aws

開發人員指南

Amazon API Gateway

Amazon API Gateway: 開發人員指南

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 建立 WebSocket API 4
誰使用 API Gateway？ 5
存取 API Gateway 6
部分 AWS 無伺服器基礎架構 6
如何開始使用 Amazon API Gateway 6
API Gateway 概念 7
在 REST API 與 HTTP API 之間進行選擇 11
..... 11
端點類型 11
安全 12
授權 12
API 管理 13
開發 13
監控 14
整合 14
REST API 主控台入門 15
步驟 1：建立 Lambda 函數 16
步驟 2：建立 REST API 16
步驟 3：建立 Lambda 代理整合 17
步驟 4：部署 API 17
步驟 5：調用 API 18
(選用) 步驟 6：清除 19
必要條件 20
註冊一個 AWS 帳戶 20
建立具有管理權限的使用者 20
入門 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：清除 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
選擇一個 HTTP 整合教學課程 59
教學：建置具有私有整合的 API 74
教學課程：建置具有整合功能的 AWS API 76
教學課程：具有三種整合的計算器 81
教學課程：在 API Gateway 中建立 REST API 做為 Amazon S3 代理 109
教學課程：建立 REST API 做為 Amazon Kinesis 代理 155
教學課程：使用 SDK 或建立邊緣最佳化 API AWSAWS CLI 199
教學課程：建置私有的 REST API 232
HTTP API 教學課程 238
與 Lambda 和 DynamoDB 搭配使用的 CRUD API 238
私有整合至 Amazon ECS 250
WebSocket API 教學課程 257
WebSocket 聊天應用 257
WebSocket Step Functions 應用 262
使用 REST API 277
開發 277
API Gateway 端點類型 278
方法 282
存取控制 298
整合 373
請求驗證 434
資料轉換 466
閘道回應 529
CORS 537
二進位媒體類型 550
呼叫 580
OpenAPI 611
發佈 625
部署 REST API 625

自訂網域名稱 667
最佳化 702
快取設定 703
內容編碼 711
分配 716
用量計劃 716
API 文件 740
產生軟體開發套件 800
將 API 當做 SaaS 銷售 827
保護 831
交互 TLS 831
用戶端憑證 837
AWS WAF 878
調節 880
私人休息 API 882
監控 897
CloudWatch 度量 898
CloudWatch 日誌 905
Firehose 911
X-Ray 912
使用 HTTP API 925
開發 925
建立 HTTP API 926
路由 927
存取控制 929
整合 947
CORS 967
參數映射 969
OpenAPI 976
發佈 985
階段 986
適用於 API 的安全性原則 988
自訂網域名稱 990
保護 996
調節 996
交互 TLS 997

監控 1003
指標 1003
日誌 1005
故障診斷 1013
Lambda 整合 1014
JWT 授權方 1016
使用 WebSocket API 1018
關於 WebSocket API 1018
管理連線使用者和用戶端應用程式 1019
呼叫後端整合 1022
將資料從後端服務傳送到連線的用戶端 1025
WebSocket 選取表示式 1026
開發 1031
建立和設定 1032
路由 1033
存取控制 1040
整合 1049
請求驗證 1057
資料轉換 1060
二進位媒體類型 1071
調用 1071
發佈 1074
階段 1075
部署一個 WebSocket API 1077
WebSocket API 的安全性原則 1079
自訂網域名稱 1081
保護 1085
每個區域的帳戶層級調節 1086
路由層級調節 1086
監控 1086
指標 1087
日誌 1088
API Gateway ARN 1095
API 和 WebSocket API 資源 1095
REST API 資源 1098
execute-api（HTTP API， WebSocket API 和其餘 API） 1103

OpenAPI 延伸 1104
x-amazon-apigateway-any-method 1105
x-amazon-apigateway-any-方法示例 1106
x-amazon-apigateway-cors 1107
x-amazon-apigateway-cors 例子 1107
x-amazon-apigateway-api-key-source 1108
x-amazon-apigateway-api-關鍵源示例 1108
x-amazon-apigateway-auth 1109
x-amazon-apigateway-auth 例子 1109
x-amazon-apigateway-authorizer 1110
x-amazon-apigateway-authorizer 其餘 API 的範例 1113
x-amazon-apigateway-authorizer 適用於 HTTP API 的範例 1116
x-amazon-apigateway-authtype 1118
x-amazon-apigateway-authtype 例子 1118
另請參閱 1120
x-amazon-apigateway-binary-媒體類型 1121
x-amazon-apigateway-binary-媒體類型的例子 1121
x-amazon-apigateway-documentation 1121
x-amazon-apigateway-documentation 例子 1121
x-amazon-apigateway-endpoint-配置 1122
x-amazon-apigateway-endpoint-配置示例 1123
x-amazon-apigateway-gateway-回應 1123
x-amazon-apigateway-gateway-響應示例 1123
x-amazon-apigateway-gateway-響應。網關響應 1124
x-amazon-apigateway-gateway-響應。網關響應示例 1124
x-amazon-apigateway-gateway-回應. 回應參數 1125
x-amazon-apigateway-gateway-回應. 回應參數範例 1125
x-amazon-apigateway-gateway-響應. 響應模板 1125
x-amazon-apigateway-gateway-響應。響應模板示例 1126
x-amazon-apigateway-importexport-版本 1126
x-amazon-apigateway-importexport-版本示例 1126
x-amazon-apigateway-integration 1127
x-amazon-apigateway-integration 例子 1131
x-amazon-apigateway-integrations 1132
x-amazon-apigateway-integrations 例子 1133
x-amazon-apigateway-integration。requestTemplates 1135

x-amazon-apigateway-integration。requestTemplates 示例 1135
x-amazon-apigateway-integration. requestParameters 1135
x-amazon-apigateway-integration.requestParameters 範例 1136
x-amazon-apigateway-integration. 回應。 1137
x-amazon-apigateway-integration.responses 範例 1138
x-amazon-apigateway-integration. 回應。 1138
x-amazon-apigateway-integration.response 範例 1139
x-amazon-apigateway-integration. responseTemplates 1140
x-amazon-apigateway-integration. 回應範本範例 1140
x-amazon-apigateway-integration. responseParameters 1141
x-amazon-apigateway-integration.responseParameters 範例 1141
x-amazon-apigateway-integration.TLSCONFIG 1141
x-amazon-apigateway-integration.tlsconfig 範例 1143
x-amazon-apigateway-minimum-壓縮大小 1144
x-amazon-apigateway-minimum-壓縮大小示例 1144
x-amazon-apigateway-policy 1144
x-amazon-apigateway-policy 範例 1144
x-amazon-apigateway-request-驗證 1145
x-amazon-apigateway-request-validator 範例 1145
x-amazon-apigateway-request-驗證器 1146
x-amazon-apigateway-request-validators 範例 1146
x-amazon-apigateway-request-驗證器. 請求驗證器 1147
x-amazon-apigateway-request-validators.requestValidator 範例 1147
x-amazon-apigateway-tag-價值 1148
x-amazon-apigateway-tag-value 範例 1148
安全 1149
資料保護 1149
資料加密 1150
網際網路流量隱私權 1151
Identity and Access Management 1151
對象 1152
使用身分來驗證 1152
使用政策管理存取權 1154
Amazon API Gateway 與 IAM 搭配運作的方式 1156
身分型政策範例 1161
資源型政策範例 1169

故障診斷 1169
使用服務連結角色 1170
記錄和監控 1175
使用 CloudTrail 1176
使用 AWS Config 1179
法規遵循驗證 1181
恢復能力 1182
基礎設施安全性 1183
組態與漏洞分析 1183
最佳實務 1183
標記 1185
可標記的 API Gateway 資源 1185
Amazon API Gateway V1 API 中的標籤繼承 1187
標籤限制和使用慣例 1188
屬性型存取控制 1188
根據資源標籤限制動作 1189
根據資源標籤允許動作 1189
拒絕標記操作 1190
允許標記操作 1191
API 參考 1193
配額和重要說明 1194
每個區域的 API Gateway 帳戶層級配額 1194
HTTP API 配額 1195
..... 1195
用於設定和執行 API 的 WebSocket API Gateway 配額 1197
設定和執行 REST API 的 API Gateway 配額 1198
建立、部署和管理 API 的 API Gateway 配額 1201
重要說明 1203
其餘 API、HTTP API 和 WebSocket API 的重要注意事項 1203
其餘 API 和 WebSocket API 的重要注意事項 1203
WebSocket API 的重要注意事項 1204
適用於 REST API 的重要說明 1204
文件歷史記錄 1210
舊版更新 1217
AWS 詞彙表 1225
..... mccxxvi

什麼是 Amazon API Gateway？

Amazon API Gateway 是一種 AWS 用於建立、發佈、維護、監控和保護任何規模的 REST、HTTP 和 WebSocket API 的服務。API 開發人員可以創建訪問 AWS 或其他 Web 服務的 API，以及存儲在AWS 雲中的數據。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 之間進行選擇 ”、使用 HTTP API、the section called “使用 API Gateway 建立 REST API”，以及the section called “開發”。

API Gateway 會建立以下 WebSocket 功能的 API：

- 遵守WebSocket協議，該協議可在客戶端和服務器之間實現有狀態的全雙工通信。
- 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 建立 WebSocket API”和the section called “關於 WebSocket API”。

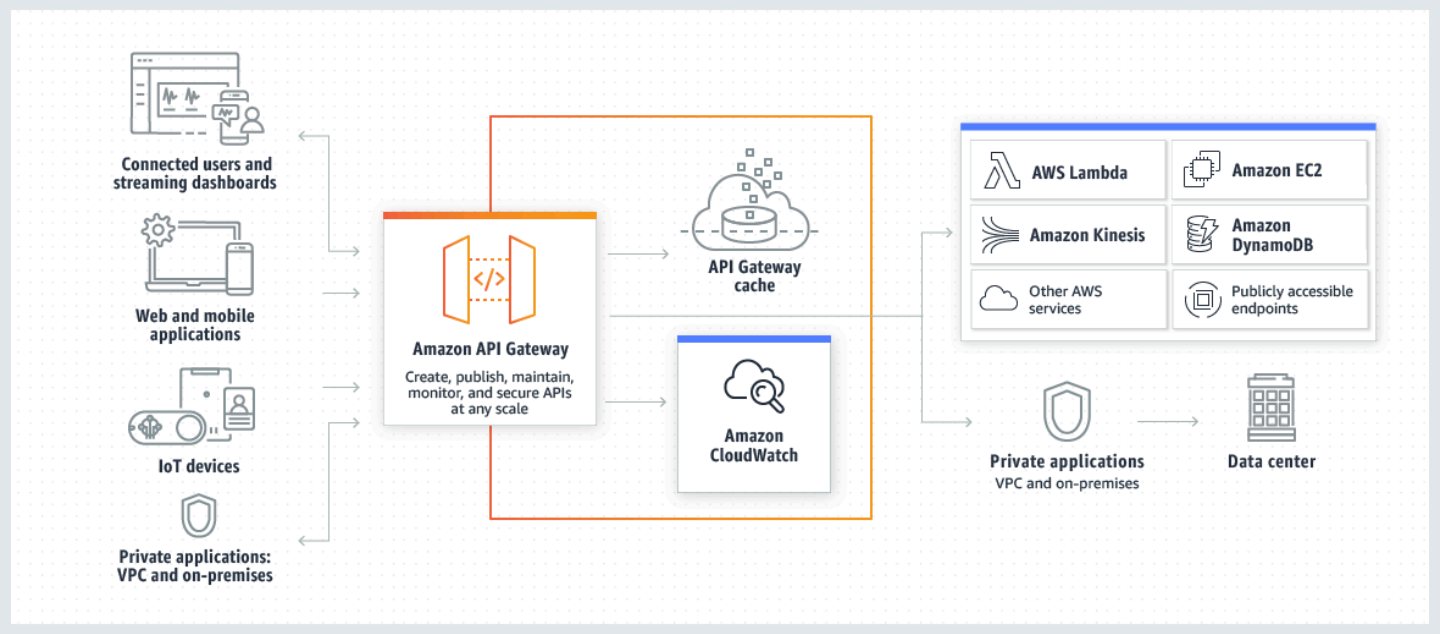
主題

- API Gateway 架構
- API Gateway 的功能
- API Gateway 使用案例
- 存取 API Gateway
- 部分 AWS 無伺服器基礎架構
- 如何開始使用 Amazon API Gateway
- Amazon API Gateway 概念

- 在 REST API 與 HTTP API 之間進行選擇
- REST API 主控台入門

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 彈性運算雲端 (Amazon EC2) 上執行的工作負載、在任何 Web 應用程式上 AWS Lambda執行的程式碼或即時通訊應用程式。

API Gateway 的功能

Amazon API Gateway 提供如下功能：

- Support 可設定狀態 (WebSocket) 和無狀態 (HTTP 和 REST) API。
- 功能強大且靈活的身份驗證機制，例如 AWS Identity and Access Management 政策、Lambda 授權者函數和 Amazon Cognito 使用者集區。
- Canary 發行部署，可讓您安全地進行變更。

- CloudTrail記錄和監控 API 使用情況和 API 更改。
- 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 度量” 及 the section called “指標”。
- 能夠使用 AWS CloudFormation 模板來啟用 API 創建。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 功能發佈的完整清單，請參閱文件歷史記錄。

API Gateway 使用案例

主題

- 使用 API Gateway 建立 REST API
- 使用 API Gateway 建立 HTTP API
- 使用 API Gateway 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 API 將請求發送到 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 之間進行選擇 ”。

使用 API Gateway 建立 WebSocket API

在 WebSocket API 中，客戶端和服務器都可以隨時向彼此發送消息。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 建立無伺服器應用程式，並 AWS Lambda 在聊天室中向個別使用者或使用者群組傳送和接收訊息。或者 AWS Lambda，您也可以根據訊息內容叫用後端服務，例如 Amazon Kinesis 或 HTTP 端點。

您可以使用 API Gateway WebSocket API 建置安全的即時通訊應用程式，而不必佈建或管理任何伺服器來管理連線或大規模資料交換。針對性使用案例包含即時的應用程式，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
- 用 AWS X-Ray 於在通過 API 傳輸到後端服務時跟踪消息
- 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 的 AWS 帳戶中的使用者。

應用程序開發人員通過調用 API API Gateway 關中的 API 開發人員創建的 WebSocket 或 REST API 來構建一個功能正常的應用程序來調用 AWS 服務。

該應用程式開發人員是 API 開發人員的客戶。應用程式開發人員不需要擁有 AWS 帳戶，前提是 API 不需要 IAM 許可，或支援透過 Amazon Cognito 使用者集區身分聯合支援的第三方聯合身分供應商授權使用者。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。它們包括使用 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) 使用 OpenAPI 的 API Gateway 延伸 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 Amazon API Gateway 中叫用 REST API和調用一個 API WebSocket 。

存取 API Gateway

您可以通過以下方式存取 Amazon API Gateway：

- AWS Management Console— 提 AWS Management Console 供用於建立和管理 API 的 Web 介面。完成必要條件中的步驟後，您可以前往 https://console.aws.amazon.com/apigateway 存取 API Gateway 主控台。
- AWS SDK — 如果您使用的是 AWS 提供 SDK 的程式設計語言，您可以使用 SDK 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 Windows PowerShell。

部分 AWS 無伺服器基礎架構

與 AWS LambdaAPI Gateway 一起構成 AWS 無伺服器基礎架構的應用程式面向部分。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。 AWS Lambda 在高可用性的運算基礎架構上執行您的程式碼。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 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：

- 入門，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

API Gateway

API Gateway 是支援下列項目的 AWS 服務：

- 建立、部署和管理 RESTful 應用程式設計介面 (API)，以公開後端 HTTP 端點、 AWS Lambda 函式或其他 AWS 服務。
- 建立、部署和管理 WebSocketAPI 以公開 AWS Lambda 函式或其他 AWS 服務。
- 透過前端 HTTP 和 WebSocket端點叫用公開的 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 資源和方法的集合。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 之間進行選擇 ”。

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 之間進行選擇 ”。

WebSocket API Gateway

與後端 HTTP 端點、Lambda 函數或其他 AWS 服務整合的 WebSocket 路由和路由金鑰的集合。您可以在一或多個階段中部署此集合。API 方法是透過前端 WebSocket 連線叫用，您可以與已註冊的自訂網域名稱建立關聯。

API 部署

API Gateway API 的 point-in-time 快照。若要可供用戶端使用，部署必須與一或多個 API 階段相關聯。

API 開發人員

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

API 端點

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

- 邊緣最佳化的 API 端點
- 私有 API 端點
- 區域 API 端點

API 金鑰

API Gateway 用來識別使用您 REST 或 WebSocket API 的應用程式開發人員的英數字串。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

應用程序創建者可能擁有或可能沒有 AWS 帳戶，並與您（即 API 開發人員）已部署的 API 進行交互。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

當新用戶端透過 WebSocket 連線連線時，您可以呼叫 API Gateway 中的整合來儲存用戶端的回呼 URL。之後您即可使用該回呼 URL 從後端系統傳送訊息到連接的用戶端。

開發人員入口網站

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

邊緣最佳化的 API 端點

部署到指定區域的 API Gateway API 的預設主機名稱，同時使用 CloudFront 分發以促進通常來自不同區 AWS 域的用戶端存取。API 要求會路由到最近的存在 CloudFront 點 (POP)，這通常會縮短不同地理位置用戶端的連線時間。

請參閱 API 端點。

整合請求

API Gateway 中 WebSocket API 路由的內部介面或 REST API 方法，您可以在其中將路由要求的主體或方法要求的參數和主體對應至後端所需的格式。

整合回應

API Gateway 中 WebSocket API 路由或 REST API 方法的內部介面，您可以在其中將從後端接收的狀態碼、標頭和裝載對應至傳回給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。

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 “透過使用 AWS CLI 建立一個 HTTP API”。

區域 API 端點

部署到指定區域並打算為相 AWS 同區域中的用戶端 (例如 EC2 執行個體) 提供服務的 API 的主機名稱。API 請求直接針對特定於區域的 API Gateway API，而無需經過任何 CloudFront 分發。對於區域內請求，區域端點會繞過不必要的往返分發 CloudFront 。

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 路由用於根據消息的內容將傳入消息導向到特定集成，例如 AWS Lambda 功能。定義 WebSocket API 時，您可以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 方法的公共接口，用於定義應用程序開發人員在請求中發送以通過 API 訪問後端所必須發送的主體。

路由回應

WebSocket API 的公共接口，用於定義應用程序開發人員應該從 API Gateway 獲得的狀態代碼，標題和主體模型。

用量計畫

使用方案可讓選取的 API 用戶端存取一或多個已部署的 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 連接

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 之間進行選擇

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。

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	✓	
區域性	✓	✓

端點類型	REST API	HTTP API
私有	✓	

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	✓	✓
後端身分驗證的憑證	✓	
AWS WAF	✓	

授權

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	✓	✓
資源政策	✓	
Amazon Cognito	✓	✓ [1]
具有 AWS Lambda 功能的自定義授權	✓	✓
JSON Web Token (JWT) [2]		✓

[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	✓	✓
API 金鑰	✓	
個別用戶端速率限制	✓	
個別用戶端使用量限制	✓	

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	✓	✓
測試叫用	✓	
快取	✓	
使用者控制的部署	✓	✓
自動部署		✓
自訂閘道回應	✓	
Canary 版本部署	✓	
請求驗證	✓	

功能	REST API	HTTP API
請求參數轉換	✓	✓
請求主體轉換	✓	

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	✓	✓
存取記 CloudWatch 錄檔	✓	✓
訪問 Amazon 數據 Firehose 的日誌	✓	
執行日誌	✓	
AWS X-Ray 追蹤	✓	

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	✓	✓
AWS 服務	✓	✓
AWS Lambda 函數	✓	✓

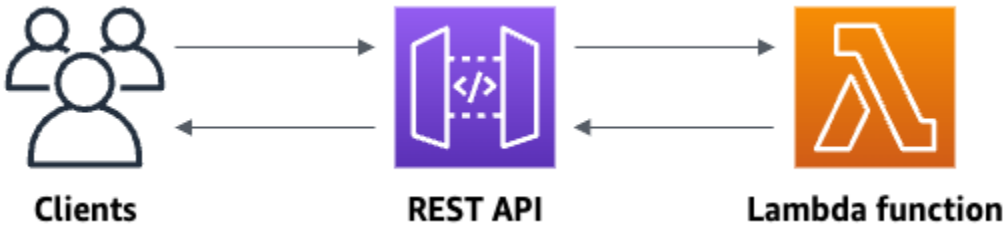
功能	REST API	HTTP API
與 Network Load Balancer 的私有整合	✓	✓
與 Application Load Balancer 的私有整合		✓
私人整合 AWS Cloud Map		✓
模擬整合	✓	

REST API 主控台入門

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 開始使用 API Gateway 的必要條件。

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：清除

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 函式

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. 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. 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**。

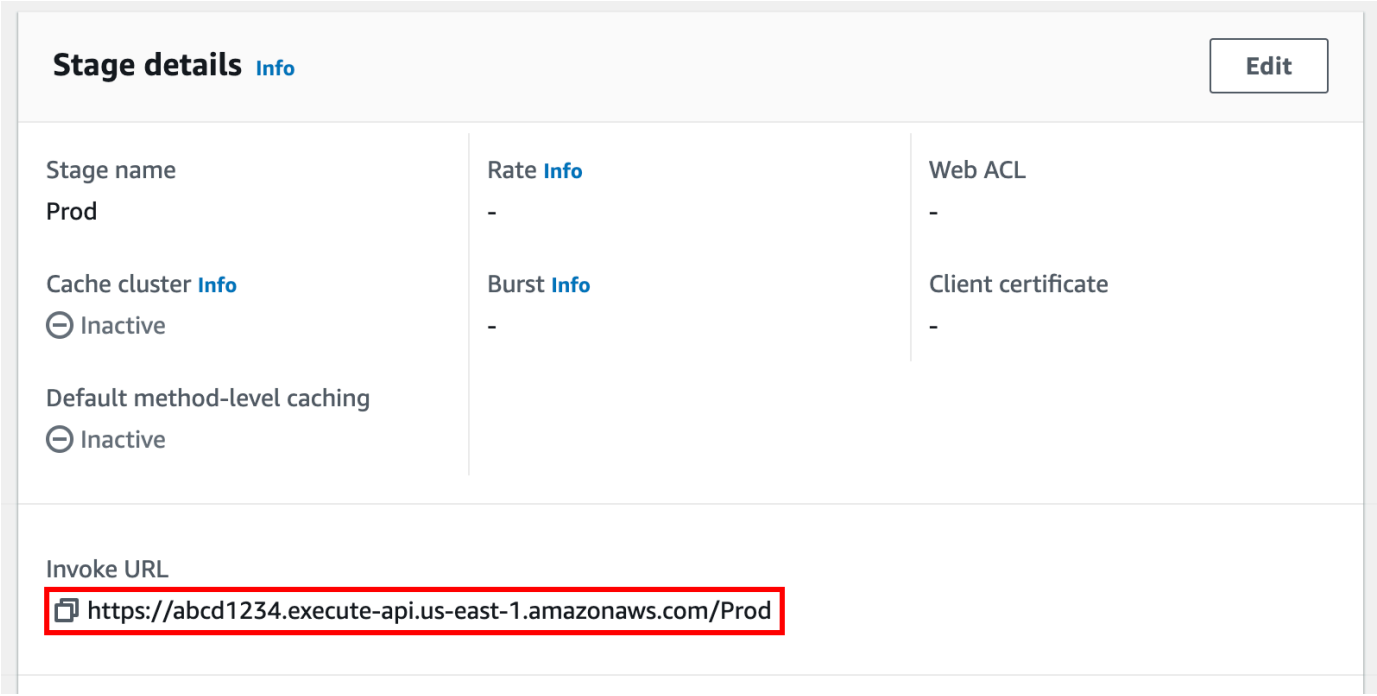
4. 在描述，請輸入描述。
5. 選擇部署。

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：清除

若要避免產生不必要的成本 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 主控台的日誌群組頁面。
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 or AWS Serverless Application Model (AWS SAM) 自動建立和清理 AWS 資源。有關某些示例 AWS CloudFormation 模板，請參閱 awsdocs GitHub 存儲庫中 API Gateway 的示例模板。

開始使用 API Gateway 的必要條件

首次使用 Amazon API Gateway 之前，請先完成下列作業：

註冊一個 AWS 帳戶

如果您沒有 AWS 帳戶，請完成以下步驟來建立一個。

若要註冊成為 AWS 帳戶

1. 開啟 https://portal.aws.amazon.com/billing/signup。
2. 請遵循線上指示進行。

   部分註冊程序需接收來電，並在電話鍵盤輸入驗證碼。

   當您註冊一個時 AWS 帳戶，將創建AWS 帳戶根使用者一個。根使用者有權存取該帳戶中的所有 AWS 服務 和資源。安全性最佳做法是將管理存取權指派給使用者，並僅使用 root 使用者來執行需要 root 使用者存取權的工作。

AWS 註冊過程完成後，會向您發送確認電子郵件。您可以隨時登錄 https://aws.amazon.com/ 並選擇我的帳戶，以檢視您目前的帳戶活動並管理帳戶。

建立具有管理權限的使用者

註冊後，請保護您的 AWS 帳戶 AWS 帳戶根使用者 AWS IAM Identity Center、啟用和建立系統管理使用者，這樣您就不會將 root 使用者用於日常工作。

保護您的 AWS 帳戶根使用者

1. 選擇 Root 使用者並輸入您的 AWS 帳戶 電子郵件地址，以帳戶擁有者身分登入。AWS Management Console在下一頁中，輸入您的密碼。

   如需使用根使用者登入的說明，請參閱 AWS 登入 使用者指南中的以根使用者身分登入。
2. 若要在您的根使用者帳戶上啟用多重要素驗證 (MFA)。

   如需指示，請參閱《IAM 使用者指南》中的為 AWS 帳戶 根使用者啟用虛擬 MFA 裝置 (主控台)。

建立具有管理權限的使用者

1. 啟用 IAM Identity Center。

   如需指示，請參閱 AWS IAM Identity Center 使用者指南中的啟用 AWS IAM Identity Center。
2. 在 IAM 身分中心中，將管理存取權授予使用者。

   若要取得有關使用 IAM Identity Center 目錄 做為身分識別來源的自學課程，請參閱《使用指南》IAM Identity Center 目錄中的「以預設值設定使用AWS IAM Identity Center 者存取」。

以具有管理權限的使用者身分登入

- 若要使用您的 IAM Identity Center 使用者簽署，請使用建立 IAM Identity Center 使用者時傳送至您電子郵件地址的簽署 URL。

  如需使用 IAM 身分中心使用者登入的說明，請參閱使用AWS 登入 者指南中的登入 AWS 存取入口網站。

指派存取權給其他使用者

1. 在 IAM 身分中心中，建立遵循套用最低權限許可的最佳做法的權限集。

   如需指示，請參閱《AWS IAM Identity Center 使用指南》中的「建立權限集」。
2. 將使用者指派給群組，然後將單一登入存取權指派給群組。

   如需指示，請參閱《AWS IAM Identity Center 使用指南》中的「新增群組」。

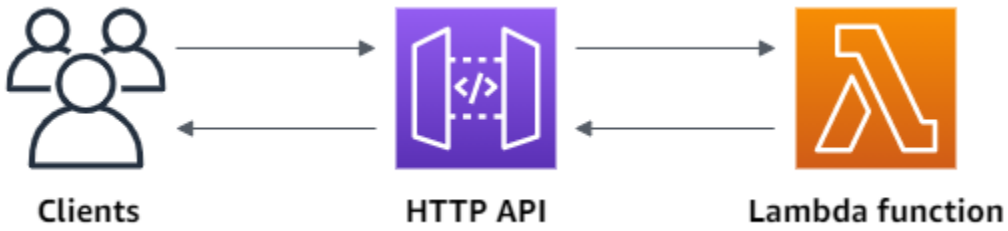
API Gateway 入門

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，您可以使用 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 “REST API 主控台入門”。
> 如需 HTTP API 和其他 API 之間差異的詳細資訊，請參閱the section called “在 REST API 與 HTTP API 之間進行選擇 ”。

在您叫用 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。



要完成這個練習，您需要一個 AWS 帳戶和一個具有控制台訪問權限的 AWS Identity and Access Management 用戶。如需詳細資訊，請參閱 必要條件。

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：清除
- 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. 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 也支援 REST WebSocket API 和 API，但 HTTP API 是本練習的最佳選擇。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 的設計具有最少的功能，因此可以以較低的價格提供它們。 WebSocket API 會維持與用戶端的持續連線，以進行全雙工通訊，此範例並不需要這樣做。

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)。

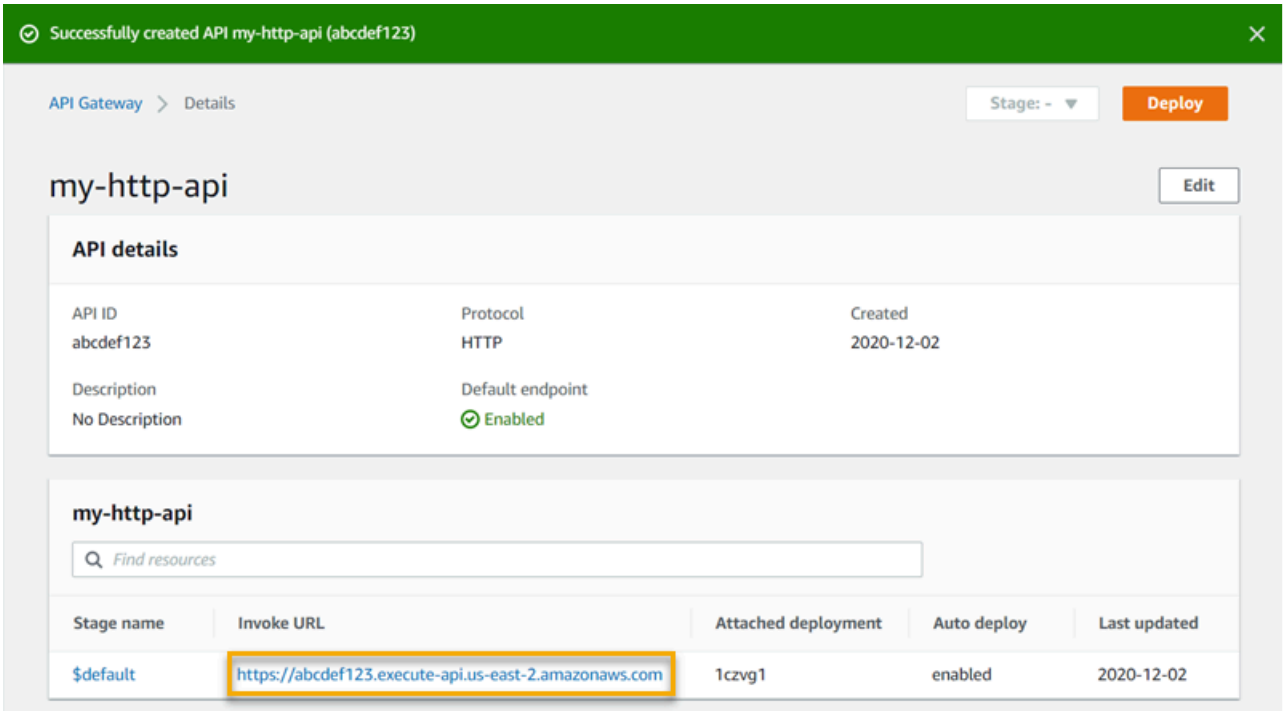
現在，您已建立具有 Lambda 整合的 HTTP API，可隨時接收來自用戶端的請求。

