。`openIdConnectUrl` 必須具備完整格式。

```
"securitySchemes": {
  "jwt-authorizer-autofind": {
    "type": "openIdConnect",
    "openIdConnectUrl": "https://cognito-idp.region.amazonaws.com/userPoolId/.well-
known/openid-configuration",
    "x-amazon-apigateway-authorizer": {
      "type": "jwt",
      "jwtConfiguration": {
        "audience": [
          "audience1",
          "audience2"
        ]
      },
      "identitySource": "$request.header.Authorization"
    }
  }
}
```

下列範例會建立 HTTP API 的 Lambda 授權方。此範例授權方使用 `Authorization` 標頭作為其身分來源。授權方會使用 `2.0` 承載格式版本，並傳回布林值，因為系統將 `enableSimpleResponses` 設定為 `true`。

```
"securitySchemes" : {
  "lambda-authorizer" : {
    "type" : "apiKey",
    "name" : "Authorization",
    "in" : "header",
    "x-amazon-apigateway-authorizer" : {
      "type" : "request",
      "identitySource" : "$request.header.Authorization",
      "authorizerUri" : "arn:aws:apigateway:us-west-2:lambda:path/2015-03-31/functions/
arn:aws:lambda:us-west-2:123456789012:function:function-name/invocations",
      "authorizerPayloadFormatVersion" : "2.0",
      "authorizerResultTtlInSeconds" : 300,
      "enableSimpleResponses" : true
    }
  }
}
```

# x-amazon-apigateway-authtype 屬性

對於 REST API，此延伸可用於定義 Lambda 授權方的自訂類型。在這種情況下，值為任意格式。例如，一個 API 可能有多個 Lambda 授權方使用不同的內部結構描述。您可以使用此延伸來識別 Lambda 授權方的內部結構描述。

更常見的情況是，在 HTTP API 和 REST API 中，也可用於定義共享相同安全性結構描述的多個作業之間的 IAM 授權。在這種情況下，術語 `awsSigv4` 是保留字詞，以及任何前綴為 `aws` 的任何術語。

此延伸適用於 OpenAPI 2 和 OpenAPI 3 中的 `apiKey` 類型安全性結構描述。

## x-amazon-apigateway-authtype 範例

下列 OpenAPI 3 範例定義了 REST API 或 HTTP API 中多個資源之間的 IAM 授權：

```
{
  "openapi" : "3.0.1",
  "info" : {
    "title" : "openapi3",
    "version" : "1.0"
  },
  "paths" : {
    "/operation1" : {
      "get" : {
```

```
        "responses" : {
          "default" : {
            "description" : "Default response"
          }
        },
        "security" : [ {
          "sigv4Reference" : [ ]
        } ]
      }
    },
    "/operation2" : {
      "get" : {
        "responses" : {
          "default" : {
            "description" : "Default response"
          }
        },
        "security" : [ {
          "sigv4Reference" : [ ]
        } ]
      }
    }
  },
  "components" : {
    "securitySchemes" : {
      "sigv4Reference" : {
        "type" : "apiKey",
        "name" : "Authorization",
        "in" : "header",
        "x-amazon-apigateway-authtype": "awsSigv4"
      }
    }
  }
}
```

下列 OpenAPI 3 範例會使用 REST API 的自訂結構描述來定義 Lambda 授權方：

```
{
  "openapi" : "3.0.1",
  "info" : {
    "title" : "openapi3 for REST API",
    "version" : "1.0"
  },
```

```
  "paths" : {
    "/protected-by-lambda-authorizer" : {
      "get" : {
        "responses" : {
          "200" : {
            "description" : "Default response"
          }
        },
        "security" : [ {
          "myAuthorizer" : [ ]
        } ]
      }
    }
  },
  "components" : {
    "securitySchemes" : {
      "myAuthorizer" : {
        "type" : "apiKey",
        "name" : "Authorization",
        "in" : "header",
        "x-amazon-apigateway-authorizer" : {
          "identitySource" : "method.request.header.Authorization",
          "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",
          "authorizerResultTtlInSeconds" : 300,
          "type" : "request",
          "enableSimpleResponses" : false
        },
        "x-amazon-apigateway-authtype": "Custom scheme with corporate claims"
      }
    }
  },
  "x-amazon-apigateway-importexport-version" : "1.0"
}
```

## 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` 和 `image/jpeg`。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-endpoint-access-mode

指定 REST API 或自訂網域名稱的端點存取模式。若要進一步了解端點存取模式，請參閱 端點存取模式。

### x-amazon-apigateway-endpoint-access-mode 範例

下列範例會將端點存取模式指定為嚴格。

```
"x-amazon-apigateway-endpoint-access-mode": "STRICT"
```

## x-amazon-apigateway-endpoint-configuration 物件

指定 API 的端點組態詳細資訊。此延伸是 OpenAPI 操作物件的延伸屬性。這個物件應該位於 Swagger 2.0 的上層廠商延伸中。對於 OpenAPI 3.0，它應該位於伺服器物件的廠商延伸下方。

| 屬性名稱 | 類型 | 描述 |
| --- | --- | --- |
| disableExecuteApiEndpoint | 布林值 | 指定用戶端是否可以使用預設 execute-api 端點叫用您的 API。根據預設，用戶端可以使用預設 https://{api_id}.execute-api.{region}.amazonaws.com 端點叫用您的 API。 |

| 屬性名稱 | 類型 | 描述 |
| --- | --- | --- |
| | | 如要要求用戶端使用自訂網域名稱來叫用 API，請指定 `true`。 |
| `vpcEndpointIds` | `String` 陣列 | 要為其建立 REST API 之 Route 53 別名記錄的 VpcEndpoint 識別符清單。僅支援 `PRIVATE` 端點類型的 REST API。 |
| `ipAddressType` | `string` | 可調用 HTTP API 的 IP 位址類型。使用 `ipv4` 可允許 IPv4 位址類型調用 HTTP API。使用 `dualstack` 可允許 IPv4 和 IPv6 位址類型調用 HTTP API。僅 HTTP API 可支援。 |

## x-amazon-apigateway-endpoint-configuration 範例

以下範例會將指定的 VPC 端點與 REST API 建立關聯。

```
"x-amazon-apigateway-endpoint-configuration": {
    "vpcEndpointIds": ["vpce-0212a4ababd5b8c3e", "vpce-01d622316a7df47f9"]
}
```

下列範例會停用 API 的預設端點。

```
"x-amazon-apigateway-endpoint-configuration": {
    "disableExecuteApiEndpoint": true
}
```

下列範例會將 HTTP API 的 IP 位址類型設定為雙堆疊。

```
"x-amazon-apigateway-endpoint-configuration": {
    "ipAddressType": "dualstack"
}
```

# 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 的對應。該擴展適用於根層級 OpenAPI 結構。

| 屬性名稱 | 類型 | 描述 |
| --- | --- | --- |
| *responseType* | x-amazon-apigateway-gateway-responses.gatewayResponse | 指定之 *responseType* 的 GatewayResponse 。 |

## 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 對應，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。

| 屬性名稱 | 類型 | 描述 |
| --- | --- | --- |
| *responseParameters* | x-amazon-apigateway-gateway-responses.responseParameters | 指定 GatewayResponse 參數，即標頭參數。該參數值可使用任何傳入請求參數值或靜態自訂值。 |
| *responseTemplates* | x-amazon-apigateway-gateway-responses.responseTemplates | 指定閘道回應的對應範本。該範本不由 VTL 引擎處理。 |
| *statusCode* | `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。僅支援 REST API。

| 屬性名稱 | 類型 | 描述 |
| --- | --- | --- |
| gatewayresponse. *param-position* .*param-name* | string | *param-position* 可以是 header、path 或 querystring 。如需詳細資訊，請參閱API Gateway 中 REST API 的參數映射。 |

## x-amazon-apigateway-gateway-responses.responseParameters 範例

下列 OpenAPI 延伸範例示範 GatewayResponse 回應參數對應運算式，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 對應範本定義為鍵值對之字串對字串的對應。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。

| 屬性名稱 | 類型 | 描述 |
| --- | --- | --- |
| *content-type* | 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 對應範本。

```
      "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 對應範本。

```
      "responseTemplates": {
        "application/json": "{ \"message\": 'API-specific errors' }"
      }
```

## x-amazon-apigateway-importexport-version

指定 HTTP API 的 API Gateway 匯入和匯出演算法版本。目前，僅支援的值為 `1.0`。若要深入了解，請參閱《API Gateway 第 2 版 API 參考》中的 exportVersion。

### x-amazon-apigateway-importexport-version example

下列範例會將匯入和匯出版本設定為 `1.0`。

```
{
    "openapi": "3.0.1",
    "x-amazon-apigateway-importexport-version": "1.0",
    "info": { ...
```

## 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。結果是 API Gateway 整合物件。

| 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 VpcLink ID。 |
| connectionType | string | 整合連線類型。有效值是私有整合的 "VPC_LINK" 或 "INTERNET" 。 |
| credentials | string | 對於 AWS IAM 角色型登入資料，請指定適當 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，請參閱使用資源政策授予許可。<br><br>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 AWS STS 區域端點。 |
| 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。 |

| 屬性名稱 | 類型 | 描述 |
| --- | --- | --- |
| 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。 |
| integrationSubtype | string | 指定 AWS 服務整合的整合子類型。僅支援 HTTP API。如需支援的整合子類型，請參閱 the section called “AWS 服務整合參考”。 |
| integrationTarget | string | 要傳送請求的 ALB 或 NLB 接聽程式。僅支援使用 VPC 連結 V2 的私有整合。如需詳細資訊，請參閱the section called “VPC 連結 V2”。 |
| 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 Integration.passthroughBehavior。 |
| payloadFormatVersion | string | 為傳送至整合的承載指定格式。HTTP API 需要此項目。對於 HTTP API，Lambda 代理整合支援的值為 1.0 和 2.0。對於所有其他整合，1.0 是唯一支援的值。如需進一步了解，請參閱the section called “AWS Lambda 整合”和the section called “AWS 服務整合參考”。 |

| 屬性名稱 | 類型 | 描述 |
| --- | --- | --- |
| requestParameters | x-amazon-apigateway-integration.requestParameters 物件 | 對於 REST API，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 querystring 、path、header 和 body。<br><br>對於 HTTP API，請求參數是一個金鑰值映射，指定傳遞給帶有特定 integrationSubtype 的 AWS_PROXY 整合。您可以提供靜態值，或映射請求資料、階段變數或在執行階段評估的內容變數。如需進一步了解，請參閱the section called “AWS 服務整合”。 |
| requestTemplates | x-amazon-apigateway-integration.requestTemplates 物件 | 指定 MIME 類型的請求承載對應範本。 |
| responses | 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所需的參數對應或承載對應。 |
| responseTransferMode | string | 整合的回應傳輸模式。使用 BUFFERED 讓 API Gateway 在開始傳輸之前等待接收完整回應。使用 STREAM 讓 API Gateway 在部分回應可用時傳回用戶端。如需詳細資訊，請參閱 the section called “串流代理整合的整合回應”。 |
| timeoutInMillis | integer | 整合逾時，介於 50 毫秒到 29,000 毫秒之間。 |

| 屬性名稱 | 類型 | 描述 |
| --- | --- | --- |
| type | string | 與指定後端整合的類型。有效值為：<br>• http 或 http_proxy ：用於與 HTTP 後端整合。<br>• aws_proxy ，用於與 AWS Lambda 函數整合。<br>• aws，用於與 AWS Lambda 函數或其他 AWS 服務的整合，例如 Amazon DynamoDB、Amazon Simple Notification Service 或 Amazon Simple Queue Service。<br>• mock：用於與 API Gateway 整合，但不呼叫任何後端。<br><br>如需整合類型的詳細資訊，請參閱 integration:type。 |
| tlsConfig | the section called “x-amazon-apigateway-integration.tlsConfig” | 指定整合的 TLS 組態。 |
| uri | 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

對於 HTTP API，您可以在 OpenAPI 定義的元件部分中定義整合。如需進一步了解，請參閱 x-amazon-apigateway-integrations 物件。

```
"x-amazon-apigateway-integration": {
    "$ref": "#/components/x-amazon-apigateway-integrations/integration1"
}
```

下列範例會建立與 Lambda 函數的整合。基於示範目的，以下範例之 requestTemplates 和 responseTemplates 中示範的映射範本範例，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-integrations 物件

定義整合的集合。您可以在 OpenAPI 定義的元件部分中定義整合，然後將整合用於多個路由。僅支援 HTTP API。

| 屬性名稱 | 類型 | 描述 |
| --- | --- | --- |
| *##* | x-amazon-apigateway-integration 物件 | 整合物件的集合。 |

## x-amazon-apigateway-integrations 範例

下列範例會建立 HTTP API，其惠定義兩個整合，並使用來參照整合 `$ref": "#/components/x-amazon-apigateway-integrations/`*integration-name*。

```
{
  "openapi": "3.0.1",
  "info":
    {
```

```
    "title": "Integrations",
    "description": "An API that reuses integrations",
    "version": "1.0"
  },
"servers": [
{
  "url": "https://example.com/{basePath}",
  "description": "The production API server",
  "variables":
    {
      "basePath":
        {
          "default": "example/path"
        }
    }
}],
"paths":
  {
    "/":
      {
        "get":
          {
            "x-amazon-apigateway-integration":
              {
                "$ref": "#/components/x-amazon-apigateway-integrations/integration1"

              }
          }
      },
     "/pets":
      {
        "get":
          {
            "x-amazon-apigateway-integration":
              {
                "$ref": "#/components/x-amazon-apigateway-integrations/integration1"

              }
          }
      },
     "/checkout":
      {
        "get":
          {
```

```
              "x-amazon-apigateway-integration":
                {
                  "$ref": "#/components/x-amazon-apigateway-integrations/integration2"
                }
            }
        }
    },
    "components": {
      "x-amazon-apigateway-integrations":
        {
          "integration1":
            {
              "type": "aws_proxy",
              "httpMethod": "POST",
              "uri": "arn:aws:apigateway:us-east-2:lambda:path/2015-03-31/functions/
arn:aws:lambda:us-east-2:123456789012:function:my-function/invocations",
              "passthroughBehavior": "when_no_templates",
              "payloadFormatVersion": "1.0"
            },
          "integration2":
            {
              "type": "aws_proxy",
              "httpMethod": "POST",
              "uri": "arn:aws:apigateway:us-east-2:lambda:path/2015-03-31/functions/
arn:aws:lambda:us-east-2:123456789012:function:example-function/invocations",
              "passthroughBehavior": "when_no_templates",
              "payloadFormatVersion" : "1.0"
            }
        }
    }
}
```

# x-amazon-apigateway-integration.requestTemplates 物件

為指定 MIME 類型的請求承載指定對應範本。

| 屬性名稱 | 類型 | 描述 |
| --- | --- | --- |
| *MIME type* | string | MIME 類型範例為 application/json 。如需建立映射 |

| 屬性名稱 | 類型 | 描述 |
| --- | --- | --- |
| | | 範本的相關資訊，請參閱the section called “映射範本轉換”。 |

## 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

對於 REST API，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。

對於 HTTP API，指定透過指定 AWS_PROXY 傳遞給 integrationSubtype 整合的參數。

| 屬性名稱 | 類型 | 描述 |
| --- | --- | --- |
| integration.request. *<param-type>* .*<param-name>* | string | 對於 REST API，該值通常是格式為 method.request. *<param-type>* .*<param-name>* 的預先定義方法請求參數，其中 <param-type> 可以是 querystring 、path、header 或 body。不過，$context. *VARIABLE_NAME* 、$stageVar |

| 屬性名稱 | 類型 | 描述 |
| --- | --- | --- |
| | | iables. *VARIABLE_NAME* 和 *STATIC_VALUE* 也是有效的值。至於 body 參數，<param-name> 是開頭沒有 $. 的 JSON 路徑表達式。 |
| *parameter* | string | 對於 HTTP API，請求參數是一個金鑰值映射，指定傳遞給帶有特定 integrationSubtype 的 AWS_PROXY 整合。您可以提供靜態值，或映射請求資料、階段變數或在執行階段評估的內容變數。如需進一步了解，請參閱the section called “AWS 服務整合”。 |

## 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 建立資源 CloudFormation，則靜態值應以單引號括住。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。

| 屬性名稱 | 類型 | 描述 |
| --- | --- | --- |
| `######` | x-amazon-apigateway-integration.response 物件 | 用於將整合回應與方法回應匹配的規則運算式，或者是用於捕獲您尚未配置的任何回應的 `default`。若是 HTTP 整合，此 Regex 會套用至整合回應狀態碼。對於 Lambda 調用，當 Lambda 函數執行擲回例外狀況時，regex 會套用至 傳回 AWS Lambda 做為失敗回應內文的錯誤資訊物件`errorMessage` 欄位。<br><br>**Note**<br>`######`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。它不會對應到 API Gateway REST API 中 IntegrationResponse 資源的任何識別符。 |

## `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 | 指定回應承載的 MIME 類型專屬對應範本。 |
| `responseParameters` | x-amazon-apigateway-integration.responseParameters 物件 | 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。 |

| 屬性名稱 | 類型 | 描述 |
| --- | --- | --- |
| 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。

| 屬性名稱 | 類型 | 描述 |
| --- | --- | --- |
| *MIME type* | 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。如需建立映射範本的相關資訊，請參閱the section called “映射範本轉換”。*MIME ##*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。您可以將 header、body 或靜態值映射到 header 類型的方法回應。僅支援 REST API。

| 屬性名稱 | 類型 | 說明 |
| --- | --- | --- |
| method.response.header. *<param-name>* | string | 具名參數值可從 header 和 body 類型的整合回應參數產生。 |

## 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-integration.tlsConfig object

指定整合的 TLS 組態。

| 屬性名稱 | 類型 | 描述 |
| --- | --- | --- |
| insecureSkipVerification | Boolean | 僅支援 REST API。指定 API Gateway 是否略過以下驗證：整合端點的憑證是由支援的憑證授權單位所發行。不建議您這樣做，但這可讓您使用由私人憑證授權單位簽署的憑證，或是自我簽署的憑證。如果啟用，API Gateway 仍會執行基本憑證驗證，包括檢查憑證的過期日期、主機名稱、根憑證授權單位是否存在，以及憑證鏈完整性。憑證鏈驗證包括 X509v3 名稱限制強制執行。如果您在insecureSkipVerification 啟用時遇到驗證錯誤，請確認您的 CA 憑證不包含名稱限制延伸模組，或在沒有該延伸模組的情況下重新發行它們。屬於私人授權單位的根憑證必須滿足下列限制： |

| 屬性名稱 | 類型 | 描述 |
| --- | --- | --- |
| | | • x509 延伸 keyUsage 必須有 keyCertSign 。<br>• x509 延伸 basicConstraints 必須有 CA:TRUE。<br><br>僅支援 HTTP 與 HTTP_PROXY 整合。<br><br>**Warning**<br>不建議啟用 insecureSkipVerification ，特別是對於與公有 HTTPS 端點的整合。如果您啟用 insecureSkipVerification ，會增加中間人攻擊的風險。 |
| serverNameToVerify | string | 僅支援 HTTP API 私人整合。如果您指定伺服器名稱，API Gateway 會使用此名稱來驗證整合憑證上的主機名稱。伺服器名稱也包含在 TLS 交握中，以支援伺服器名稱指示 (SNI) 或虛擬託管。 |

## x-amazon-apigateway-integration.tlsConfig examples

下列 OpenAPI 3.0 範例可讓 insecureSkipVerification 進行 REST API HTTP 代理整合。