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：清除

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 主 CloudWatch 控台中，開啟日誌群組頁面。
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 [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 AWS CloudFormation 或來自動建立和清理 AWS 資源 AWS SAM。如需 AWS CloudFormation 範本範例，請參閱範例 AWS CloudFormation 範本。

後續步驟

在此範例中，您使 AWS Management Console 用建立簡單的 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 Amazon 簡單佇列服務和 Kinesis Data Streams 等服務 AWS Step Functions
- 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。當您進入這個討論區時， AWS 可能會要求您登入。

若要直接從取得 API Gateway 的說明 AWS，請參閱「Support」頁面上的AWS 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。

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
- 選擇一個 HTTP 整合教學課程
- 教學：建置具有 API Gateway 私有整合的 REST API
- 教學課程：建置具有 AWS 整合功能的 API Gateway 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
- 教學課程：在 API Gateway 中建立 REST API 做為 Amazon S3 代理
- 教學課程：在 API Gateway 中建立 REST API 做為 Amazon Kinesis 代理
- 教學課程：使用 SDK 或建立邊緣最佳化 API AWSAWS CLI
- 教學課程：建置私有的 REST API

HTTP API 教學課程

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

WebSocket API 教學課程

- 教學課程：使用 WebSocket API、Lambda 和 DynamoDB 建置無伺服器聊天應用程式

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
- 選擇一個 HTTP 整合教學課程
- 教學：建置具有 API Gateway 私有整合的 REST API
- 教學課程：建置具有 AWS 整合功能的 API Gateway 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
- 教學課程：在 API Gateway 中建立 REST API 做為 Amazon S3 代理
- 教學課程：在 API Gateway 中建立 REST API 做為 Amazon Kinesis 代理
- 教學課程：使用 SDK 或建立邊緣最佳化 API AWSAWS CLI
- 教學課程：建置私有的 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請求標頭、路徑變數、查詢字串參數、主體和 API 組態資料的任意組合。您只需要選擇一個 Lambda 函數。API Gateway 會設定整合請求和整合回應。設置完成後，您的 API 方法可以在不修改現有設置的情況下進化。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。您必須將作為請求參數提供的客戶端的任何輸入數據映射到適當的集成請求主體。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，您可以在與建立 API 的帳戶不同的帳戶中使用 Lambda 函數。

主題

- 教學課程：建置具有 Lambda 代理整合的 Hello World REST API
- 教學課程：建置具有 Lambda 非代理整合的 API Gateway REST API
- 教學課程：建置具有跨帳戶 Lambda 代理整合的 API Gateway REST API

教學課程：建置具有 Lambda 代理整合的 Hello World 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 函數
- 建立 "Hello, World!" API
- 部署和測試 API

建立 "Hello, World!" Lambda 函數

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，選擇一個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 (建立函式)。
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. 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。

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

e. 將 Policy templates (政策範本) 欄位留白。

f. 選擇 Create function (建立函式)。

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 = function(event, context, callback) {
    console.log('Received event:', JSON.stringify(event, null, 2));
    var res ={
        "statusCode": 200,
        "headers": {
            "Content-Type": "*/*"
        }
    };
    var greeter = 'World';
    if (event.greeter && event.greeter!=="") {
        greeter =  event.greeter;
    } else if (event.body && event.body !== "") {
        var body = JSON.parse(event.body);
        if (body.greeter && body.greeter !== "") {
            greeter = body.greeter;
        }
    } else if (event.queryStringParameters &&
 event.queryStringParameters.greeter && event.queryStringParameters.greeter !==
 "") {
        greeter = event.queryStringParameters.greeter;
    } else if (event.multiValueHeaders && event.multiValueHeaders.greeter &&
 event.multiValueHeaders.greeter != "") {
        greeter = event.multiValueHeaders.greeter.join(" and ");
    } else if (event.headers && event.headers.greeter && event.headers.greeter !
= "") {
        greeter = event.headers.greeter;
    }

    res.body = "Hello, " + greeter + "!";
    callback(null, res);
};
```

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. 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. 選取 / 資源，然後選擇建立資源。
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. 對於 Lambda 函數，請選取 AWS 區域 您建立 Lambda 函數的位置，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測試GET請求，您可以在瀏覽器網址列中輸入 API helloworld 資源的 URL。

要為 API 的helloworld資源創建 URL，請將資源helloworld和查詢字符串參數附加?greeter=John到調用 URL。您的網址看起來應如下所示。

```
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!
```

教學課程：建置具有 Lambda 非代理整合的 API Gateway 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 函數可能會導致您的 AWS 帳戶收取費用。

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'];

console.log('Loading function');

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

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

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

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

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
    try:
        if event['name']:
            name = event['name']
    except KeyError:
        name = 'you'
    try:
        if event['city']:
            city = event['city']
    except KeyError:
        city = '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，該函數 CloudWatch 通過調用將其執行記錄到 Amazon `console.log(...)`。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，請為 Lambda 函數設定具有適當政策的 IAM 角色，以建立 CloudWatch 串流並將記錄項目新增至串流。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 許可控制 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. 請在以下位置開啟 AWS Lambda 主控台。 https://console.aws.amazon.com/lambda/
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. 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。
   d. 在角色名稱中輸入角色名稱 (例如 **`GetStartedLambdaIntegrationRole`**)。
   e. 在 Policy templates (政策範本) 中，選擇 Simple microservice permissions (簡易微服務許可)。
   f. 選擇 Create function (建立函式)。

5. 在 Configure function (設定函數) 窗格的 Function code (函數程式碼) 中，執行下列操作：

   a. 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!"
}
```

輸出也會寫入「 CloudWatch 記錄檔」。

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 CloudWatchCreateLogGroup建立 Lambda 函數的區域中呼叫您帳戶的任何 CloudWatch 資源。此原則也允許建立 GetStartedLambdaIntegration Lambda 函數的 CloudWatch 串流和記錄事件。

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

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

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

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

```
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

此信任關係和內嵌政策的組合使 Lambda 函數可以叫用`console.log()`函數將事件 CloudWatch 記錄到日誌。

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. 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. 對於 Lambda 函數，請選取 AWS 區域 您建立 Lambda 函數的位置，然後輸入函數名稱。
6. 選擇 [方法要求設定]。

   現在，您開啟 URL 路徑變數、查詢字串參數和標頭的要求驗證程式，以確保已定義所有必要資料。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. 針對名稱，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. 針對名稱，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. 在整合要求索引標籤的整合要求設定下，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 Save (儲存)。

測試呼叫 API 方法

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

```
POST /Seattle?time=morning
day:Wednesday

{
    "callerName": "John"
}
```

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

測試 **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. 請登入 AWS Management Console 並開啟 AWS Lambda 主控台，網址為 https://console.aws.amazon.com/lambda/。
2. 從函數清單中選擇，選擇「GetStartedLambdaIntegration動作」，然後選擇「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，選擇「角色動作」，然後選擇「刪除角色」。依照主控台中的步驟刪除角色。

教學課程：建置具有跨帳戶 Lambda 代理整合的 API Gateway REST API

您現在可以使用來自不同 AWS 帳戶的 AWS Lambda 函數作為 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

若要建立 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. 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 (建立函式)。
3. 選擇 Author from scratch (從頭開始撰寫)。
4. 在 Author from scratch (從頭開始撰寫) 下，進行下列操作：

   a. 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. 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。

   d. 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 “教學課程：具有 Lambda 代理整合的 Hello World 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. 選取 / 資源，然後選擇建立資源。
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您的第二個帳戶設定的 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. 請登入 AWS Management Console 並開啟 AWS Lambda 主控台，網址為 https://console.aws.amazon.com/lambda/。
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，透過 PetStore 網站的 HTTP 整合來建立和測試簡單的 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。

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

> **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 會以 PetStore 網站概觀進行回應。這是 `MOCK` 整合類型範例。
- `GET /pets`：適用於與相同具名後端 `/pets` 資源整合之 API `/pets` 資源的讀取存取。後端會傳回中可用寵物的頁面 PetStore。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`。

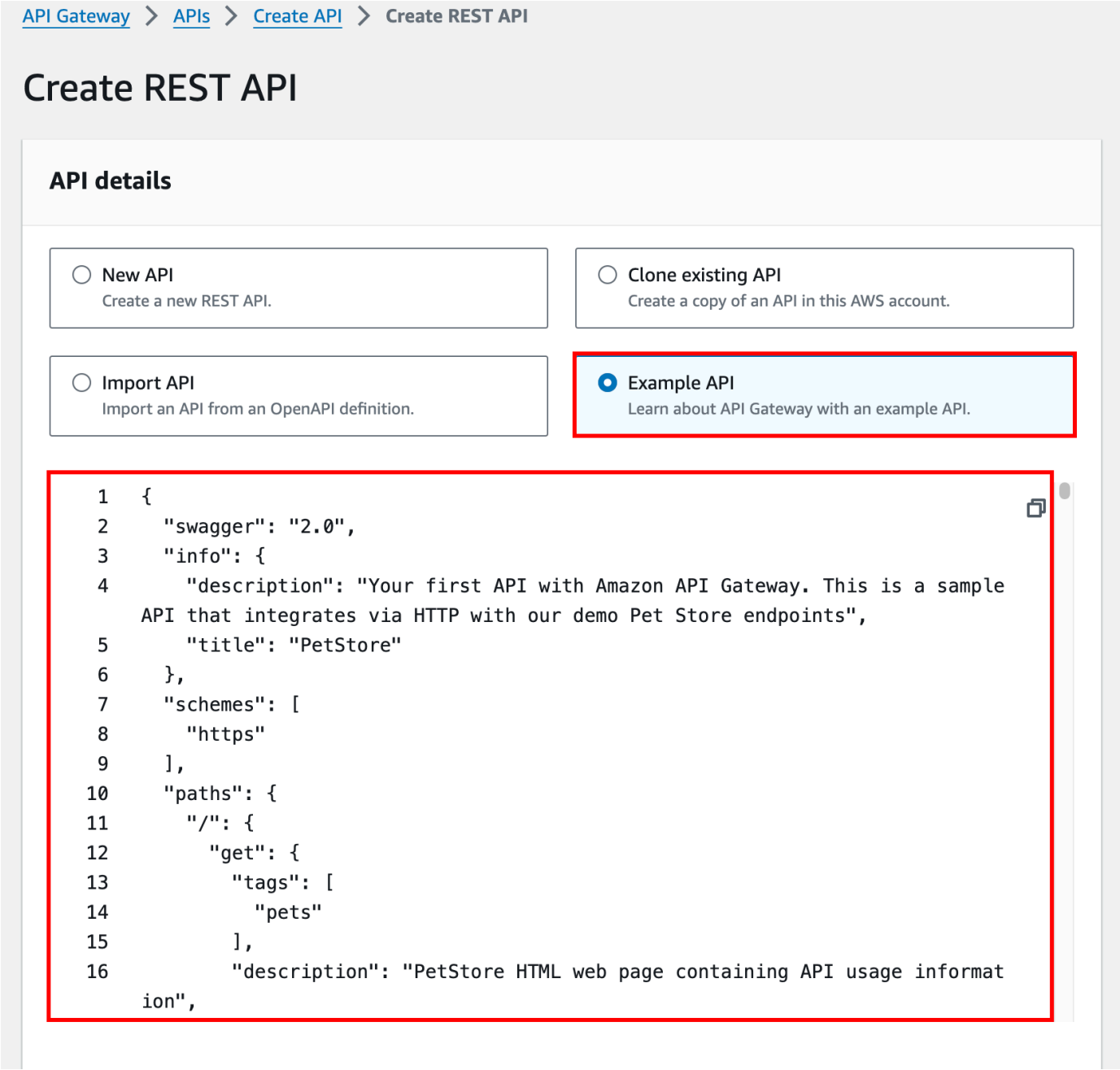
- `POST /pets`：適用於與後端 `/pets` 資源整合之 API `/petstore/pets` 資源的寫入存取。收到正確的請求後，後端將指定的寵物添加到， PetStore 並將結果返回給調用者。此整合也是 HTTP。
- `GET /pets/{petId}`：適用於指定為傳入請求 URL 之路徑變數的 `petId` 值所識別寵物的讀取存取。這個方法也有 HTTP 整合類型。後端會傳回在中找到的指定寵物 PetStore。後端 HTTP 端點的 URL 是 `http://petstore-demo-endpoint.execute-api.com/petstore/pets/`*n*，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。

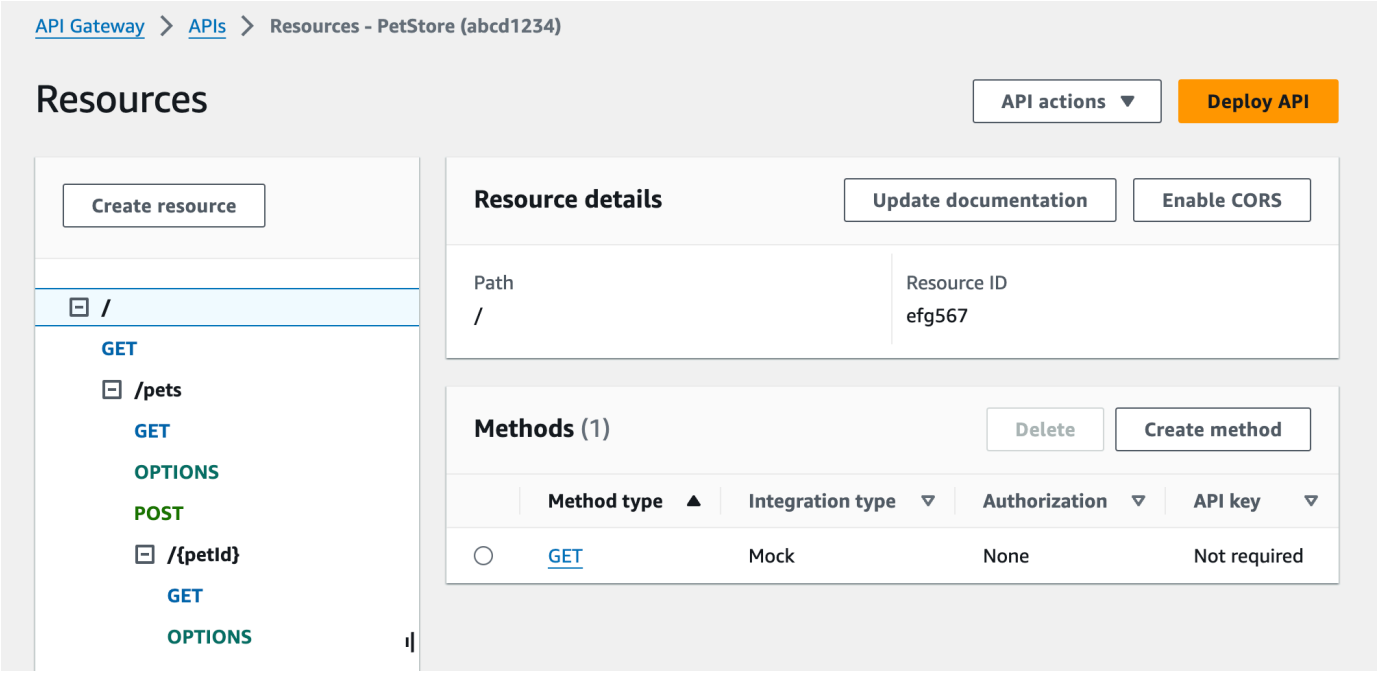
匯入、建置和測試範例 API

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。



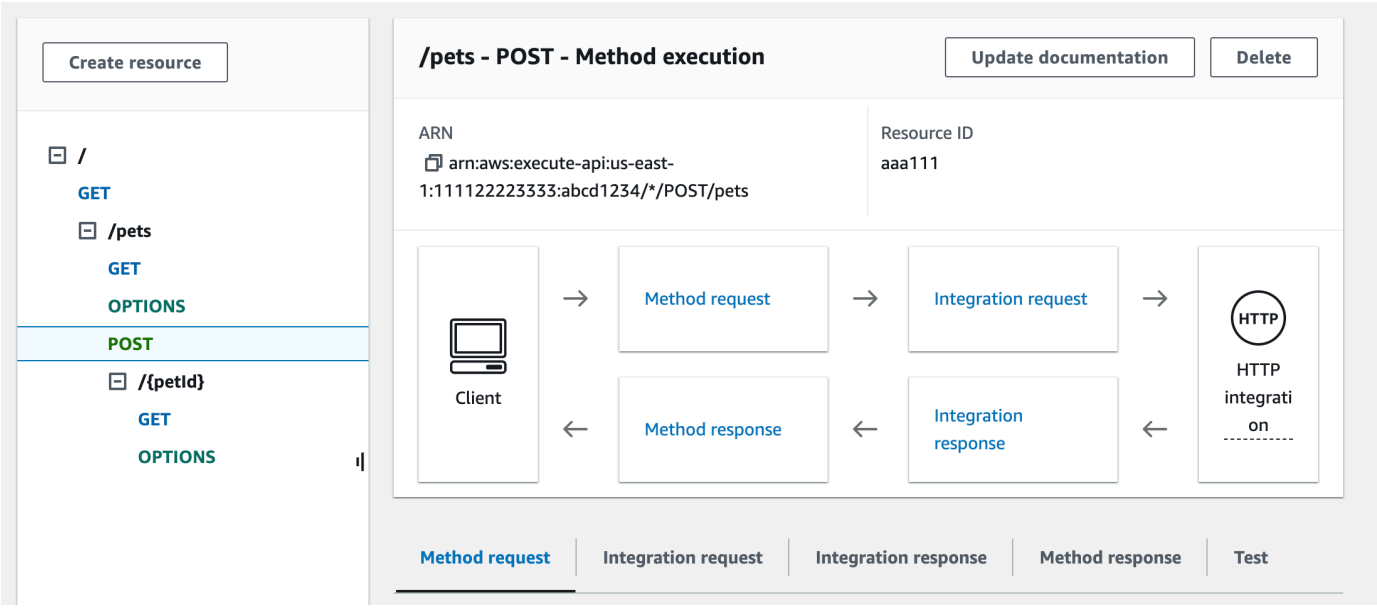
您可以先捲動 OpenAPI 定義以取得此範例 API 的詳細資訊，再選擇建立 API。

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。

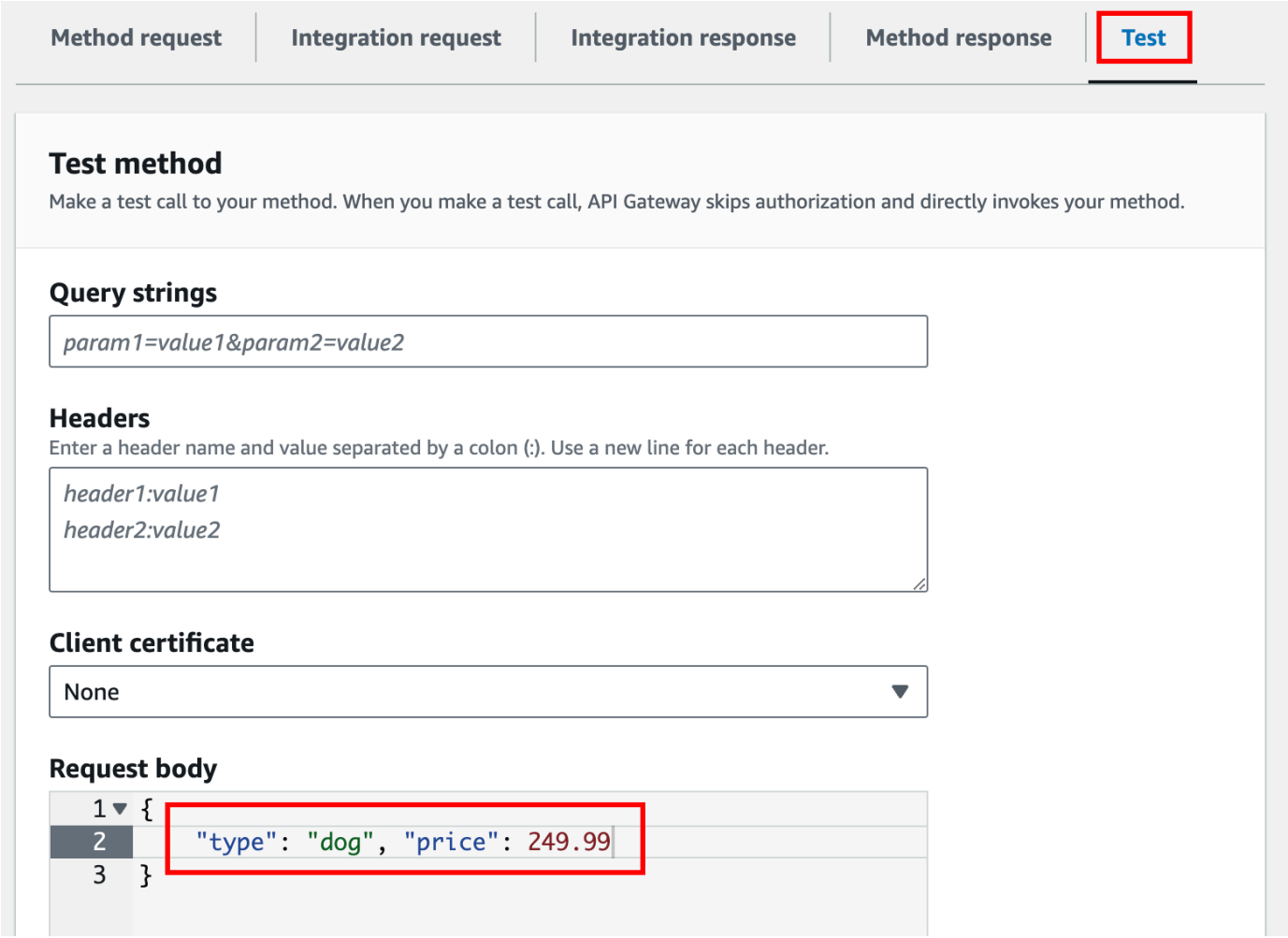
用戶端會使用 API 透過方法請求來存取後端功能。API Gateway 會在必要時先將用戶端請求轉譯為整合請求中後端可接受的形式，再將傳入請求轉送至後端。轉換後的請求稱為整合請求。同樣地，後端會傳回整合回應中的 API Gateway 回應。API Gateway 接著會先將它路由至 Method Response (方法回應)，再將它傳送至用戶端。同樣地，若有必要，API Gateway 可以將後端回應資料映射至用戶端所預期的形式。

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。這些涉及方法與其對應整合之間的資料映射。目前，我們專注於測試 API 以提供用 end-to-end 戶體驗。

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。



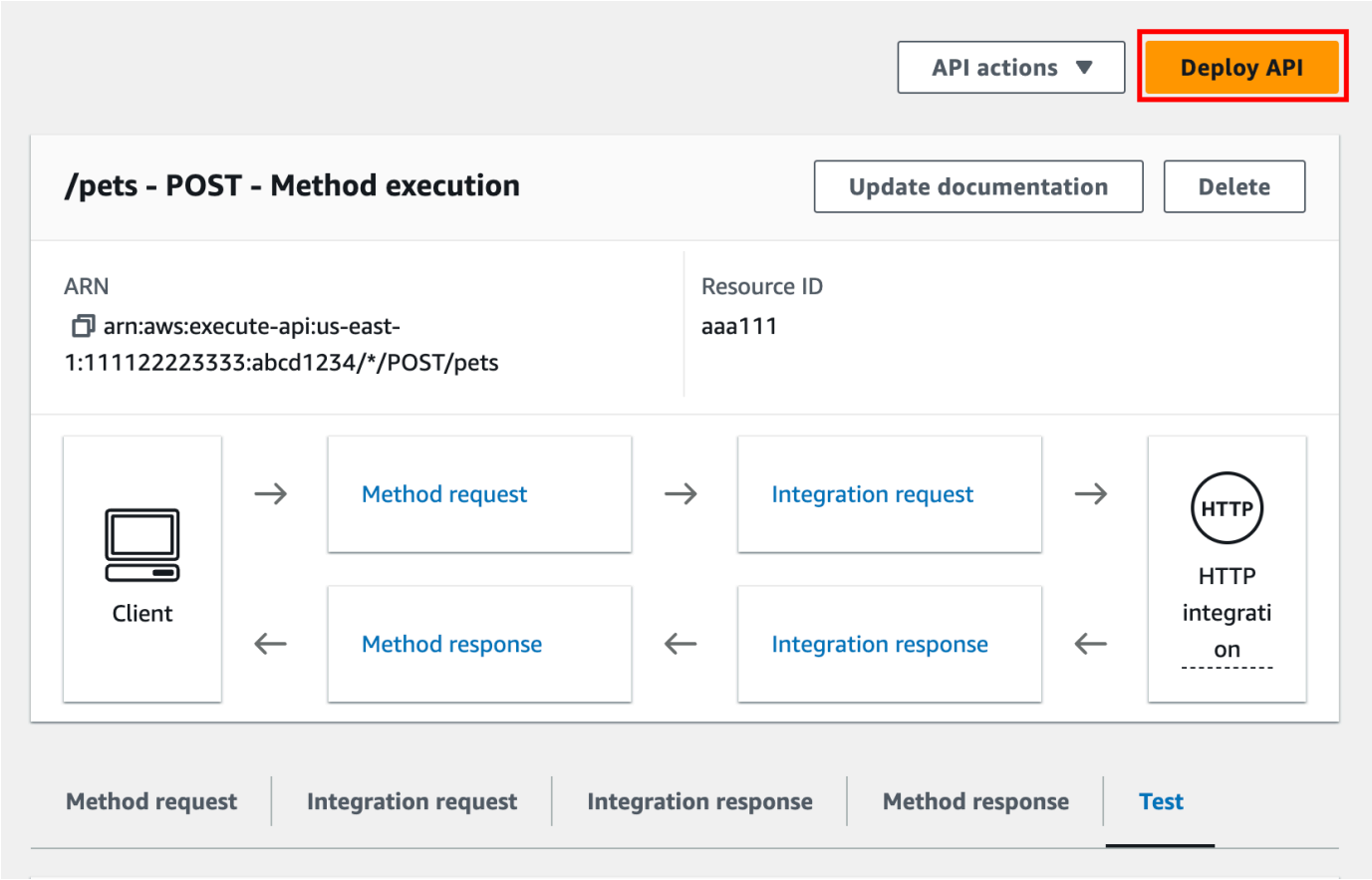
輸入指定我們要添加到 PetStore 網站上的寵物列表中的寵物的屬性。

8. 結果顯示如下：

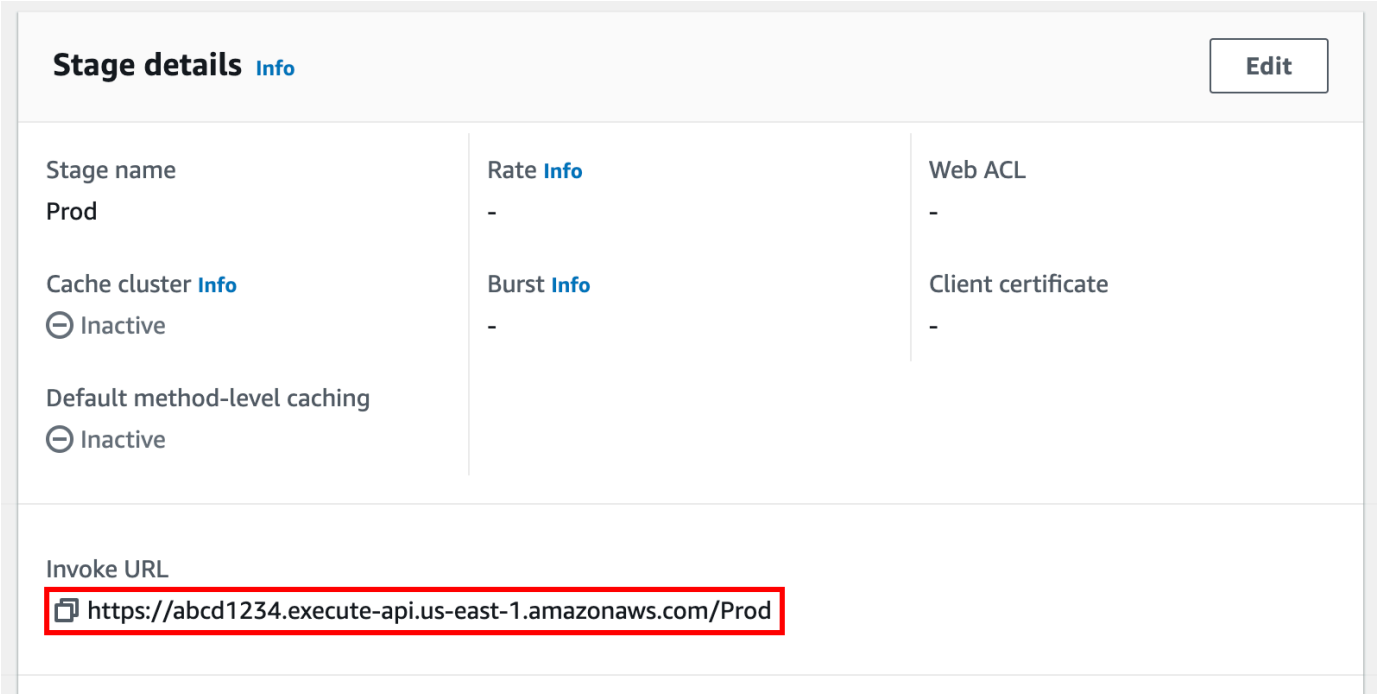
輸出的日誌項目會顯示從方法請求到整合請求，以及從整合回應到方法回應的狀態變更。這適用於故障診斷任何會導致請求失敗的映射錯誤問題。在這個範例中，不會套用任何映射：方法請求承載會透過整合請求傳遞至後端；且同樣地，後端回應會透過整合回應傳遞至方法回應。

若要使用 API Gateway test-invoke-request 功能以外的用戶端測試 API，您必須先將 API 部署到階段。

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 Signature 第 4 版 (SigV4) 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：建置具有 HTTP 非代理整合的 API”。

選擇一個 HTTP 整合教學課程

若要建立具有 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，您只需要根據後端需求設定 HTTP 方法和 HTTP 端點 URI 即可。我們建議您盡可能使用 HTTP 代理整合，以充分利用簡化的 API 設定。

如果您需要轉換後端的用戶端要求資料或轉換用戶端的後端回應資料，則可能需要使用 HTTP 自訂整合。

主題

- 教學課程：建置具有 HTTP 代理整合的 REST API
- 教學課程：建置具有非 HTTP 代理整合的 REST API

教學課程：建置具有 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，透過一般代理資源與 PetStore網站整合`{proxy+}`，並建立的 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 (教學課程：建置具有非 HTTP 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。

若要建立 API

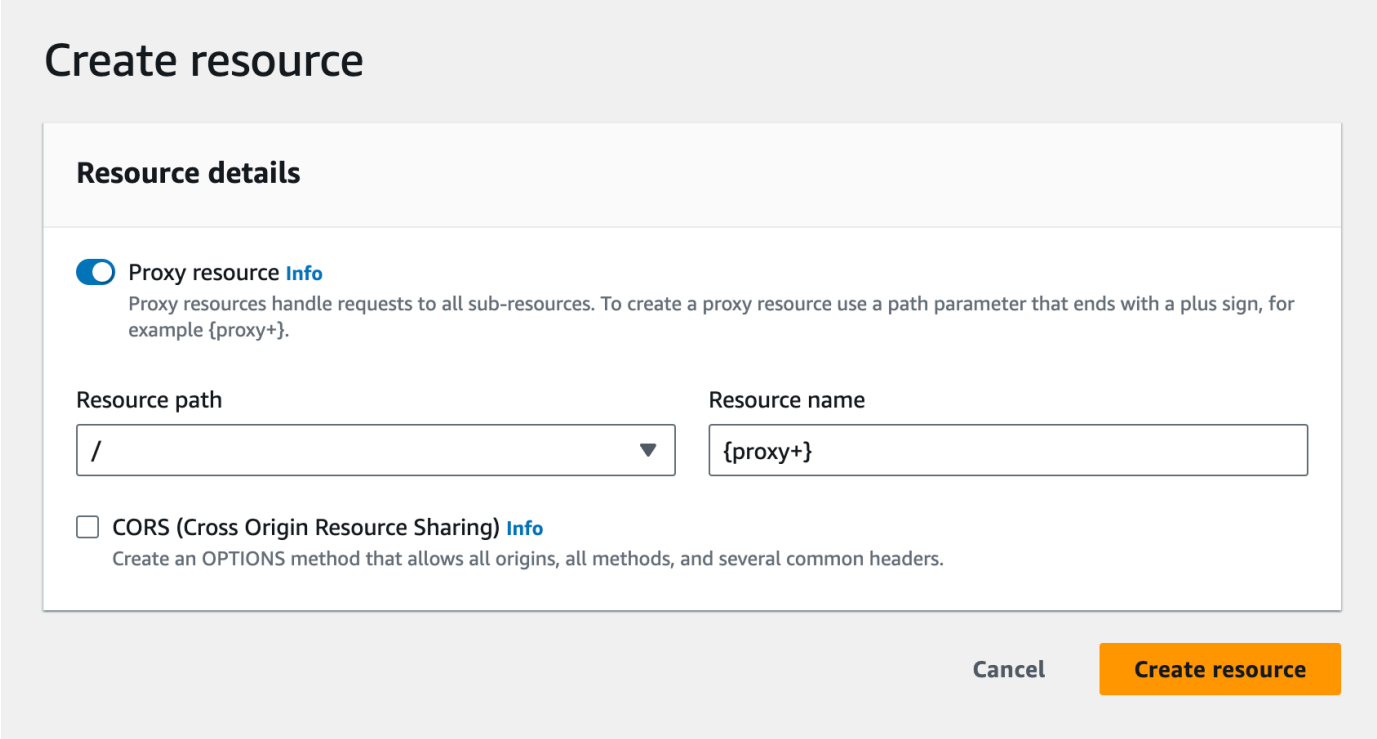
1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 如果這是您第一次使用 API Gateway，您會看到服務功能的介紹頁面。在 REST API 下方，選擇 Build (組建)。當 Create Example API (建立範例 API) 快顯出現時，選擇 OK (確定)。

   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. 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。

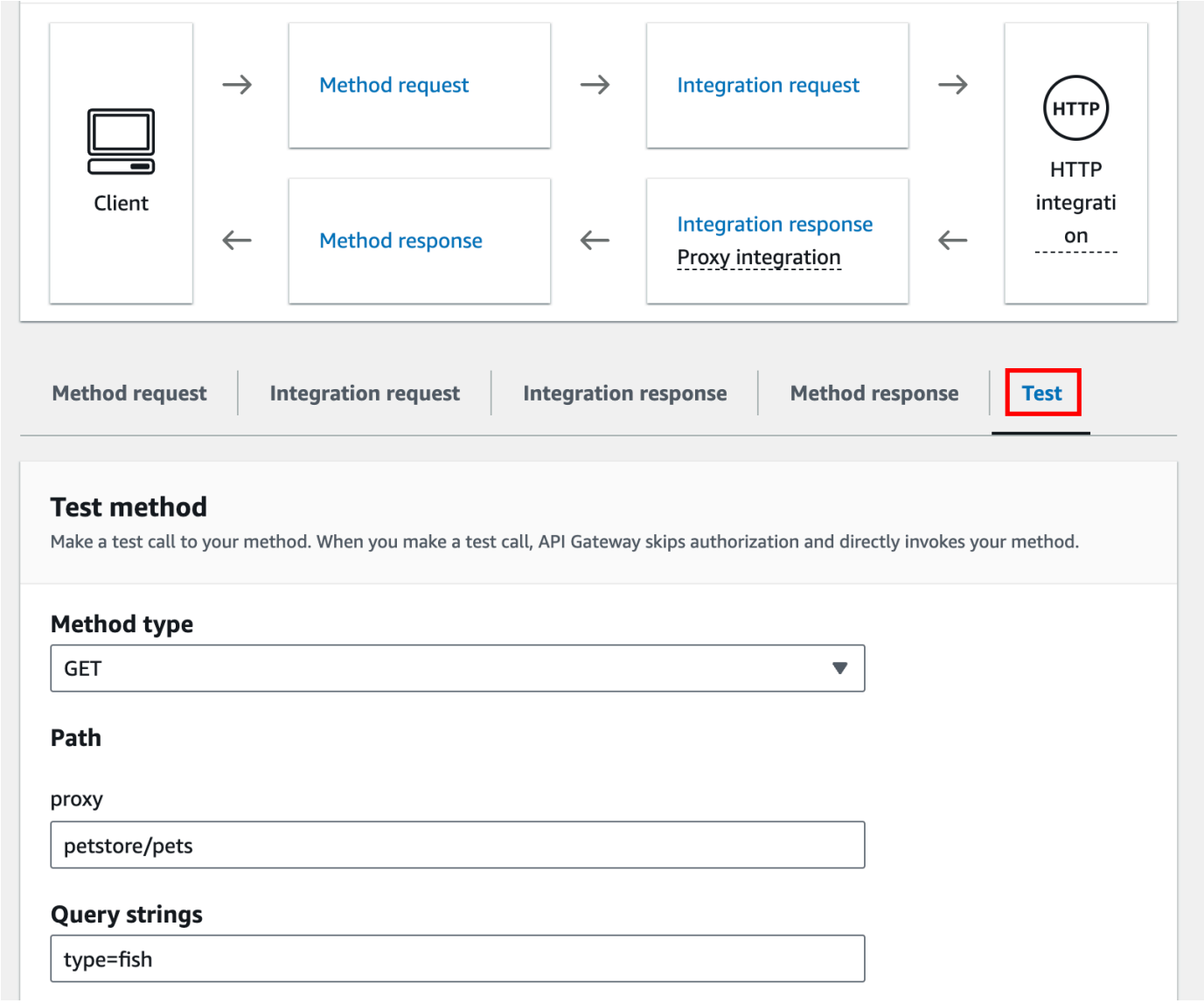
- 如果用戶端提供正確的輸入。

例如，此處使用的 PetStore API 不會公開資/petstore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。

通過代理資源使用 HTTP 代理集成來測試與 PetStore 網站集成的 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。如果您打電話`GET /petstore/pets/1`，則會收到具有以下有效負載的`200 OK`響應，因為該請求受 PetStore網站支持。

```
{
  "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。

教學課程：建置具有非 HTTP 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。

若要建立 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. 選擇建立 API。

Resources (資源) 樹狀目錄顯示不含任何方法的根資源 (/)。在本練習中，我們將使用 PetStore 網站的 HTTP 自定義集成構建 API (HTTP：//petstore-demo-endpoint執行應用程序/寵物店/寵物。) 出於說明目的，我們將創建一個`/pets`資源作為根的子項，並在此資源上公開 GET 方法，供客戶端從 PetStore 網站中檢索可用的 Pets 項目列表。

建立 /pets 資源

1. 選取 / 資源，然後選擇建立資源。
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**。

   該 PetStore 網站允許您在給定頁面上按寵物類型（例如「狗」或「貓」）檢索Pet項目列表。
8. 針對內容處理，選取傳遞。
9. 選擇 URL 查詢字串參數。

   該 PetStore 網站使用type和page查詢字符串參數來接受輸入。您可以將查詢字串參數新增至方法要求，並將它們對應至整合要求的對應查詢字串參數。
10. 若要新增查詢字串參數，請執行下列動作：

    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。這些輸入參數必須對應至整合的查詢字串參數，才能將輸入值轉寄至後端的 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. 針對名稱，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. 針對名稱，輸入 **page**。
   g. 對於映射自，輸入 **method.request.querystring.page**。

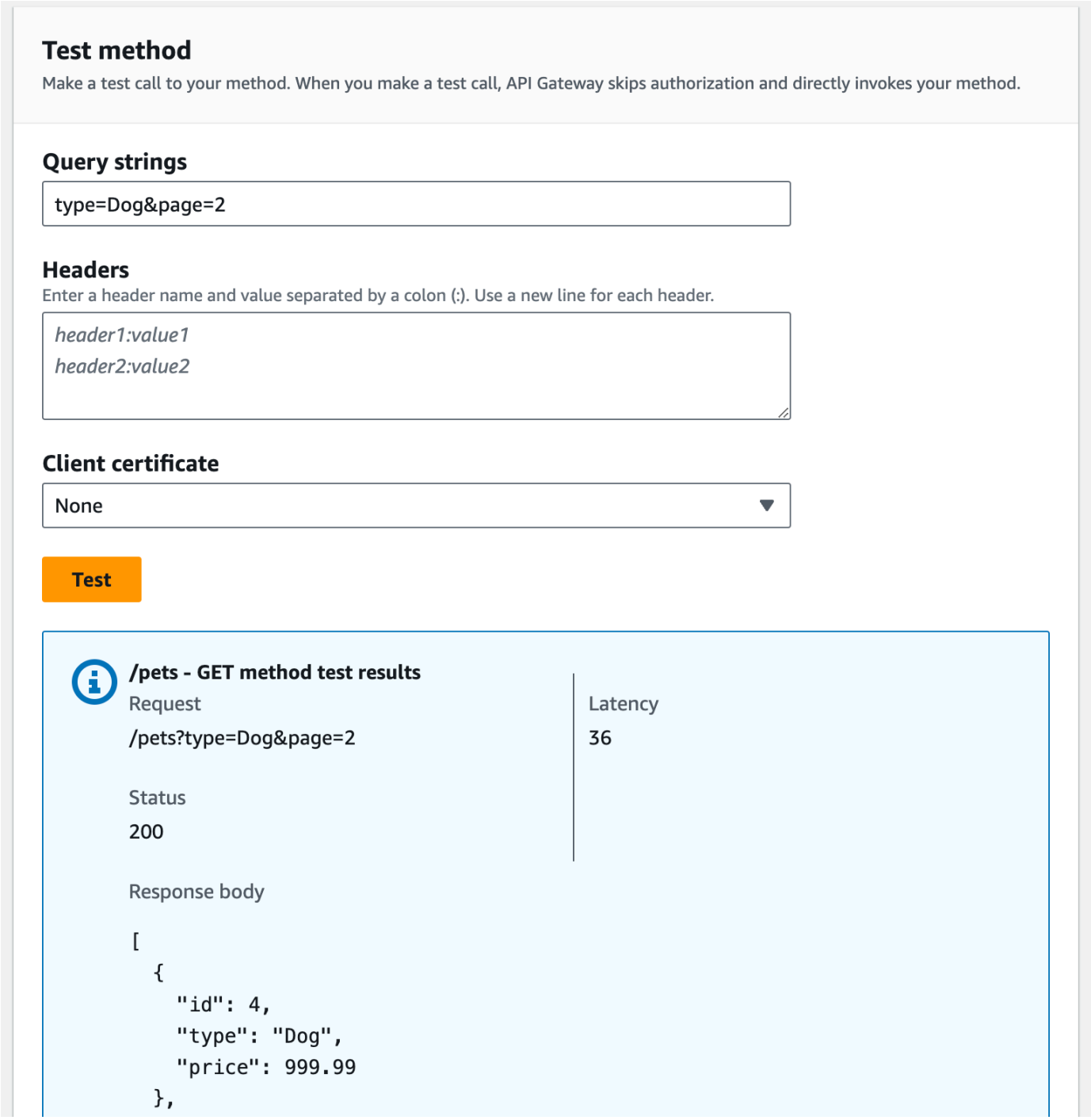
h. 讓快取保持關閉。

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

本教學課程說明如何在 API 的方法請求 URL 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。

現在，您會將 {petId} 路徑參數映射至 HTTP 端點中的 {id} 路徑參數。

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 會建立名為 petId 的整合請求路徑參數。這不適用於您的後端。HTTP 端點會使用 {id} 作為路徑參數。將 petId 重新命名為 **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，請參閱在 API Gateway 中設定資料轉換。
- 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：建置具有 API Gateway 私有整合的 REST API

您可以建立具有私有整合的 API Gateway API，讓您的客戶能夠在 Amazon Virtual Private Cloud (Amazon VPC)內存取 HTTP/HTTPS 資源。這種 VPC 資源是 VPC 中，Network Load Balancer 後方之 EC2 執行個體上的 HTTP/HTTPS 端點。Network Load Balancer 會封裝 VPC 資源，並將傳入請求路由到目標資源。

當用戶端呼叫 API 時，API Gateway 會透過預先設定的 VPC 連結連線到 Network Load Balancer。VPC 連結由的 API Gateway 資源封裝。VpcLink它負責將 API 方法請求轉送到 VPC 資源，再將後端回應傳回給發起人。對於 API 開發人員而言，VpcLink 功能等同於整合端點。

若要建立具有私有整合的 API，您必須建立新的 VpcLink，或選擇已連線到 Network Load Balancer 的現有 ，且該 Network Load Balancer 是以所需的 VPC 資源為目標。您必須擁有適當的許可，才能建立和管理 VpcLink。然後，您要設定 API 方法，將 VpcLink 或 HTTP 設定為整合類型、將 HTTP_PROXY 設定為整合連線類型，並設定整合 VPC_LINK 上的 VpcLink 識別符，整合此方法與 connectionId。

> **Note**
>
> Network Load Balancer 和 API 必須擁有相同 AWS 帳戶。

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

1. 建立 VPC 資源，在同區域的您的帳戶下建立或選擇 Network Load Balancer，然後新增 EC2 執行個體，將資源裝載為 Network Load Balancer 的目標。如需更多詳細資訊，請參閱 設定 API Gateway 私有整合的 Network Load Balancer。

2. 授予建立私有整合 VPC 連結的許可。如需更多詳細資訊，請參閱 授予建立 VPC 連結的許可。

使用在目標群組中設定的 VPC 資源建立您的 VPC 資源和 Network Load Balancer 之後，請依下列說明來建立 API，並在私有整合中透過 `VpcLink` 整合它與 VPC 資源。

建立具有私有整合的 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. 建立邊緣最佳化或區域性的 REST API。
4. 選取您的 API。
5. 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，選取 [使用階段變數]，然後在下面的文字方塊中輸入 **`${stageVariables.vpcLinkId}`**。

      在將 API 部署到階段後定義 `vpcLinkId` 階段變數，並將其值設定為 `VpcLink` 的 ID。

   f. 針對端點 URL 輸入 URL，例如 `http://myApi.example.com`。

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

   g. 選擇建立方法。

      使用代理整合，API 已備妥可供部署。否則，您需要繼續設定適當的方法回應和整合回應。
6. 選擇部署 API，然後執行下列操作：

   a. 針對階段，選取新階段。

   b. 針對階段名稱，輸入階段名稱。

   c. 在描述，請輸入描述。

d. 選擇部署。

7. 在階段詳細資訊區段下，記下產生的叫用 URL。您需要它來呼叫 API。執行這項操作之前，您必須設定 `vpcLinkId` 階段變數。
8. 在階段窗格中，選擇階段變數標籤，然後執行下列動作：

   a. 選擇管理變數，然後選擇新增階段變數。

   b. 針對名稱，輸入 **`vpcLinkId`**。

   c. 針對值，輸入 `VPC_LINK` 的 ID，例如 *`gix6s7`*。

   d. 選擇儲存。

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

教學課程：建置具有 AWS 整合功能的 API Gateway REST API

如需了解如何建立 API Gateway API 以公開整合的 Lambda 函數，請參閱教學課程：建置具有 Lambda 代理整合的 Hello World REST API 和選擇 AWS Lambda 整合教學課程 主題。此外，您還可以建立 API Gateway API 來公開其他 AWS 服務，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 服務都支持專用 API 以公開其功能。不過，應用程式協定或程式設計界面可能會因服務而不同。具有整合功能的 API Gateway `AWS` API 具有為用戶端提供一致的應用程式通訊協定以存取不同 AWS 服務的優點。

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 方法集成了一個單一的 AWS 操作。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。然後，Lambda 函數會剖析並處理其他 AWS 動作的要求。

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 選擇 AWS Lambda 整合教學課程 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

- 必要條件

- 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

必要條件

開始此演練前，請執行下列操作：

1. 完成「開始使用 API Gateway 的必要條件」中的步驟。
2. 建立名為 `MyDemoAPI` 的新 API。如需詳細資訊，請參閱 教學課程：建置具有非 HTTP 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 取得 Amazon SNS 中與您 AWS 帳戶相關聯的主題清單。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。

若要建立服 AWS 務代理執行角色

1. 登入 AWS Management Console 並開啟身分與存取權管理主控台，網址為 https://console.aws.amazon.com/iam/。
2. 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]，然後選取 [允許 API Gateway 將記錄檔推送至 CloudWatch 記錄]。
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，您會建立可讓 AWS 服務 Proxy 與服 AWS 務互動的資源。

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，您會建立讓 AWS 服務 Proxy 與服務互動的 GET 方法。 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. 在中 AWS 區域，選取 AWS 區域 您建立 Amazon SNS 主題的位置。
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 服務代理所依賴的 IAM 資源，則該 AWS 服務代理和任何依賴它的 API 將不再起作用。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 [ApigateWay]AWSProxyExecRole，然後選擇 [角色動作]、[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 [ApigateWay]AWSProxyExecPolicy，然後選擇 [策略動作] > [刪除]。出現提示時，選擇 Delete (刪除)。

本演練到此結束。如需將 API 建立為 AWS 服務 Proxy 的詳細討論，請參閱教學課程：在 API Gateway 中建立 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：在 API Gateway 中建立 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

非代理整合入門教學課程只會使用 `Lambda Function` 整合。`Lambda Function` 整合是 `AWS Service` 整合類型的特殊案例，此整合類型會為您執行許多整合設定，例如自動新增 Lambda 函數叫用所需的資源型許可。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：

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

```
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

您建立的角色必須具有 Lambda InvokeFunction權限。否則，API 發起人將會收到 `500 Internal Server Error` 回應。若要將此許可給與角色，請將下列 IAM 政策連接至其中：

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

以下是如何完成此全部操作的方式：

建立 API Gateway 可擔任的 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。

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

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

b. 選擇 Review policy (檢閱政策)。

c. 在 Review Policy (檢閱政策) 下，執行下列操作：

i. 在 Name (名稱) 中輸入名稱，例如 **lambda_execute**。

ii. 選擇 Create Policy (建立政策)。

8. 在原本的 Create Role (建立角色) 主控台視窗中，執行下列動作：

a. 在 Attach permissions policies (連接許可政策) 下方，從下拉式清單中選擇您的 **lambda_execute** 政策。

如果您在清單中沒有看到您的政策，請選擇清單頂端的重新整理按鈕。(請不要重新整理瀏覽器頁面！)

b. 選擇 Next: Add Tags (下一步：新增標籤)。

c. 選擇 Next:Review (下一步：檢閱)。

d. 在 Role name (角色名稱) 中輸入名稱，例如 **lambda_invoke_function_assume_apigw_role**。

e. 選擇 Create Role (建立角色)。

9. 從角色清單中選擇您的 **lambda_invoke_function_assume_apigw_role**。

10. 選擇 Trust Relationships (信任關係) 標籤。

11. 選擇 Edit trust relationship (編輯信任關係)。

12. 用以下內容取代現有政策：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "",
```

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

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

建立 **Calc** Lambda 函數

接下來，您會使用 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. 選擇 建立函式。
6. 在 Lambda 主控台中，複製下列 Lambda 函數，並將其貼入程式碼編輯器。

   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
```

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

若要建立 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. 選擇建立 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，讓其可透過瀏覽器叫用。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. 在中 AWS 區域，選取 AWS 區域 您建立 Lambda 函數的位置。
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`**。對於 ***`account-id`***，輸入您的 AWS 帳戶 ID。在中***`us-east-2`***，輸入 AWS 區域 您建立 Lambda 函數的位置。
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 [方法要求設定]。
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，您可以在 /calc 資源上為 GET 方法設定查詢字串參數，以便它可以代表後端 Lambda 函數接收輸入。

    若要建立查詢字串參數，請執行下列動作：

    a. 選擇新增查詢字串。

    b. 針對名稱，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')"
}
```

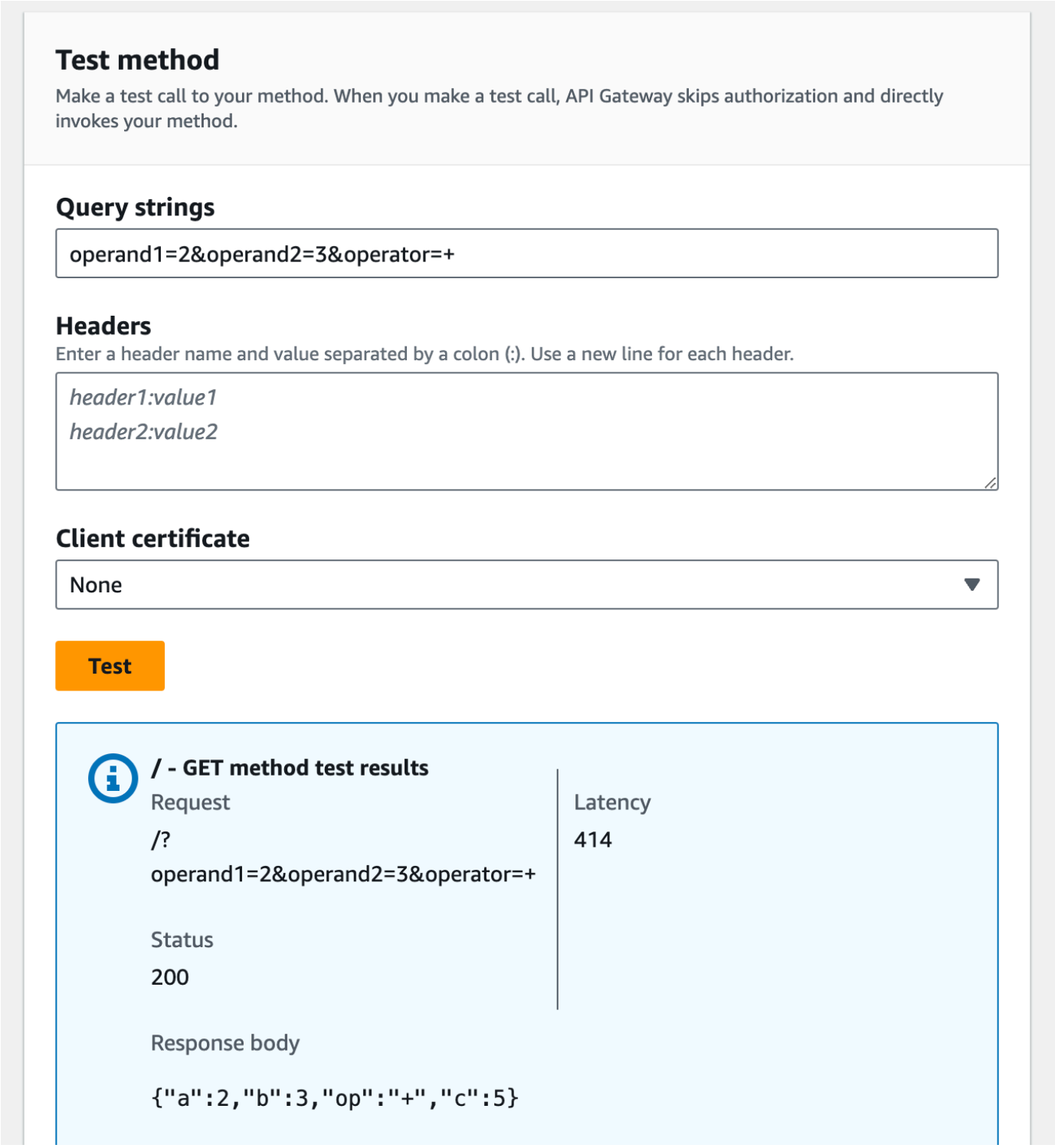
7. 選擇儲存。

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. 在中 AWS 區域，選取 AWS 區域 您建立 Lambda 函數的位置。
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**。對於 ***account-id***，輸入您的 AWS 帳戶 ID。在中***us-east-2***，輸入 AWS 區域 您建立 Lambda 函數的位置。
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. 針對名稱，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. 針對名稱，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. 針對名稱，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。

若要啟用 POST 方法的要求驗證

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。

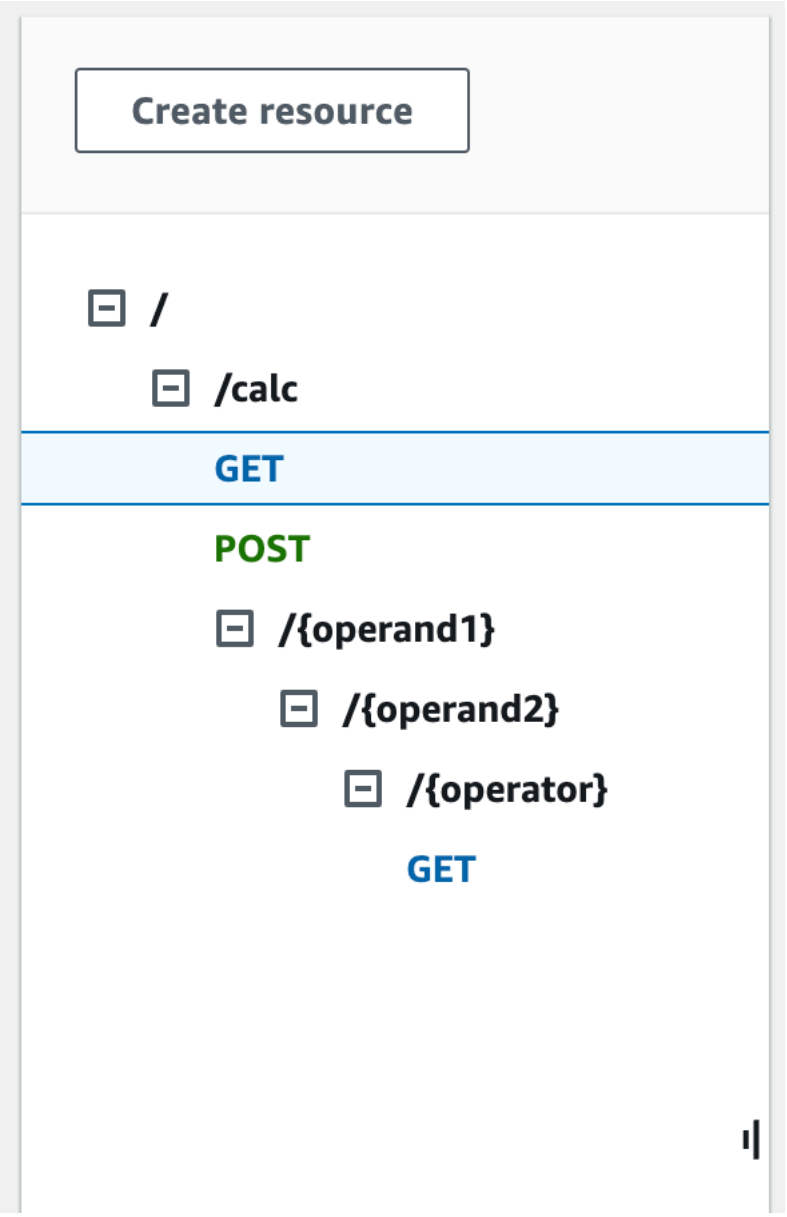
   您應該會看到下列輸出：

```
{
  "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對應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. 對於 Lambda 函數，請選取 AWS 區域 您建立 Lambda 函數的位置並輸入**`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。

建立對應範本

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')",
```