```
"x-amazon-apigateway-integration": {
  "uri": "http://petstore-demo-endpoint.execute-api.com/petstore/pets",
  "responses": {
     default": {
       "statusCode": "200"
      }
  },
  "passthroughBehavior": "when_no_match",
  "httpMethod": "ANY",
  "tlsConfig" : {
    "insecureSkipVerification" : true
  }
  "type": "http_proxy",
}
```

下面的 OpenAPI 3.0 範例指定 `serverNameToVerify` 來進行 HTTP API 私有整合。

```
"x-amazon-apigateway-integration" : {
  "payloadFormatVersion" : "1.0",
  "connectionId" : "abc123",
  "type" : "http_proxy",
  "httpMethod" : "ANY",
  "uri" : "arn:aws:elasticloadbalancing:us-west-2:123456789012:listener/app/my-load-
balancer/50dc6c495c0c9188/0467ef3c8400ae65",
  "connectionType" : "VPC_LINK",
  "tlsConfig" : {
     "serverNameToVerify" : "example.com"
  }
}
```

# x-amazon-apigateway-minimum-compression-size

為 REST API 指定一個的最低壓縮大小。若要啟用壓縮，請指定介於 0 到 10485760 之間的整數。如需進一步了解，請參閱API Gateway 中 REST API 的承載壓縮。

## x-amazon-apigateway-minimum-compression-size example

下列範例會為 REST API 指定一個 `5242880` 位元組的最低壓縮大小。

```
"x-amazon-apigateway-minimum-compression-size": 5242880
```

# x-amazon-apigateway-policy

指定 REST API 的資源政策。如需進一步了解以資源為基礎的政策，請參閱使用 API Gateway 資源政策來控制對 REST API 的存取。如需資源策略範例，請參閱 API Gateway 資源政策範例。

## `x-amazon-apigateway-policy` 範例

下列範例會指定 REST API 的資源政策。資源策略會拒絕 (封鎖) 來自指定來源 IP 位址區塊的 API 傳入流量。匯入時`arn:aws:execute-api:`*region*`:`*account-id*`:`*api-id*`/*`，會使用目前的區域、AWS 您的帳戶 ID 和目前的 REST API ID `"execute-api:/*"`轉換為 。

```
"x-amazon-apigateway-policy": {
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": "*",
            "Action": "execute-api:Invoke",
            "Resource": [
                "execute-api:/*"
            ]
        },
        {
            "Effect": "Deny",
            "Principal": "*",
            "Action": "execute-api:Invoke",
            "Resource": [
               "execute-api:/*"
            ],
            "Condition" : {
                "IpAddress": {
                    "aws:SourceIp": "192.0.2.0/24"
                }
            }
        }
    ]
}
```

# 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，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-validators" : {
    "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-validators 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 REST API。

| 屬性名稱 | 類型 | 描述 |
| --- | --- | --- |
| *request_validator_name* | x-amazon-apigateway-request-validators.requestValidator 物件 | 指定包含具名驗證程式的驗證規則。例如：<br>`"basic" : { "validateRequestBody" : true, "validateRequestParameters" : true },`<br>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 x-amazon-apigateway-request-validator 屬性 屬性值的參考。 |

## x-amazon-apigateway-request-validators 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-validators" : {
    "basic" : {
      "validateRequestBody" : true,
      "validateRequestParameters" : true
    },
    "params-only" : {
      "validateRequestBody" : false,
      "validateRequestParameters" : true
    }
  },
  ...
```

```
}
```

## x-amazon-apigateway-request-validators.requestValidator 物件

將請求驗證程式的驗證規則指定為 x-amazon-apigateway-request-validators 物件 對應定義的一部分。

| 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-validators.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
}
```

## x-amazon-apigateway-security-policy

指定 REST API 的安全政策。如果您建立以 開頭的安全政策"SecurityPolicy_"，您還必須設定端點存取模式。若要進一步了解安全政策，請參閱 APIs中 REST API 的安全政策。

### x-amazon-apigateway-security-policy 範例

下列範例指定 REST API SecurityPolicy_TLS13_1_3_2025_09 的 。

```
"x-amazon-apigateway-security-policy": "SecurityPolicy_TLS13_1_3_2025_09"
```

# x-amazon-apigateway-tag-value 屬性

指定 HTTP API 的 AWS 標籤值。您可以使用 `x-amazon-apigateway-tag-value` 屬性做為根層級 OpenAPI 標籤物件的一部分，來指定 HTTP API 的 AWS 標籤。如果您指定不含 `x-amazon-apigateway-tag-value` 屬性的標籤名稱，則 API Gateway 會為值建立含有空字串的標籤。

如需進一步了解標記的資訊，請參閱 API Gateway 資源的標記。

| 屬性名稱 | 類型 | 描述 |
| --- | --- | --- |
| `name` | `String` | 指定標籤金鑰。 |
| `x-amazon-apigateway-tag-value` | `String` | 指定標籤值。 |

## `x-amazon-apigateway-tag-value` 範例

下列範例會為 HTTP API 指定兩個標籤：

- "Owner": "Admin"
- "Prod": ""

```
"tags": [
    {
      "name": "Owner",
      "x-amazon-apigateway-tag-value": "Admin"
    },
    {
      "name": "Prod"
    }
]
```

# Amazon API Gateway 的安全性

的雲端安全性 AWS 是最高優先順序。身為 AWS 客戶，您可以受益於資料中心和網路架構，這些架構專為滿足最安全敏感組織的需求而建置。

安全性是 AWS 與您之間的共同責任。共同責任模型將其描述為雲端「的」安全性和雲端「中」的安全性：

- 雲端的安全性 – AWS 負責保護在 AWS Cloud 中執行 AWS 服務的基礎設施。 AWS 也為您提供可安全使用的服務。在AWS 合規計畫中，第三方稽核人員會定期測試和驗證我們安全的有效性。若要了解適用於 Amazon API Gateway 的合規計劃，請參閱AWS 合規計劃範圍內的服務。
- 雲端的安全性 – 您的責任取決於您使用 AWS 的服務。您也必須對其他因素負責，包括資料的機密性、您公司的要求和適用法律和法規。

本文件有助於您了解如何在使用 API Gateway 時套用共同責任模型。下列各主題將說明如何設定 API Gateway，以達成您的安全性與合規目標。您也會了解如何使用其他 AWS 服務來協助您監控和保護 API Gateway 資源。

如需詳細資訊，請參閱 Amazon API Gateway 的安全概觀。

主題

- Amazon API Gateway 的資料保護
- 適用於 Amazon API Gateway 的 Identity and Access Management
- 在 Amazon API Gateway 中進行記錄和監控
- Amazon API Gateway 的合規驗證
- Amazon API Gateway 中的復原功能
- Amazon API Gateway 的基礎設施安全性
- Amazon API Gateway 的漏洞分析
- Amazon API Gateway 的安全最佳實務

# Amazon API Gateway 的資料保護

共同責任模型適用於 AWS Amazon API Gateway 中的資料保護。如此模型所述， AWS 負責保護執行所有 的全域基礎設施 AWS 雲端。您負責維護在此基礎設施上託管內容的控制權。您也同時負責所使

用 AWS 服務 的安全組態和管理任務。如需資料隱私權的詳細資訊，請參閱資料隱私權常見問答集。如需歐洲資料保護的詳細資訊，請參閱一般資料保護規則 (GDPR) 中心。

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

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

我們強烈建議您絕對不要將客戶的電子郵件地址等機密或敏感資訊，放在標籤或自由格式的文字欄位中，例如名稱欄位。這包括當您使用 API Gateway 或使用主控台、API AWS CLI、 AWS SDKs的其他 AWS 服務 時。您在標籤或自由格式文字欄位中輸入的任何資料都可能用於計費或診斷日誌。如果您提供外部伺服器的 URL，我們強烈建議請勿在驗證您對該伺服器請求的 URL 中包含憑證資訊。

## Amazon API Gateway 的資料加密

資料保護是指保護傳輸中的資料 (即資料往返於 API Gateway 時) 以及靜態資料 (即資料存放在 時AWS

### Amazon API Gateway 的靜態資料加密

如果您選擇為 REST API 啟用快取功能，就可以啟用快取加密。如需進一步了解，請參閱API Gateway 中 REST API 的快取設定。

如需關於資料保護的詳細資訊，請參閱 AWS 安全部落格上的 AWS 共同責任模型和歐盟《一般資料保護規範》(GDPR) 部落格文章。

加密和解密憑證的私有金鑰

當您為私有 API 建立自訂網域名稱時，您的 ACM 憑證和私有金鑰會使用具有別名 aws/acm 的 AWS 受管 KMS 金鑰進行加密。您可以在 AWS KMS 主控台的 AWS 受管金鑰下，檢視具有此別名的金鑰 ID。

API Gateway 不會直接存取您的 ACM 資源。而是使用 AWS TLS Connection Manager 來保護和存取憑證的私有金鑰。當您使用 ACM 憑證為私有 API 建立 APIs Gateway 自訂網域名稱時，API Gateway 會為您的憑證與 AWS TLS Connection Manager 建立關聯。方法是針對具有字首 aws/acm 的 AWS 受管金鑰在 AWS KMS 中建立授權。授權是一種政策工具，允許 TLS Connection Manager 使用密碼編譯操作中的 KMS 金鑰。授權可讓承授者主體 (TLS Connection Manager) 呼叫 KMS 金鑰上指定的授予操作，以解密憑證的私有金鑰。TLS Connection Manager 接著會使用憑證和解密的 (純文字) 私有金鑰，與 API Gateway 服務的用戶端建立安全連線 (SSL/TLS 工作階段)。當憑證與私有 API 的 API Gateway 自訂網域名稱取消關聯時，授權即會失效。

如果您想要移除對 KMS 金鑰的存取權，建議您使用 AWS 管理主控台 或 AWS CLI 中的 `update-service` 命令，取代或刪除服務的憑證。

API Gateway 的加密內容

加密內容是選用的一組鍵值對，其中包含有關您的私有金鑰可能用途的情境式內容資訊。AWS KMS 將加密內容與加密資料綁定，並將其用作其他驗證資料，以支援驗證加密。

當您的 TLS 金鑰與 API Gateway 和 TLS Connection Manager 搭配使用時，您的 API Gateway 服務名稱會包含在用來加密靜態金鑰的加密內容中。您可以透過檢視 CloudTrail 日誌中的加密內容，如下一節所示，或查看 ACM 主控台的關聯的資源標籤，以驗證憑證和私有金鑰所使用的 API Gateway 自訂網域名稱。

若要解密資料，則必須在請求中包含相同的加密內容。API Gateway 在所有的 AWS KMS 密碼編譯操作中使用相同的加密內容，其中金鑰為 `aws:apigateway:arn`，而值為 API Gateway `PrivateDomainName` 資源的 Amazon Resource Name (ARN)。

下列範例展示操作輸出中的加密內容，例如 `CreateGrant`。