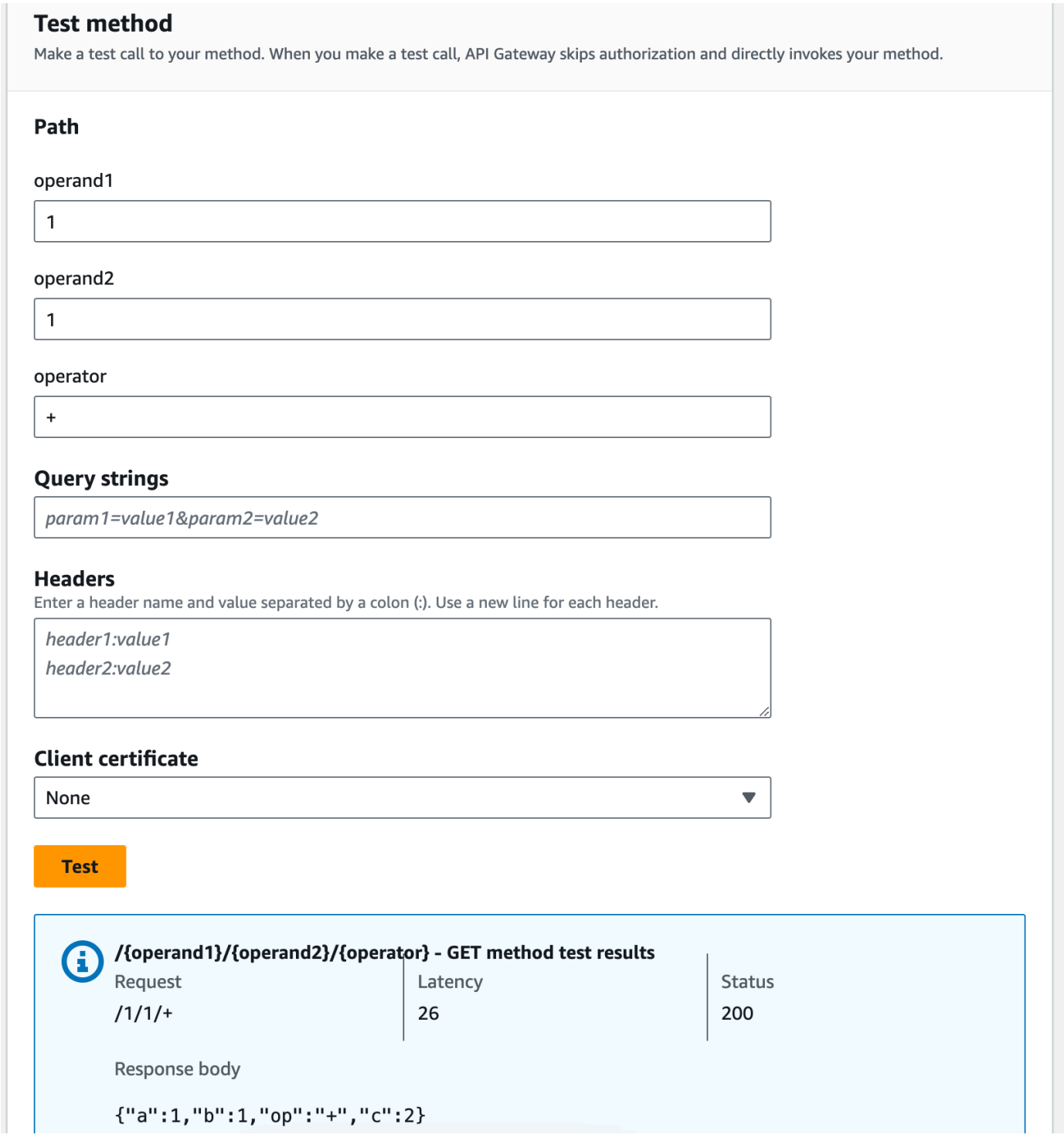
```
   "op":
 #if($input.params('operator')=='%2F')"/"#{else}"$input.params('operator')"#end
}
```

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。

建立對應範本

1. 在整合回應索引標籤上，於預設 - 回應下，選擇編輯。
2. 選擇對應範本。
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
    }
  }
}
```

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

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

- 在 API 根資源上公開 GET，以列出發起人的所有 Amazon S3 儲存貯體。
- 在 Folder 資源上公開 GET，以檢視 Amazon S3 儲存貯體中的所有物件清單。
- 在 Folder/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，請參閱使用 OpenAPI 設定 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。

若要建立服 AWS 務代理執行角色

1. 登入 AWS Management Console 並開啟身分與存取權管理主控台，網址為 https://console.aws.amazon.com/iam/。
2. 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]，然後選取 [允許 API Gateway 將記錄檔推送至 CloudWatch 記錄]。
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

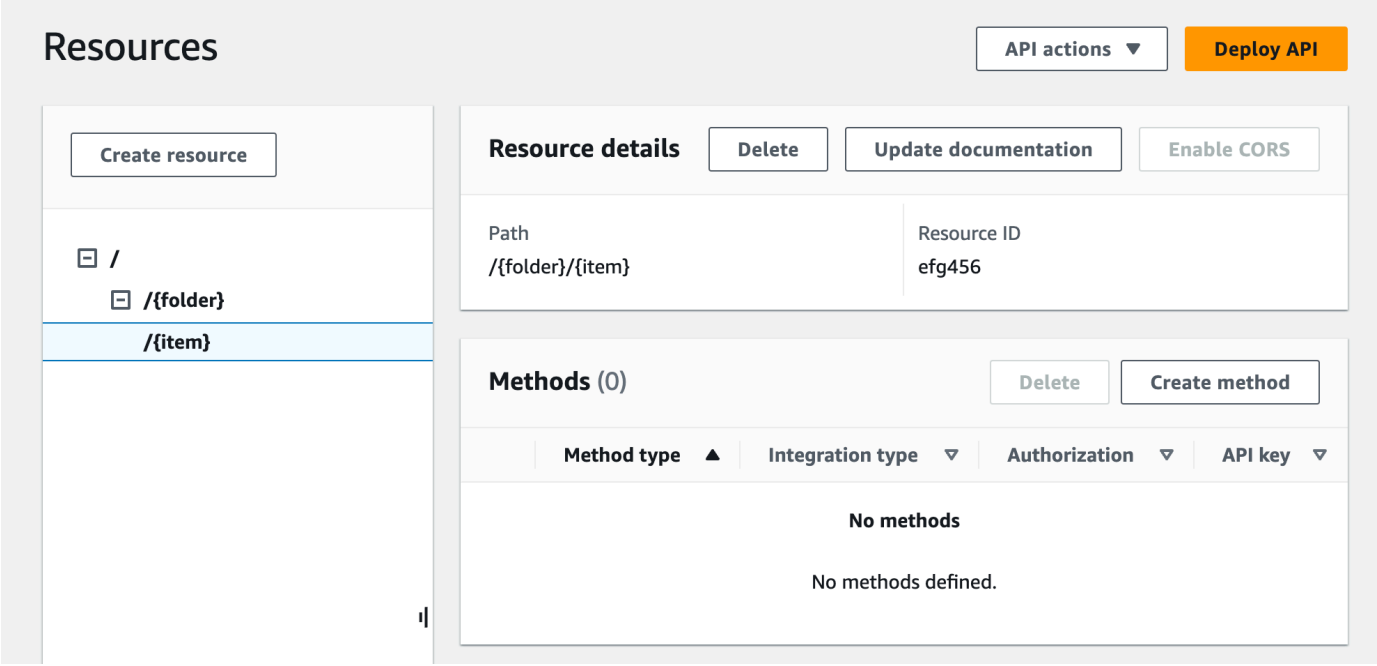
您可以使用 API 的根 (/) 資源做為經驗證呼叫者 Amazon S3 儲存貯體的容器。您也可以建立`Folder`和`Item`資源來分別代表特定的 Amazon S3 儲存貯體和特定的 Amazon S3 物件。發起人會將資料夾名稱和物件金鑰指定為請求 URL 一部分的路徑參數。

> **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. 在同一個 AWS 區域 你創建了你的 Amazon S3 存儲桶，創建一個名為 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. 選取 API 的根資源，然後選擇建立資源。
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

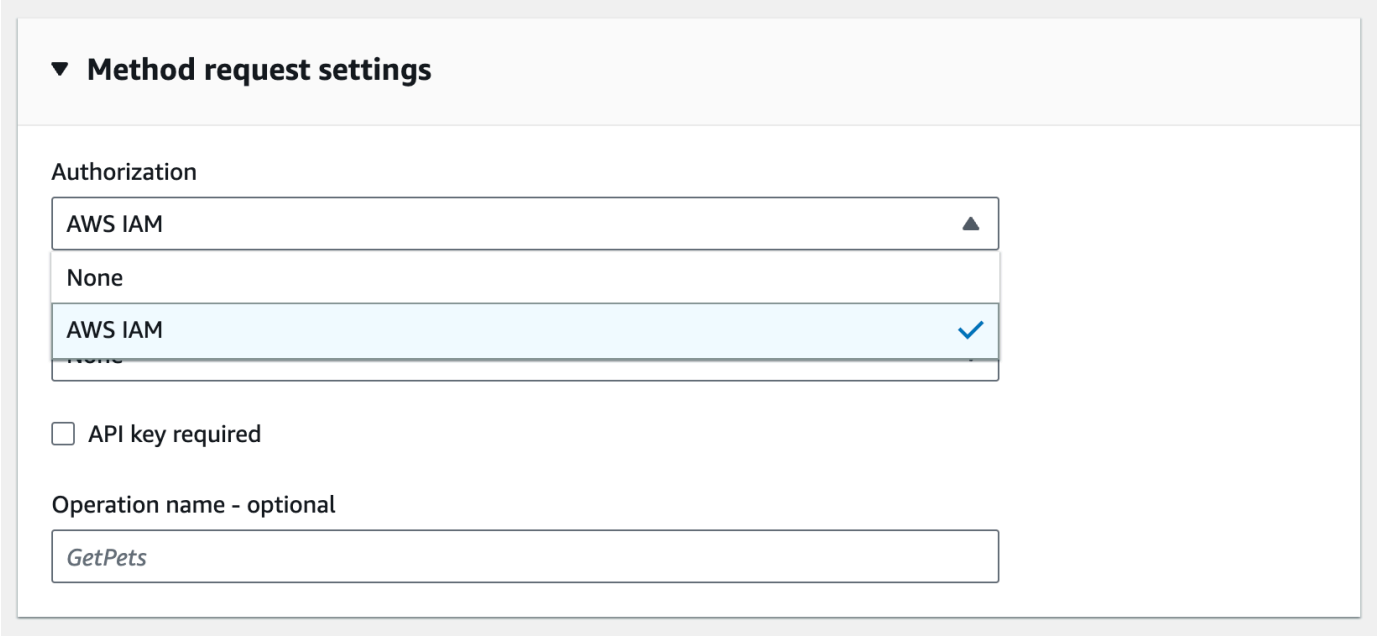
取得發起人的 Amazon S3 儲存貯體清單包含在 Amazon S3 上呼叫 GET 服務動作。在 API 的根資源 (/) 上，建立 GET 方法。設定 GET 方法以與 Amazon S3 整合，如下所示。

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. 在中 AWS 區域，選取您 AWS 區域 在哪裡建立 Amazon S3 儲存貯體。
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 [方法要求設定]。

    您可以使用方法請求設置來控制誰可以調用您的 API 的這個方法。
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 正確返回成功的響應和異常給調用者，您可以在方法響應中聲明 200，400 和 500 響應。您可以使用 200 個回應的預設對應，以便未在此處宣告之狀態碼的後端回應作為 200 個回應傳回給呼叫者。

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。

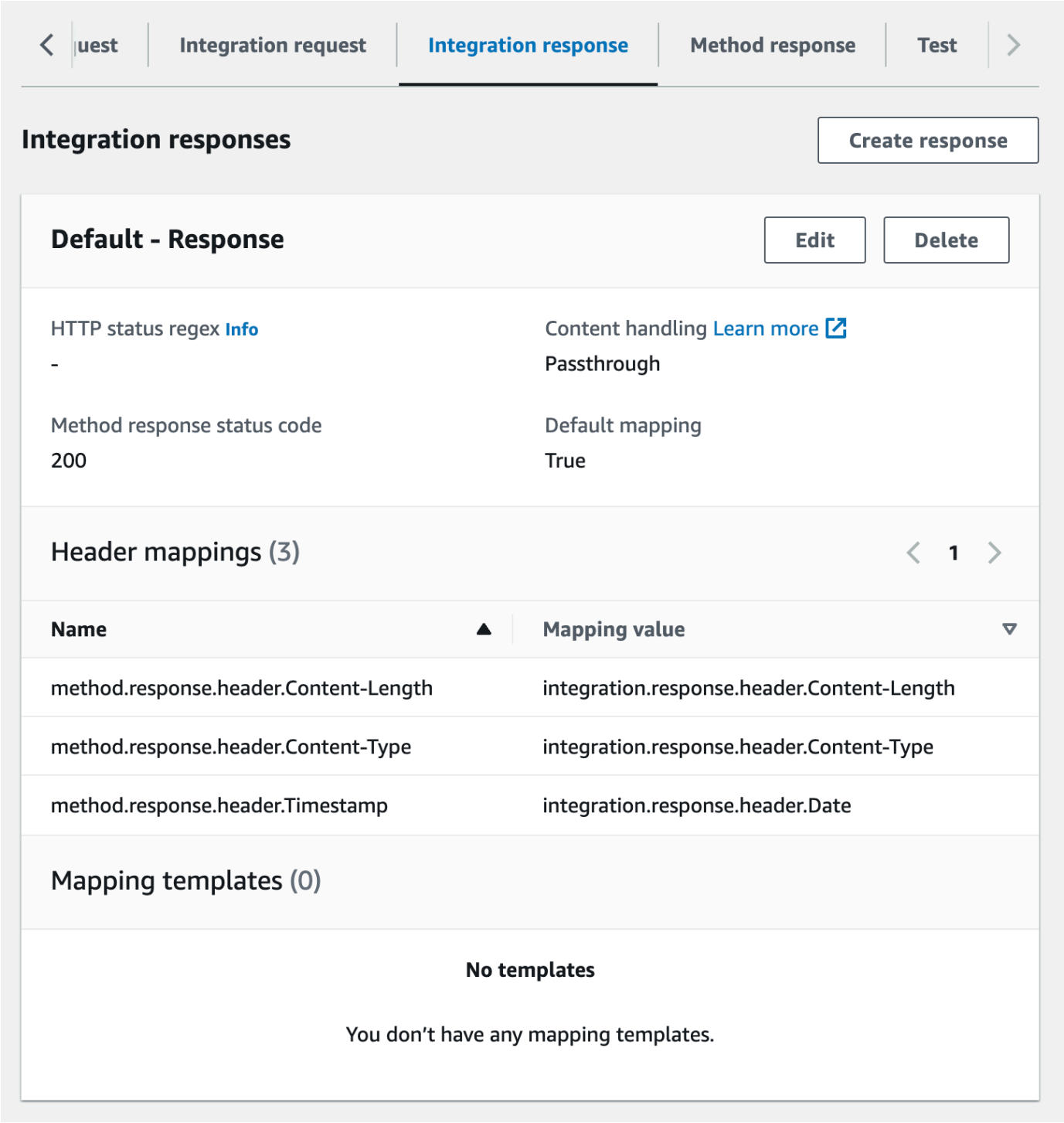
   您未針對此回應設定任何標頭。

由於 Amazon S3 的成功整合回應會以 XML 承載的形式傳回儲存貯體清單，而 API Gateway 的預設方法回應會傳回 JSON 承載，因此您必須將後端 Content-Type 標頭參數值對應至前端對應項目。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，您還希望向客戶端顯示其他頭參數，如日期和內容長度。

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：

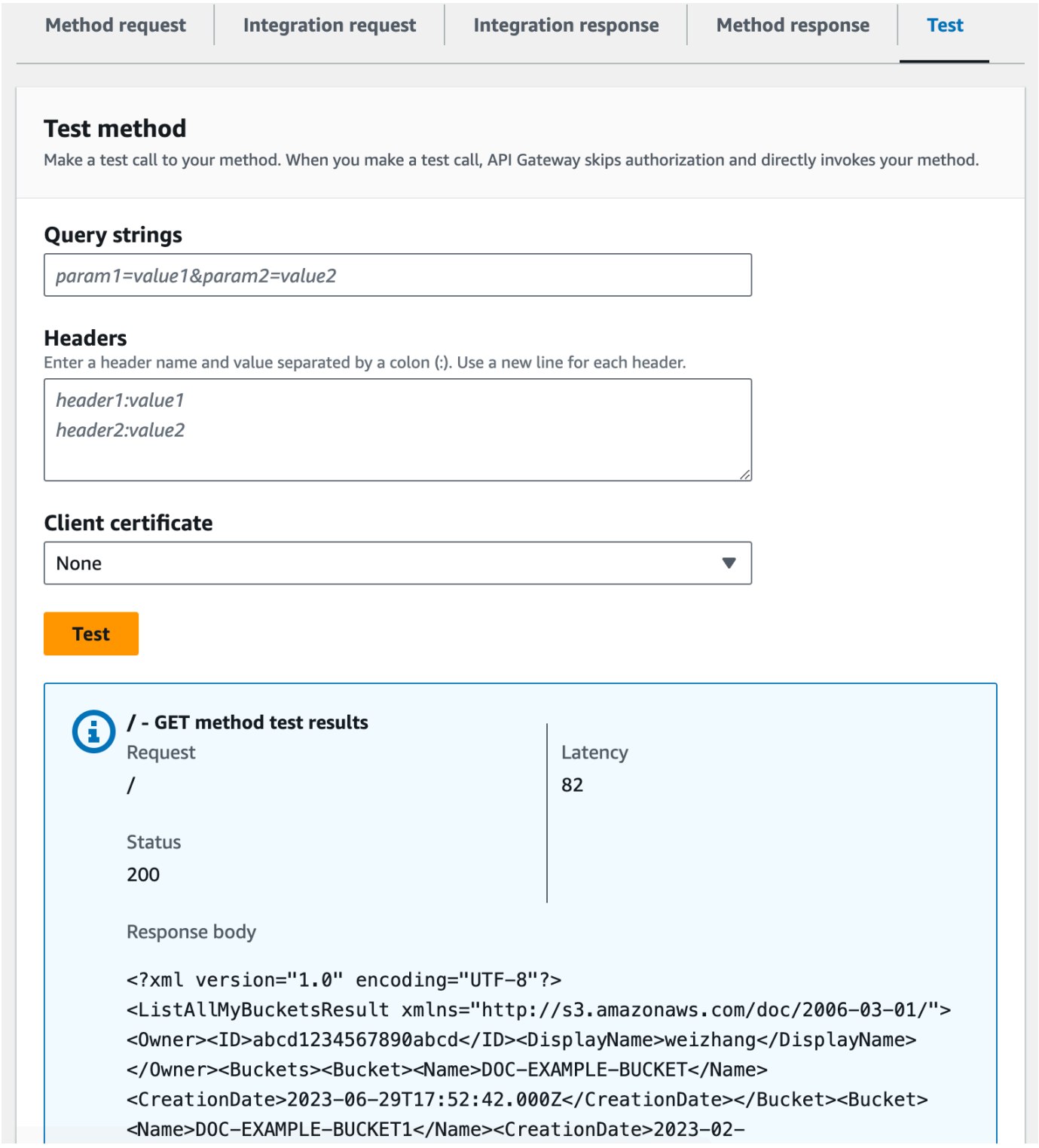


6. 在整合回應索引標籤上的整合回應下，選擇建立回應。
7. 對於 HTTP 狀態 regex，輸入 **4\d{2}**。這會將所有 4xx HTTP 回應狀態碼對應到方法回應。
8. 針對方法回應狀態碼，選取 **400**。
9. 選擇建立。
10. 重複下列步驟，為 500 方法回應建立整合回應。對於 HTTP 狀態 regex，輸入 **5\d{2}**。

作為一個好的做法，您可以測試到目前為止配置的 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：


Method request
Integration request
Integration response
Method response
Test
Test method
Make a test call to your method. When you make a test call, API Gateway skips authorization and directly invokes your method.
Query strings
param1=value1&param2=value2
Headers
Enter a header name and value separated by a colon (:). Use a new line for each header.
header1:value1
header2:value2
Client certificate
None
Test
/ - GET method test results
Request
/
Status
200
Latency
82
Response body
<?xml version="1.0" encoding="UTF-8"?>
<ListAllMyBucketsResult xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
<Owner><ID>abcd1234567890abcd</ID><DisplayName>weizhang</DisplayName>
</Owner><Buckets><Bucket><Name>DOC-EXAMPLE-BUCKET</Name>
<CreationDate>2023-06-29T17:52:42.000Z</CreationDate></Bucket><Bucket>
<Name>DOC-EXAMPLE-BUCKET1</Name><CreationDate>2023-02-

公開 API 方法來存取 Amazon S3 儲存貯體

若要使用 Amazon S3 儲存貯體，請公開/{資料夾} 資源上的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. 在中 AWS 區域，選取您 AWS 區域 在哪裡建立 Amazon S3 儲存貯體。
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}**

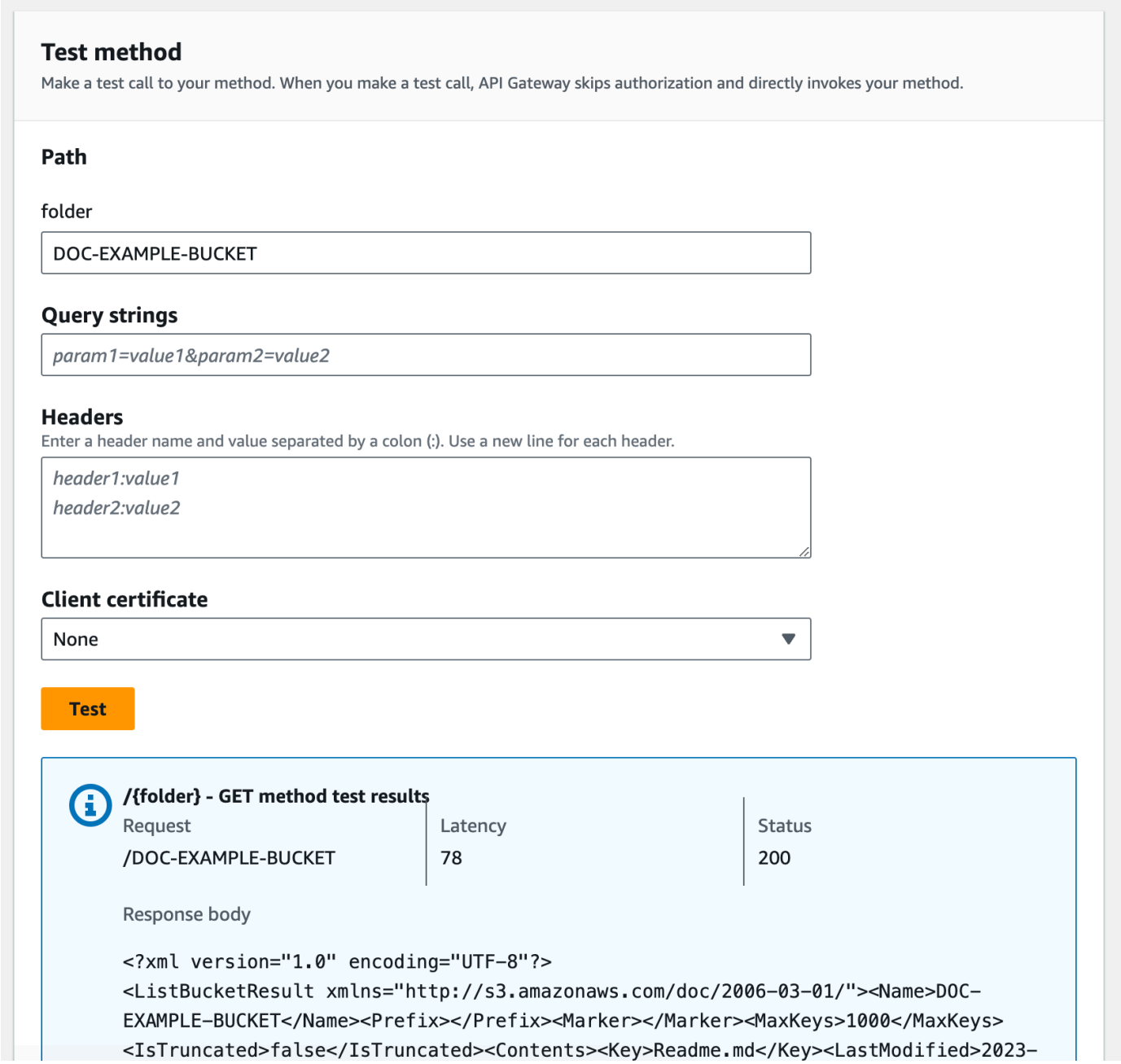
1. 在整合請求索引標籤上，於整合請求設定下，選擇編輯。
2. 選擇 URL 路徑參數，然後選擇新增路徑參數。
3. 針對名稱，輸入 **bucket**。
4. 對於對應來源，輸入 **method.request.path.folder**。
5. 選擇儲存。

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. 在中 AWS 區域，選取您 AWS 區域 在哪裡建立 Amazon S3 儲存貯體。
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 路徑參數對應至整合請求的 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}**

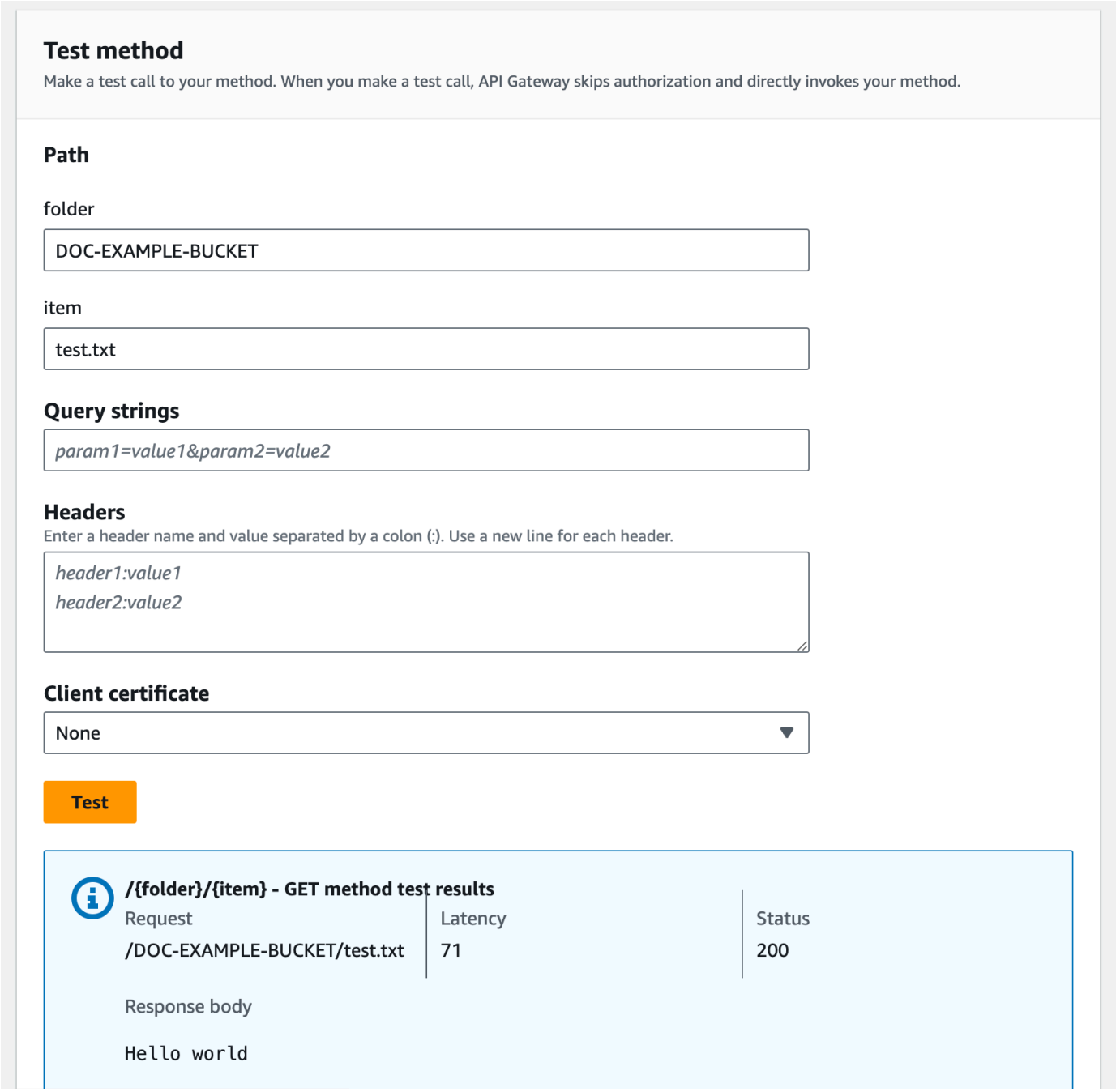
1. 在整合請求索引標籤上，於整合請求設定下，選擇編輯。
2. 選擇 URL 路徑參數。
3. 選擇新增路徑參數。
4. 針對名稱，輸入 **bucket**。

5. 對於對應來源，輸入 **method.request.path.folder**。
6. 選擇新增路徑參數。
7. 針對名稱，輸入 **object**。
8. 對於對應來源，輸入 **method.request.path.item**。
9. 選擇儲存。

測試 **/{folder}/{object} 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 (DOC-EXAMPLE-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. 將受影響文件的媒體類型註冊到 API 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，請參閱整合傳遞行為和選取 VTL 對應範本。

承載大小上限為 10 MB。請參閱設定和執行 REST API 的 API Gateway 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，請參閱使用 OpenAPI 設定 REST API。

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

要了解如何使用支持 AWS IAM 授權的郵差調用此 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，我們現在會示範如何使用 Postman 呼叫 API，它支援 AWS IAM 授權。

使用 Postman 呼叫 Amazon S3 代理 API

1. 部署或重新部署 API。請記下 API 的基本 URL，而此 API 顯示在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`。分別在和輸入欄位中SecretKey輸入 IAM 使用者的存取金鑰 ID AccessKey和秘密存取金鑰。在 [AWS 地區] 文字方塊中輸入 API 部署的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 `apig-demo-5` 的儲存貯體新增至 *`{region}`* 區域中的 Amazon S3 帳戶：

   > **Note**
   >
   > 請確定儲存貯體名稱必須為全域唯一。

   a. 從下拉式方法清單中選擇 PUT，並輸入方法 URL (`https://`*`api-id`*`.execute-api.`*`aws-region`*`.amazonaws.com/`*`stage`*`/`*`folder-name`*

b. 將 Content-Type 標頭值設定為 application/xml。您可能需要先刪除任何現有標頭，再設定內容類型。

c. 選擇內文選單項目，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 **apig-demo-5** 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/apig-demo-5/Readme.txt HTTP/1.1
Host: 9gn28ca086.execute-api.{region}.amazonaws.com
Content-Type: application/xml
X-Amz-Date: 20161015T062647Z
Authorization: AWS4-HMAC-SHA256 Credential=access-key-id/20161015/{region}/execute-
api/aws4_request, SignedHeaders=content-length;content-type;host;x-amz-date,
 Signature=ccadb877bdb0d395ca38cc47e18a0d76bb5eaf17007d11e40bf6fb63d28c705b
Cache-Control: no-cache
Postman-Token: 6135d315-9cc4-8af8-1757-90871d00847e

Hello, World!
```

如果一切順利，您應該會收到含有空白承載的 200 OK 回應。

6. 若要取得剛剛新增至 Readme.txt 儲存貯體之 apig-demo-5 檔案的內容，請執行 GET 請求，如下所示：

```
GET /S3/apig-demo-5/Readme.txt HTTP/1.1
Host: 9gn28ca086.execute-api.{region}.amazonaws.com
Content-Type: application/xml
X-Amz-Date: 20161015T063759Z
Authorization: AWS4-HMAC-SHA256 Credential=access-key-id/20161015/{region}/
execute-api/aws4_request, SignedHeaders=content-type;host;x-amz-date,
 Signature=ba09b72b585acf0e578e6ad02555c00e24b420b59025bc7bb8d3f7aed1471339
Cache-Control: no-cache
Postman-Token: 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 apig-demo-5 儲存貯體中的項目，請提交下列請求：