```
"constraints": {
"encryptionContextEquals": {
"aws:acm:arn": "arn:aws:acm:us-west-2:859412291086:certificate/9177097a-
f0ae-4be1-93b1-19f911ea4f88",
"aws:apigateway:arn": "arn:aws:apigateway:us-west-2:859412291086:/domainnames/denytest-
part1.pdx.sahig.people.aws.dev+cbaeumzjhg"
}
},
"operations": [
"Decrypt"
],
"granteePrincipal": "tlsconnectionmanager.amazonaws.com"
```

### Amazon API Gateway 的傳輸中資料加密

使用 Amazon API Gateway 建立的 API 只會公開 HTTPS 端點。API Gateway 不支援未加密的 (HTTP) 端點。

API Gateway 會管理預設 `execute-api` 端點的憑證。如果您要設定自訂網域名稱，請指定網域名稱的憑證。最佳實務是不要關聯憑證。

為了提高安全性，您可以選擇要為 API Gateway 自訂網域強制執行的最低 Transport Layer Security (TLS) 通訊協定版本。WebSocket API 與 HTTP API 僅支援 TLS 1.2 版。如需進一步了解，請參閱在 API Gateway 中為您的自訂網域選擇安全政策。

您也可以在帳戶中使用自訂 SSL 憑證來設定 Amazon CloudFront 分佈，並搭配區域 API 使用。接著，您可以根據安全與合規要求來使用 TLS 1.1 以上版本，為 CloudFront 分佈設定安全政策。

如需關於資料保護的詳細資訊，請參閱 AWS 安全部落格上的在 API Gateway 中保護您的 REST API及 AWS 共同責任模型和歐盟《一般資料保護規範》(GDPR) 部落格文章。

### 網際網路流量隱私權

使用 Amazon API Gateway 時，您可以建立只能從 Amazon Virtual Private Cloud (VPC) 存取的私有 REST API。VPC 會使用界面 VPC 端點，也就是您在 VPC 中建立的端點網路界面。您可以使用資源政策，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 Direct Connect 在內部部署網路與 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。如需進一步了解，請參閱the section called “私有 REST API”。

## 適用於 Amazon API Gateway 的 Identity and Access Management

AWS Identity and Access Management (IAM) 是 AWS 服務 ，可協助管理員安全地控制對 AWS 資源的存取。IAM 管理員可以控制身分驗證 (已登入) 和授權 (具有許可) 來使用 API Gateway 資源。IAM 是您可以免費使用 AWS 服務 的 。

主題

- 目標對象
- 使用身分驗證
- 使用政策管理存取權
- Amazon API Gateway 與 IAM 搭配運作的方式

- Amazon API Gateway 身分型政策範例
- Amazon API Gateway 資源型政策範例
- 疑難排解 Amazon API Gateway 身分和存取
- 使用 API Gateway 的服務連結角色

## 目標對象

使用方式 AWS Identity and Access Management (IAM) 會根據您的角色而有所不同：

- 服務使用者 — 若無法存取某些功能，請向管理員申請所需許可 (請參閱 疑難排解 Amazon API Gateway 身分和存取)
- 服務管理員 — 負責設定使用者存取權並提交相關許可請求 (請參閱 Amazon API Gateway 與 IAM 搭配運作的方式)
- IAM 管理員 — 撰寫政策以管理存取控制 (請參閱 Amazon API Gateway 身分型政策範例)

## 使用身分驗證

身分驗證是您 AWS 使用身分憑證登入 的方式。您必須以 AWS 帳戶根使用者、IAM 使用者或擔任 IAM 角色身分進行身分驗證。

您可以使用身分來源的登入資料，例如 AWS IAM Identity Center (IAM Identity Center)、單一登入身分驗證或 Google/Facebook 登入資料，以聯合身分的形式登入。如需有關登入的詳細資訊，請參閱《AWS 登入 使用者指南》中的如何登入您的 AWS 帳戶。

對於程式設計存取， AWS 提供 SDK 和 CLI 以密碼編譯方式簽署請求。如需詳細資訊，請參閱《IAM 使用者指南》中的 API 請求的AWS 第 4 版簽署程序。

### AWS 帳戶 根使用者

當您建立 時 AWS 帳戶，您會從一個名為 AWS 帳戶 theroot 使用者的登入身分開始，該身分具有對所有 AWS 服務 和 資源的完整存取權。強烈建議不要使用根使用者來執行日常任務。有關需要根使用者憑證的任務，請參閱《IAM 使用者指南》中的需要根使用者憑證的任務。

### IAM 使用者和群組

IAM 使用者https://docs.aws.amazon.com/IAM/latest/UserGuide/id_users.html是一種身分具備單人或應用程式的特定許可權。建議以臨時憑證取代具備長期憑證的 IAM 使用者。如需詳細資訊，請參閱《IAM 使用者指南》中的要求人類使用者使用聯合身分提供者來 AWS 使用臨時憑證存取 。

IAM 群組會指定 IAM 使用者集合，使管理大量使用者的許可權更加輕鬆。如需詳細資訊，請參閱《IAM 使用者指南》中的 IAM 使用者的使用案例。

### IAM 角色

IAM 角色https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles.html的身分具有特定許可權，其可以提供臨時憑證。您可以透過從使用者切換到 IAM 角色（主控台）或呼叫 AWS CLI 或 AWS API 操作來擔任角色。如需詳細資訊，請參閱《IAM 使用者指南》中的擔任角色的方法。

IAM 角色適用於聯合身分使用者存取、臨時 IAM 使用者許可、跨帳戶存取權與跨服務存取，以及在 Amazon EC2 執行的應用程式。如需詳細資訊，請參閱《IAM 使用者指南》中的 IAM 中的快帳戶資源存取。

## 使用政策管理存取權

您可以透過建立政策並將其連接到身分或資源 AWS 來控制 AWS 中的存取。政策定義與身分或資源相關聯的許可。當委託人提出請求時 AWS ， 會評估這些政策。大多數政策會以 JSON 文件 AWS 形式存放在 中。如需進一步了解 JSON 政策文件，請參閱《IAM 使用者指南》中的 JSON 政策概觀。

管理員會使用政策，透過定義哪些主體可在哪些條件下對哪些資源執行動作，以指定可存取的範圍。

預設情況下，使用者和角色沒有許可。IAM 管理員會建立 IAM 政策並將其新增至角色，供使用者後續擔任。IAM 政策定義動作的許可，無論採用何種方式執行。

### 身分型政策

身分型政策是附加至身分 (使用者、使用者群組或角色) 的 JSON 許可政策文件。這類政策控制身分可對哪些資源執行哪些動作，以及適用的條件。如需了解如何建立身分型政策，請參閱《IAM 使用者指南》中的透過客戶管理政策定義自訂 IAM 許可。

身分型政策可分為內嵌政策 (直接內嵌於單一身分) 與受管政策 (可附加至多個身分的獨立政策)。如需了解如何在受管政策及內嵌政策之間做選擇，請參閱《IAM 使用者指南》中的在受管政策與內嵌政策之間選擇。

### 資源型政策

資源型政策是附加到資源的 JSON 政策文件。範例包括 IAM 角色信任政策與 Amazon S3 儲存貯體政策。在支援資源型政策的服務中，服務管理員可以使用它們來控制對特定資源的存取權限。您必須在資源型政策中指定主體。

資源型政策是位於該服務中的內嵌政策。您無法在以資源為基礎的政策中使用來自 IAM 的 AWS 受管政策。

### 存取控制清單 (ACL)

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

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

### 其他政策類型

AWS 支援其他政策類型，可設定更多常見政策類型授予的最大許可：

- 許可界限 — 設定身分型政策可授與 IAM 實體的最大許可。如需詳細資訊，請參閱《 IAM 使用者指南》中的 IAM 實體許可界限。
- 服務控制政策 (SCP) — 為 AWS Organizations中的組織或組織單位指定最大許可。如需詳細資訊，請參閱《AWS Organizations 使用者指南》中的服務控制政策。
- 資源控制政策 (RCP) — 設定您帳戶中資源可用許可的上限。如需詳細資訊，請參閱《AWS Organizations 使用者指南》中的資源控制政策 (RCP)。
- 工作階段政策 — 在以程式設計方式為角色或聯合身分使用者建立臨時工作階段時，以參數形式傳遞的進階政策。如需詳細資訊，請參《IAM 使用者指南》中的工作階段政策。

### 多種政策類型

當多種類型的政策適用於請求時，產生的許可會更複雜而無法理解。若要了解如何 AWS 在涉及多個政策類型時決定是否允許請求，請參閱《IAM 使用者指南》中的政策評估邏輯。

## Amazon API Gateway 與 IAM 搭配運作的方式

在您使用 IAM 來管理 API Gateway 的存取權之前，您應該了解提供哪些 IAM 功能與 API Gateway 搭配使用。若要全面了解 API Gateway 和其他 AWS 服務如何與 IAM 搭配使用，請參閱《IAM 使用者指南》中的與 IAM AWS 搭配使用的 服務。

主題

- API Gateway 身分類型政策

- API Gateway 資源類型政策
- 根據 API Gateway 標籤授權
- API Gateway IAM 角色

## API Gateway 身分類型政策

使用 IAM 身分型政策，您可以指定允許或拒絕的動作和資源，以及在何種條件下允許或拒絕動作。API Gateway 支援特定動作、資源和條件索引鍵。如需有關 API Gateway 特定動作、資源和條件索引鍵的詳細資訊，請參閱 Amazon API Gateway Management 的動作、資源和條件索引鍵和 Amazon API Gateway Management V2 的動作、資源和條件索引鍵。有關您在 JSON 政策中使用之所有元素的資訊，請參閱 IAM 使用者指南中的 IAM JSON 政策元素參考。

下列範例顯示允許使用者僅建立或更新私有 REST API 的身分型政策。如需更多範例，請參閱 the section called “身分型政策範例”。

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "ScopeToPrivateApis",
      "Effect": "Allow",
      "Action": [
        "apigateway:PATCH",
        "apigateway:POST",
        "apigateway:PUT"
      ],
      "Resource": [
        "arn:aws:apigateway:us-east-1::/restapis",
        "arn:aws:apigateway:us-east-1::/restapis/??????????"
      ],
      "Condition": {
        "ForAllValues:StringEqualsIfExists": {
          "apigateway:Request/EndpointType": "PRIVATE",
          "apigateway:Resource/EndpointType": "PRIVATE"
        }
      }
    },
    {
```

```
      "Sid": "AllowResourcePolicyUpdates",
      "Effect": "Allow",
      "Action": [
          "apigateway:UpdateRestApiPolicy"
      ],
      "Resource": [
          "arn:aws:apigateway:us-east-1::/restapis/*"
      ]
    }
  ]
}
```

動作

JSON 原則的 `Action` 元素描述您可以用來允許或拒絕原則中存取的動作。

API Gateway 中的政策動作會在動作之前使用下列字首：`apigateway:`。政策陳述式必須包含 `Action` 或 `NotAction` 元素。API Gateway 會定義自己的一組動作，描述您可以使用此服務執行的任務。

API 管理 `Action` 表達式具有格式 `apigateway:`*`action`*，其中*`##`*是下列 API Gateway 動作之一：GET、POST、PUT、DELETE、PATCH (以更新資源)，或 * (此為前述所有動作)。

`Action` 表達式的一些範例包括：

- **`apigateway:*`** 是所有 API Gateway 動作。
- **`apigateway:GET`** 只有 API Gateway 中的 GET 動作。

若要在單一陳述式中指定多個動作，請用逗號分隔，如下所示：

```
"Action": [
      "apigateway:action1",
      "apigateway:action2"
```

有關用於特定 API Gateway 操作之 HTTP 動詞的資訊，請參閱 Amazon API Gateway 第 1 版 API 參考 (REST API) 和 Amazon API Gateway 第 2 版 API 參考 (WebSocket 和 HTTP API)。

如需詳細資訊，請參閱the section called “身分型政策範例”。

Resources

管理員可以使用 AWS JSON 政策來指定誰可以存取內容。也就是說，哪個主體在什麼條件下可以對什麼資源執行哪些動作。

`Resource` JSON 政策元素可指定要套用動作的物件。最佳實務是使用其 Amazon Resource Name (ARN) 來指定資源。若動作不支援資源層級許可，使用萬用字元 (*) 表示該陳述式適用於所有資源。

```
"Resource": "*"
```

API Gateway 資源具有以下 ARN 格式：

```
arn:aws:apigateway:region::resource-path-specifier
```

例如，若要指定具有 ID *api-id* 及其子資源的 REST API，例如陳述式中的授權者，請使用下列 ARN：

```
"Resource": "arn:aws:apigateway:us-east-2::/restapis/api-id/*"
```

若要指定屬於特定帳戶的所有 REST API 和子資源，請使用萬用字元 (*)：

```
"Resource": "arn:aws:apigateway:us-east-2::/restapis/*"
```

若要查看 API Gateway 資源類型及其 ARN 的清單，請參閱API Gateway Amazon Resource Name (ARN) 參考資料。

條件索引鍵

管理員可以使用 AWS JSON 政策來指定誰可以存取內容。也就是說，哪個主體在什麼條件下可以對什麼資源執行哪些動作。

`Condition` 元素會根據定義的條件，指定陳述式的執行時機。您可以建立使用條件運算子的條件運算式 (例如等於或小於)，來比對政策中的條件和請求中的值。若要查看所有 AWS 全域條件索引鍵，請參閱《IAM 使用者指南》中的AWS 全域條件內容索引鍵。

API Gateway 會定義自己的一組條件金鑰，也支援使用一些全域條件金鑰。若要查看 API Gateway 條件索引鍵清單，請參閱《IAM 使用者指南》中的 Amazon API Gateway 條件索引鍵。若要了解您可以搭配條件索引鍵使用哪些動作和資源，請參閱 Amazon API Gateway 定義的動作。

如需有關標記的資訊 (包括屬性型存取控制)，請參閱 標記。

範例

若要了解 API Gateway 身分型政策的範例，請參閱Amazon API Gateway 身分型政策範例。

## API Gateway 資源類型政策

資源型政策是 JSON 政策文件，這些文件會指定指定的主體可對 API Gateway 資源以及在怎樣的條件下執行哪些動作。API Gateway 支援適用於 REST API 的資源型許可政策。您可以使用資源政策來控制誰可以叫用 REST API。如需詳細資訊，請參閱the section called “使用 API Gateway 資源政策”。

範例

若要了解 API Gateway 資源型政策的範例，請參閱API Gateway 資源政策範例。

## 根據 API Gateway 標籤授權

您可以將標籤連接到 API Gateway 資源，或是在請求中將標籤傳遞給 API Gateway。若要根據標籤控制存取，請使用 `apigateway:ResourceTag/`*`key-name`*、`aws:RequestTag/`*`key-name`* 或 `aws:TagKeys` 條件金鑰，在原則的條件元素中，提供標籤資訊。如需標記 API Gateway 資源的詳細資訊，請參閱the section called “屬性型存取控制”。

若要了解身分型政策範例，以根據該資源上的標籤來限制存取資源，請參閱使用標籤來控制對 API Gateway REST API 資源的存取。

## API Gateway IAM 角色

IAM 角色是您 AWS 帳戶中具有特定許可的實體。

搭配 API Gateway 使用暫時登入資料

您可以搭配聯合使用暫時憑證、擔任 IAM 角色，或是擔任跨帳戶角色。您可以透過呼叫 AssumeRole 或 GetFederationToken 等 AWS STS API 操作來取得臨時安全登入資料。

API Gateway 支援使用臨時登入資料。

服務連結角色

服務連結角色可讓 AWS 服務存取其他服務中的資源，以代表您完成 動作。服務連結角色會顯示在您的 IAM 帳戶中，並由該服務所擁有。IAM 管理員可以檢視，但不能編輯服務連結角色的許可。

API Gateway 支援服務連結角色。如需關於建立或管理 API Gateway 服務連結角色的資訊，請參閱使用 API Gateway 的服務連結角色。

服務角色

服務可代表您擔任服務角色。服務角色可讓服務存取其他服務中的資源，以代表您完成動作。服務角色會出現在您的 IAM 帳戶中，並由該帳戶擁有，因此管理員可以變更此角色的許可。不過，這樣可能會破壞此服務的功能。

API Gateway 支援服務角色。

# Amazon API Gateway 身分型政策範例

預設情況下，IAM 使用者和角色沒有建立或修改 API Gateway 資源的許可。他們也無法使用 AWS 管理主控台 AWS CLI或 AWS SDKs 執行任務。IAM 管理員必須建立 IAM 政策，授予使用者和角色在指定資源上執行特定 API 作業的所需許可。管理員接著必須將這些政策連接至需要這些許可的 IAM 使用者或群組。

如需有關如何建立 IAM 政策的資訊，請參閱《IAM 使用者指南》中在 JSON 索引標籤上建立政策。如需有關 API Gateway 特定動作、資源和條件索引鍵的資訊，請參閱 Amazon API Gateway Management 的動作、資源和條件索引鍵和 Amazon API Gateway Management V2 的動作、資源和條件索引鍵。

主題


- 政策最佳實務
- 允許使用者檢視他們自己的許可
- 簡易讀取許可
- 僅建立 REQUEST 或 JWT 授權者
- 要求停用預設 execute-api 端點
- 允許使用者僅建立或更新私有 REST API
- 要求 API 路由具有授權
- 防止使用者建立或更新 VPC 連結
- 使用路由規則的範例政策


## 政策最佳實務

身分型政策會判斷您帳戶中的某個人員是否可以建立、存取或刪除 API Gateway 資源。這些動作可能會讓您的 AWS 帳戶產生費用。當您建立或編輯身分型政策時，請遵循下列準則及建議事項：

- 開始使用 AWS 受管政策並邁向最低權限許可 – 若要開始將許可授予您的使用者和工作負載，請使用將許可授予許多常見使用案例的 AWS 受管政策。它們可在您的 中使用 AWS 帳戶。我們建議您定義特定於使用案例 AWS 的客戶受管政策，以進一步減少許可。如需更多資訊，請參閱《IAM 使用者指南》中的 AWS 受管政策或任務職能的AWS 受管政策。
- 套用最低權限許可 – 設定 IAM 政策的許可時，請僅授予執行任務所需的許可。為實現此目的，您可以定義在特定條件下可以對特定資源採取的動作，這也稱為最低權限許可。如需使用 IAM 套用許可的更多相關資訊，請參閱《IAM 使用者指南》中的 IAM 中的政策和許可。
- 使用 IAM 政策中的條件進一步限制存取權 – 您可以將條件新增至政策，以限制動作和資源的存取。例如，您可以撰寫政策條件，指定必須使用 SSL 傳送所有請求。如果透過特定 等使用服務動作 AWS 服務，您也可以使用條件來授予其存取權 CloudFormation。如需詳細資訊，請參閱《IAM 使用者指南》中的 IAM JSON 政策元素：條件。
- 使用 IAM Access Analyzer 驗證 IAM 政策，確保許可安全且可正常運作 – IAM Access Analyzer 驗證新政策和現有政策，確保這些政策遵從 IAM 政策語言 (JSON) 和 IAM 最佳實務。IAM Access Analyzer 提供 100 多項政策檢查及切實可行的建議，可協助您撰寫安全且實用的政策。如需詳細資訊，請參閱《IAM 使用者指南》中的使用 IAM Access Analyzer 驗證政策。
- 需要多重要素驗證 (MFA) – 如果您的案例需要 IAM 使用者或 中的根使用者 AWS 帳戶，請開啟 MFA 以提高安全性。如需在呼叫 API 操作時請求 MFA，請將 MFA 條件新增至您的政策。如需詳細資訊，請參閱《IAM 使用者指南》中的透過 MFA 的安全 API 存取。

如需 IAM 中最佳實務的相關資訊，請參閱《IAM 使用者指南》中的 IAM 安全最佳實務。

## 允許使用者檢視他們自己的許可

此範例會示範如何建立政策，允許 IAM 使用者檢視附加到他們使用者身分的內嵌及受管政策。此政策包含在主控台或使用 或 AWS CLI AWS API 以程式設計方式完成此動作的許可。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
```

```
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

## 簡易讀取許可

此範例政策提供使用者許可，以取得在 us-east-1 a123456789 AWS 區域中識別符 之 HTTP 或 WebSocket API 所有資源的相關資訊。資源 `arn:aws:apigateway:us-east-1::/apis/a123456789/*` 包括 API 的所有子資源，例如授權者和部署。

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "apigateway:GET"
      ],
      "Resource": [
        "arn:aws:apigateway:us-east-1::/apis/a123456789/*"
      ]
    }
  ]
```

```
}
```

## 僅建立 REQUEST 或 JWT 授權者

此範例政策允許使用者僅使用 REQUEST 或 JWT 授權者建立 API，包括透過匯入。在政策的 Resource 區段中，arn:aws:apigateway:us-east-1::/apis/?????????? 要求資源最多有 10 個字元，其中排除 API 的子資源。本範例會在 ForAllValues 區段中使用 Condition，因為使用者可以透過匯入 API 來一次建立多個授權者。

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "OnlyAllowSomeAuthorizerTypes",
      "Effect": "Allow",
      "Action": [
        "apigateway:PUT",
        "apigateway:POST",
        "apigateway:PATCH"
      ],
      "Resource": [
        "arn:aws:apigateway:us-east-1::/apis",
        "arn:aws:apigateway:us-east-1::/apis/??????????",
        "arn:aws:apigateway:us-east-1::/apis/*/authorizers",
        "arn:aws:apigateway:us-east-1::/apis/*/authorizers/*"
      ],
      "Condition": {
        "ForAllValues:StringEqualsIfExists": {
          "apigateway:Request/AuthorizerType": [
            "REQUEST",
            "JWT"
          ]
        }
      }
    }
  ]
}
```

## 要求停用預設 execute-api 端點

此範例政策允許使用者建立、更新或匯入 API，而且要求 DisableExecuteApiEndpoint 為 true。如果 DisableExecuteApiEndpoint 為 true，則用戶端無法使用預設 execute-api 端點來叫用 API。

我們使用 BoolIfExists 條件來處理呼叫以更新沒有填入 DisableExecuteApiEndpoint 條件索引鍵的 API。當使用者嘗試建立或匯入 API 時，一律會填入 DisableExecuteApiEndpoint 條件索引鍵。

由於 apis/* 資源也會擷取子資源 (例如授權者或方法)，所以我們明確地將其範圍設定為僅使用 Deny 陳述式的 API。

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "DisableExecuteApiEndpoint",
      "Effect": "Allow",
      "Action": [
        "apigateway:PATCH",
        "apigateway:POST",
        "apigateway:PUT"
      ],
      "Resource": [
        "arn:aws:apigateway:us-east-1::/apis",
        "arn:aws:apigateway:us-east-1::/apis/*"
      ],
      "Condition": {
        "BoolIfExists": {
          "apigateway:Request/DisableExecuteApiEndpoint": true
        }
      }
    },
    {
      "Sid": "ScopeDownToJustApis",
      "Effect": "Deny",
      "Action": [
        "apigateway:PATCH",
        "apigateway:POST",
```

```
        "apigateway:PUT"
      ],
      "Resource": [
        "arn:aws:apigateway:us-east-1::/apis/*/*"
      ]
    }
  ]
}
```

## 允許使用者僅建立或更新私有 REST API

此範例政策使用條件索引鍵，要求使用者僅建立 PRIVATE API，並禁止可能會將 API 從 PRIVATE 變更為其他類型 (例如 REGIONAL) 的更新。

我們使用 ForAllValues 以要求每個新增至 API 中的 EndpointType 都是 PRIVATE。我們使用資源條件索引鍵來允許任何 API 更新，只要它是 PRIVATE。條件索引鍵存在時才會套用 ForAllValues。

我們使用非窮盡比對器 (?) 來明確比對 API ID，以防止允許授權者等非 API 資源。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "ScopePutToPrivateApis",
            "Effect": "Allow",
            "Action": [
                "apigateway:PUT"
            ],
            "Resource": [
                "arn:aws:apigateway:us-east-1::/restapis",
                "arn:aws:apigateway:us-east-1::/restapis/??????????"
            ],
            "Condition": {
                "ForAllValues:StringEquals": {
                    "apigateway:Resource/EndpointType": "PRIVATE"
                }
            }
        },
```

```
        {
            "Sid": "ScopeToPrivateApis",
            "Effect": "Allow",
            "Action": [
                "apigateway:DELETE",
                "apigateway:PATCH",
                "apigateway:POST"
            ],
            "Resource": [
                "arn:aws:apigateway:us-east-1::/restapis",
                "arn:aws:apigateway:us-east-1::/restapis/??????????"
            ],
            "Condition": {
                "ForAllValues:StringEquals": {
                    "apigateway:Request/EndpointType": "PRIVATE",
                    "apigateway:Resource/EndpointType": "PRIVATE"
                }
            }
        },
        {
            "Sid": "AllowResourcePolicyUpdates",
            "Effect": "Allow",
            "Action": [
                "apigateway:UpdateRestApiPolicy"
            ],
            "Resource": [
                "arn:aws:apigateway:us-east-1::/restapis/*"
            ]
        }
    ]
}
```

## 要求 API 路由具有授權

如果路由沒有授權，此政策會導致嘗試建立或更新路由 (包括透過匯入) 失敗。如果金鑰不存在，例如沒有建立或更新路由時，則 `ForAnyValue` 會評估為 False。我們使用 `ForAnyValue`，因為可以透過匯入建立多個路由。

JSON

```
{
```

```
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "AllowUpdatesOnApisAndRoutes",
      "Effect": "Allow",
      "Action": [
        "apigateway:POST",
        "apigateway:PATCH",
        "apigateway:PUT"
      ],
      "Resource": [
        "arn:aws:apigateway:us-east-1::/apis",
        "arn:aws:apigateway:us-east-1::/apis/??????????",
        "arn:aws:apigateway:us-east-1::/apis/*/routes",
        "arn:aws:apigateway:us-east-1::/apis/*/routes/*"
      ]
    },
    {
      "Sid": "DenyUnauthorizedRoutes",
      "Effect": "Deny",
      "Action": [
        "apigateway:POST",
        "apigateway:PATCH",
        "apigateway:PUT"
      ],
      "Resource": [
        "arn:aws:apigateway:us-east-1::/apis",
        "arn:aws:apigateway:us-east-1::/apis/*"
      ],
      "Condition": {
        "ForAnyValue:StringEqualsIgnoreCase": {
          "apigateway:Request/RouteAuthorizationType": "NONE"
        }
      }
    }
  ]
}
```

## 防止使用者建立或更新 VPC 連結

此政策可防止使用者建立或更新 VPC 連結。VPC 連結可以公開 Amazon VPC 內的資源給 VPC 以外的用戶端。

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "DenyVPCLink",
      "Effect": "Deny",
      "Action": [
        "apigateway:POST",
        "apigateway:PUT",
        "apigateway:PATCH"
      ],
      "Resource": [
        "arn:aws:apigateway:us-east-1::/vpclinks",
        "arn:aws:apigateway:us-east-1::/vpclinks/*"
      ]
    }
  ]
}
```

## 使用路由規則的範例政策

下列範例政策說明如何使用 RoutingRule 條件索引鍵，來控制使用者如何將流量從其自訂網域名稱路由至其 REST API。您可以使用這些範例，為使用者可制訂的路由規則建立更精細的政策。如需詳細資訊，請參閱the section called “路由規則”。

防止使用者變更自訂網域名稱路由請求的方式

此政策可防止使用者建立或更新 `BasePathMapping`、`ApiMapping` 或 `RoutingRule`。這些資源全都可能會變更自訂網域名稱將請求路由至 API 的方式。

JSON

```
{
```

```
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "DenyAccessBasePathMappingsApiMappingsRoutingRules",
      "Effect": "Deny",
      "Action": "apigateway:*",
      "Resource": [
        "arn:aws:apigateway:us-east-1::/domainnames/example.com/basepathmappings/
*",
        "arn:aws:apigateway:us-east-1::/domainnames/example.com/apimappings/*",
        "arn:aws:apigateway:us-east-1:111122223333:/domainnames/example.com/
routingrules/*"
      ]
    }
  ]
}
```

允許使用者更新特定優先順序的路由規則

此政策只允許使用者將路由規則更新為 1001 到 2000 之間的優先順序。您可以使用此規則將生產規則與較低優先順序的規則分開，然後允許使用者修改較低優先順序的規則，而不會影響生產規則。

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "UpdatingRoutingRulePriorityBetween1001And2000",
      "Effect": "Allow",
      "Action": "apigateway:UpdateRoutingRule",
      "Resource": "arn:aws:apigateway:us-east-1:111122223333:/domainnames/
example.com/routingrules/*",
      "Condition": {
        "NumericGreaterThanEquals": {
          "apigateway:Resource/Priority": 1001,
          "apigateway:Request/Priority": 1001
        },
       "NumericLessThanEquals": {
          "apigateway:Resource/Priority": 2000,
          "apigateway:Request/Priority": 2000
        }
```

```
      }
    }
  ]
}
```

允許使用者更新特定基本路徑值的路由規則或基本路徑映射

此政策只允許使用者更新開頭為 orders 的任何基本路徑的基本路徑映射，或更新與開頭為 orders 的基本路徑相符的路由規則。在此政策中，使用者可以更新 orders/create 或 orders123 的基本路徑映射或路由規則，但不能更新 payment/orders。

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
      {
        "Sid": "AllowUpdateRoutingRuleUnderPathOrders",
        "Effect": "Allow",
        "Action": "apigateway:UpdateRoutingRule",
        "Resource": "arn:aws:apigateway:us-east-1:111122223333:/domainnames/
example.com/routingrules/*",
        "Condition": {
            "ForAllValues:StringLike": {
                "apigateway:Request/ConditionBasePaths": ["orders*"],
                "apigateway:Resource/ConditionBasePaths": ["orders*"]
            },
            "Null":{
                 "apigateway:Request/ConditionBasePaths":"false",
                 "apigateway:Resource/ConditionBasePaths":"false"
          }
        }
      }
  ]
}
```

允許使用者將路由模式更新為特定值

此政策只允許使用者將路由模式更新為 API_MAPPING_ONLY 和 ROUTING_RULE_THEN_API_MAPPING。如需路由模式的詳細資訊，請參閱 the section called “為您的自訂網域名稱設定路由模式”。

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
        {
           "Sid": "AllowUpdateRoutingModeToAnythingWithApiMapping",
           "Effect": "Allow",
           "Action": ["apigateway:PATCH"],
           "Resource": "arn:aws:apigateway:us-east-1::/domainnames/example.com",
              "Condition": {
               "StringLike": {
                   "apigateway:Request/RoutingMode":"*API_MAPPING*"
               }
           }
       }
    ]
}
```

## Amazon API Gateway 資源型政策範例

如需資源型的政策範例，請參閱the section called “API Gateway 資源政策範例”。

## 疑難排解 Amazon API Gateway 身分和存取

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

主題

- 我未獲授權，不得在 API Gateway 中執行動作
- 我未獲得執行 iam:PassRole 的授權
- 我想要允許 AWS 帳戶外的人員存取我的 API Gateway 資源

## 我未獲授權，不得在 API Gateway 中執行動作

如果您收到錯誤，告知您未獲授權執行動作，您的政策必須更新，允許您執行動作。

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

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 apigateway:GetWidget on resource: my-example-widget because no identity-based policy
 allows the GetWidget action
```

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

如果您需要協助，請聯絡您的 AWS 管理員。您的管理員提供您的簽署憑證。

## 我未獲得執行 iam:PassRole 的授權

如果您收到錯誤，告知您無權執行 iam:PassRole 動作，您的政策必須更新，允許您將角色傳遞給 API Gateway。

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

當名為 marymajor 的 IAM 使用者嘗試使用主控台在 API Gateway 中執行動作時，發生下列範例錯誤。但是，動作請求服務具備服務角色授予的許可。Mary 沒有將角色傳遞給服務的許可。

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

在這種情況下，Mary 的政策必須更新，允許她執行 iam:PassRole 動作。

如果您需要協助，請聯絡您的 AWS 管理員。您的管理員提供您的簽署憑證。

## 我想要允許 AWS 帳戶外的人員存取我的 API Gateway 資源

您可以建立一個角色，讓其他帳戶中的使用者或您組織外部的人員存取您的資源。您可以指定要允許哪些信任物件取得該角色。針對支援基於資源的政策或存取控制清單 (ACL) 的服務，您可以使用那些政策來授予人員存取您的資源的許可。

若要進一步了解，請參閱以下內容：

- 若要了解 API Gateway 是否支援這些功能，請參閱 Amazon API Gateway 與 IAM 搭配運作的方式。
- 若要了解如何 AWS 帳戶 在您擁有的 資源之間提供存取權，請參閱《IAM 使用者指南》中的在您擁有 AWS 帳戶 的另一個 中提供存取權給 IAM 使用者。
- 若要了解如何將資源的存取權提供給第三方 AWS 帳戶，請參閱《IAM 使用者指南》中的將存取權提供給第三方 AWS 帳戶 擁有。
- 如需了解如何透過聯合身分提供存取權，請參閱《IAM 使用者指南》中的將存取權提供給在外部進行身分驗證的使用者 (聯合身分)。
- 如需了解使用角色和資源型政策進行跨帳戶存取之間的差異，請參閱《IAM 使用者指南》中的 IAM 中的跨帳戶資源存取。

## 使用 API Gateway 的服務連結角色

Amazon API Gateway 使用 AWS Identity and Access Management (IAM) 服務連結角色。服務連結角色是直接連結至 API Gateway 的一種特殊 IAM 角色類型。服務連結角色由 API Gateway 預先定義，並包含該服務代表您呼叫其他 AWS 服務所需的所有許可。

服務連結角色可讓設定 API Gateway 更為簡單，因為您不必手動新增必要的許可。API Gateway 會定義其服務連結角色的許可，除非另外定義，否則只有 API Gateway 可擔任該角色。定義的許可包括信任政策和許可政策，並且該許可政策不能附加到任何其他 IAM 實體。

您必須先刪除相關的資源，才能刪除服務連結角色。這可保護您的 API Gateway 資源，避免您不小心移除資源的存取許可。

如需有關支援服務連結角色的其他服務的資訊，請參閱可搭配 IAM 運作的AWS 服務，並尋找 Service-Linked Role (服務連結角色) 欄顯示為 Yes (是) 的服務。選擇具有連結的 Yes (是)，以檢視該服務的服務連結角色文件。

### API Gateway 的服務連結角色許可

API Gateway 使用名為 AWSServiceRoleForAPIGateway 的服務連結角色 – 允許 API Gateway 代表您存取 Elastic Load Balancing、Amazon Data Firehose 和其他服務資源。

AWSServiceRoleForAPIGateway 服務連結角色信任下列服務擔任該角色：

- `ops.apigateway.amazonaws.com`

角色許可政策允許 API Gateway 在指定資源上完成下列動作：

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "elasticloadbalancing:AddListenerCertificates",
                "elasticloadbalancing:RemoveListenerCertificates",
                "elasticloadbalancing:ModifyListener",
                "elasticloadbalancing:DescribeListeners",
                "elasticloadbalancing:DescribeLoadBalancers",
                "xray:PutTraceSegments",
                "xray:PutTelemetryRecords",
                "xray:GetSamplingTargets",
                "xray:GetSamplingRules",
                "logs:CreateLogDelivery",
                "logs:GetLogDelivery",
                "logs:UpdateLogDelivery",
                "logs:DeleteLogDelivery",
                "logs:ListLogDeliveries",
                "servicediscovery:DiscoverInstances"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "firehose:DescribeDeliveryStream",
                "firehose:PutRecord",
                "firehose:PutRecordBatch"
            ],
            "Resource": "arn:aws:firehose:*:*:deliverystream/amazon-apigateway-*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "acm:DescribeCertificate",
                "acm:GetCertificate"
            ],
```

```
            "Resource": "arn:aws:acm:*:*:certificate/*"
        },
        {
            "Effect": "Allow",
            "Action": "ec2:CreateNetworkInterfacePermission",
            "Resource": "arn:aws:ec2:*:*:network-interface/*"
        },
        {
            "Effect": "Allow",
            "Action": "ec2:CreateTags",
            "Resource": "arn:aws:ec2:*:*:network-interface/*",
            "Condition": {
                "ForAllValues:StringEquals": {
                    "aws:TagKeys": [
                        "Owner",
                        "VpcLinkId"
                    ]
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:ModifyNetworkInterfaceAttribute",
                "ec2:DeleteNetworkInterface",
                "ec2:AssignPrivateIpAddresses",
                "ec2:CreateNetworkInterface",
                "ec2:DeleteNetworkInterfacePermission",
                "ec2:DescribeNetworkInterfaces",
                "ec2:DescribeAvailabilityZones",
                "ec2:DescribeNetworkInterfaceAttribute",
                "ec2:DescribeVpcs",
                "ec2:DescribeNetworkInterfacePermissions",
                "ec2:UnassignPrivateIpAddresses",
                "ec2:DescribeSubnets",
                "ec2:DescribeRouteTables",
                "ec2:DescribeSecurityGroups"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "servicediscovery:GetNamespace",
            "Resource": "arn:aws:servicediscovery:*:*:namespace/*"
```

```
        },
        {
            "Effect": "Allow",
            "Action": "servicediscovery:GetService",
            "Resource": "arn:aws:servicediscovery:*:*:service/*"
        }
    ]
}
```

您必須設定許可，IAM 實體 (如使用者、群組或角色) 才可建立、編輯或刪除服務連結角色。如需詳細資訊，請參閱《IAM 使用者指南》中的服務連結角色許可。

## 建立 API Gateway 的服務連結角色

您不需要手動建立服務連結角色，當您在 、 或 AWS API 中建立 API AWS 管理主控台、自訂網域名稱 AWS CLI或 VPC 連結時，API Gateway 會為您建立服務連結角色。

若您刪除此服務連結角色，之後需要再次建立，您可以在帳戶中使用相同程序重新建立角色。當您建立 API、自訂網域名稱或 VPC 連結時，API Gateway 會再次為您建立服務連結角色。

## 編輯 API Gateway 的服務連結角色

API Gateway 不允許您編輯 AWSServiceRoleForAPIGateway 服務連結角色。因為可能有各種實體會參考服務連結角色，所以您無法在建立角色之後變更其名稱。然而，您可使用 IAM 來編輯角色描述。如需詳細資訊，請參閱《IAM 使用者指南》中的編輯服務連結角色。

## 刪除 API Gateway 的服務連結角色

若您不再使用需要服務連結角色的功能或服務，我們建議您刪除該角色。如此一來，您就沒有未主動監控或維護的未使用實體。然而，在手動刪除服務連結角色之前，您必須先清除資源。

> **Note**
>
> 如果 API Gateway 服務在您試圖刪除資源時正在使用該角色，刪除可能會失敗。若此情況發生，請等待數分鐘後並再次嘗試操作。

刪除 AWSServiceRoleForAPIGateway 所使用的 API Gateway 資源

1. 在以下網址開啟 API Gateway 主控台：https://console.aws.amazon.com/apigateway/。

2. 瀏覽至使用服務連結角色的 API、自訂網域名稱或 VPC 連結。
3. 使用主控台刪除資源。
4. 重複此程序，刪除使用服務連結角色的所有 API、自訂網域名稱或 VPC 連結。

使用 IAM 手動刪除服務連結角色

使用 IAM 主控台 AWS CLI、 或 AWS API 來刪除 AWSServiceRoleForAPIGateway 服務連結角色。如需詳細資訊，請參閱 IAM 使用者指南中的刪除服務連結角色。