```
GET /S3/apig-demo-5 HTTP/1.1
Host: 9gn28ca086.execute-api.{region}.amazonaws.com
Content-Type: application/xml
X-Amz-Date: 20161015T064324Z
Authorization: AWS4-HMAC-SHA256 Credential=access-key-id/20161015/{region}/
execute-api/aws4_request, SignedHeaders=content-type;host;x-amz-date,
 Signature=4ac9bd4574a14e01568134fd16814534d9951649d3a22b3b0db9f1f5cd4dd0ac
Cache-Control: no-cache
Postman-Token: 9c43020a-966f-61e1-81af-4c49ad8d1392
```

如果成功，除非先將更多檔案新增至儲存貯體再提交此請求，否則您應該會收到 200 OK 回應，而其 XML 承載顯示所指定儲存貯體中的單一項目。

```
<?xml version="1.0" encoding="UTF-8"?>
<ListBucketResult xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
    <Name>apig-demo-5</Name>
    <Prefix></Prefix>
    <Marker></Marker>
    <MaxKeys>1000</MaxKeys>
    <IsTruncated>false</IsTruncated>
    <Contents>
        <Key>Readme.txt</Key>
        <LastModified>2016-10-15T06:26:48.000Z</LastModified>
        <ETag>"65a8e27d8879283831b664bd8b7f0ad4"</ETag>
        <Size>13</Size>
        <Owner>
            <ID>06e4b09e9d...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：在 API Gateway 中建立 REST API 做為 Amazon Kinesis 代理

本頁面說明如何建立和設定與 AWS 類型整合的 REST API，以存取 Kinesis。

> **Note**
>
> 若要整合 API Gateway API 與 Kinesis，您必須選擇可使用 API Gateway 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 的`/streams`資源上公開 HTTP GET 方法，並將該方法與 Kinesis 中的ListStreams動作整合，以列出呼叫者帳戶中的串流。
- 在 API 的`/streams/{stream-name}`資源上公開 HTTP POST 方法，並將該方法與 Kinesis 中的CreateStream動作整合在一起，以便在呼叫者的帳戶中建立具名串流。
- 在 API 的`/streams/{stream-name}`資源上公開 HTTP GET 方法，並將該方法與 Kinesis 中的DescribeStream動作整合，以描述呼叫者帳戶中的具名串流。
- 在 API 的`/streams/{stream-name}`資源上公開 HTTP DELETE 方法，並將該方法與 Kinesis 中的DeleteStream動作整合，以刪除呼叫者帳戶中的串流。
- 在 API 的`/streams/{stream-name}/record`資源上公開 HTTP PUT 方法，並將該方法與 Kinesis 中的PutRecord動作整合。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，並將該方法與 Kinesis 中的PutRecords動作整合。這可讓用戶端將資料記錄清單新增至具名串流。

- 在 API 的`/streams/{stream-name}/records`資源上公開 HTTP GET 方法，並將該方法與 Kinesis 中的GetRecords動作整合。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 的`/streams/{stream-name}/sharditerator`資源上公開 HTTP GET 方法，並將該方法與 Kinesis 中的GetShardIterator動作整合。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「使用 OpenAPI 設定 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。

若要建立服 AWS 務代理執行角色

1. 登入 AWS Management Console 並開啟身分與存取權管理主控台，網址為 https://console.aws.amazon.com/iam/。
2. 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]，然後選取 [允許 API Gateway 將記錄檔推送至 CloudWatch 記錄]。
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。

創建一個 API 作為 Kinesis 代理

使用下列步驟可在 API Gateway 主控台中建立 API。

若要建立 API 做為 Kinesis 的 AWS 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. 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. 在中 AWS 區域，選取 AWS 區域 您建立 Kinesis 串流的位置。

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. 針對名稱，輸入 **`Content-Type`**。

c. 對於對應來源，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。

要ListStreams求會採用下列 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。

我們公開GET to-be-named `Stream`資源上的POST、和 `Delete` 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 /stream 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/{串流名稱} 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. 在中 AWS 區域，選取 AWS 區域 您建立 Kinesis 串流的位置。
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` 方法的請求參數對應相同。
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/{串流名稱} 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. 在中 AWS 區域，選取 AWS 區域 您建立 Kinesis 串流的位置。
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` 方法的請求參數對應相同。
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/{串流名稱} 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. 在中 AWS 區域，選取 AWS 區域 您建立 Kinesis 串流的位置。
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 方法的請求參數對應相同。

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。新增資料記錄涉及呼叫 Kinesis 中的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。

在 Kinesis 中讀取數據相當於調用動GetRecords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：

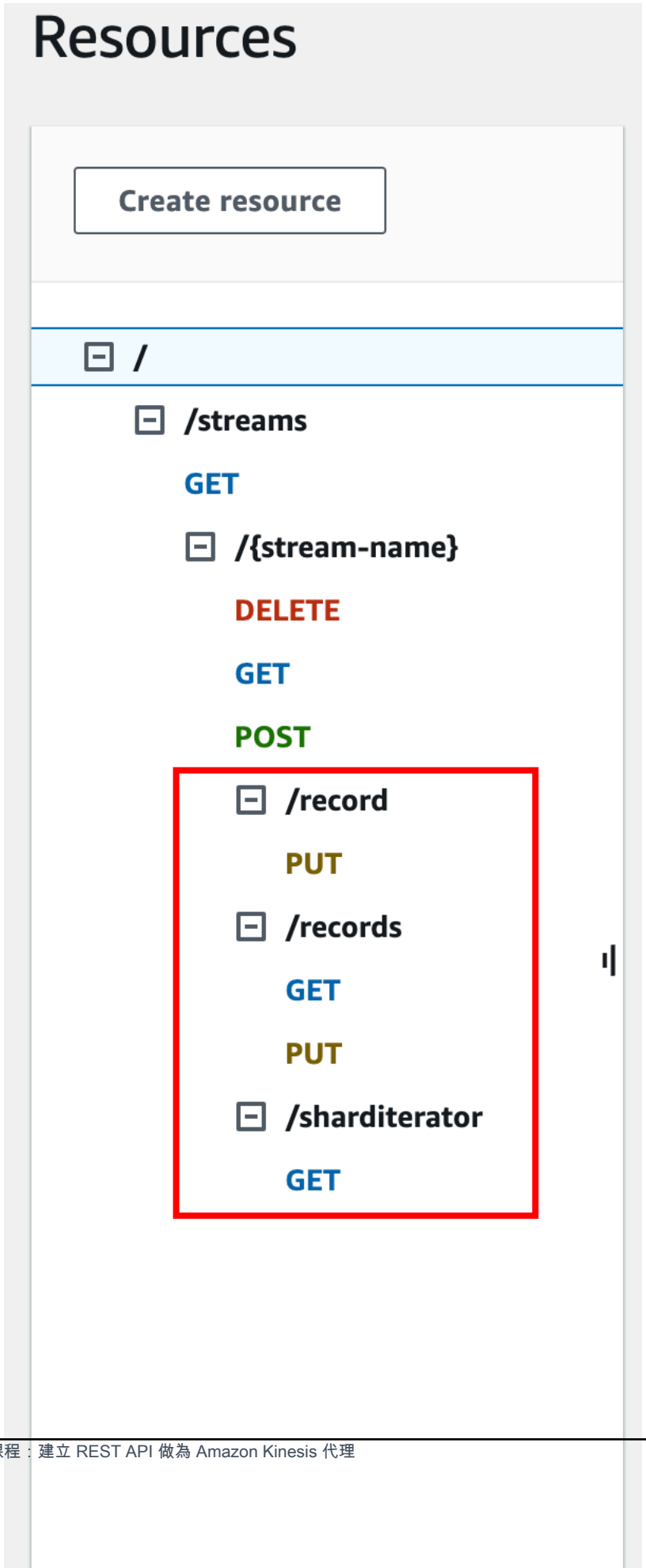
```
POST /?Action=GetShardIterator HTTP/1.1
Host: kinesis.region.domain
Authorization: AWS4-HMAC-SHA256 Credential=..., ...
...
Content-Type: application/x-amz-json-1.1
Content-Length: PayloadSizeBytes

{
    "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/{串流名稱} 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：


Resources
Create resource
/
/streams
GET
/{stream-name}
DELETE
GET
POST
/record
PUT
/records
GET
PUT
/sharditerator
GET

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 /記錄，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. 在中 AWS 區域，選取 AWS 區域 您建立 Kinesis 串流的位置。
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` 方法的請求參數對應相同。

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. 在中 AWS 區域，選取 AWS 區域 您建立 Kinesis 串流的位置。
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 方法的請求參數對應相同。

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/分析器資源，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. 在中 AWS 區域，選取 AWS 區域 您建立 Kinesis 串流的位置。
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。

    動GetShardIterator作需要輸入 ShardId值。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. 針對名稱，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. 在中 AWS 區域，選取 AWS 區域 您建立 Kinesis 串流的位置。
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 要求標頭。

GetRecords 動作需要 ShardIterator 值的輸入。要傳遞一個客戶端提供的ShardIterator值，我們添加一個Shard-Iterator頭參數的方法請求。

13. 選擇新增標頭。
14. 針對名稱，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 SDK 或建立邊緣最佳化 API AWSAWS CLI

下列教學課程說明如何建立支援`GET /pets`和方`GET /pets/{petId}`法的 PetStore API。這些方法與 HTTP 端點集成在一起。您可以遵循本教學課程使用 SDK JavaScript、Python 的 SDK (博托 3) 或. AWS 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

- 建立 (_R)
- 建立資源 (_I)
- 推入方法

- 方法回應
- 整合
- 整合 _ 回應
- 建立部署 (_D)

AWS CLI

- create-rest-api
- 創建資源
- 放入方法
- put-method-response
- 投入整合
- put-integration-response
- 建立部署

如需 JavaScript v3 適用之 AWS SDK 的詳細資訊，請參閱 AWS SDK 的用途為何 JavaScript？ 。如需有關適 SDK for Python (Boto3) 資訊，請參閱。AWS SDK for Python (Boto3)如需有關的詳細資訊 AWS CLI，請參閱什麼是 AWS CLI? 。

設定邊緣最佳化 PetStore API

在本教學課程中，範例指令會將預留位置值用於值 ID (例如 API ID 和資源 ID)。完成自學課程時，請用您自己的值取代這些值。

若要使用 SDK 設定邊緣最佳化 PetStore API AWS

1. 使用下列範例來建立`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)
}
})();
```

成功的呼叫會在輸出中傳回 API ID 和 API 的根資源 ID，如下所示：

```
{
  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)
```

成功的呼叫會在輸出中傳回 API ID 和 API 的根資源 ID，如下所示：

```
{'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 具有的 API ID 以abcd1234及的根資源識別碼efg567。您可以在 API 的設置中使用這些值。

2. 接下來，在根下附加一個子資源，並將其指定RootResourceId為parentId屬性值。使用以下示例為您的 API 創建/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)
}
})();
```

成功的呼叫會在輸出中傳回有關資源的資訊，如下所示：

```
{
    "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)
```

成功的呼叫會在輸出中傳回有關資源的資訊，如下所示：

```
{'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資源具有的資源 ID aaa111。您可以在 API 的設置中使用此值。

3. 接下來，在資源下附加子資/pets源。此資源/{petId}具有 {petId} .To 使路徑部分成為路徑參數的 path 參數，請將其括在一對大括號中。{ }使用以下示例為您的 API 創建/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)
}
})();
```

成功的呼叫會在輸出中傳回有關資源的資訊，如下所示：

```
{
    "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)
```

成功的呼叫會在輸出中傳回有關資源的資訊，如下所示：

```
{'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}'
```

如果成功，此命令會傳回下列回應：

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

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

4. 在下面的兩個步驟中，您將 HTTP 方法添加到您的資源。在本自學課程中，您將方法設定為設定為，將方法設定`authorization-type`為具有開放存取權`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”。

   下列範例會在`/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)
}
})();
```

成功的呼叫會傳回下列輸出：

```
{
    "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)
```

成功的呼叫會傳回下列輸出：

```
{'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
```

以下是此命令的成功輸出：

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

5. 下列範例會在/pets/{petId}資源上新增 GET HTTP 方法，並將requestParameters屬性設定為將用戶端提供的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)
}
})();
```

成功的呼叫會傳回下列輸出：

```
{
    "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)
```

成功的呼叫會傳回下列輸出：

```
{'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
```

以下是此命令的成功輸出：

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

6. 使用下列範例來新增方法的 200 OK 方GET /pets法回應：

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)
}
})();
```

成功的呼叫會傳回下列輸出：

```
{
    "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)
```

成功的呼叫會傳回下列輸出：

```
{'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. 使用下列範例來新增方法的 200 OK 方GET /pets/{petId}法回應：

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)
```

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

成功的呼叫會傳回下列輸出：

```
{
    "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)
```

成功的呼叫會傳回下列輸出：

```
{'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. 下列範例會針對具有 HTTP 端點的方GET /pets法設定整合。HTTP 端點是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)
}
})();
```

成功的呼叫會傳回下列輸出：

```
{
    "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)
```

成功的呼叫會傳回下列輸出：

```
{'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. 下列範例會針對具有 HTTP 端點的方GET /pets/{petId}法設定整合。HTTP 端點是http://petstore-demo-endpoint.execute-api.com/petstore/pets/{id}。在此步驟中，您會將 path 參數petId對應至的整合端點路徑參數id。

   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)
```

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

成功的呼叫會傳回下列輸出：

```
{
    "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)
```

成功的呼叫會傳回下列輸出：

```
{'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. 下列範例會新增整合的GET /pets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)
}
})();
```

成功的呼叫會傳回下列輸出：

```
{
    "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)
```

成功的呼叫會傳回下列輸出：

```
{'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. 下列範例會新增整合的GET /pets/{petId}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)
}
})();
```

成功的呼叫會傳回下列輸出：

```
{
    "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)
```

成功的呼叫會傳回下列輸出：

```
{'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 可以查詢 PetStore 網站上的可用寵物，並檢視指定識別碼的個別寵物。在您的客戶呼叫 API 之前，您必須先部署它。我們建議您在部署 API 之前先對其進行測試。

12. 下列範例會測試方GET /pets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. 下列範例會測試 a 為 3 petId 的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 之後，您可以將其部署到階段。

14. 下列範例會將您的 API 部署到名為test的階段。當您將 API 部署到階段時，API 呼叫者可以叫用您的 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"
}
```

客戶現在可以調用您的 API。您可以通過在瀏覽器中輸入 `https://abcd1234.execute-api.us-west-2.amazonaws.com/test/pets` URL 來測試此 API，並`abcd1234`用 API 的標識符代替。

有關如何使用 SDK 建立或更新 API 的更多範例 AWS CLI，請參閱使用 AWS SDK 進行 API Gateway 的動作。 AWS

自動化 API 的設定

而不是創建您的 API step-by-step，您可以使用 OpenAPI 或地形來創建您的 API 來自動創建和清理 AWS 資源。 AWS CloudFormation

OpenAPI 定义

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 AWS 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'
```

地形配置

如需有關地形的詳細資訊，請參閱地形。

```
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
}
```

教學課程：建置私有的 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。

此堆疊需要約 30 分鐘的時間完成。首先，您可以使用 AWS CloudFormation 範本建立 Amazon VPC、VPC 端點和 AWS Lambda 函數，以及啟動用來測試 API 的 Amazon EC2 執行個體。接下來，您可以使 AWS Management Console 用建立私有 API 並附加資源策略，該策略僅允許從您的 VPC 端點進行存取。最後，測試您的 API。

若要完成本教學課程，您需要一個 AWS 帳戶和具有主控台存取權限的 AWS Identity and Access Management 使用者。如需詳細資訊，請參閱 必要條件。

在本教學課程中，您需使用 AWS Management Console。如需建立此 API 和所有相關資源的 AWS CloudFormation 範本，請參閱範本。

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：部署您的 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：清除
- 後續步驟：自動化 AWS CloudFormation

步驟 1：建立相依性

下載並解壓縮此 AWS CloudFormation 模板。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 AWS CloudFormation 堆疊的步驟

1. 請在以下位置開啟 AWS CloudFormation 主控台。 https://console.aws.amazon.com/cloudformation
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. 對於功能，請確認 AWS CloudFormation 可以在您的帳戶中建立 IAM 資源。

9. 選擇提交。

AWS CloudFormation 佈建 API 的依賴關係，這可能需要幾分鐘的時間。當 AWS CloudFormation 堆疊狀態為「建立 _ 完成」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. 對於 VPC 端點識別碼，請從堆疊的輸出中輸入 VPC 端點識別碼。 AWS CloudFormation
6. 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. 選取 / 資源，然後選擇建立方法。
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. 對於 Lambda 函數，請在步驟 1 中選擇您使用 AWS 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。使用堆疊輸出中的 VPC 端點識別碼取代 v *PCE* ID。 AWS CloudFormation

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

5. 選擇儲存變更。

步驟 5：部署您的 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private-api-tutorial使用 AWS CloudFormation 範本建立的名稱為執行個體。
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 REST API 和 AWS 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 AWS CloudFormation 堆疊的步驟

1. 請在以下位置開啟 AWS CloudFormation 主控台。 https://console.aws.amazon.com/cloudformation
2. 選取您的 AWS CloudFormation 堆疊。
3. 選擇刪除，然後確認您的選擇。

後續步驟：自動化 AWS CloudFormation

您可以自動建立和清理本自學課程中涉及的所有 AWS 資源。如需完整的 AWS 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

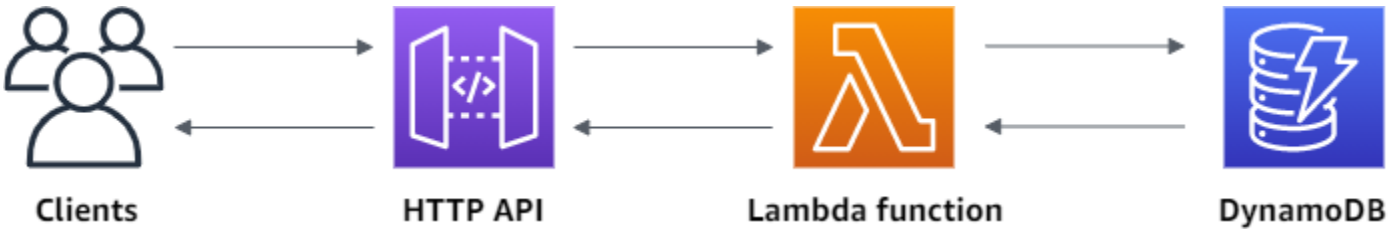
- 教學課程：使用 Lambda 和 DynamoDB 建置 CRUD API
- 教學課程：透過私有整合至 Amazon ECS 服務來建置 HTTP API

教學課程：使用 Lambda 和 DynamoDB 建置 CRUD 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，您可以使用 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。



要完成這個練習，您需要一個 AWS 帳戶和一個具有控制台訪問權限的 AWS Identity and Access Management 用戶。如需詳細資訊，請參閱 必要條件。

在本教學課程中，您需使用 AWS Management Console。如需建 AWS SAM 立此 API 及所有相關資源的範本，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：清除
- 後續步驟：使用 AWS SAM 或自動化 AWS 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 (分區索引鍵)，請輸入 **`id`**。
5. 選擇 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。

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 策略範本建立新角色。
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，並以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. 選擇 Next (下一步)。
5. 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。
6. 檢閱 API Gateway 為您建立的階段，然後選擇 Next (下一步)。
7. 選擇 Create (建立)。

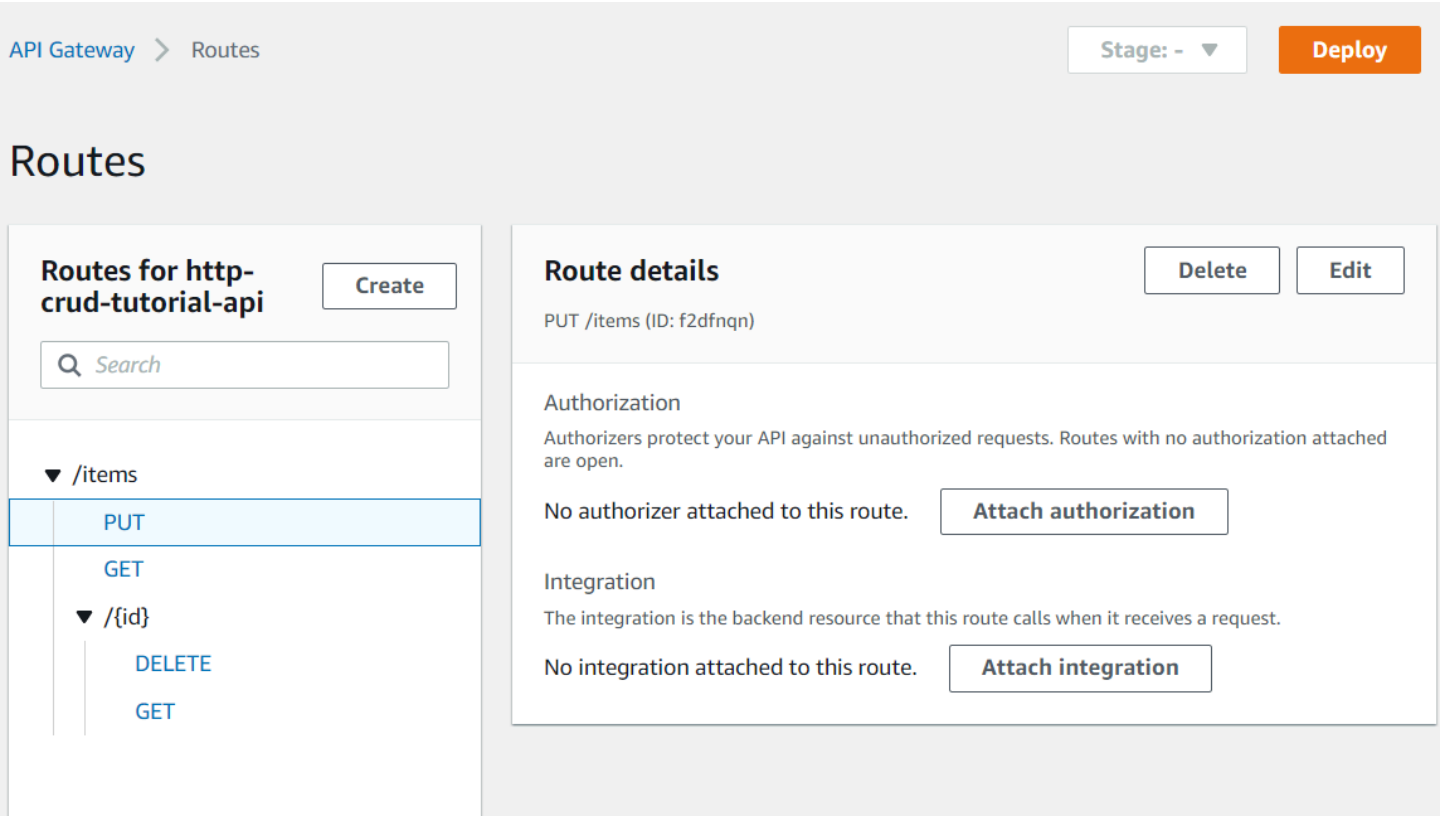
步驟 4：建立路由

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)。

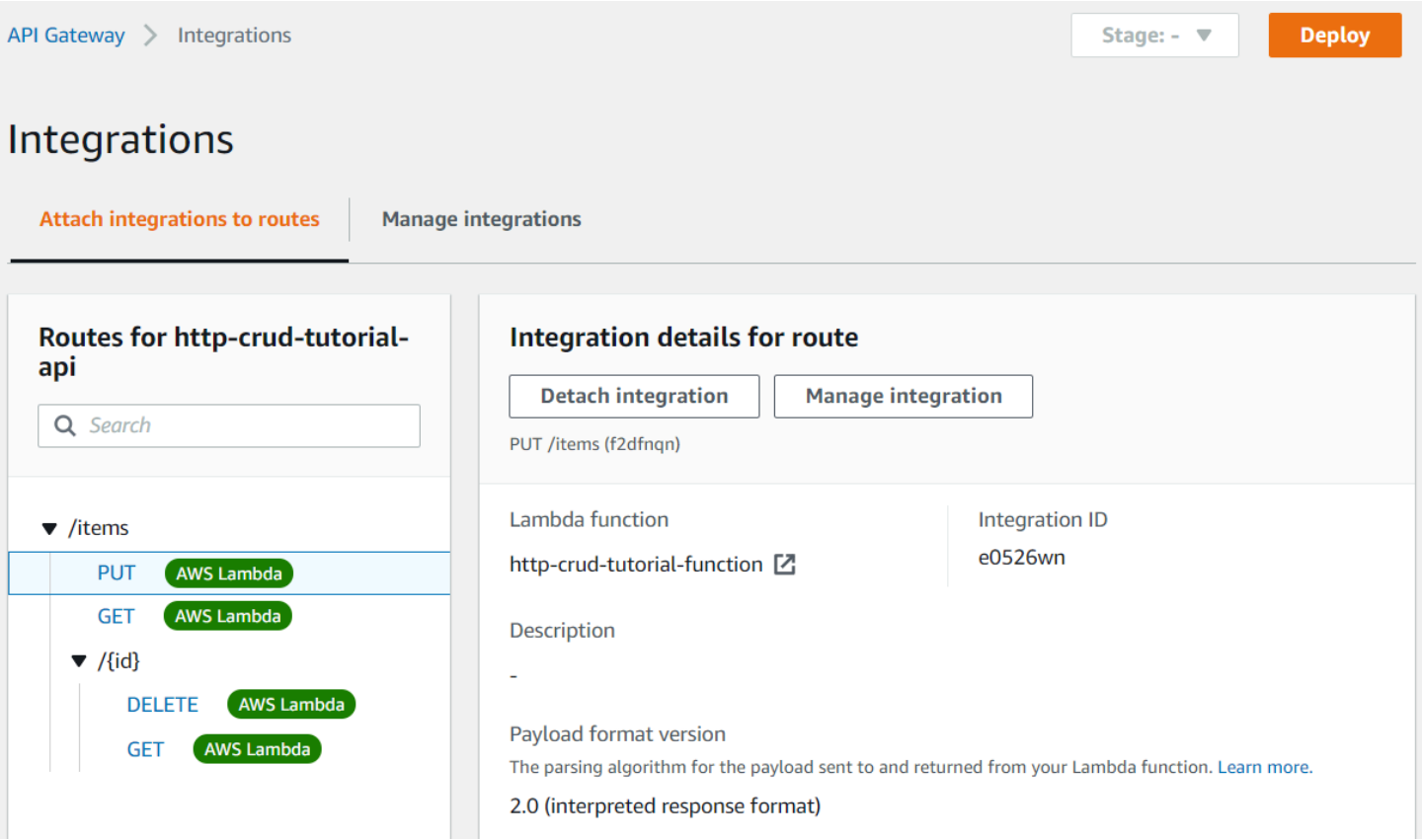
步驟 6：將整合連接至路由

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。

所有路由都會顯示已貼附 AWS Lambda 整合。



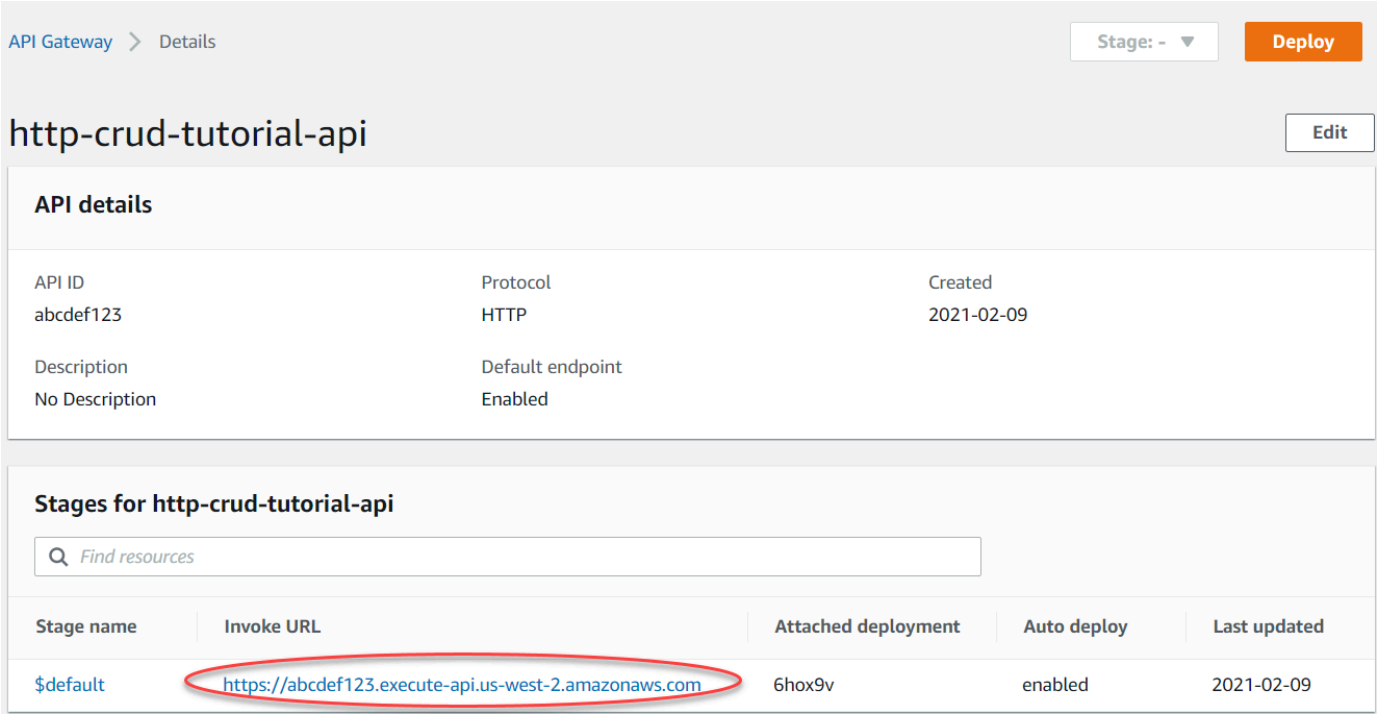
現在您擁有包含路由和整合的 HTTP API，您可以測試您的 API 了。

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：清除

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 主 CloudWatch 控台中，開啟日誌群組頁面。
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 AWS CloudFormation

您可以使用 AWS CloudFormation 或來自動建立和清理 AWS 資源 AWS SAM。如需本教學課程的 AWS SAM 範本範例，請參閱 `template.yaml`。