### API Gateway 服務連結角色的支援區域

API Gateway 在所有提供服務的區域中支援使用服務連結的角色。如需詳細資訊，請參閱 AWS 服務端點。

### AWS 受管政策的 API Gateway 更新

檢視自此服務開始追蹤這些變更以來，API Gateway AWS 受管政策更新的詳細資訊。如需有關此頁面變更的自動提醒，請訂閱 API Gateway 文件歷程記錄頁面上的 RSS 摘要。

| 變更 | 描述 | 日期 |
| --- | --- | --- |
| 已新增 `acm:GetCertificate` 支援到 `AWSServiceRoleForAPIGateway` 政策。 | 所以此 `AWSServiceRoleForAPIGateway` 政策現在包含呼叫 ACM `GetCertificate` API 動作的許可。 | 2021 年 7 月 12 日 |
| API Gateway 開始追蹤變更 | API Gateway 開始追蹤其 AWS 受管政策的變更。 | 2021 年 7 月 12 日 |

# 在 Amazon API Gateway 中進行記錄和監控

監控是維護 API Gateway 及您 AWS 解決方案之可靠性、可用性和效能的重要部分。您應該從 AWS 解決方案的所有部分收集監控資料，以便在發生多點失敗時，可更輕鬆地偵錯。AWS 提供多種工具，讓您可以監控 API Gateway 資源及回應潛在的事件：

Amazon CloudWatch Logs

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 CloudWatch Logs 來記錄 API 呼叫。如需更多詳細資訊，請參閱 the section called “CloudWatch 日誌”。

Amazon CloudWatch 警示

您可以使用 CloudWatch 警示觀察單一指標一段指定的時間。如果指標超過指定的閾值，則會傳送一則通知至 Amazon Simple Notification Service 主題或 AWS Auto Scaling 政策。當指標處於特定狀態時，CloudWatch 警示不會叫用動作。必須是狀態已變更並維持了所指定的時間長度，才會呼叫動作。如需更多詳細資訊，請參閱 the section called “CloudWatch 指標”。

Firehose 的存取記錄功能

為了協助偵錯 API 用戶端存取相關問題，您可以啟用 Firehose 以記錄 API 呼叫。如需更多詳細資訊，請參閱 the section called “Firehose”。

AWS CloudTrail

CloudTrail 會提供由使用者、角色或 AWS 服務在 API Gateway 中採取動作的紀錄。您可以利用 CloudTrail 所收集的資訊來判斷向 API Gateway 發出的請求，以及發出請求的 IP 地址、人員、時間和其他詳細資訊。如需更多詳細資訊，請參閱 the section called “使用 CloudTrail”。

AWS X-Ray

X-Ray 是一項 AWS 服務，可收集您應用程式提供請求的資料，並使用此資料建構服務地圖，藉以辨識應用程式的問題與最佳化的機會。如需更多詳細資訊，請參閱 the section called “設定 AWS X-Ray”。

AWS Config

AWS Config 提供您帳戶中 AWS 資源組態的詳細檢視。您可以了解資源如相關聯，可以取得組態變更歷程記錄，並且了解關係和組態隨時間產生的變化。您可以使用 AWS Config 來定義評估資料合規性之資源組態的規則。AWS Config 規則代表您 API Gateway 資源的理想組態設定。如果資源違反規則並標示為不合規，AWS Config 可以使用 Amazon Simple Notification Service (Amazon SNS) 主題來提醒您。如需詳細資訊，請參閱the section called “使用 AWS Config”。

## 使用 記錄 Amazon API Gateway API 呼叫 AWS CloudTrail

與 AWS CloudTrail 整合的 Amazon API Gateway 是一項服務，可提供由使用者、角色或 AWS 服務所採取之動作的記錄。CloudTrail 會將 API Gateway 服務的所有 REST API 呼叫擷取為事件。擷取的呼叫包括來自 API Gateway 主控台的呼叫，以及對 API Gateway 服務 API 的程式碼呼叫。您可以利用

CloudTrail 收集的資訊來判斷向 API Gateway 發出的請求，以及發出請求的 IP 位址、時間和其他詳細資訊。

> **Note**
>
> TestInvokeAuthorizer 和 TestInvokeMethod 不會在 CloudTrail 中記錄。

每一筆事件或日誌專案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：

- 該請求是使用根使用者還是使用者憑證提出。
- 請求是否代表 IAM Identity Center 使用者提出。
- 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憑證。
- 該請求是否由另一項 AWS 服務服務提出。

當您建立帳戶 AWS 帳戶 時CloudTrail 會在 中處於作用中狀態，而且您會自動存取 CloudTrail 事件歷史記錄。CloudTrail 事件歷史記錄為 AWS 區域中過去 90 天記錄的管理事件，提供可檢視、可搜尋、可下載且不可變的記錄。如需詳細資訊，請參閱「AWS CloudTrail 使用者指南」中的使用 CloudTrail 事件歷史記錄。檢視事件歷史記錄不會產生 CloudTrail 費用。

如需 AWS 帳戶 過去 90 天內持續記錄的事件，請建立線索或 CloudTrail Lake 事件資料存放區。

CloudTrail 追蹤

線索能讓 CloudTrail 將日誌檔案交付至 Amazon S3 儲存貯體。使用 建立的所有線索 AWS 管理主控台 都是多區域。您可以使用 AWS CLI建立單一或多區域追蹤。建議您建立多區域追蹤，因為您擷取 AWS 區域 帳戶中所有 的活動。如果您建立單一區域追蹤，您只能檢視追蹤 AWS 區域中記錄的事件。如需追蹤的詳細資訊，請參閱《AWS CloudTrail 使用者指南》中的為您的 AWS 帳戶建立追蹤和為組織建立追蹤。

您可以透過建立追蹤，免費將持續管理事件的一個複本從 CloudTrail 傳遞至您的 Amazon S3 儲存貯體，但這樣做會產生 Amazon S3 儲存費用。如需 CloudTrail 定價的詳細資訊，請參閱 AWS CloudTrail 定價。如需 Amazon S3 定價的相關資訊，請參閱 Amazon S3 定價。

CloudTrail Lake 事件資料存放區

CloudTrail Lake 讓您能夠對事件執行 SQL 型查詢。CloudTrail Lake 會將分列式 JSON 格式的現有事件轉換為 Apache ORC 格式。ORC 是一種單欄式儲存格式，針對快速擷取資料進行了最佳化。系統會將事件彙總到事件資料存放區中，事件資料存放區是事件的不可變集合，其依據為您透過套

用進階事件選取器選取的條件。套用於事件資料存放區的選取器控制哪些事件持續存在並可供您查詢。如需 CloudTrail Lake 的詳細資訊，請參閱AWS CloudTrail 《 使用者指南》中的使用 AWS CloudTrail Lake。

CloudTrail Lake 事件資料存放區和查詢會產生費用。建立事件資料存放區時，您可以選擇要用於事件資料存放區的定價選項。此定價選項將決定擷取和儲存事件的成本，以及事件資料存放區的預設和最長保留期。如需 CloudTrail 定價的詳細資訊，請參閱 AWS CloudTrail 定價。

## CloudTrail 中的 API Gateway 管理事件

管理事件提供有關在 資源上執行的管理操作的資訊 AWS 帳戶。這些也稱為控制平面操作。根據預設，CloudTrail 記錄管理事件。

Amazon API Gateway 會將所有 API Gateway 動作記錄為管理事件，但 TestInvokeAuthorizer 和 TestInvokeMethod 除外。如需 API Gateway 記錄到 CloudTrail 的 Amazon API Gateway 動作清單，請參閱 Amazon API Gateway API 參考。

## API Gateway 事件範例

一個事件代表任何來源提出的單一請求，並包含請求 API 操作的相關資訊、操作的日期和時間、請求參數等。CloudTrail 日誌檔案不是公有 API 呼叫的已排序堆疊追蹤，因此事件不會以任何特定順序顯示。

以下範例說明 API Gateway `GetResource` 動作的 CloudTrail 事件：

```
{
    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 ...
    ]
}
```

如需有關 CloudTrail 記錄內容的資訊，請參閱《AWS CloudTrail 使用者指南》中的 CloudTrail record contents。

## 使用 監控 API Gateway API 組態 AWS Config

您可以使用 AWS Config，記錄對您的 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 可以追蹤下列項目的變更：

- 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 規則 此功能可讓您定義組態規則，並自動偵測、追蹤和提醒這些規則的違規情況。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資源組態。

您可以使用 AWS Config 主控台或 在 AWS Config 帳戶中啟用 AWS CLI。選取您要追蹤變更的資源類型。如果您先前已設定 AWS Config 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。支援 中的 Amazon API Gateway AWS Config 適用於所有 AWS 公有區域和 AWS GovCloud (US)。如需受支援區域的完整清單，請參閱《 AWS 一般參考》中的 Amazon API Gateway 端點和配額。

主題

- 支援的資源類型
- 設定 AWS Config
- 設定 AWS Config 記錄 API Gateway 資源
- 在 AWS Config 主控台中檢視 API Gateway 組態詳細資訊
- 使用 AWS Config 規則評估 API Gateway 資源

## 支援的資源類型

下列 API Gateway 資源類型與 整合， AWS Config 並記錄在AWS Config 支援 AWS 的資源類型和資源關係中：

- `AWS::ApiGatewayV2::Api` (WebSocket 和 HTTP API)
- `AWS::ApiGateway::RestApi` (REST API)
- `AWS::ApiGatewayV2::Stage` (WebSocket 和 HTTP API 階段)
- `AWS::ApiGateway::Stage` (REST API 階段)

如需 的詳細資訊 AWS Config，請參閱 AWS Config 開發人員指南。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 API 以傳播變更。否則，您會在 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

若要最初設定 AWS Config，請參閱《AWS Config 開發人員指南》中的下列主題。

- AWS Config 使用主控台設定
- AWS Config 使用 設定 AWS CLI

## 設定 AWS Config 記錄 API Gateway 資源

根據預設， 會 AWS Config 記錄環境執行所在區域中探索的所有支援區域資源類型的組態變更。您可以自訂 AWS Config ，僅記錄特定資源類型的變更，或全域資源的變更。

若要了解區域資源與全域資源，並了解如何自訂您的 AWS Config 組態，請參閱選取哪些資源 AWS Config 記錄。

## 在 AWS Config 主控台中檢視 API Gateway 組態詳細資訊

您可以使用 AWS Config 主控台來尋找 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 組態主控台中尋找 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，以及所選資源的其他資訊。
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)。

若要進一步了解如何尋找資源並檢視此頁面的資訊，請參閱《 AWS Config 開發人員指南》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，其代表 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，或定義自訂規則。 AWS Config 會持續追蹤資源組態的變更，以判斷這些變更是否違反規則中的任何條件。 AWS Config 主控台會顯示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 可以使用 Amazon Simple Notification Service 開發人員指南 (Amazon SNS) 主題提醒您。如要透過程式設計方式使用這些 AWS Config 提醒中的資料，請使用 Amazon Simple Queue Service (Amazon SQS) 佇列做為 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 開發人員指南中的使用規則評估資源。

## Amazon API Gateway 的合規驗證

我們的新 AWS 註冊體驗並非針對受管制的工作負載而設計。如果您正在使用我們新的 AWS 註冊體驗，但想要將 AWS 用於受管制的工作負載，您可以註冊 AWS （進階） 或為您的 AWS 環境啟用進階功能。

若要了解 是否 AWS 服務 在特定合規計劃範圍內，請參閱AWS 服務 合規計劃範圍內然後選擇您感興趣的合規計劃。如需一般資訊，請參閱AWS 合規計劃。

您可以使用 下載第三方稽核報告 AWS Artifact。如需詳細資訊，請參閱在 中下載報告 AWS Artifact。

您使用 時的合規責任 AWS 服務 取決於資料的機密性、您公司的合規目標，以及適用的法律和法規。如需使用 時合規責任的詳細資訊 AWS 服務，請參閱 AWS 安全文件。

## Amazon API Gateway 中的復原功能

AWS 全球基礎設施是以 AWS 區域與可用區域為中心建置的。AWS區域提供多個分開且隔離的實際可用區域，並以低延遲、高輸送量和高度備援聯網功能相互連結。透過可用區域，您可以設計與操作的應

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

API Gateway 屬於全受管區域服務，會在每個區域的多個可用區域中運作，並利用可用區域的備援將基礎設施故障率以最大幅度降低成一種可用性風險類別。API Gateway 旨在自動從可用區域的故障中復原。

如需 AWS 區域與可用區域的相關資訊，請參閱 AWS 全球基礎設施。

為了防止您的 API 被太多的請求淹沒，API Gateway 會調節向您 API 提出的請求。具體而言，API Gateway 會對您帳戶中所有 API 的穩定狀態速率和請求提交爆增次數配置限制。您可以為您的 API 設定自訂調節。如需詳細資訊，請參閱 調節對 REST API 的請求，以提高 API Gateway 的輸送量。

您可以使用 Route 53 運作狀態檢查，控制從主要區域的 API Gateway API 到次要區域的 API Gateway API 的 DNS 備援。如需範例，請參閱the section called “DNS 備援”。

# Amazon API Gateway 的基礎設施安全性

Amazon API Gateway 是受管服務，受到 AWS 全球網路安全的保護。如需 AWS 安全服務以及如何 AWS 保護基礎設施的資訊，請參閱AWS 雲端安全。若要使用基礎設施安全的最佳實務來設計您的 AWS 環境，請參閱安全支柱 AWS Well-Architected Framework 中的基礎設施保護。

您可以使用 AWS 發佈的 API 呼叫，透過網路存取 API Gateway。使用者端必須支援下列專案：

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

您可從任何網路位置呼叫這些 API 操作，而 API Gateway 確實可支援資源型存取政策，以供納入依來源 IP 地址為主的限制。您也可以使用資源型政策來控制從特定 Amazon Virtual Private Cloud (Amazon VPC) 的端點或特定 VPC 的存取。實際上，這只會隔離網路中特定 VPC 對指定 API Gateway 資源 AWS 的網路存取。

# Amazon API Gateway 的漏洞分析

組態和 IT 控制是 AWS 與身為我們客戶的您共同的責任。如需詳細資訊，請參閱 AWS 共同的責任模型。

# Amazon API Gateway 的安全最佳實務

在您開發和實作自己的安全政策時，可考慮使用 API Gateway 提供的多種安全功能。以下最佳實務為一般準則，並不代表完整的安全解決方案。這些最佳實務可能不適用或無法滿足您的環境需求，因此請將其視為實用建議就好，而不要當作是指示。

實作最低權限存取

使用 IAM 政策來實作建立、讀取、更新或刪除 API Gateway API 的最低權限存取。如需進一步了解，請參閱適用於 Amazon API Gateway 的 Identity and Access Management。API Gateway 提供數個選項來控制對您所建立 API 的存取。如需進一步了解，請參閱在 API Gateway 中控制和管理對 REST API 的存取、在 API Gateway 中控制和管理對 WebSocket API 的存取和使用 API Gateway 中的 JWT 授權方來控制對 HTTP API 的存取。

實作記錄

使用 CloudWatch Logs 或 Amazon Data Firehose 將請求記錄到您的 API。如需進一步了解，請參閱在 API Gateway 中監控 REST API、在 API Gateway 中設定 WebSocket API 的記錄和在 API Gateway 中設定 HTTP API 的日誌。

實作 Amazon CloudWatch 警示

您可以使用 CloudWatch 警示觀察單一指標一段指定的時間。如果指標超過指定的閾值，則會將通知傳送至 Amazon Simple Notification Service 主題或 AWS Auto Scaling 政策。當指標處於特定狀態時，CloudWatch 警示不會叫用動作。必須是狀態已變更並維持了所指定的時間長度，才會呼叫動作。如需詳細資訊，請參閱the section called “CloudWatch 指標”。

啟用 AWS CloudTrail

CloudTrail 提供由使用者、角色或 API Gateway 中的 AWS 服務所採取之動作的記錄。您可以利用 CloudTrail 所收集的資訊來判斷向 API Gateway 發出的請求，以及發出請求的 IP 地址、人員、時間和其他詳細資訊。如需詳細資訊，請參閱the section called “使用 CloudTrail”。

啟用 AWS Config

AWS Config 提供帳戶中 AWS 資源組態的詳細檢視。您可以了解資源如相關聯，可以取得組態變更歷程記錄，並且了解關係和組態隨時間產生的變化。您可以使用 AWS Config 來定義評估資源組態以進行資料合規的規則。 AWS Config 規則代表 API Gateway 資源的理想組態設定。如果資源違反規則並標記為不合規， AWS Config 可以使用 Amazon Simple Notification Service (Amazon SNS) 主題提醒您。如需詳細資訊，請參閱the section called “使用 AWS Config”。

## 使用 AWS Security Hub CSPM

透過使用 AWS Security Hub CSPM 監視您 API Gateway 的使用狀況，因為它關係到安全最佳實務。Security Hub CSPM 使用安全控制來評估資源組態和安全標準，以協助您符合各種合規架構。如需使用 Security Hub CSPM 評估 API Gateway 資源的詳細資訊，請參閱AWS Security Hub《使用者指南》中的 Amazon API Gateway 控制項。

# API Gateway 資源的標記

標籤是您指派或 AWS 指派給 AWS 資源的中繼資料標籤。每個 標籤都有兩個部分：

- 標籤鍵 (例如，`CostCenter`、`Environment` 或 `Project`)。標籤鍵會區分大小寫。
- 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。您可以在 AWS 帳單與成本管理 儀表板上啟用這些標籤。 AWS 會使用標籤來分類您的成本，並傳送每月成本分配報告給您。如需詳細資訊，請參閱《AWS Billing 使用者指南》中的使用成本分配標籤。
- 識別和組織您的 AWS 資源。許多 AWS 服務支援標記，因此您可以將相同的標籤指派給來自不同 服務的資源，以指出資源相關。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。