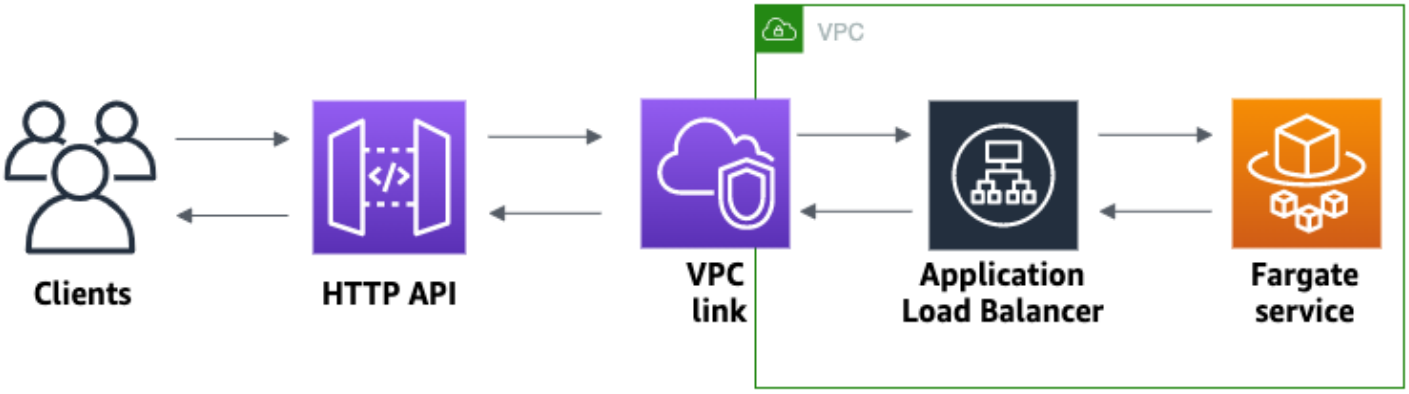
如需 AWS CloudFormation 範本，請參閱AWS CloudFormation 範例範本。

教學課程：透過私有整合至 Amazon ECS 服務來建置 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，您可以使用 AWS 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，您需要一個 AWS 帳戶和具有主控台存取權限的 AWS Identity and Access Management 使用者。如需詳細資訊，請參閱 必要條件。

在本教學課程中，您需使用 AWS Management Console。如需建立此 API 和所有相關資源的 AWS CloudFormation 範本，請參閱範本。

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：自動化 AWS 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 AWS CloudFormation 範本，以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 AWS CloudFormation 堆疊的步驟

1. 請在以下位置開啟 AWS CloudFormation 主控台。 https://console.aws.amazon.com/cloudformation
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，請確認 AWS CloudFormation 可以在您的帳戶中建立 IAM 資源。
9. 選擇提交。

AWS CloudFormation 提供 ECS 服務，這可能需要幾分鐘的時間。當 AWS CloudFormation 堆疊的狀態為「建立 _ 完成」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. 在主導航窗格上，選擇 VPC 鏈接，然後選擇創建。

   您可能需要選擇功能表圖示才能開啟主導覽窗格。
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。名稱的開頭應為PrivateIntegrationsStack。
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. 選擇 Create (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. 選擇 Next (下一步)。
5. 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。
6. 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)。
7. 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)。

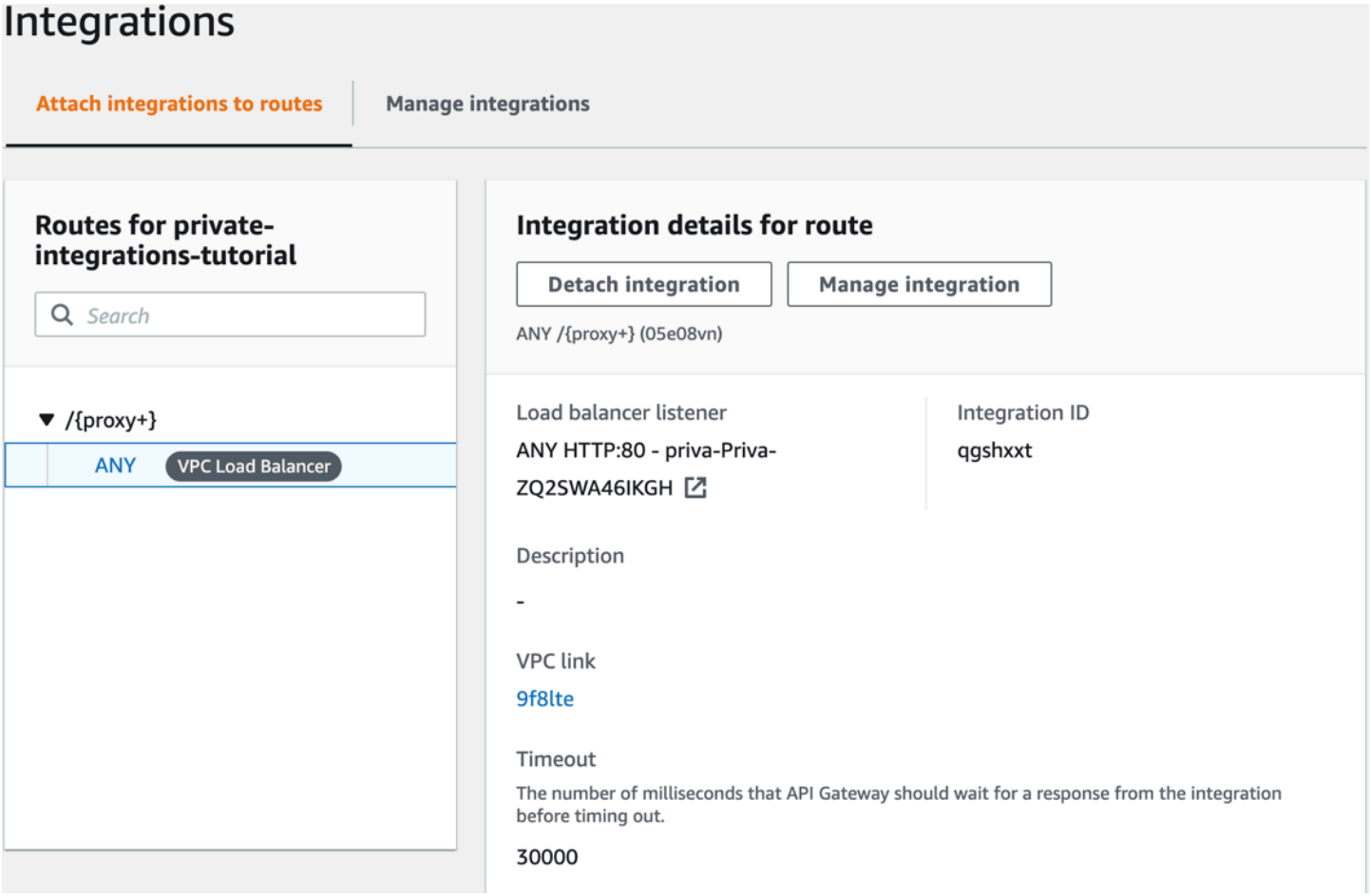
步驟 5：建立整合

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 AWS 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。



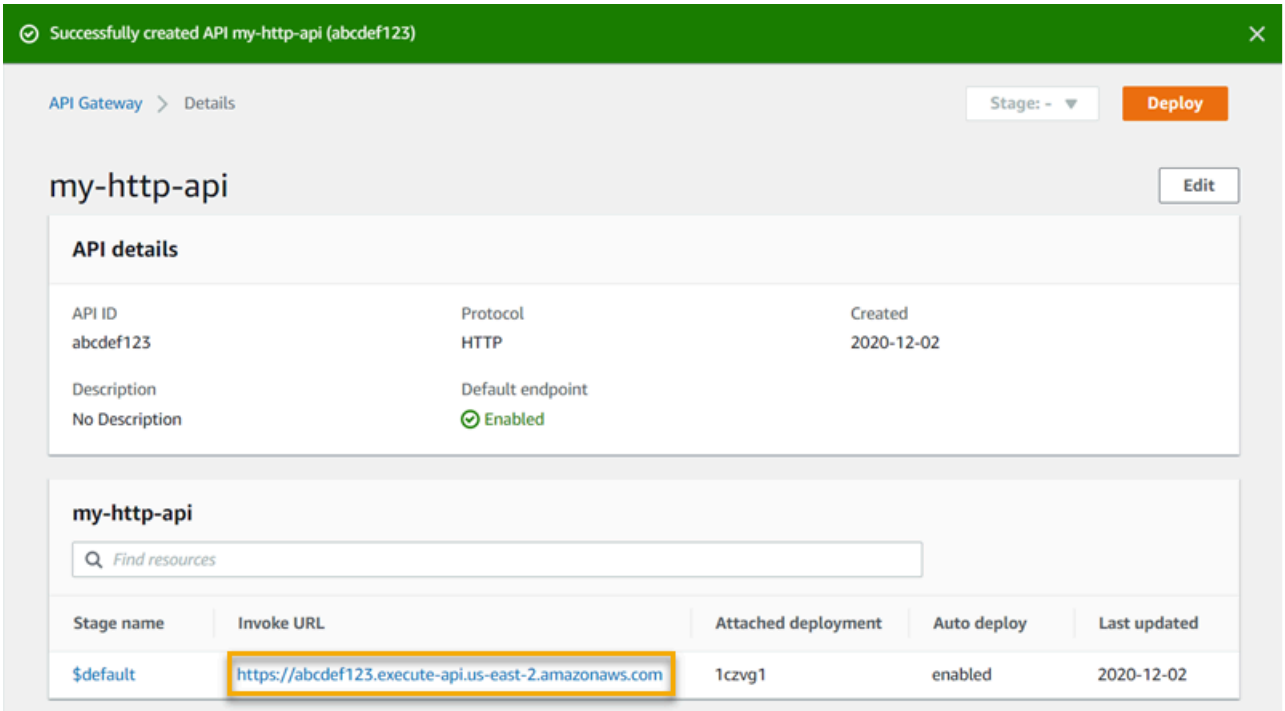
現在您已準備好測試 API。

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 人雲端連結、 AWS 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 AWS CloudFormation 堆疊的步驟

1. 請在以下位置開啟 AWS CloudFormation 主控台。 https://console.aws.amazon.com/cloudformation
2. 選取您的 AWS CloudFormation 堆疊。
3. 選擇刪除，然後確認您的選擇。

後續步驟：自動化 AWS CloudFormation

您可以自動建立和清理本自學課程中涉及的所有 AWS 資源。如需完整的 AWS CloudFormation 範本範例，請參閱 template.yaml。

Amazon API Gateway WebSocket API 教程

下列教學課程提供實作練習，協助您瞭解 API Gateway WebSocket API。

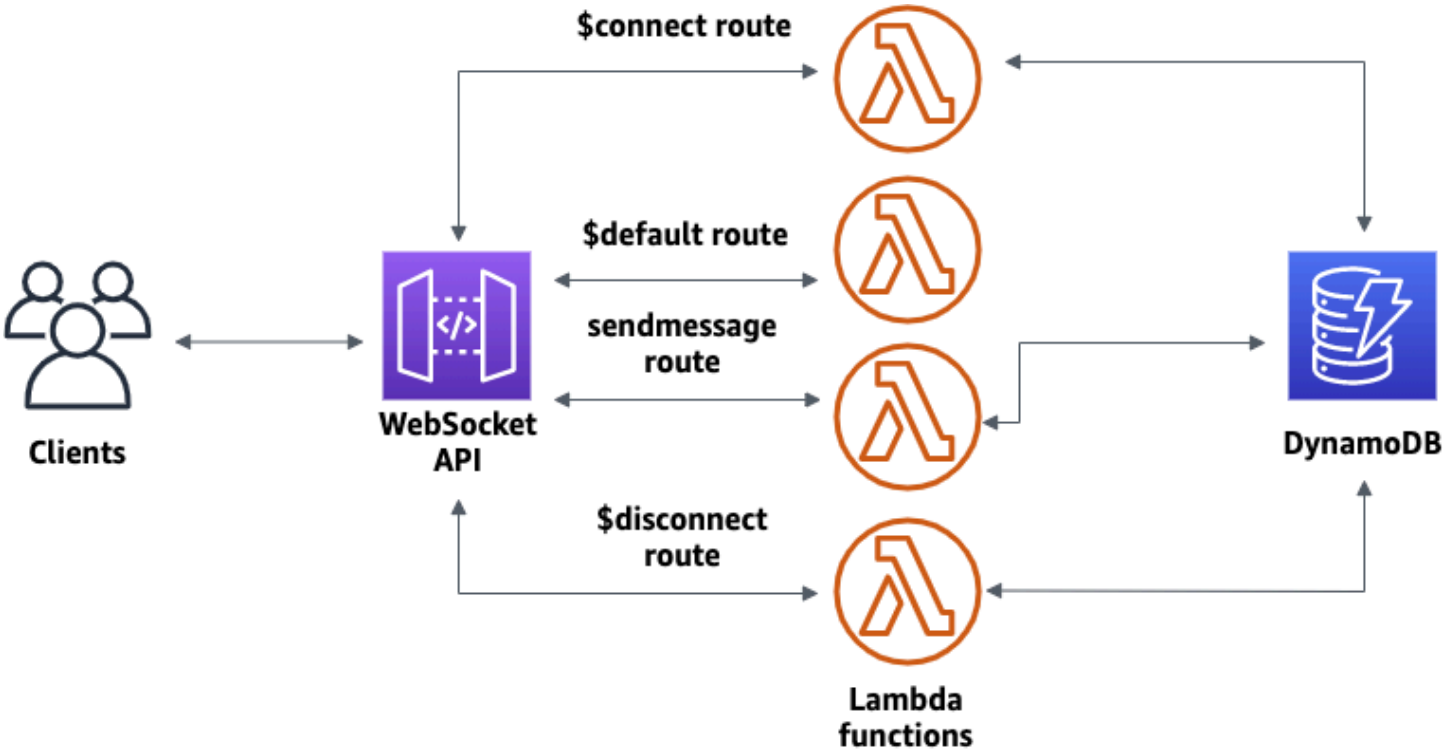
主題

- 教學課程：使用 WebSocket API、Lambda 和 DynamoDB 建置無伺服器聊天應用程式
- 教學課程：使用三種整合類型建置無伺服器應用程式

教學課程：使用 WebSocket API、Lambda 和 DynamoDB 建置無伺服器聊天應用程式

在本教學中，您將使用 WebSocket API 建立無伺服器聊天應用程式。使用 WebSocket API，您可以支持客戶端之間的雙向通信。客戶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 AWS CloudFormation 範本建立可處理 API 請求的 Lambda 函數，以及用來儲存用戶端 ID 的 DynamoDB 資料表。然後，您將使用 API Gateway 主控台建立與 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，您需要一個 AWS 帳戶和具有主控台存取權限的 AWS Identity and Access Management 使用者。如需詳細資訊，請參閱 必要條件。

您還需要 `wscat` 來連線到 API。如需詳細資訊，請參閱 the section called “用wscat於連接到 WebSocket API 並向其發送消息”。

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：自動化 AWS 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。 AWS 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 AWS CloudFormation 堆疊的步驟

1. 開啟主 AWS CloudFormation 控台，網址為 https://console.aws.amazon.com/cloudformation。
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. 對於功能，請確認 AWS CloudFormation 可以在您的帳戶中建立 IAM 資源。
9. 選擇提交。

AWS CloudFormation 規定範本中指定的資源。完成資源佈建可能需要幾分鐘的時間。當 AWS CloudFormation 堆疊的狀態為「建立 _ 完成」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 來處理用戶端連線，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 的步驟

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，選擇構建。
3. 對於 API name (API 名稱)，輸入 **`websocket-chat-app-tutorial`**。
4. 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。
5. 選擇下一步。
6. 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。
7. 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。

8. 選擇下一步。
9. 在 Attach integrations (連接整合) 下，為每個路由和 Integration type (整合類型) 選擇 Lambda。

   對於 Lambda，請選擇您 AWS CloudFormation 在步驟 1 中建立的對應 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。
10. 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。
11. 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. 請記下您的 API WebSocket 網址。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 會叫用您 API 的 `$disconnect` 路由。您 API 的 `$disconnect` 路由 Lambda 整合會從 DynamoDB 中移除連線 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 AWS CloudFormation 堆疊和 WebSocket API。

若要刪除 WebSocket API 的步驟

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 AWS CloudFormation 堆疊的步驟

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

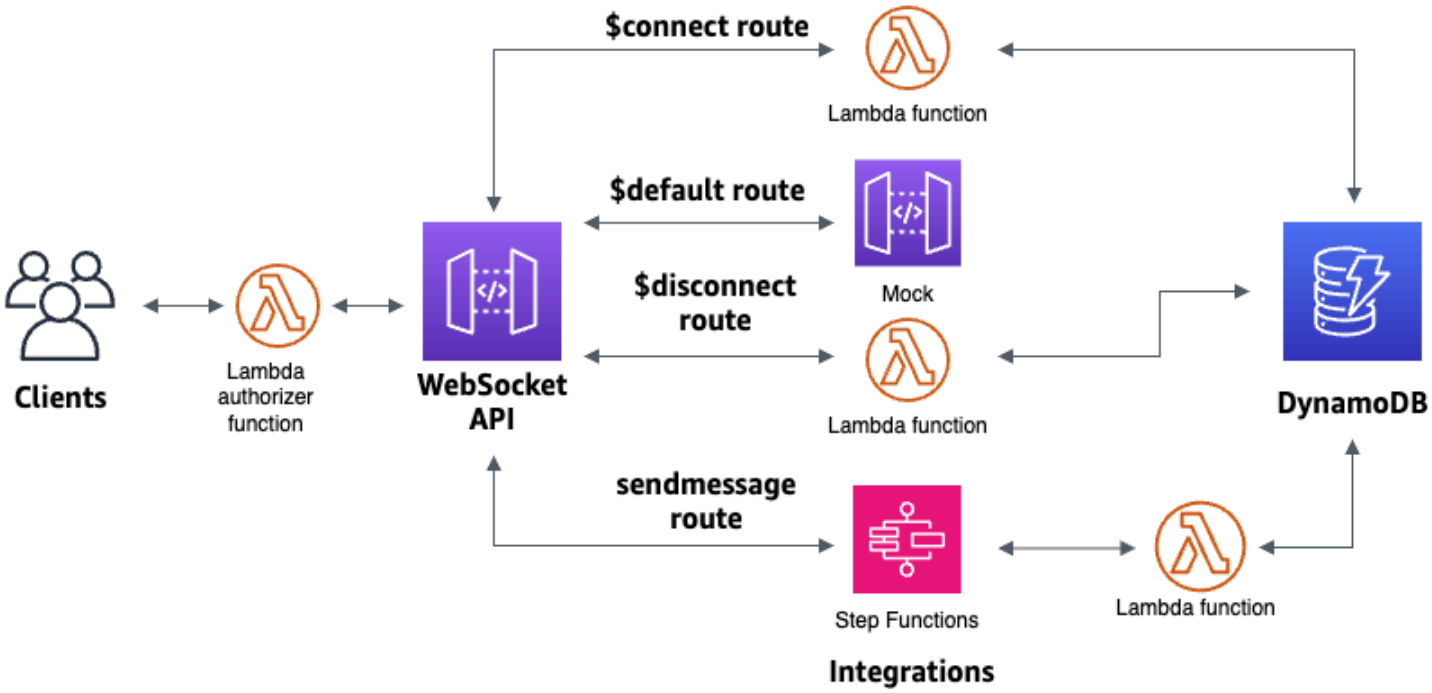
後續步驟：自動化 AWS CloudFormation

您可以自動建立和清理本自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 AWS CloudFormation 範本，請參閱聊天應用程式。

教學課程：使用三種整合類型建置無伺服器應用程式

在本教學課程中，您會使用 WebSocket API 建立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。

本教學課程說明如何向連線的用戶端廣播訊息，並包含 Lambda 授權者、模擬整合以及 Step Functions 的非代理整合範例。



使用 AWS CloudFormation 範本建立資源後，您將使用 API Gateway 主控台建立與 AWS 資源整合的 WebSocket API。您會將 Lambda 授權器附加至 API，並使用 Step Functions 建立 AWS 服務整合，以啟動狀態機器執行。Step Functions 數狀態機器將調用 Lambda 函數，該函數將消息發送給所有連接的客戶端。

建立 API 之後，您將測試與 API 的連線，並驗證訊息是否已傳送和接收。本教學課程大約需要 45 分鐘才能完成。

主題

- 必要條件
- 步驟 1：建立資源
- 步驟 2：建立 WebSocket API
- 步驟 3：建立 Lambda 授權者
- 步驟 4：創建模擬雙向集成
- 第 5 步：使用步 Step Functions 創建非代理集成
- 步驟 6：測試您的 API
- 步驟 7：清除
- 後續步驟

必要條件

您需要以下的事前準備：

- AWS 帳戶和具有控制台訪問權限的 AWS Identity and Access Management 用戶。如需詳細資訊，請參閱 必要條件。
- `wscat`連接到您的 API。如需詳細資訊，請參閱 the section called “用`wscat`於連接到 WebSocket API 並向其發送消息”。

我們建議您先完成 WebSocket 聊天應用程式教學課程，然後再開始本教學課程。若要完成 WebSocket 聊天應用程式教學課程，請參閱the section called “WebSocket 聊天應用”。

步驟 1：建立資源

下載並解壓縮的應用程式建立範本。 AWS CloudFormation您將使用此範本建立下列項目：

- 用於處理 API 請求並授權 API 存取權的 Lambda 函數。
- 用於儲存用戶端識別碼和 Lambda 授權者傳回的主要使用者識別碼的 DynamoDB 表格。
- Step Functions 狀態機將消息發送到連接的客戶端。

建立 AWS CloudFormation 堆疊的步驟

1. 請在以下位置開啟 AWS CloudFormation 主控台。 https://console.aws.amazon.com/cloudformation
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. 對於功能，請確認 AWS CloudFormation 可以在您的帳戶中建立 IAM 資源。
9. 選擇提交。

AWS CloudFormation 規定範本中指定的資源。完成資源佈建可能需要幾分鐘的時間。選擇 [輸出] 索引標籤以查看您建立的資源及其 ARN。當 AWS CloudFormation 堆疊的狀態為「建立 _ 完成」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 來處理用戶端連線，並將要求路由到您在步驟 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，選擇構建。
3. 對於 API name (API 名稱)，輸入 **`websocket-step-functions-tutorial`**。
4. 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。
5. 選擇下一步。
6. 對於預定義的路線，選擇添加 $ 連接，添加 $ 斷開，添加 $ 默認值。

   $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。當沒有其他路由匹配請求時，API Gateway 調用 $ 默認路由。建立 API 之後，您將建立自訂路由以連線至 Step Functions 式。
7. 選擇下一步。
8. 對於 $ 連接的集成，請執行以下操作：

   a. 針對整合類型，選擇 Lambda。

   b. 對於 Lambda 函數，請選擇您 AWS CloudFormation 在步驟 1 中使用建立的對應 $ 連接 Lambda 函數。Lambda 函數名稱應該以開頭**`websocket-step`**。
9. 對於 $ 斷開的集成，請執行以下操作：

   a. 針對整合類型，選擇 Lambda。

   b. 對於 Lambda 函數，請選擇您 AWS CloudFormation 在步驟 1 中使用建立的對應 $ 中斷 Lambda 函數。Lambda 函數名稱應該以開頭**`websocket-step`**。
10. 對於集成 $ 默認，選擇模擬。

    在模擬集成中，API Gateway 在沒有集成後端的情況下管理路由響應。
11. 選擇下一步。

12. 檢閱 API Gateway 為您建立的階段。根據預設，API Gateway 會建立一個名為生產的階段，並自動將您的 API 部署到該階段。選擇下一步。
13. 選擇 Create and deploy (建立和部署)。

步驟 3：建立 Lambda 授權者

若要控制 WebSocket API 的存取權限，您需要建立 Lambda 授權者。 AWS CloudFormation 範本為您建立了 Lambda 授權者函數。您可以在 Lambda 主控台中看到 Lambda 函數。名稱的開頭應為**`websocket-step-functions-tutorial-AuthorizerHandler`**。除非`Authorization`標頭為，否則此 Lambda 函數會拒絕對 WebSocket API 的所有呼叫。`AllowLambda` 函數也會將`$context.authorizer.principalId`變數傳遞至您的 API，稍後會在 DynamoDB 表格中用來識別 API 呼叫者。

在此步驟中，您將 $ 連接路由設定為使用 Lambda 授權器。

若要建立 Lambda 授權者

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 [建立授權者]。
4. 針對授權者名稱，輸入**`LambdaAuthorizer`**。
5. 針對「授權者 ARN」，輸入範本所建立之授權者的名稱。 AWS CloudFormation 名稱的開頭應為**`websocket-step-functions-tutorial-AuthorizerHandler`**。

   > **Note**
   >
   > 我們建議您不要將此範例授權者用於生產 API。

6. 針對識別來源型態，選擇「表頭」。在 Key (索引鍵) 欄位，輸入 **`Authorization`**。
7. 選擇建立授權方。

創建授權者後，將其附加到 API 的 $ 連接路由。

要將授權者附加到 $ 連接路由

1. 在主導覽窗格中，選擇「路由」。
2. 選擇 $ 連接路線。

3. 在路由請求設定區段中，選擇編輯。
4. 針對「授權」，選擇下拉式選單，然後選取您的請求授權者。
5. 選擇儲存變更。

步驟 4：創建模擬雙向集成

接下來，您創建 $ 默認路由的雙向模擬集成。模擬集成使您可以在不使用後端的情況下向客戶端發送響應。當您建立 $ 預設路由的整合時，您可以向用戶端展示如何與您的 API 互動。

您配置 $ 默認路由通知客戶端使用發送消息路由。

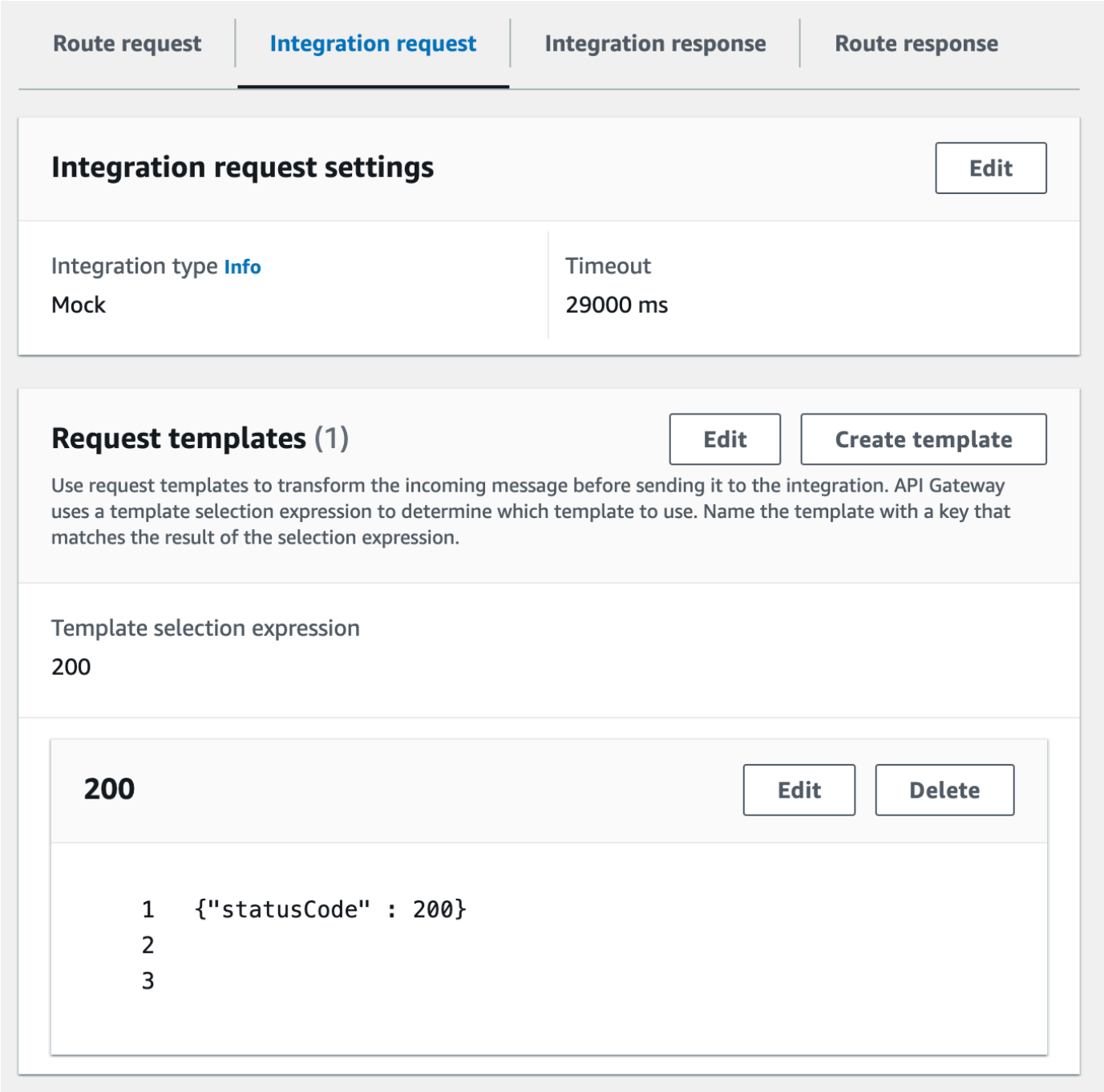
若要建立模擬整合

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 選擇 $ 預設路由，然後選擇整合要求索引標籤。
3. 針對要求範本，選擇編輯。
4. 針對「範本選取表示式」，輸入**200**，然後選擇 「編輯」。
5. 在「整合要求」標籤上，針對「要求範本」選擇「建立範本」。
6. 對於「範本金鑰」，輸入**200**。
7. 針對「產生範本」，輸入下列對應範本：

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

選擇建立範本。

結果應如下列範例所示：



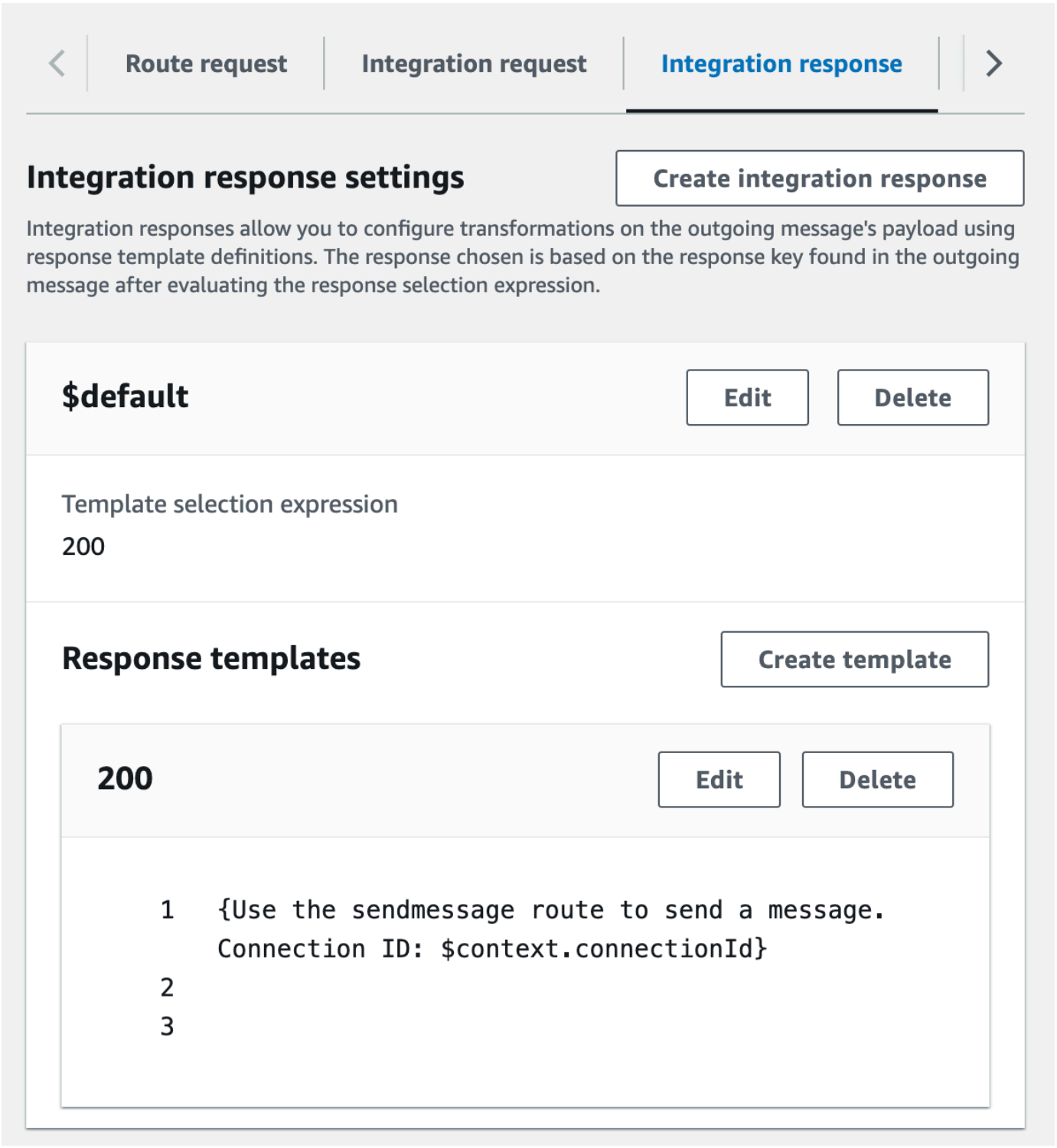
8. 在 $ 預設路由窗格中，選擇啟用雙向通訊。
9. 選擇整合回應標籤，然後選擇建立整合回應。
10. 針對「回應鍵」，輸入**$default**。
11. 對於「範本選取表示式」，輸入**200**。
12. 選擇建立回應。
13. 在「回應範本」下，選擇「建立範本」

14. 對於「範本金鑰」，輸入**200**。
15. 在「回應範本」中，輸入下列對應範本：

```
{"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：


Route request
Integration request
Integration response
Integration response settings
Create integration response
Integration responses allow you to configure transformations on the outgoing message's payload using response template definitions. The response chosen is based on the response key found in the outgoing message after evaluating the response selection expression.
$default
Edit
Delete
Template selection expression
200
Response templates
Create template
200
Edit
Delete
1 {Use the sendmessage route to send a message.
Connection ID: $context.connectionId}
2
3

第 5 步：使用步 Step Functions 創建非代理集成

接下來，您創建一個發送消息路由。客戶端可以調用發送消息路由廣播消息到所有連接的客戶端。傳送訊息路由與非 Proxy AWS 服務整合。 AWS Step Functions整合會叫用 AWS CloudFormation 範本為您建立之「Step Functions」狀態機器的StartExecution指令。

若要建立非代理整合

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. 針對「整合類型」選擇「AWS 服務」。
5. 在「AWS 區域」中，輸入您部署 AWS CloudFormation 範本的「區域」。
6. 對於AWS 維修，請選擇 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 AWS 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)。

接下來，您可以建立對應範本，將要求參數傳送至 Step Functions 狀態機器。

建立對應範本

1. 選擇傳送訊息路由，然後選擇整合要求索引標籤。
2. 在「要求範本」區段中，選擇 「編輯」。
3. 對於「範本選取表示式」，輸入**\$default**。
4. 選擇編輯。
5. 在「要求範本」區段中，選擇「建立範本」。
6. 對於「範本金鑰」，輸入**\$default**。
7. 針對「產生範本」，輸入下列對應範本：

```
#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"
}
```

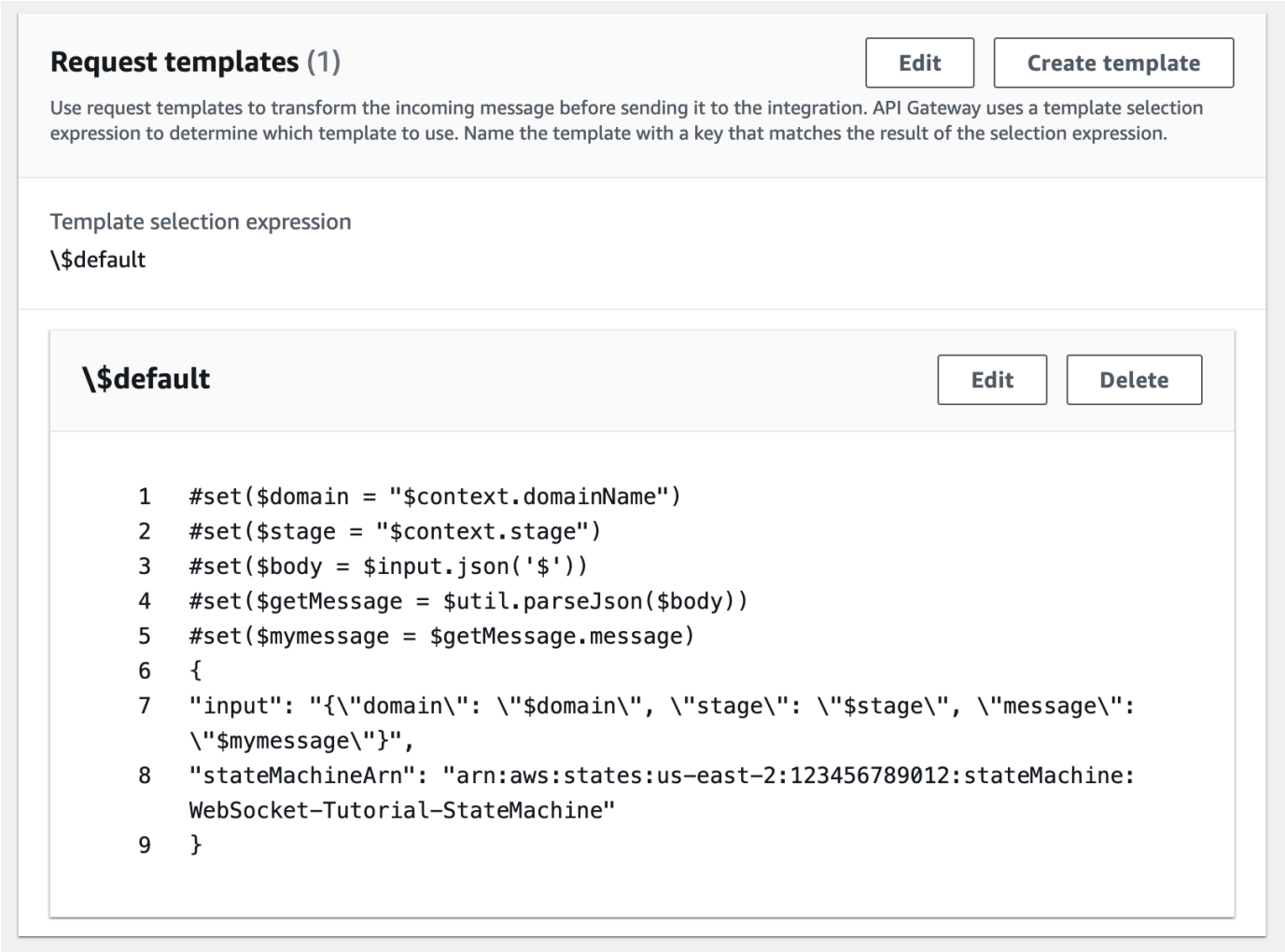
*stateMachineArn*用建立的狀態機的 ARN 取代 AWS CloudFormation。

對應範本會執行下列作業：

- 創建使$domain用上下文變量的變量domainName。
- 創建使$stage用上下文變量的變量stage。

  $domain和$stage變數是建立回呼 URL 所必需的。
- 接受傳入的 sendmessage JSON 訊息，並擷取message屬性。
- 建立狀態機器的輸入。輸入是 WebSocket API 的域和階段以及來自sendmessage路由的消息。

8. 選擇建立範本。



您可以在 $connect 或 $ 中斷連線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識別碼，而不呼叫 Lambda 函數。

步驟 6：測試您的 API

接下來，您將部署並測試您的 API，以確保它可以正常運作。您將使用該`wscat`命令連接到 API，然後，您將使用斜杠命令發送 ping 框架以檢查到 WebSocket API 的連接。

部署 API

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 在主導覽窗格中，選擇「路由」。
3. 選擇部署 API。
4. 在「舞台」中選擇「製作」。

5. (選擇性) 對於部署說明，請輸入說明。
6. 選擇部署。

部署 API 之後，您可以叫用它。使用呼叫網址呼叫您的 API。

為您的 API 獲取調用網址

1. 選擇您的 API。
2. 選擇 Stages (階段)，然後選擇 production (產品)。
3. 請記下您的 API WebSocket 網址。URL 看起來應該會像這樣：`wss://abcdef123.execute-api.us-east-2.amazonaws.com/production`。

現在您有了調用 URL，您可以測試與 WebSocket API 的連接。

若要測試與您的 API 的連線

1. 使用下列命令來連線到 API。首先，您可以通過調用/ping路徑來測試連接。

```
wscat -c wss://abcdef123.execute-api.us-east-2.amazonaws.com/production -H
 "Authorization: Allow" --slash -P
```

```
Connected (press CTRL+C to quit)
```

2. 輸入以下指令以 ping 控制框。您可以使用控制框架從客戶端保持活動的目的。

```
/ping
```

結果應如下列範例所示：

```
< Received pong (data: "")
```

現在您已經測試了連接，您可以測試您的 API 是否正常工作。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 的路由要求選取表示式來決定要叫用的路由。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, from Step Functions!"}
```

與路由相關聯的 Step Functions 數狀態機調用帶有消息和回調 URL 的 Lambda 函數。Lambda 函數會呼叫 API Gateway 管理 API，並將訊息傳送給所有連線的用戶端。所有用戶端都會收到下列訊息：

```
< hello, from Step Functions!
```

現在您已經測試了 WebSocket API，您可以斷開與 API 的連接。

中斷與 API 的連線

- 若要中斷與 API 的連線，請按 `CTRL+C`。

  當客戶端與您的 API 斷開連接時，API Gateway 會調用 API 的 $ 斷開路由。您 API 的 $ 中斷連線路由的 Lambda 整合會從 DynamoDB 中移除連線識別碼。

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 AWS CloudFormation 堆疊和 WebSocket API。

若要刪除 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，選擇您的網絡插槽 API。
3. 依次選擇 Actions (動作)、Delete (刪除)，然後確認您的選擇。

刪除 AWS CloudFormation 堆疊的步驟

1. 請在以下位置開啟 AWS CloudFormation 主控台。 https://console.aws.amazon.com/cloudformation
2. 選取您的 AWS CloudFormation 堆疊。
3. 選擇刪除，然後確認您的選擇。

後續步驟

您可以自動建立和清理本自學課程中涉及的所有 AWS 資源。如需自動執行此教學課程中這些動作的範 AWS CloudFormation 本範例，請參閱 ws-sfn.zip。

使用 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的集合。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 API Gateway 中開發 REST API
- 發佈 REST API 以供客戶叫用
- 最佳化 REST API 的效能
- 將您的 REST API 分配給用戶端
- 保護您的 REST API
- 監控 REST API

在 API 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資源來表示可以包含資源實體集合的 API。

每個Resource實體都可以有一或多個方法資源。A Method 是由客戶端提交的傳入請求，並在請求參數和正文中表示。它定義了用於客戶端訪問暴露的應用程序編程接口Resource。若要Method與後端端點 (也稱為整合端點) 整合，請建立整合資源。這會將傳入的要求轉送至指定的整合端點 URI。如有必要，您可以轉換請求參數或請求主體以滿足後端要求。

對於回應，您可以建立MethodResponse資源來表示用戶端收到的要求回應，並建立IntegrationResponse資源來代表後端傳回的要求回應。您可以設定整合回應來轉換後端回應資料，再將資料傳回用戶端或將後端回應依現狀傳遞到用戶端。

為了協助您的客戶了解 API，您也可以在建立 API 期間或之後提供 API 的文件。若要啟用此功能，請為支援的 API 實體新增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。

如需如何建立 API 的簡介，請參閱the section called “教學課程：具有 Lambda 代理整合的 Hello World API”。若要進一步了解您在開發 REST API 時可能會使用的 API Gateway 功能的相關資訊，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、或其中一個 AWS SDK 來執行的 AWS CLI概念性資訊和程序。

主題

- API Gateway API 端點類型
- API Gateway 中其餘 API 的方法
- 在 API Gateway 中控制和管理 REST API 的存取
- 設定 REST API 整合
- 在 API Gateway 中使用請求驗證
- 設定 REST API 的資料轉換
- API Gateway 中的閘道回應
- 為 REST API 資源啟用 CORS
- 使用 REST API 的二進位媒體類型
- 在 Amazon API Gateway 中叫用 REST API
- 使用 OpenAPI 設定 REST API

API Gateway 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 流量的來源位置，API 端點類型可以是邊緣最佳化、區域或私有。

邊緣最佳化的 API 端點

邊緣最佳化的 API 端點通常會將要求路由到最近的存在 CloudFront 點 (POP)，如果您的用戶端分散在地理位置上，這可能會有所幫助。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 在將請求轉發到您的來源之前，按 cookie 名稱以自然順序對 HTTP 餅乾進行排序。如需有關 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

地區 API 端點適用於相同地區的用戶端。當在 EC2 執行個體上執行的用戶端呼叫相同區域中的 API，或者當 API 打算為少量高需求的用戶端提供服務時，區域 API 可減少連線額外負荷。

對於地區 API，您使用的任何自訂網域名稱都特定於部署 API 的地區。如果您部署在多個區域中的區域性 API，它可以在所有區域中擁有相同的自訂網域名稱。您可以使用自訂網域搭配 Amazon Route 53 來執行任務，例如以延遲為基礎的路由。如需詳細資訊，請參閱the section called “設定區域性自訂網域名稱”及the section called “建立邊緣最佳化自訂網域名稱”。

所有區域和 API 端點會依原狀傳遞所有標頭名稱。

> **Note**
>
> 在 API 用戶端分散各地的情況下，使用區域 API 端點與您自己的 Amazon 分 CloudFront 發，以確保 API Gateway 不會將 API 與服務 CloudFront 控制的分發產生關聯。有關此用例的更多信息，請參閱如何使用自己的 CloudFront 發行版設置 API Gateway？ 。

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 “私人休息 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、或 API Gateway 的 AWS CLI AWS SDK 來變更現有的 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：

- 從邊緣最佳化到區域或私有
- 從區域到邊緣優化或私有
- 從私人到區域

您不能將私有 API 改為邊緣最佳化 API。

如果您要將公用 API 從邊緣最佳化變更為區域，反之亦然，請注意，邊緣最佳化 API 的行為可能與地區 API 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。但是，區域 API 會通過此標題傳遞，儘管它可能會將標題名稱重新映射到其他名稱。了解這些差異有助於您決定如何將邊緣優化 API 更新為區域 API，或從區域 API 更新為邊緣優化 API。

主題

- 使用 API Gateway 主控台變更 API 端點類型
- 使用變 AWS CLI 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。

將地區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. 建立可授與 VPC 或 VPC 端點存取權的資源原則。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. (選擇性) 對於 VPC 端點識別碼，請選取您要與私有 API 建立關聯的 VPC 端點識別碼。
8. 選擇儲存變更。
9. 重新部署您的 API，變更才會生效。

使用變 AWS CLI 更 API 端點類型

若要使用 AWS CLI 更新 API ID update-rest-api為的邊緣最佳化 API*{api-id}*，請依照下列方式呼叫：

```
aws apigateway update-rest-api \
    --rest-api-id {api-id} \
    --patch-operations op=replace,path=/endpointConfiguration/types/EDGE,value=REGIONAL
```

成功回應會有 200 OK 狀態碼與類似下列的承載：

```
{

    "createdDate": "2017-10-16T04:09:31Z",
    "description": "Your first API with Amazon API Gateway. This is a sample API that
 integrates via HTTP with our demo Pet Store endpoints",
    "endpointConfiguration": {
        "types": "REGIONAL"
    },
    "id": "0gsnjtjck8",
    "name": "PetStore imported as edge-optimized"
}
```

反之，將區域 API 更新成邊緣最佳化 API，如下所示：

```
aws apigateway update-rest-api \
    --rest-api-id {api-id} \
    --patch-operations op=replace,path=/endpointConfiguration/types/REGIONAL,value=EDGE
```

由put-rest-api於用於更新 API 定義，因此不適用於更新 API 端點類型。

API Gateway 中其餘 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，您可以定義的 API 方法要求`GET /pets/{petId}`，其中`{petId}`是可以在執行階段取得數字的 path 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。

必要條件

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課程：建置具有非 HTTP 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 API Gateway 支援的 HTTP 與 HTTP 代理整合憑證授權機構。

> **Note**
>
> 當您使用 REST API 主控台建立方法時，您可以同時設定整合要求和方法要求。如需詳細資訊，請參閱 the section called “ 使用主控台設定整合請求”。

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

建立RestApi資源之後，設定方法要求包括執行下列工作：

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（創建資源和 put 方法）
- 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。

針對PetStore 範例 API，根資源 (/) 會公開寵物商店。/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。

使用 AWS CLI，您可以調用get-resources命令來找出 API 的哪些資源可用：

```
aws apigateway get-resources --rest-api-id <apiId> \
                             --region <region>
```

結果會列出目前可用的 API 資源。對於 PetStore 範例 API，此清單如下所示：

```
{
    "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系之後，請呼叫下列命令新增子資源。

```
aws apigateway create-resource --rest-api-id <apiId> \
                               --region <region> \
                               --parent-id <parentId> \
                               --path-part <resourceName>
```

例如，要在 PetStore 網站上添加要銷售的寵物食品，請將food資源添加到根 (/) ，方法是將資源設置path-partparent-id為food和svzr2028x8。結果看起來如下：

```
{
    "path": "/food",
    "pathPart": "food",
    "id": "xdsvhp",
    "parentId": "svzr2028x8"
}
```

使用代理資源來簡化 API 設定

隨著業務的增長， PetStore 所有者可能會決定添加食物，玩具和其他寵物相關物品出售。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。Greedy 路徑參數名稱 proxy 可以取代為其他字串，就如同處理一般路徑參數名稱一樣。

使用 AWS CLI，您可以呼叫下列命令，在 root (/{proxy+}) 下設定代理資源：

```
aws apigateway create-resource --rest-api-id <apiId> \
                               --region <region> \
                               --parent-id <rootResourceId> \
                               --path-part {proxy+}
```

結果類似如下：

```
{
    "path": "/{proxy+}",
    "pathPart": "{proxy+}",
    "id": "234jdr",
    "parentId": "svzr2028x8"
}
```

在 PetStore API 範例中，您可以使用 /{proxy+} 來表示 /pets 與 /pets/{petId}。此代理資源還可以引用任何其他（現有或 to-be-added）資源 /food/{type}/{item}/toys/{type}/{item}，例如，等等/food/{type}/{subtype}/{item}，或/toys/{type}/{subtype}/{item}，等等。後端開發人員會決定資源階層，而用戶端開發人員則負責了解此階層。API Gateway 只會將用戶端提交的任何項目傳遞到後端。

一個 API 可以有多個代理資源。例如，API 中允許下列代理資源。

```
/{proxy+}
/parent/{proxy+}
/parent/{child}/{proxy+}
```

當代理資源有非代理同層級資源時，則會從代理資源的顯示方式中排除這些同層級資源。在上述範例中，/{proxy+} 是指根資源下除了 /parent[/*] 資源以外的任何資源。換言之，對特定資源提出的方法請求，會優先於對資源階層中同層級之一般資源提出的方法請求。

代理資源不能有任何子資源。{proxy+} 後面的任何 API 資源是多餘且模棱兩可的。API 中不允許下列代理資源。

```
/{proxy+}/child
/parent/{proxy+}/{child}
```

```
/parent/{child}/{proxy+}/{grandchild+}
```

設定 HTTP 方法

API 方法請求是由 API Gateway 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。

下列 AWS CLI 命令顯示如何針對指定的 resource (6sxz2j) 建立ANY動詞的方法要求，並使用 IAM 許可來控制其存取權。

```
aws apigateway put-method --rest-api-id vaz7da96z6 \
       --resource-id 6sxz2j \
       --http-method ANY \
       --authorization-type AWS_IAM \
       --region us-west-2
```

若要使用不同的授權類型來建立 API 方法請求，請參閱「the section called “設定方法請求授權”」。

設定方法請求參數

方法請求參數可讓用戶端提供完成方法請求所需的輸入資料或執行內容。方法參數可以是路徑參數、標頭或查詢字串參數。設定方法請求時，您必須宣告必要的請求參數以提供給用戶端。對於非代理整合，您可以將這些請求參數轉換成與後端需求相容的格式。

例如，對於 GET /pets/{petId} 方法請求，{petId} 路徑變數是必要的請求參數。您可以在呼叫 AWS CLI的 put-method 命令時宣告此路徑參數，如下所示：

```
aws apigateway put-method --rest-api-id vaz7da96z6 \
```

```
       --resource-id rjkmth \
       --http-method GET \
       --authorization-type "NONE" \
       --region us-west-2 \
       --request-parameters method.request.path.petId=true
```

如果不需要參數，您可以在 `false` 中將它設定為 `request-parameters`。例如，如果 `GET /pets` 方法使用一個選用的查詢字串參數 `type` 與一個選用的標頭參數 `breed`，您可以使用下列 CLI 命令來宣告這些參數，並假設 `/pets` 資源 `id` 為 `6sxz2j`：

```
aws apigateway put-method --rest-api-id vaz7da96z6 \
       --resource-id 6sxz2j \
       --http-method GET \
       --authorization-type "NONE" \
       --region us-west-2 \
       --request-parameters
 method.request.querystring.type=false,method.request.header.breed=false
```

除了此縮寫格式之外，您還可以使用 JSON 字串來設定 `request-parameters` 值：

```
'{"method.request.querystring.type":false,"method.request.header.breed":false}'
```

有了這項設定，用戶端就可依類型查詢寵物：

```
GET /pets?type=dog
```

此外，用戶端可以查詢貴賓狗品種的狗，如下所示：

```
GET /pets?type=dog
breed:poodle
```

如需如何將方法請求參數映射到整合請求參數的資訊，請參閱「the section called “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。要設置方法請求模型，請在調用 AWS CLI put-method命令時將"*<media-type>*":"*<model-name>*"格式的鍵值對添加到 requestModels map 中。

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，要在示 PetStore 例 API 的POST /pets方法請求的 JSON 有效負載上設置模型，可以調用以下 AWS CLI 命令：

```
aws apigateway put-method \
       --rest-api-id vaz7da96z6 \
       --resource-id 6sxz2j \
       --http-method POST \
       --authorization-type "NONE" \
       --region us-west-2 \
       --request-models '{"application/json":"petModel"}'
```

在此範例中，petModel 是描述寵物之 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：

```
{
    "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。來自不同的用戶只有在允許在 API 所有者中擔任 AWS 帳戶 具有必要權限調用`execute-api:Invoke`操作的角色時，才 AWS 帳戶 可以調用 API 方法。如需跨帳戶許可的資訊，請參閱使用 IAM 角色。

您可以使用 AWS CLI AWS SDK 或 REST API 客戶端，例如郵差，它實現簽名版本 4（Sigv4）簽名。

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：

```
aws apigateway create-request-validator \
    --rest-api-id 7zw9uyk9kl \
    --name bodyOnlyValidator \
    --validate-request-body  \
    --no-validate-request-parameters
```

此 CLI 命令會建立僅限本文的請求驗證程式。範例輸出如下：

```
{
    "validateRequestParameters": false,
```

```
    "validateRequestBody": true,
    "id": "jgpyy6",
    "name": "bodyOnlyValidator"
}
```

透過此請求驗證程式，您可以在設定方法請求時啟用請求驗證：

```
aws apigateway put-method \
    --rest-api-id 7zw9uyk9kl
    --region us-west-2
    --resource-id xdsvhp
    --http-method PUT
    --authorization-type "NONE"
    --request-parameters '{"method.request.querystring.type": false,
 "method.request.querystring.page":false}'
    --request-models '{"application/json":"petModel"}'
    --request-validator-id jgpyy6
```

您必須將請求參數宣告為必要，才能將它包含在請求驗證中。如果在請求驗證中使用頁面的查詢字串參數，上述範例的 `request-parameters` 映射必須指定為 `'{"method.request.querystring.type": false, "method.request.querystring.page":true}'`。

在 API Gateway 中設定方法回應

API 方法回應可封裝用戶端將會接收的 API 方法請求輸出。該輸出資料包含 HTTP 狀態碼、一些標頭，並可能包含本文。

透過非代理整合，指定的回應參數與本文可從相關聯的整合回應資料映射，或根據映射指派特定靜態值。這些映射是在整合回應中指定。此映射可以是依現狀傳遞整合回應的相同轉換。

透過代理整合，API Gateway 會自動將後端回應傳遞至方法回應。您不需要設定 API 方法回應。不過，若使用 Lambda 代理整合，Lambda 函數必須傳回此輸出格式的結果，API Gateway 才能成功將整合回應映射至方法回應。

以程式設計方式，方法回應設定等於建立 API Gateway 的MethodResponse資源，並設定 statusCode、responseParameters 和 response Models 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。下列 AWS CLI 命令會建立方法回應 200。

```
aws apigateway put-method-response \
       --region us-west-2  \
       --rest-api-id vaz7da96z6 \
       --resource-id 6sxz2j \
       --http-method GET \
       --status-code 200
```

設定方法回應參數

方法回應參數可定義用戶端所收到以回應相關聯方法請求的標頭。回應參數也可根據 API 方法的整合回應中所指定的映射，來指定 API Gateway 映射整合回應參數的目標。

若要設定方法回應參數，請新增至 responseParameters 格式之 MethodResponse 索引鍵/值組的 "{parameter-name}":"{boolean}" 映射。下面的 CLI 命令顯示了設置my-header標頭的一個例子。

```
aws apigateway put-method-response \
        --region us-west-2 \
        --rest-api-id vaz7da96z6 \
        --resource-id 6sxz2j \
        --http-method GET \
        --status-code 200  \
        --response-parameters method.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。設定回應模型之前，您必須先在 API Gateway 中建立模型。若要這樣做，您可以呼叫 create-model 命令。下列範例示範如何建立 PetStorePet 模型，以描述 GET /pets/{petId} 方法請求的回應本文。

```
aws apigateway create-model \
    --region us-west-2 \
    --rest-api-id vaz7da96z6 \
    --content-type application/json \
    --name PetStorePet \
    --schema '{ \
                "$schema": "http://json-schema.org/draft-04/schema#", \
                "title": "PetStorePet", \
                "type": "object", \
                "properties": { \
                  "id": { "type": "number" }, \
                  "type": { "type": "string" }, \
                  "price": { "type": "number" } \
                } \
            }'
```

結果會建立為 API Gateway `Model` 資源。

要設置方法響應模型以定義有效負載格式，請將「應用/json」:PetStorePet」鍵值對添加到資源`requestModels`映射。`MethodResponse`以下 AWS CLI 命令`put-method-response`顯示了如何完成此操作：

```
aws apigateway put-method-response \
        --region us-west-2 \
        --rest-api-id vaz7da96z6 \
        --resource-id 6sxz2j \
        --http-method GET \
        --status-code 200  \
        --response-parameters method.response.header.my-header=false \
        --response-models '{"application/json":"PetStorePet"}'
```

當您為 API 產生強型別開發套件時，需要設定方法回應模型。它可確保輸出在 Java 或 Objective-C 中轉換成適當的類別。在其他情況下，設定模型為選擇性。

使用 API Gateway 主控台設定方法

當您使用 REST API 主控台建立方法時，您可以同時設定整合要求和方法要求。根據預設，API Gateway 會為您的`200`方法建立方法回應。

下列指示說明如何編輯方法要求設定，以及如何為您的方法建立其他方法回應。

主題

- 在 API Gateway 主控台中編輯 API Gateway 方法要求
- 使用 API Gateway 主控台設定 API Gateway 方法回應

在 API Gateway 主控台中編輯 API Gateway 方法要求

這些指示假設您已經建立了方法要求。若要取得有關如何建立方法的更多資訊，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] 索引標籤。
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：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "execute-api:Invoke"
      ],
      "Resource": [
        "resource-statement"
      ]
    }
  ]
}
```

      在此訪問策略中，####是您方法的 ARN。您可以在 [資源] 頁面上選取方法來尋找方法的 ARN。如需設定 IAM 許可的詳細資訊，請參閱使用 IAM 許可控制 API 的存取。

      若要建立 IAM 角色，您可以調整以下教學課程中的指示???。

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 使用者集區作為授權方來控制對 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「在 API Gateway 中使用請求驗證」。

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「在 API Gateway 中使用請求驗證」。

   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「在 API Gateway 中使用請求驗證」。

   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「了解資料模型」。

   > **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] 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. 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 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 角色和政策提供彈性且強大的存取控制，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 “用戶端憑證”。
- 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。如需更多詳細資訊，請參閱 the section called “用量計劃”。

使用 API Gateway 資源政策控制對 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 Management Console、 AWS CLI 或 AWS SDK，為 API Gateway 中的任何 API 端點類型附加資源策略。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 可在 API Gateway 資源策略中使用的條件金鑰