如需有關使用標籤的秘訣，請參閱白皮書 AWS 標記策略。

下列各節會提供 Amazon API Gateway 標籤的詳細資訊。

主題

- 可標記的 API Gateway 資源
- 使用標籤來控制對 API Gateway REST API 資源的存取

## 可標記的 API Gateway 資源

您可以在 Amazon API Gateway V2 API 中為下列 HTTP API 或 WebSocket API 資源設定標籤：

- `Api`
- `DomainName`
- `Stage`

- VpcLink

此外，您還可以在 Amazon API Gateway V1 API 中為下列 REST API 資源設定標籤：

- ApiKey
- ClientCertificate
- DomainName
- DomainNameAccessAssociation
- RestApi
- Stage
- UsagePlan
- VpcLink

您無法直接在其他資源上設定標籤。不過，在 Amazon API Gateway V1 API 中，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，屬性型存取控制 的該 RestApi 之下列子資源會繼承該標籤：
  - Authorizer
  - Deployment
  - Documentation
  - GatewayResponse
  - Integration
  - Method
  - Model
  - Resource
  - ResourcePolicy
  - Setting
  - Stage
- 如果 DomainName 上已設定標籤，則該標籤會由其下的任何 BasePathMapping、ApiMapping 和 RoutingRule 資源繼承。
- 如果在 UsagePlan 上設定標籤，它之下的任何 UsagePlanKey 資源會繼承該標籤。

> **Note**
>
> 標籤繼承僅適用於屬性型存取控制。例如，您無法使用繼承的標籤來監控其中的成本 AWS Cost Explorer。若要使用標籤進行成本分配，建議您在 Stage 資源等子資源上建立標籤。當您呼叫資源的 GetTags 時，API Gateway 不會傳回繼承的標籤。

## Amazon API Gateway V1 API 中的標籤繼承

過去，您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

搭配 API Gateway 資源使用標籤時，適用以下限制和使用慣例：

- 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。
- 索引鍵和值允許的字元包括可用 UTF-8 表示的英文字母、數字、空格，還有以下特殊字元：. : + = @ _ / - (連字號)。Amazon EC2 資源允許任何字元。
- 標籤金鑰與值皆區分大小寫。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金鑰或值。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 REST API 資源的存取

AWS Identity and Access Management 政策中的條件是您用來指定 API Gateway 資源許可的語法的一部分。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”。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鍵。

使用屬性型存取控制的標籤，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。

## 根據資源標籤限制動作

以下範例政策會授與使用者在所有資源上執行所有動作的許可，只要這些資源不具有值為 prod 的標籤 Environment。

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "apigateway:*",
      "Resource": "*"
    },
    {
      "Effect": "Deny",
      "Action": [
        "apigateway:*"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/Environment": "prod"
        }
      }
    }
  ]
}
```

## 根據資源標籤允許動作

以下範例政策可讓使用者在 API 閘道資源上執行所有動作，只要這些資源具有值為 Development 的標籤 Environment。Deny 陳述式可防止使用者變更 Environment 標籤的值。

JSON

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

```
    {
      "Sid": "ConditionallyAllow",
      "Effect": "Allow",
      "Action": [
        "apigateway:*"
      ],
      "Resource": [
        "arn:aws:apigateway:*::*"
      ],
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/Environment": "Development"
        }
      }
    },
    {
      "Sid": "AllowTagging",
      "Effect": "Allow",
      "Action": [
        "apigateway:*"
      ],
      "Resource": [
        "arn:aws:apigateway:*::/tags/*"
      ]
    },
    {
      "Sid": "DenyChangingTag",
      "Effect": "Deny",
      "Action": [
        "apigateway:*"
      ],
      "Resource": [
        "arn:aws:apigateway:*::/tags/*"
      ],
      "Condition": {
        "ForAnyValue:StringEquals": {
          "aws:TagKeys": "Environment"
        }
      }
    }
  ]
}
```

## 拒絕標記操作

以下範例政策可讓使用者執行所有 API Gateway 的動作，變更標籤除外。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "apigateway:*"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Effect": "Deny",
            "Action": [
                "apigateway:*"
            ],
            "Resource": "arn:aws:apigateway:*::/tags*"
        }
    ]
}
```

## 允許標記操作

下列範例政策可讓使用者取得所有 API Gateway 的資源，並且變更這些資源的標籤。若要取得資源的標籤，使用者必須具有該資源的 GET 許可。若要更新資源的標籤，使用者必須具有該資源的 PATCH 許可。

JSON

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

```
            "Effect": "Allow",
            "Action": [
                "apigateway:GET",
                "apigateway:PUT",
                "apigateway:POST",
                "apigateway:DELETE"
            ],
            "Resource": [
                "arn:aws:apigateway:*::/tags/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "apigateway:GET",
                "apigateway:PATCH"
            ],
            "Resource": [
                "arn:aws:apigateway:*::*"
            ]
        }
    ]
}
```

# 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，API Gateway APIs可在 AWS SDKs中使用。

如果您使用的是 AWS SDK 存在的語言，您可能偏好使用 SDK，而不是直接使用 API Gateway REST APIs。軟體開發套件可簡化身分驗證、與您的開發環境輕鬆整合，並可輕鬆存取 API Gateway 命令。

以下是可找到 AWS SDKs和 API Gateway REST API 參考文件的位置：

- 適用於 Amazon Web Services 的工具
- Amazon API Gateway REST API 參考
- Amazon API Gateway WebSocket 和 HTTP API 參考

## API Gateway 服務端點

端點是做為 AWS Web 服務進入點的 URL。API Gateway 支援下列端點類型：

- the section called “IPv4 端點”
- 雙堆疊端點 (同時支援 IPv4 和 IPv6)
- FIPS 端點

當您提出請求時，您可以指定要使用的端點。如果您沒有指定端點，則預設使用 IPv4 端點。若要使用不同的端點類型，您必須在請求中將其指定。如需如何執行此作業的範例，請參閱 the section called “指定端點”。如需可用端點的列表，請參閱 Amazon API Gateway 端點。

### IPv4 端點

IPv4 端點僅支援 IPv4 流量。IPv4 端點適用於所有區域。

如果您指定一般端點、`apigateway.amazonaws.com`，我們會使用 `us-east-1` 的端點。若要使用不同的區域，請指定其相關聯的端點。例如，如果您指定 `apigateway.us-east-2.amazonaws.com` 做為端點，我們會將您的請求導向 `us-east-2` 端點。

IPv4 端點名稱使用以下命名慣例：

- `apigateway.`*`region`*`.amazonaws.com`

例如，eu-west-1 區域的 IPv4 端點名稱是 apigateway.eu-west-1.amazonaws.com。

## 雙堆疊 (IPv4 和 IPv6) 端點

雙堆疊端點同時支援 IPv4 和 IPv6 流量。當您請求雙堆疊端點時，端點 URL 會解析為 IPv6 或 IPv4 位址，這點取決於您的網路和用戶端使用的通訊協定。

雙堆疊端點名稱使用以下命名慣例：

- apigateway.*region*.api.aws

例如，eu-west-1 區域的雙堆疊端點名稱是 apigateway.eu-west-1.api.aws。

## 指定端點

下列範例示範如何使用適用於 的 AWS CLI 指定 us-east-2區域的端點apigateway。

- **雙堆疊**

```
aws apigateway get-rest-apis --region us-east-2 --endpoint-url https://apigateway.us-east-2.api.aws
```

- IPv4

```
aws apigateway get-rest-apis --region us-east-2 --endpoint-url https://apigateway.us-east-2.amazonaws.com
```

下列範例示範如何使用適用於 的 AWS CLI 指定 us-east-2區域的端點apigatewayv2。

- **雙堆疊**

```
aws apigatewayv2 get-apis --region us-east-2 --endpoint-url https://apigateway.us-east-2.api.aws
```

- IPv4

```
aws apigatewayv2 get-apis --region us-east-2 --endpoint-url https://apigateway.us-east-2.amazonaws.com
```

# 在 IAM 原則中使用 IPv6 地址

如果您使用任何 IAM 使用者政策或 API Gateway 資源政策來控制 API Gateway 或任何 API Gateway API 的存取權，請確認您的政策已更新並包含 IPv6 位址範圍。若政策未更新，因而無法處理 IPv6 位址，則可能在用戶端開始使用雙堆疊端點時，影響用戶端對 API Gateway 的存取。如需使用 IAM 管理存取許可的詳細資訊，請參閱 適用於 Amazon API Gateway 的 Identity and Access Management

篩選 IP 地址的 IAM 原則使用IP 地址條件運算子。下列身分政策允許 54.240.143.* 範圍內的 IP 位址取得識別碼為 a123456789 的所有 HTTP 或 WebSocket API 資源的相關資訊。超出此範圍的任何 IP 位址都將遭拒，而無法存取 API 中所有資源的相關資訊。因為所有 IPv6 位址都在允許的範圍外，所以此政策會阻止 IPv6 位址存取有關 API 的資訊。

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "IPAllow",
      "Effect": "Allow",
      "Principal": "*",
      "Action": "apigateway:GET",
      "Resource": "arn:aws:apigateway:us-east-1::/apis/a123456789/*",
      "Condition": {
         "IpAddress": {"aws:SourceIp": "54.240.143.0/24"}
      }
    }
  ]
}
```

您可以修改 API 政策的 Condition 元素，以允許 IPv4 (54.240.143.0/24) 與 IPv6 (2001:DB8:1234:5678::/64) 位址範圍，如以下範例所示。您可以使用範例中所示的相同類型 Condition 區塊，來更新您的 IAM 使用者政策和 API Gateway 資源政策。

```
    "Condition": {
      "IpAddress": {
         "aws:SourceIp": [
           "54.240.143.0/24",
            "2001:DB8:1234:5678::/64"
```

```
            ]
        }
    }
```

# Amazon API Gateway 配額

下列配額適用於所有 Amazon API Gateway API 類型。

## 每個區域的 API Gateway 帳戶層級配額

以下配額會根據 Amazon API Gateway 中的每個區域套用至每個區域。

| 資源或操作 | 預設配額 | 可以提高 |
| --- | --- | --- |
| HTTP API、REST API、WebSocket API 和 WebSocket Callback API 間每個帳戶、每個區域的調節配額。 | 每秒 10,000 個請求 (RPS)，加上字符儲存貯體演算法提供的額外高載容量，最多使用 5,000 個請求的儲存貯體容量。<br><br>Note<br>該高載配額是由 API Gateway 服務團隊根據各區域中該帳戶整體 RPS 配額而決定。客戶無法針對該配額進行控制或請求變更。 | 是 |
| 入口網站每個區域每個帳戶無存取控制的調節配額 | 每秒 250，000 個請求 | 否 |
| 入口網站每個區域每個帳戶的存取控制調節配額 | 每秒 10，000 個請求 | 否 |

* 對於下列區域，預設限流配額為 2500 RPS，預設暴增配額為 1250 RPS：非洲 (開普敦)、歐洲 (米蘭)、亞太區域 (雅加達)、中東 (阿拉伯聯合大公國)、亞太區域 (海德拉巴)、亞太區域 (墨爾本)、歐洲 (西班牙)、歐洲 (蘇黎世)、以色列 (特拉維夫)、加拿大西部 (卡加利)、亞太區域 (馬來西亞)、亞太區域 (泰國) 和墨西哥 (中部)。

# 建立、部署和管理 API 的 API Gateway 配額

下列固定配額適用於使用 、API Gateway 主控台或 API Gateway REST API 及其 SDKs 在 API Gateway 中建立 AWS CLI、部署和管理 API。這些配額無法增加。

| 動作 | 預設配額 | 可以提高 |
| --- | --- | --- |
| CreateApiKey | 每個帳戶每秒 5 個請求 | 否 |
| CreateDeployment | 每個帳戶每 5 秒 1 個請求。 | 否 |
| CreateDocumentationVersion | 每個帳戶每 20 秒 1 個請求。 | 否 |
| CreateDomainName | 每個帳戶每 30 秒 1 個請求。 | 否 |
| CreateResource | 每個帳戶每秒 5 個請求 | 否 |
| 區域或私有 API 的 CreateRestApi | 每個帳戶每 3 秒 1 個請求。 | 否 |
| 邊緣最佳化 API 的 CreateRestApi | 每個帳戶每 30 秒 1 個請求。 | 否 |
| CreateVpcLink (V2) | 每個帳戶每 15 秒 1 個請求。 | 否 |
| DeleteApiKey | 每個帳戶每秒 5 個請求 | 否 |
| DeleteDomainName | 每個帳戶每 30 秒 1 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 |
| DeleteVpcLink (V2) | 每個帳戶每 30 秒 1 個請求。 | 否 |
| ImportDocumentationParts | 每個帳戶每 30 秒 1 個請求。 | 否 |
| 區域或私有 API 的 ImportRestApi | 每個帳戶每 3 秒 1 個請求。 | 否 |

| 動作 | 預設配額 | 可以提高 |
| --- | --- | --- |
| 邊緣最佳化 API 的 ImportRestApi | 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 |
| UpdateDomainName | 每個帳戶每 30 秒 1 個請求。 | 否 |
| UpdateUsagePlan | 每個帳戶每 20 秒 1 個請求。 | 否 |
| 建立入口網站 | 每 3 秒 1 個請求 | 否 |
| 更新入口網站 | 每分鐘 2 個請求 | 否 |
| 取得入口網站 | 每秒 10 個請求 | 否 |
| 列出入口網站 | 每秒 10 個請求 | 否 |
| 發佈入口網站 | 每分鐘 2 個請求 | 否 |
| DeletePortal | 每分鐘 2 個請求 | 否 |
| PreviewPortal | 每 3 秒 1 個請求 | 否 |
| DisablePortal | 每分鐘 2 個請求 | 否 |
| GetPortalProduct | 每秒 10 個請求 | 否 |
| ListPortalProduct | 每秒 5 個請求 | 否 |
| CreatePortalProduct | 每秒 2 個請求 | 否 |
| UpdatePortalProduct | 每秒 0.5 個請求 | 否 |
| DeletePortalProduct | 每秒 1 個請求 | 否 |
| PutPortalProductSharingPolicy | 每 3 秒 1 個請求 | 否 |

| 動作 | 預設配額 | 可以提高 |
| --- | --- | --- |
| GetPortalProductSharingPolicy | 每秒 10 個請求 | 否 |
| DeletePortalProductSharingPolicy | 每 3 秒 1 個請求 | 否 |
| CreateProductRestEndpointPage | 每 3 秒 1 個請求 | 否 |
| GetProductRestEndpointPage | 每秒 10 個請求 | 否 |
| UpdateProductRestEndpointPage | 每 3 秒 1 個請求 | 否 |
| DeleteProductRestEndpointPage | 每 3 秒 1 個請求 | 否 |
| ListProductRestEndpointPages | 每秒 10 個請求 | 否 |
| CreateProductPage | 每 3 秒 1 個請求 | 否 |
| GetProductPage | 每秒 10 個請求 | 否 |
| UpdateProductPage | 每 3 秒 1 個請求 | 否 |
| DeleteProductPage | 每 3 秒 1 個請求 | 否 |
| ListProductPages | 每秒 10 個請求 | 否 |
| 其他操作 | 配額不可超過帳戶配額總計。 | 否 |
| 操作總計 | 每秒 10 個請求 (rps)，爆量配額為每秒 40 個請求。 | 否 |

# 在 API Gateway 中設定和執行 REST API 的配額

下列配額適用於在 Amazon API Gateway 中設定和執行 REST API。如果配額是依每個 API 計算，則只能在特定 API 上增加配額，無法對帳戶中的所有 API 增加配額。

如需帳戶層級配額的相關資訊，請參閱 配額和重要說明。

| 資源或操作 | 預設配額 | 可以提高 |
| --- | --- | --- |
| 每個區域每個帳戶的公有自訂網域名稱 | 120 | 是 |
| 每個區域每個帳戶的私有自訂網域名稱 | 50 | 是 |
| 每個區域的區域 API | 600 | 否 |
| 每個區域的邊緣最佳化 API | 120 | 否 |
| 每個區域每個帳戶的私有 API | 600 | 否 |
| 每個帳戶的網域名稱存取關聯 | 100 | 是 |
| 每個網域的多層 API 映射 | 200 | 否 |
| 每個網域的路由規則 | 50 | 是 |
| 邊緣最佳化 API 的 URL 長度 (以字元為單位) | 8192 | 否 |

| 資源或操作 | 預設配額 | 可以提高 |
| --- | --- | --- |
| 區域 API URL 的長度，以字元為單位 | 10240 | 否 |
| 私有 API URL 的長度，以字元為單位 | 8192 | 否 |
| API Gateway 資源政策的長度 (以字元為單位) | 8192 | 是 |
| 每個區域每個帳戶的 API 金鑰 | 10000 | 否 |
| 每個區域每個帳戶的 API Gateway 產生的用戶端憑證 | 60 | 是 |
| 每個區域每個帳戶的 ACM 用戶端憑證 | 500 | 否 |
| 每個 API (AWS Lambda 和 Amazon Cognito) 的授權方 | 10 | 是<br><br>若要增加此配額，請連絡 AWS 支援中心 |

| 資源或操作 | 預設配額 | 可以提高 |
| --- | --- | --- |
| 每個 API 的文件部分 | 2000 | 是<br><br>若要增加此配額，請連絡 AWS 支援中心 |
| 每個 API 資源 | 300 | 是 |
| 每個 API 階段 | 10 | 是 |
| 每個階段的階段變數 | 100 | 否 |
| 階段變數中的金鑰長度 (以字元為單位) | 64 | 否 |
| 階段變數中的數值長度 (以字元為單位) | 512 | 否 |
| 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) |

| 資源或操作 | 預設配額 | 可以提高 |
| --- | --- | --- |
| 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 |
| 區域 API 的整合逾時 | 50 毫秒 - 所有整合類型的 29 秒，包括 Lambda、Lambda 代理、HTTP、HTTP 代理和 AWS 整合。 | 是 * |
| 邊緣最佳化 API 的整合逾時 | 50 毫秒 - 所有整合類型的 29 秒，包括 Lambda、Lambda 代理、HTTP、HTTP 代理和 AWS 整合。 | 否 |
| 私有 API 的整合逾時 | 50 毫秒 - 所有整合類型的 29 秒，包括 Lambda、Lambda 代理、HTTP、HTTP 代理和 AWS 整合。 | 是 * |
| 所有標頭值的總合併大小，包括標頭名稱、值、任何換行符號和空格 | 20480 位元組 | 否 |
| 一個私有 API 所有標頭值的總合併大小 | 8000 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 `#foreach ... #end` 迴圈的反覆運算數目 | 1000 | 否 |
| 映射範本大小上限 | 300 KB | 否 |
| 存取日誌範本大小 | 3 KB | 否 |

| 資源或操作 | 預設配額 | 可以提高 |
| --- | --- | --- |
| 方法 ARN 長度 | 1600 個位元組。如果您的方法包含路徑參數，且用戶端使用的值超過 ARN 長度，則您的 API 將傳回 `414 Request URI too long` 回應。 | 否 |
| 用量計劃中單一階段的方法層級調節設定 | 20 | 是<br><br>若要增加此配額，請連絡 AWS 支援中心 |
| 每個 API 的資料大小 | 400 KB | 否 |
| 信任庫中的憑證數量 | 1,000 個憑證，總物件大小上限為 1 MB。 | 否 |
| 閒置連線逾時 | 310 秒 | 否 |
| 使用 restapi:import 或 restapi:put 時的 API 定義檔案大小上限 | 6 MB | 否 |

* 您無法將整合逾時設定為小於 50 毫秒。您可以將整合逾時提高到 29 秒以上，但這樣做可能需要減少您帳戶的區域層級限流配額。

## 減少配額的最佳實務

以下最佳實務可能有助於減少您目前的資源數量，以避免增加您的配額。請確定這些建議適用於您的 API 架構。

每個區域的 API

若要減少每個區域的 API 數量，請匯出任何未使用的 API，然後從 API Gateway 將其刪除。如需詳細資訊，請參閱the section called “匯出 REST API”。

每個 API 階段

若要減少每個 API 的階段數量，請將您的 API 拆分為多個 API。

每個 API 資源

使用 `{proxy+}` 路徑來減少資源數量。如需詳細資訊，請參閱the section called “設定代理整合與代理資源”。

API 映射

若要減少自訂網域名稱的 API 映射數量，請使用具有單一層級的 API 映射，例如 `/prod`。具有單一層級的 API 映射不會計入 API 映射配額。

每個 API 的授權方

若要減少每個 API 的授權方數量，請對不同的 API 方法重複使用授權方。

每個 API 的文件部分

若要減少每個 API 的文件組件數量，請在匯入 API 時使用 `ignore=documentation`。如需詳細資訊，請參閱the section called “匯入 API 文件”。

您也可以使用內容繼承，允許某些文件組件從規格更普通的 API 實體繼承內容。如需詳細資訊，請參閱the section called “API Gateway 中的 API 文件表示”。

API Gateway 資源政策的長度 (以字元為單位)

若要減少資源政策的長度，請使用 AWS WAFV2 來保護您的 API。如需詳細資訊，請參閱the section called “AWS WAF”。

如果您的政策包含 IP 位址，您也可以使用範圍而非特定值來定義 IP 位址。

每個 API 金鑰的用量計畫

若要減少每個 API 金鑰的用量計畫數量，則讓每個用量計畫使用一個 API 金鑰，並將您的用量計畫與多個 API 建立關聯。我們不建議在多個用量計畫之間共用一個 API 金鑰。

# 在 API Gateway 中設定和執行 HTTP API 的配額

下列配額適用於在 Amazon API Gateway 中設定和執行 HTTP API。如果配額是依每個 API 計算，則只能在特定 API 上增加配額，無法對帳戶中的所有 API 增加配額。如需帳戶層級配額的相關資訊，請參閱 配額和重要說明

| 資源或操作 | 預設配額 | 可以提高 |
| --- | --- | --- |
| 每 API 的路由 | 300 | 是 |
| 每 API 的整合 | 300 | 否 |
| 最大整合逾時 | 30 秒 | 否 |
| 每個 API 階段 | 10 | 是 |
| 每個網域的多層 API 映射 | 200 個 API 映射。此限制不包括具有單一層級的 API 映射，例如 `/prod`。 | 否 |
| 每個階段的標籤 | 50 | 否 |
| 請求行與標頭值的總大小 | 10240 個位元組 | 否 |
| 承載大小 | 10 MB | 否 |
| 每個區域每個帳戶的自訂網域數 | 120 | 是 |
| 存取日誌範本大小 | 3 KB | 否 |
| Amazon CloudWatch Logs 日誌項目 | 1 MB | 否 |
| 每個 API 的授權方 | 10 | 是<br><br>若要增加此配 |

| 資源或操作 | 預設配額 | 可以提高 |
| --- | --- | --- |
| | | 額，請連絡 AWS 支援中心 |
| 每個授權方的對象 | 50 | 否 |
| 每個路由的範圍 | 10 | 否 |
| JSON Web Key Set 端點的逾時 | 1500 毫秒 | 否 |
| 來自 JSON Web 密鑰集端點的回應大小 | 150000 個位元組 | 否 |
| OpenID Connect 探索端點的逾時 | 1500 毫秒 | 否 |
| Lambda 授權方回應逾時 | 10000 毫秒 | 否 |
| 每個區域每個帳戶的 VPC 連結 | 10 | 是 |
| 每個 VPC 連結的子網路數 | 10 | 是 |
| 每個階段的階段變數 | 100 | 否 |
| 階段變數中的金鑰長度 (以字元為單位) | 64 | 否 |

| 資源或操作 | 預設配額 | 可以提高 |
| --- | --- | --- |
| 階段變數中的數值長度 (以字元為單位) | 512 | 否 |

## 減少配額的最佳實務

以下最佳實務可能有助於減少您目前的資源數量，以避免增加您的配額。請確定這些建議適用於您的 API 架構。

每個區域的 API

若要減少每個區域的 API 數量，請匯出任何未使用的 API，然後從 API Gateway 將其刪除。如需詳細資訊，請參閱the section called “Export (匯出)”。

每個 API 階段

若要減少每個 API 的階段數量，請將您的 API 拆分為多個 API。

每個 API 的授權方

若要減少每個 API 的授權方數量，請對不同的 API 方法重複使用授權方。

# 在 API Gateway 中設定入口網站的配額

下列配額適用於在 API Gateway 中建立入口網站。如需詳細資訊，請參閱API Gateway 入口網站。

| 資源或操作 | 預設配額 | 可以提高 |
| --- | --- | --- |
| 每個帳戶的入口網站 | 15 | 是 |
| 每個入口網站的入口網站產品 | 200 | 否 |

| 資源或操作 | 預設配額 | 可以提高 |
| --- | --- | --- |
| 每個帳戶的入口網站產品 | 500 | 否 |
| 每個入口網站產品的產品 REST 端點頁面 | 40 | 是 |
| 每個入口網站產品的產品頁面 | 40 | 是 |
| 標誌大小 | 200 KB | 否 |
| 每個產品 REST 端點頁面的文件頁面大小 | 32，000 個字元 | 否 |
| 產品頁面的自訂頁面大小 | 32，000 個字元 | 否 |
| 每個入口網站的自訂網域名稱 | 1 | 否 |
| 每個入口網站的授權方 | 1 | 否 |

# 在 API Gateway 中設定和執行 WebSocket 的配額

下列配額適用於在 Amazon API Gateway 中設定和執行 WebSocket API。如果配額是依每個 API 計算，則只能在特定 API 上增加配額，無法對帳戶中的所有 API 增加配額。如需帳戶層級配額的相關資訊，請參閱 配額和重要說明

| 資源或操作 | 預設配額 | 可以提高 |
| --- | --- | --- |
| 每個帳戶 (所有 WebSocket API) 在每個區域每秒的新連線 | 500 | 是 |
| 並行連線 | 不適用 * | 不適用 |
| AWS Lambda 每個 API 的授權方 | 10 | 是<br><br>若要增加此配額，請連絡 AWS 支援中心 |
| AWS Lambda 授權方結果大小 | 8 KB | 否 |
| 每 API 的路由 | 300 | 是 |
| 每 API 的整合 | 300 | 是<br><br>若要增加此配額，請連絡 AWS 支援中心 |
| 整合逾時 | 50 毫秒 - 所有整合類型的 29 秒，包括 Lambda、Lambda 代理、HTTP、HTTP 代理和 AWS 整合。 | 否 |
| 每個 API 階段 | 10 | 是 |

| 資源或操作 | 預設配額 | 可以提高 |
| --- | --- | --- |
| WebSocket 框架大小 | 32 KB | 否 |
| 訊息承載大小 | 128 KB ** | 否 |
| 映射範本大小上限 | 300 KB | 否 |
| WebSocket API 連線持續時間 | 2 小時 | 否 |
| 閒置連線逾時 | 10 分鐘 | 否 |
| WebSocket API 的 URL 長度 (以字元為單位) | 4096 | 否 |
| 存取日誌範本大小 | 3 KB | 否 |

* API Gateway 不會在並行連線上強制執行配額。並行連線的最大數目取決於新連線每秒速率，以及兩小時期間的連線數目上限。例如，預設配額為每秒 500 個新連線，如果用戶端在兩小時內以最大速率連線，則 API Gateway 最多可提供 3,600,000 個並行連線。

** 由於 WebSocket 框架大小配額為 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。這適用於 `@connections` 命令。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。

# Amazon API Gateway 重要說明

下一節詳細說明可能會影響您使用 API Gateway 的備註。

主題

- 適用於 HTTP API 的 Amazon API Gateway 重要說明
- 適用於 HTTP 和 WebSocket API 的 Amazon API Gateway 重要說明
- 適用於 REST 和 WebSocket API 的 Amazon API Gateway 重要說明
- 適用於 WebSocket API 的 Amazon API Gateway 重要說明

- 適用於 REST API 的 Amazon API Gateway 重要說明

## 適用於 HTTP API 的 Amazon API Gateway 重要說明

- HTTP API 會將傳入 `X-Forwarded-*` 標頭轉譯為標準 `Forwarded` 標頭，並附加傳出 IP、主機和通訊協定。
- 如果沒有承載且 Content-Length 為 0，API Gateway 會將 Content-type 標頭新增至您的請求。

## 適用於 HTTP 和 WebSocket API 的 Amazon API Gateway 重要說明

- 適用於 HTTP 和 WebSocket API 的 Amazon API Gateway 未正式支援第 4A 版簽署程序。

## 適用於 REST 和 WebSocket API 的 Amazon API Gateway 重要說明

- 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。(如需更多詳細資訊，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。
- 映射範本的大小上限為 300 KB。

## 適用於 WebSocket API 的 Amazon API Gateway 重要說明

- 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，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 (|) 和大括號字元 ({ 或 })，而且必須進行 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。
- REST API 會先解碼 URL 編碼的請求參數，然後再將它們傳遞至後端整合。對於 UTF-8 請求參數，REST API 會解碼參數，然後將參數當作 Unicode 傳遞至後端整合。REST API 在將百分號字元 (%) 傳遞到後端整合之前對其進行 URL 編碼。
- 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、底線、連字號、句號、逗號、冒號和大括號。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}" 卻無效。
  - 模型名稱只能包含英數字元。
  - 針對輸入參數，只支援下列屬性：`name`、`in`、`required`、`type`、`description`。其他屬性一概忽略。
  - `securitySchemes` 類型 (若已使用) 必須是 `apiKey`。不過，透過 Lambda 授權方支援 OAuth 2 和 HTTP 基本驗證；透過供應商延伸完成 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。
  - 在任何結構描述物件中，不支援 `discriminator` 參數。
  - 不支援 `example` 標籤。
  - 不支援 `nullable` 欄位。
  - 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後更新模型。
- 對於 OpenAPI 3.0，您無法在相同結構描述內擁有對定義使用 $ref 的 oneOf、anyOf 或 allOf。您可以直接輸入結構描述，或定義個別 API Gateway 模型資源。如需詳細資訊，請參閱the section called “建立更複雜的模型”。
- oneOf 不支援 OpenAPI 2.0 或 SDK 開發套件產生。
- 不支援 readOnly 欄位。
- $ref 無法用於參考其他檔案。
- 在 OpenAPI 文件根中，不支援 "500": {"$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"
    }
  }
```

- API Gateway 不會使用 OpenAPI 規格中定義的根層級安全性，因此您需要在操作層級中定義安全性，以便適當應用。
- 不支援 default 關鍵字。

- 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，可能會將下列資料表清單中的標頭捨棄或修改：在此資料表中：
  - Remapped 表示標頭名稱從 *<string>* 變更為 X-Amzn-Remapped-*<string>*。

    Remapped Overwritten 表示標頭名稱從 *<string>* 變更為 X-Amzn-Remapped-*<string>* 並且覆寫原來的值。

| 標頭名稱 | 請求 (**http/http_proxy /lambda**) | 回應 (**http/http_proxy /lambda**) |
| --- | --- | --- |
| 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 | 傳遞 * | 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 |

| 標頭名稱 | 請求 (**http/http_proxy /lambda**) | 回應 (**http/http_proxy /lambda**) |
| --- | --- | --- |
| Date | 傳遞 | 已重新對應覆寫 |
| Expect | 已捨棄 | 已捨棄 |
| Host | 覆寫至整合端點 | 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 |

| 標頭名稱 | 請求 (**http/http_proxy /lambda**) | 回應 (**http/http_proxy /lambda**) |
| --- | --- | --- |
| 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 | 已重新映射 |

* 如果其包含簽章版本 4 的簽章或使用 AWS_IAM 授權，則 Authorization 標頭將被丟棄。

- API Gateway 所產生之 API 的 Android 軟體開發套件使用 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 開發套件不同，API Gateway 所產生之 API 的 JavaScript 軟體開發套件不支援 500 層級錯誤重試。
- 方法測試呼叫會使用預設的 application/json 內容類型，並忽略任何其他內容類型的規格。
- 傳遞 X-HTTP-Method-Override 標頭以傳送請求至 API 時，API Gateway 即會覆寫該方法。為了將標頭傳遞至後端，您需要將該標頭新增至整合請求中。
- 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 binaryMediaTypes 清單中新增第一個 Accept 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。透過新增 image/webp 至 binaryMediaTypes 清單，端點就能收到二進位格式的 JPEG 檔案。
- 目前不支援自訂 413 REQUEST_TOO_LARGE 的預設閘道回應。
- API Gateway 包含所有整合回應的 Content-Type 標頭。內容類型預設為 application/json。

# 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。如需此文件更新的相關通知，您可以在頂端功能表面板中選擇 RSS 按鈕來訂閱 RSS 摘要。

- 文件最近更新時間：2025 年 12 月 2 日

| 變更 | 描述 | 日期 |
| --- | --- | --- |
| 新增 API Gateway REST API 做為 Amazon Bedrock AgentCore Gateway 的目標 | API Gateway 現在支援新增 REST API 做為 Amazon Bedrock AgentCore Gateway 的目標。如需詳細資訊，請參閱新增 API Gateway REST API 做為 Amazon Bedrock AgentCore Gateway 的目標。 | 2025 年 12 月 2 日 |
| 與 Application Load Balancer 的私有整合 | API Gateway 現在支援與適用於 REST APIs Application Load Balancer 進行私有整合。如需詳細資訊，請參閱 REST APIs 的私有整合。 | 2025 年 11 月 21 日 |
| REST APIs和自訂網域名稱的安全政策 | API Gateway 現在支援 REST APIsTLS 安全政策。如需詳細資訊，請參閱 API Gateway 中 REST APIs的安全政策。 | 2025 年 11 月 19 日 |
| REST APIs的回應串流 | API Gateway REST APIs 現在支援在用戶端可用時逐步串流回應承載。如需詳細資訊，請參閱串流代理整合的整合回應。 | 2025 年 11 月 19 日 |
| 開發人員入口網站 | API Gateway 現在支援入口網站，這是您的客戶探索和測試 APIs和入口網站產品的集中位 | 2025 年 11 月 19 日 |

| | | |
|---|---|---|
| | 置，其代表您要共用的服務或功能。如需詳細資訊，請參閱 API Gateway 入口網站。 | |
| 適用於 REST API 的 SIGv4a | API Gateway 現在可支援使用適用於 REST API 的第 4a 版簽署程序。如需詳細資訊，請參閱調用 API 的 API Gateway 許可模型。 | 2025 年 8 月 19 日 |
| REST API 的路由規則 | API Gateway 現在可支援使用自訂網域名稱的 REST API 路由規則。您可以根據 HTTP 標頭值、URL 基本路徑或兩者的組合，動態路由傳送傳入的請求。如需詳細資訊，請參閱透過自訂網域名稱將流量傳送至 API。 | 2025 年 6 月 3 日 |
| 新增了雙堆疊 (IPv4 和 IPv6) 端點支援 | API Gateway 現在可針對 REST、HTTP 和 WebSocket API 支援雙堆疊端點，以及支援自訂網域名稱。如需詳細資訊，請參閱 IP 位址類型。 | 2025 年 3 月 28 日 |
| 私有 API 的自訂網域名稱 | 已新增對私有 API 的自訂網域的支援。如需詳細資訊，請參閱私有 API 的自訂網域名稱。 | 2024 年 11 月 21 日 |
| 新增對 TLS 1.3 的支援 | API Gateway 現在支援區域性 REST API、HTTP API 和 WebSocket API 上的 TLS 1.3。如需詳細資訊，請參閱在 API Gateway 中選擇自訂網域的安全政策。 | 2024 年 2 月 15 日 |
| REST API 和 WebSocket API 主控台更新 | REST API 和 WebSocket API 的更新後主控台資訊。 | 2023 年 12 月 10 日 |

| | | |
|---|---|---|
| 文件更新 | 更新了概念性資訊，並針對 API Gateway REST API 製作了資料轉換和請求驗證主題的新教學課程。如需詳細資訊，請參閱在 API Gateway 中使用請求驗證和設定 REST API 的資料轉換。 | 2023 年 6 月 22 日 |
| 設定多區域 API Gateway 的 DNS 備援 | 新增使用 Amazon Route 53 運作狀態檢查的支援，以控制從主要 中的 API Gateway REST API AWS 區域 到次要 區域中的 DNS 容錯移轉。如需詳細資訊，請參閱設定 DNS 備援的自訂運作狀態檢查。 | 2022 年 10 月 31 日 |
| 文件更新 | 更新了 REST API 和 HTTP API 這兩種 API 的核心功能摘要。如需詳細資訊，請參閱在 REST API 和 HTTP API 這兩種 API 之間選擇。 | 2022 年 5 月 31 日 |
| 受管政策更新 | 已新增 `acm:GetCertificate` 支援到 `AWSServiceRoleForAPIGateway` 政策。如需詳細資訊，請參閱使用 API Gateway 的服務連結角色。 | 2021 年 7 月 12 日 |
| HTTP API 的參數映射 | 新增對 HTTP API 參數映射的支援。如需詳細資訊，請參閱轉換 API 請求和回應。 | 2021 年 1 月 7 日 |
| 停用 REST API 的預設端點 | 新增了對於停用 REST API 預設端點的支援。如需詳細資訊，請參閱停用 REST API 的預設端點。 | 2020 年 10 月 29 日 |

| | | |
|---|---|---|
| 交互 TLS 驗證 | 新增了對於 REST API 和 HTTP API 交互 TLS 驗證的支援。如需詳細資訊，請參閱設定 REST API 的交互 TLS 驗證和設定 HTTP API 的交互 TLS 驗證。 | 2020 年 9 月 17 日 |
| HTTP API AWS Lambda 授權方 | 新增對 HTTP APIs AWS Lambda 授權方的支援。如需詳細資訊，請參閱使用 HTTP APIs的授權 AWS Lambda 方。 | 2020 年 9 月 9 日 |
| HTTP API AWS 服務整合 | 新增對 HTTP APIs AWS 服務整合的支援。如需詳細資訊，請參閱使用 HTTP APIs AWS 的服務整合。 | 2020 年 8 月 20 日 |
| HTTP API 萬用字元自訂網域 | 新增了對於 HTTP API 萬用字元自訂網域名稱的支援。如需詳細資訊，請參閱萬用字元自訂網域名稱。 | 2020 年 8 月 10 日 |
| 無伺服器開發人員入口網站改善功能 | 將使用者管理新增至管理員面板，並支援匯出 API 定義。如需詳細資訊，請參閱使用無伺服器開發人員入口網站將您的 API Gateway API 編目。 | 2020 年 6 月 25 日 |
| WebSocket API `Sec-WebSocket-Protocol` 支援 | 新增對 `Sec-WebSocket-Protocol` 欄位的支援 如需詳細資訊，請參閱 設定需要 WebSocket 子通訊協定的 $connect 路由。 | 2020 年 6 月 16 日 |

| | | |
|---|---|---|
| HTTP API 匯出 | 新增了對於匯出 HTTP API 的 OpenAPI 3.0 定義的支援。如需詳細資訊，請參閱從 API Gateway 匯出 HTTP API。 | 2020 年 4 月 20 日 |
| 安全性文件 | 新增安全性文件。如需詳細資訊，請參閱 Amazon API Gateway 的安全性。 | 2020 年 3 月 31 日 |
| 重整文件 | 重整開發人員指南。 | 2020 年 3 月 12 日 |
| HTTP API 一般可用性 | 已發行的 HTTP API 正式發行。如需詳細資訊，請參閱使用 HTTP API。 | 2020 年 3 月 12 日 |
| HTTP API 記錄 | 在 HTTP API 日誌中新增對 `$context.integrationErrorMessage` 的支援。如需詳細資訊，請參閱 HTTP API 記錄變數。 | 2020 年 2 月 26 日 |
| AWS OpenAPI 匯入的 變數 | 新增對 OpenAPI 定義中 AWS 變數的支援。如需詳細資訊，請參閱適用於 OpenAPI 匯入的 AWS 變數。 | 2020 年 2 月 17 日 |
| HTTP API | 在測試版中發行 HTTP API。如需詳細資訊，請參閱 HTTP API。 | 2019 年 12 月 4 日 |
| 萬用字元自訂網域名稱 | 新增對萬用字元自訂網域名稱的支援。如需詳細資訊，請參閱萬用字元自訂網域名稱。 | 2019 年 10 月 21 日 |

| | | |
|---|---|---|
| Amazon Data Firehose 記錄 | 新增了對於 Amazon Data Firehose 做為存取記錄資料目的地的支援。如需詳細資訊，請參閱使用 Amazon Data Firehose 做為 API Gateway 存取記錄目的地。 | 2019 年 10 月 15 日 |
| 用於呼叫私有 API 的 Route53 別名 | 新增對其他 Route53 別名 DNS 記錄的支援以呼叫私有 API。如需詳細資訊，請參閱使用 Route53 別名存取您的私有 API。 | 2019 年 9 月 18 日 |
| WebSocket API 的標籤型存取控制 | 新增支援 WebSocket API 的標籤型存取控制。如需詳細資訊，請參閱可以標記的 API Gateway 資源。 | 2019 年 6 月 27 日 |
| 自訂網域的 TLS 版本選擇 | 新增支援自訂網域所部署 API 的 Transport Layer Security (TLS) 版本選擇。請參閱在 API Gateway 中選擇自訂網域的最低 TLS 版本中的相關附註。 | 2013 年 6 月 20 日 |
| 私有 API 的 VPC 端點政策 | 新增將端點政策連接至界面 VPC 端點的支援，以提高私有 API 的安全性。如需詳細資訊，請參閱在 API Gateway 中使用私有 API 的 VPC 端點政策。 | 2019 年 6 月 4 日 |
| 文件已更新 | 已重寫 Amazon API Gateway 入門。將教學課程移至 Amazon API Gateway 教學。 | 2019 年 5 月 29 日 |

| | | |
|---|---|---|
| REST API 的標籤型存取控制 | 新增支援 REST API 的標籤型存取控制。如需詳細資訊，請參閱使用標籤搭配 IAM 政策以控制對 API Gateway 資源的存取。 | 2019 年 5 月 23 日 |
| 文件已更新 | 已重寫 6 個主題：何謂 Amazon API Gateway？、教學課程：建置具有 HTTP 代理整合的 API、教學課程：建立具有三個非代理整合的計算 REST API、API Gateway 對應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。 | 2019 年 4 月 5 日 |
| 無伺服器開發人員入口網站改善功能 | 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布 API。如需詳細資訊，請參閱使用開發人員入口網站將您的 API 編目。 | 2019 年 3 月 28 日 |
| 的支援 AWS Config | 新增 的支援 AWS Config。如需詳細資訊，請參閱使用 監控 API Gateway API 組態 AWS Config。 | 2019 年 3 月 20 日 |
| 的支援 CloudFormation | 已將 API Gateway V2 API 新增至 CloudFormation 範本參考。如需詳細資訊，請參閱 Amazon API Gateway V2 資源類型參考。 | 2019 年 2 月 7 日 |

| | | |
|---|---|---|
| 支援 WebSocket API | 新增對 WebSocket API 的支援。如需詳細資訊，請參閱在 Amazon API Gateway 中建立 WebSocket API。 | 2018 年 12 月 18 日 |
| 可透過 取得無伺服器開發人員入口網站 AWS Serverless Application Repository | 現在可從 AWS Serverless Application Repository (除了 GitHub 以外) 取得 Amazon API Gateway 開發人員入口網站無伺服器應用程式。如需詳細資訊，請參閱使用開發人員入口網站將您的 API Gateway API 編目。 | 2018 年 11 月 16 日 |
| 的支援 AWS WAF | 新增對 AWS WAF (Web 應用程式防火牆) 的支援。如需詳細資訊，請參閱使用 控制 API 的存取 AWS WAF。 | 2018 年 11 月 5 日 |
| 無伺服器開發人員入口網站 | 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，請參閱使用開發人員入口網站將您的 API Gateway API 編目。 | 2018 年 10 月 29 日 |
| 支援多值標頭和查詢字串參數 | 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，請參閱支援多值標頭和查詢字串參數。 | 2018 年 10 月 4 日 |
| OpenAPI 支援 | Amazon API Gateway 現在支援 OpenAPI 3.0 以及 OpenAPI (Swagger) 2.0。 | 2018 年 9 月 27 日 |

| | | |
|---|---|---|
| 文件已更新 | 新增了一個主題：Amazon API Gateway 資源政策如何影響授權工作流程。 | 2018 年 9 月 27 日 |
| 主動 AWS X-Ray 整合 | 您現在可以使用 AWS X-Ray 來追蹤和分析使用者請求中經過 APIs 到基礎服務的延遲。如需詳細資訊，請參閱使用 追蹤 API Gateway API 執行 AWS X-Ray。 | 2018 年 9 月 6 日 |
| 快取功能改良 | 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，請參閱啟用 API 快取以提升回應能力。 | 2018 年 8 月 20 日 |
| Service Limits 已修訂 | 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，請參閱限制。 | 2018 年 7 月 13 日 |
| 覆寫 API 請求和回應參數和標頭 | 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，請參閱使用對應範本來覆寫 API 請求和回應參數和標頭。 | 2018 年 7 月 12 日 |

| | | |
|---|---|---|
| 用量計劃的方法層級調節 | 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，請參閱調節 API 請求以達到更高的輸送量。 | 2018 年 7 月 11 日 |
| API Gateway 開發人員指南更新通知現在可透過 RSS 發送 | 《API Gateway 開發人員指南》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。先前發布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。 | 2018 年 27 月 6 日 |

# 舊版更新

下表說明 2018 年 6 月 27 日前每個《API Gateway 開發人員指南》版本的重要變更。

| 變更 | 描述 | 變更日期 |
|---|---|---|
| 私有 API | 為您透過界面 VPC 端點公開的私有 API 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 | 使用來自不同 AWS 帳戶的 AWS Lambda 函數做為 Lambda 授權方函數或 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the section called “設定跨帳戶 Lambda 授權方”、the section called “教學：建置具 cross-account Lambda 代理整 | 2018 年 4 月 2 日 |

| 變更 | 描述 | 變更日期 |
| --- | --- | --- |
| | 合的 REST API”及the section called “為 REST API 設定跨帳戶 Amazon Cognito 授權方”。 | |
| API 的資源政策 | 使用 API Gateway 資源政策，讓使用不同 AWS 帳戶的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 區塊叫用 API。如需詳細資訊，請參閱the section called “使用 API Gateway 資源政策”。 | 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 the section called “設定標籤”。 | 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the section called “內容編碼”。 | 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the section called “在 API Gateway 中選擇 API 金鑰來源”。 | 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the section called “針對 REST API 使用 Amazon Cognito 使用者集區做為授權方”。 | 2017 年 12 月 14 日 |
| 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 VpcLink 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the section called “教學：建立具私有整合的 REST API”及the section called “設定私有整合”。 | 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the section called “設定 Canary Release 部署”。 | 2017 年 11 月 28 日 |

| 變更 | 描述 | 變更日期 |
| --- | --- | --- |
| 存取記錄 | 使用您所選格式之 $context 變數產生的資料，記錄用戶端對 API 的存取。如需詳細資訊，請參閱the section called “CloudWatch 日誌”。 | 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the section called “在 API Gateway 中產生 API 的 Ruby SDK”及the section called “使用由 API Gateway 為 REST API 產生的 Ruby 軟體開發套件”。 | 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the section called “API Gateway 端點類型”及the section called “設定區域性自訂網域名稱”。 | 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，請參閱使用 API Gateway Lambda 授權方。 | 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，請參閱設定閘道回應以自訂錯誤回應。 | 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 `integration.response.body` 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，請參閱處理 API Gateway 中的自訂 Lambda 錯誤。 | 2017 年 6 月 6 日 |

| 變更 | 描述 | 變更日期 |
| --- | --- | --- |
| 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，請參閱調節對 REST API 的請求，以提高 API Gateway 的輸送量。 | 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API Gateway 中的 REST API 的請求驗證。 | 2017 年 4 月 11 日 |
| 與 ACM 整合 | 為您 API 的自訂網域名稱使用 ACM 憑證。您可以在 中建立憑證， AWS Certificate Manager 或將現有的 PEM 格式憑證匯入 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API Gateway 中公有 REST API 的自訂網域名稱。 | 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 軟體開發套件，並在您的 Java 用戶端中使用軟體開發套件呼叫 API。如需詳細資訊，請參閱使用由 API Gateway 產生的 Java 軟體開發套件來執行 REST API。 | 2017 年 1 月 13 日 |
| 與 整合 AWS Marketplace | 透過 SaaS 產品在用量計劃中銷售您的 API AWS Marketplace。使用 AWS Marketplace 推廣您的 API。代表您依賴 AWS Marketplace 進行客戶帳單。讓 API Gateway 處理使用者授權和用量量測。如需詳細資訊，請參閱將 API 當做 SaaS 銷售。 | 2016 年 12 月 1 日 |
| 提供您 API 的文件支援 | 在 API Gateway 的 DocumentationPart 資源中新增 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 DocumentationVersion。將文件版本匯出至外部檔案 (例如 Swagger 檔案)，來發布 API 文件。如需詳細資訊，請參閱API Gateway 中 REST API 的文件。 | 2016 年 12 月 1 日 |

| 變更 | 描述 | 變更日期 |
| --- | --- | --- |
| 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主體識別符。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，請參閱來自 API Gateway Lambda 授權方的輸出。 | 2016 年 12 月 1 日 |
| 支援二進位承載 | 設定您 API 上的 binaryMediaTypes，以支援請求或回應的二進位承載。設定 Integration 或 IntegrationResponse 上的 `contentHandling` 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API Gateway 中 REST API 的二進位媒體類型。 | 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 ANY 形式的 Greedy 路徑參數以及全部截獲的 `{proxy+}` 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，請參閱設定代理整合與代理資源。 | 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API Gateway 中 REST API 的用量計畫和 API 金鑰。 | 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，請參閱使用 Amazon Cognito 使用者集區做為授權方，藉以控制對 REST API 的存取。 | 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。您可以在 API Gateway 主控台和 Amazon CloudWatch 主控台中進行檢視。如需詳細資訊，請參閱Amazon API Gateway 維度和指標。 | 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，請參閱輪換匯入至 ACM 的憑證。 | 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，請參閱教學課程：匯入範例來建立 REST API及教學：建立具有 HTTP 非代理整合的 REST API。 | 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「在 API Gateway 中使用 OpenAPI 開發 REST API」。 | 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()` 的更多相關資訊，請參閱用於 API Gateway 資料轉換的變數。 | 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，請參閱在 API Gateway 中排清 API 階段快取及使 API Gateway 快取項目失效。改善用於管理 API 請求調節的主控台體驗。如需詳細資訊，請參閱調節對 REST API 的請求，以提高 API Gateway 的輸送量。 | 2016 年 3 月 25 日 |
| 使用自訂授權啟用和呼叫 API Gateway API | 建立並設定 AWS Lambda 函數以實作自訂授權。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，請參閱使用 API Gateway Lambda 授權方。 | 2016 年 2 月 11 日 |

| 變更 | 描述 | 變更日期 |
| --- | --- | --- |
| 使用 Swagger 定義檔和延伸來匯入和匯出 API Gateway API | 使用 Swagger 規格搭配 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 匯出 API，將 API Gateway API 匯出至 Swagger 定義檔。如需詳細資訊，請參閱在 API Gateway 中使用 OpenAPI 開發 REST API及從 API Gateway 匯出 REST API。 | 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API Gateway 中 REST API 的參數映射範例。 | 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，請參閱在 API Gateway 中使用 REST API 的階段變數。 | 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API Gateway 中 REST API 的 CORS。 | 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，請參閱在 API Gateway 中使用 API Gateway 產生的憑證進行後端身分驗證。 | 2015 年 9 月 22 日 |
| 方法的模擬整合 | 了解如何模擬整合 API 與 Amazon API Gateway。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，請參閱用於 API Gateway 資料轉換的變數。 | 2015 年 8 月 28 日 |

| 變更 | 描述 | 變更日期 |
| --- | --- | --- |
| Swagger 整合 | 使用 GitHub 上的 Swagger 匯入工具，將 Swagger API 定義匯入 Amazon API Gateway。進一步了解 API Gateway 的 OpenAPI 擴充功能，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 用於 API Gateway 資料轉換的變數參數和其函數。 | 2015 年 7 月 18 日 |
| 初始公有版本 | 這是《API Gateway 開發人員指南》的初始公開版本。 | 2015 年 7 月 9 日 |

本文為英文版的機器翻譯版本，如內容有任何歧義或不一致之處，概以英文版為準。