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。

有關 AWS 服務如何決定應允許還是拒絕給定請求的詳細信息，請參閱確定請求是允許還是拒絕。

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。如需進一步了解，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。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": "*",
            "Action": "execute-api:Invoke",
            "Resource": "arn:aws:execute-api:region:account-id:*"
        },
        {
            "Effect": "Deny",
            "Principal": "*",
            "Action": "execute-api:Invoke",
            "Resource": "arn:aws:execute-api:region:account-id:*",
            "Condition": {
                "NotIpAddress": {
                    "aws:SourceIp": "123.4.5.6/24"
                }
            }
        }
    ]
}
```

API Gateway 資源政策如何影響授權工作流程

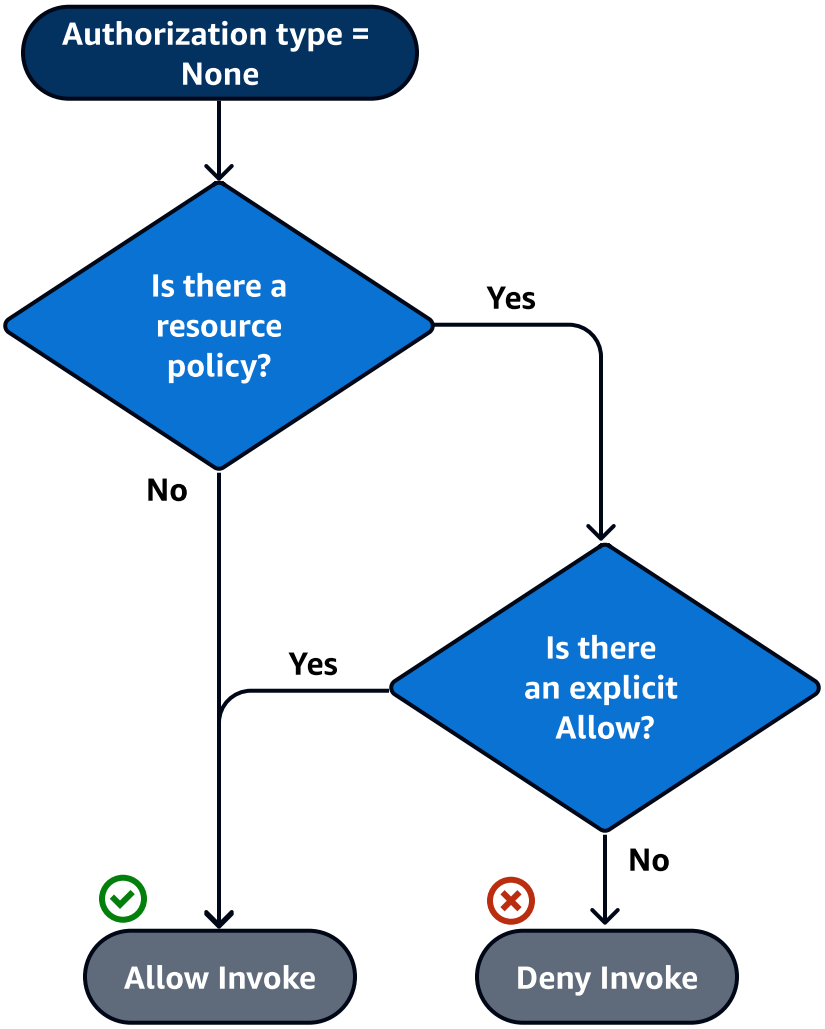
API Gateway 評估連接到您 API 的資源政策時，其結果會受到您為 API 定義的身分驗證類型所影響，如下節流程圖所示。

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。

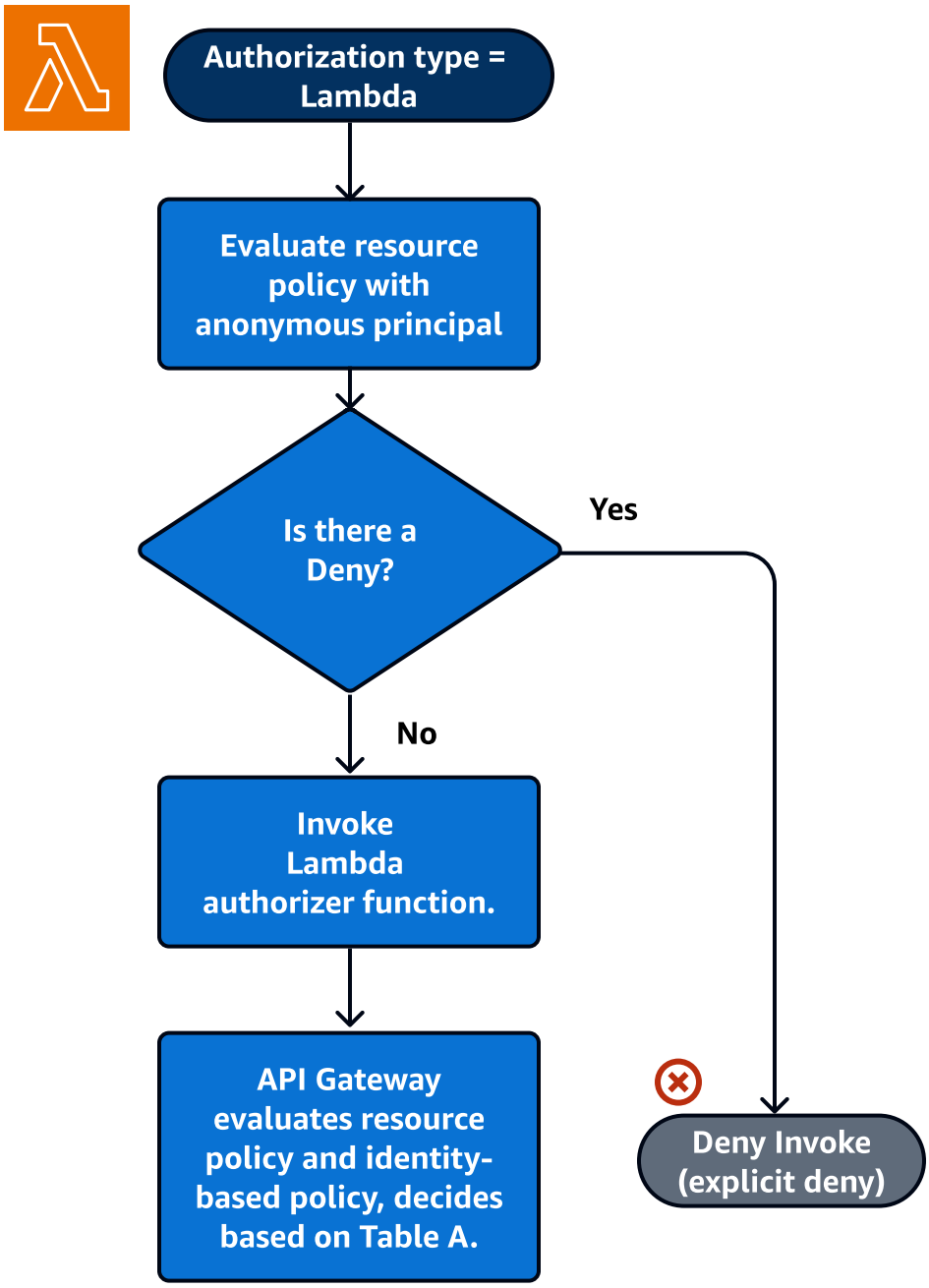
```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": "*",
            "Action": "execute-api:Invoke",
            "Resource": "arn:aws:execute-api:region:account-id:api-id/",
            "Condition": {
```

```
                "IpAddress": {
                    "aws:SourceIp": ["192.0.2.0/24", "198.51.100.0/24" ]
                }
            }
        }
    ]
}
```

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。結果是根據表 A 決定的。

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。




```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Principal": "*",
            "Action": "execute-api:Invoke",
            "Resource": [
```

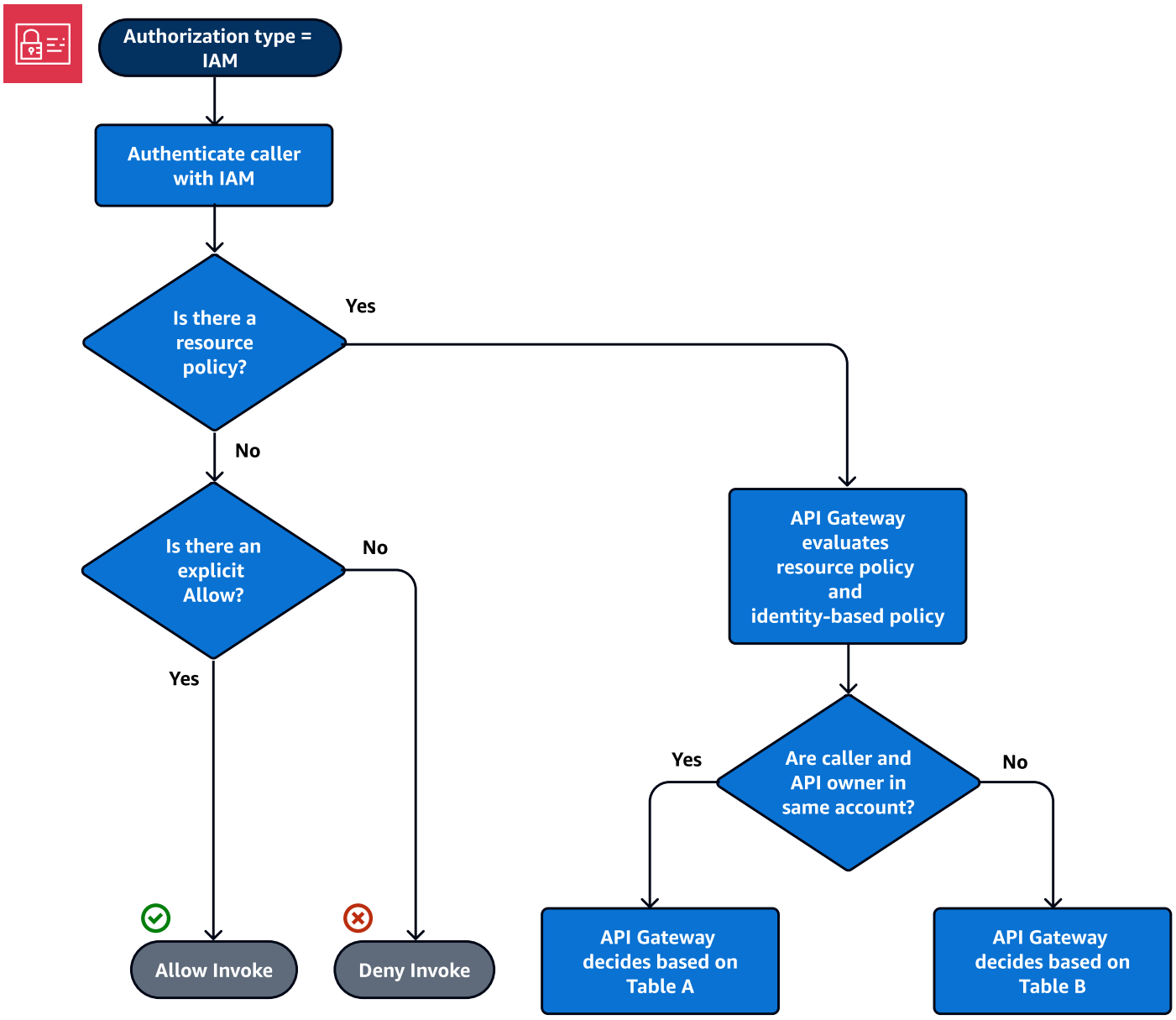
```
                "arn:aws:execute-api:region:account-id:api-id/"
            ],
            "Condition" : {
                "StringNotEquals": {
                    "aws:SourceVpce": "vpce-1a2b3c4d"
                }
            }
        }
    ]
}
```

IAM 身分驗證和資源政策

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。結果會根據呼叫者與 API 擁有者之間的位置相同 AWS 帳戶 還是獨立 AWS 帳戶而有所不同。

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。



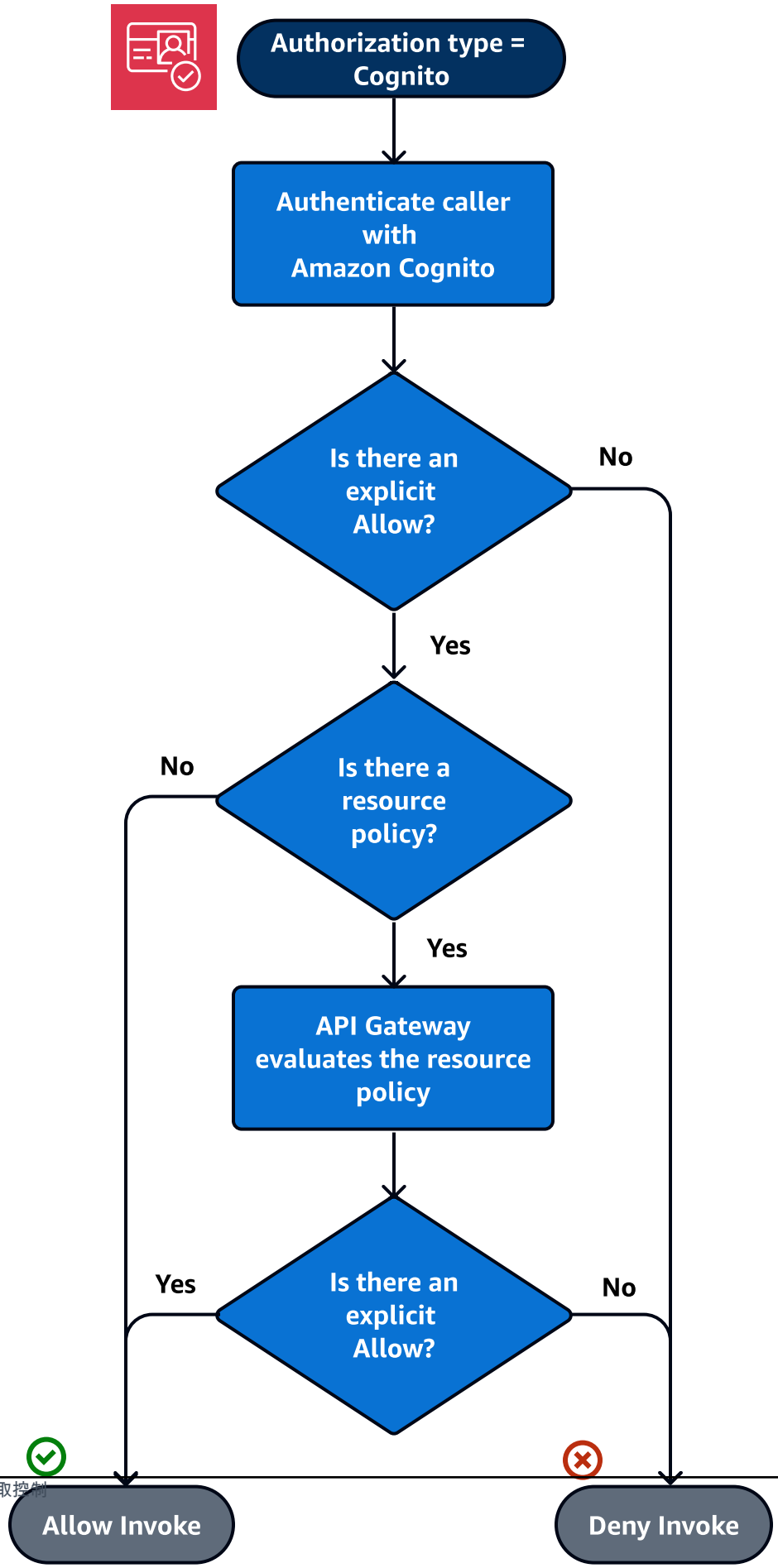
以下是跨帳戶資源政策的範例。假設 IAM 政策包含允許效果，則該資源政策僅允許來自 VPC ID 為 *vpc-2f09a348* 的 VPC 的呼叫。如需詳細資訊，請參閱表 B。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": "*",
            "Action": "execute-api:Invoke",
            "Resource": [
```

```
                "arn:aws:execute-api:region:account-id:api-id/"
            ],
            "Condition" : {
                "StringEquals": {
                    "aws:SourceVpc": "vpc-2f09a348"
                    }
            }
        }
    ]
}
```

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)。如需詳細資訊，請參閱表 B。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": "*",
            "Action": "execute-api:Invoke",
            "Resource": "arn:aws:execute-api:region:account-id:api-id/",
            "Condition": {
                "IpAddress": {
                    "aws:SourceIp": ["192.0.2.0/24", "198.51.100.0/24" ]
                }
            }
        }
    ]
}
```

政策評估結果表

表 A 列出 API Gateway API 的存取權由 IAM 政策或 Lambda 授權者以及 API Gateway 資源政策所控制時產生的行為，這兩項政策都是相同 AWS 帳戶的。

表 A：帳戶 A 呼叫帳戶 A 擁有的 API

IAM 政策 (或 Lambda 授權者)	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
拒絕	Allow	明確拒絕

IAM 政策 (或 Lambda 授權者)	API Gateway 資源政策	產生行為
拒絕	不允許也不拒絕	明確拒絕
拒絕	拒絕	明確拒絕

表 B 列出 API Gateway API 的存取權由 IAM 政策或 Amazon Cognito 使用者集區授權者和 API Gateway 資源政策控制時產生的行為，這些政策不同。 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要求資源政策和 IAM 政策或 Amazon Cognito 使用者集區授權者都明確授予存取權。

表 B：帳戶 B 呼叫帳戶 A 擁有的 API

IAM 政策 (或 Amazon Cognito 使用者集區授權者)	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、您的 AWS 帳戶識別碼以及與資源策略相關聯的 REST API 識別碼來建立完整的 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：允許其他 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：允許其他 AWS 帳戶中的角色使用 API

下列範例資源策略會透過簽章版本 4 (SIGv4) 通訊協定，將一個 AWS AWS 帳戶中的 API 存取權授予不同帳戶中的兩個角色。具體而言，授與所*account-id-2*識別之 AWS 帳戶的開發人員和系統管理員角色，可針對您 AWS 帳戶中的`pets`資源 (API) 執行`GET`動作的動作。`execute-api:Invoke`

```
{
   "Version": "2012-10-17",
   "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": [
                    "arn:aws:iam::account-id-2:role/developer",
                    "arn:aws:iam::account-id-2: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。

```
{
    "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" ]
                }
            }
        }
    ]
}
```

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。

```
{
    "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：

```
{
    "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：

```
{
    "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

若要允許使用者呼叫 API 執行服務來存取您的 API，您必須建立 API Gateway 資源原則，並將原則附加至 API。當您將政策附加到 API 時，它會將策略中的權限套用至 API 中的方法。如果您更新資源策略，則需要部署 API。

主題

- 必要條件
- 將資源策略附加到 API Gateway API
- 疑難排解資源策略

必要條件

若要更新 API Gateway 資源政策，您需要`apigateway:UpdateRestApiPolicy`權限和`apigateway:PATCH`權限。

對於邊緣最佳化或區域 API，您可以在建立 API 時或部署後將資源政策附加至 API。對於私有 API，您無法在沒有資源策略的情況下部署 API。如需詳細資訊，請參閱 the section called “私人休息 API”。

將資源策略附加到 API Gateway API

下列程序說明如何將資源原則附加至 API Gateway API。

AWS Management Console

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 API 並將資源策略附加至其中，請依照下列方式呼叫create-rest-api命令：

```
aws apigateway create-rest-api \
    --name "api-name" \
    --policy "{\"jsonEscapedPolicyDocument\"}"
```

若要使用 AWS CLI 將資源策略附加至現有 API，請依照下列方式呼叫update-rest-api命令：

```
aws apigateway update-rest-api \
    --rest-api-id api-id \
    --patch-operations op=replace,path=/
policy,value='"{\"jsonEscapedPolicyDocument\"}"'
```

AWS CloudFormation

您可以使 AWS CloudFormation 用建立包含資源策略的 API。下列範例會使用範例資源策略建立 REST API the section called “範例：拒絕根據來源 IP 地址或範圍的 API 流量”。

```
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
  ApiDeployment:
    Type: 'AWS::ApiGateway::Deployment'
    DependsOn:
      - MethodGet
    Properties:
      RestApiId: !Ref Api
      StageName: test
```

疑難排解資源策略

下列疑難排解指引可能有助於解決資源策略的問題。

我的 API 返回 {「消息」：「用戶：匿名未被授權執行：執行 API：調用資源：ARN：aws：執行 API：我們東部 -1：********/****/****/"}

在您的資源策略中，如果您將主參與者設定為 AWS 主參與者，如下所示：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            ""Principal": {
                "AWS": [
                    "arn:aws:iam::account-id:role/developer",
```

```
                "arn:aws:iam::account-id:role/Admin"
            ]
        },
        "Action": "execute-api:Invoke",
        "Resource": [
            "execute-api:/*"
        ]
    },
    ...
    }
```

您必須對 API 中的每個方法使用AWS_IAM授權，否則您的 API 會返回先前的錯誤消息。如需如何開啟方法AWS_IAM授權的更多指示，請參閱the section called “方法”。

我的資源策略未更新

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”。

AWS 可在 API Gateway 資源策略中使用的條件金鑰

下表包含可用於每種授權類型之 API Gateway API 資源原則中的 AWS 條件金鑰。

如需有關 AWS 條件索引鍵的詳細資訊，請參閱AWS 全域條件內容索引鍵

條件金鑰表格

條件鍵	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
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

條件鍵	條件	需要 **AuthN**？	授權類型
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
aws:SecureTransport	無	否	全部
aws:SourceArn	無	否	全部

條件鍵	條件	需要 AuthN？	授權類型
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 許可控制 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：

- 使用者和群組位於 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。請按照 IAM 使用者指南 的 為第三方身分提供者 (聯合) 建立角色 中的指示進行操作。
- 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 中的指示進行操作。
  - (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。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 與後端中的 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) 的權限。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：

```
{
  "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 管理主控台中建立此類政策，或呼叫建 AWS CLI 立政策命令或對應的 SDK 方法。

若要讓 API Gateway 呼叫整合式 AWS 服務，您還必須附加至此角色適當的 IAM 許可政策，以呼叫整合式 AWS 服務。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：

```
{
    "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 服務實施存取控制，您可以使用以呼叫者為基礎的許可模型 (其中權限原則直接附加至呼叫者的使用者或群組)，或是以角色為基礎的權限模型 (其中，權限原則會附加至 API Gateway 可以承擔的 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。您可以利用此優勢，要求使用者僅透過 API Gateway API 存取 AWS 服務。

控制對 API 的呼叫存取權

在本節中，您將了解如何撰寫 IAM 政策陳述式，以控制誰可以在 API Gateway 中呼叫已部署的 API。在本例中，您也將尋找政策陳述式參考，包括與 API 執行服務相關之 `Action` 與 `Resource` 欄位的格式。您也應研究 the section called “資源政策如何影響授權工作流程” 中的 IAM 一節。

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。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Permission",
      "Action": [
        "execute-api:Execution-operation"
      ],
      "Resource": [
        "arn:aws:execute-api:region:account-id:api-id/stage/METHOD_HTTP_VERB/Resource-path"
      ]
    }
  ]
}
```

在本例中，*Permission* 將根據您要授予或撤銷內含許可，取代為 `Allow` 或 `Deny`。*Execution-operation* 將取代為 API 執行服務支援的操作。*METHOD_HTTP_VERB* 代表指定資源支援的 HTTP 動詞。*Resource-path* 是支援上述 *METHOD_HTTP_VERB* 之已部署 API `Resource` 執行個體的 URL 路徑預留位置。如需詳細資訊，請參閱 在 API Gateway 中執行 API 之 IAM 政策的陳述式參考。

> **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：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "execute-api:Invoke"
      ],
      "Resource": [
        "arn:aws:execute-api:us-east-1:account-id:api-id/*/GET/pets"
      ]
    },
    {
      "Effect": "Deny",
      "Action": [
        "execute-api:Invoke"
      ],
      "Resource": [
        "arn:aws:execute-api:us-east-1:account-id:api-id/*/POST/pets"
      ]
    }
  ]
}
```

若要授予使用者許可，讓他們可以檢視設定為 GET /pets/*{petId}* 之 API 所公開的特定寵物，您可以在 IAM 政策中包含以下陳述式：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
```

```
      "Action": [
        "execute-api:Invoke"
      ],
      "Resource": [
        "arn:aws:execute-api:us-east-1:account-id:api-id/*/GET/pets/a1b2"
      ]
    }
  ]
}
```

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，用於在客戶端請求時使 API 緩存失效。

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：

- 地*#*是對應*****於該方法部署的 API 的 AWS 區 AWS 域（例如所有區域的**`us-east-1`**或所有區域）。
- *`###### REST API #`*有者的 12 位數 AWS 帳戶識別碼。
- *`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:*:*:***適用於任何階段中的任何資源路徑，適用於任何 AWS 區域中的任何 API。
- **arn:aws:execute-api:us-east-1:*:***針對任何階段中的任何資源路徑，適用於 AWS 區域中的任何 API us-east-1。
- **arn:aws:execute-api:us-east-1:*:*api-id*/***對於任何階段中的任何資源路徑，對於在 us-east-1 AWS 區域中具有 *api-id* 標識符的 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 階段的資源路徑。

如需進一步了解，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)：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "execute-api:Invoke"
```

```
      ],
      "Resource": [
        "arn:aws:execute-api:us-east-1:*:a123456789/test/POST/mydemoresource/*"
      ]
    }
  ]
}
```

下列範例政策陳述式會提供使用者許可，在已部署識別符為 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：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "execute-api:Invoke"
      ],
      "Resource": [
        "arn:aws:execute-api:*:*:a123456789/*/*/petstorewalkthrough/pets"
      ]
    }
  ]
}
```

建立政策並連接到使用者

若要讓使用者呼叫 API 管理服務或 API 執行服務，您必須建立 IAM 政策，以控制對 API Gateway 實體的存取。

若要使用 JSON 政策編輯器來建立政策

1. 登入 AWS Management Console 並開啟身分與存取權管理主控台，網址為 https://console.aws.amazon.com/iam/。
2. 在左側的導覽窗格中，選擇 Policies (政策)。

   如果這是您第一次選擇 Policies (政策)，將會顯示 Welcome to Managed Policies (歡迎使用受管政策) 頁面。選擇 Get Started (開始使用)。
3. 在頁面頂端，選擇 Create policy (建立政策)。
4. 在政策編輯器中，選擇 JSON 選項。

5. 輸入下列 JSON 政策文件：

```
{
  "Version": "2012-10-17",
  "Statement" : [
    {
      "Effect" : "Allow",
      "Action" : [
        "action-statement"
      ],
      "Resource" : [
        "resource-statement"
      ]
    },
    {
      "Effect" : "Allow",
      "Action" : [
        "action-statement"
      ],
      "Resource" : [
        "resource-statement"
      ]
    }
  ]
}
```

6. 選擇下一步。

> **Note**
>
> 您可以隨時切換視覺化與 JSON 編輯器選項。不過，如果您進行變更或在視覺化編輯器中選擇下一步，IAM 就可能會調整您的政策結構，以便針對視覺化編輯器進行最佳化。如需詳細資訊，請參閱 IAM 使用者指南中的調整政策結構。

7. 在檢視與建立頁面上，為您正在建立的政策輸入政策名稱與描述 (選用)。檢視此政策中定義的許可，來查看您的政策所授予的許可。
8. 選擇 Create policy (建立政策) 儲存您的新政策。

在此陳述式中，視需要替代 *action-statement* 與 *resource-statement*，並新增其他陳述式，以指定要讓使用者管理的 API Gateway 實體、使用者可呼叫的 API 方法，或指定兩者。根據預設，除非有明確對應的 陳述式，否則 `Allow` 使用者沒有許可。

您剛建立 IAM 政策。在您將其連接之前，它不會有任何效果。

若要提供存取權，請新增權限至您的使用者、群組或角色：

- 使用者和群組位於 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。請按照 IAM 使用者指南 的 為第三方身分提供者 (聯合) 建立角色 中的指示進行操作。
- 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 中的指示進行操作。
  - (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。

將 IAM 政策文件連接到 IAM 群組

1. 從主要導覽窗格中，選擇 Groups (群組)。
2. 在所選擇的群組下，選擇 Permissions (許可) 標籤。
3. 選擇 Attach policy (連接政策)。
4. 選擇您之前建立的政策文件，然後選擇 Attach policy (連接政策)。

若要讓 API Gateway 代表您呼叫其他 AWS 服務，請建立 Amazon API Gateway 類型的 IAM 角色。

建立 Amazon API Gateway 角色類型

1. 從主要導覽窗格中，選擇 Roles (角色)。
2. 選擇 Create New Role (建立新角色)。
3. 針對 Role name (角色名稱) 輸入名稱，然後選擇 Next Step (下一步)。
4. 在 Select Role Type (選取角色類型) 下的 AWS Service Roles (服務角色) 中，選擇 Amazon API Gateway 旁的 Select (選取)。
5. 選擇可用的受管 IAM 許可政策，例如，GatewayPushToCloudWatchLog如果您希望 API Gateway 登入指標 CloudWatch，請在 [連接政策] 下選擇 [下一步]。

6. 在 Trusted Entities (信任的實體) 下，確認 apigateway.amazonaws.com 列為項目，然後選擇 Create Role (建立角色)。
7. 在新建立的角色中，選擇 Permissions (許可) 標籤，然後選擇 Attach Policy (連接政策)。
8. 選擇之前建立的自訂 IAM 政策文件，然後選擇 Attach Policy (連接政策)。

在 API Gateway 中使用私有 API 的 VPC 端點政策

若要改善私有 API 的安全性，您可以建立 VPC 端點原則。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。

您可能想要建立 VPC 端點原則來執行下列動作：

- 僅允許特定組織或資源存取您的 VPC 端點並叫用您的 API。
- 使用單一原則，並避免使用工作階段型或角色型政策來控制 API 的流量。
- 在從內部部署移轉至時，收緊應用程式的安全性範圍 AWS。

VPC 端點政策考量事項

- 調用者的身分是根據 `Authorization` 標頭值進行評估。根據您的 `authorizationType` 而定，這可能會導致 `403 IncompleteSignatureException` 或 `403 InvalidSignatureException` 錯誤。下表顯示每個 `authorizationType` 的 `Authorization` 標頭值。

authorizationType	**Authorization** 標頭是否已評估？	允許的 **Authorization** 標頭值
`NONE`，具有預設的完整存取政策	否	未通過
`NONE`，具有自訂存取政策	是	必須是有效的 SigV4 值
`IAM`	是	必須是有效的 SigV4 值
`CUSTOM` 或 `COGNITO_USER_POOLS`	否	未通過

- 如果政策限制了對特定 IAM 主體的存取權限，例如`arn:aws:iam::account-id:role/developer`，您必須將 API 方法`authorizationType`的方法設定為`AWS_IAM`或`NONE`。如需如何設定方法的更`authorizationType`多指示，請參閱the section called “方法”。
- VPC 端點政策可與 API Gateway 資源政策搭配使用。API Gateway 資源策略指定哪些主體可以存取 API。端點策略指定誰可以存取 VPC，以及可從 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 端點原則。

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：

- 可執行動作的委託人。
- 可執行的動作。
- 可對其執行動作的資源。

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。

```
{
    "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。

```
{
    "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，由於政策會限制特定 IAM 主體的存取權，因此您必須將AWS_IAM方法authorizationType的設定為或。NONE

```
{
    "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/*"
            ]
        }
    ]
```

```
}
```

使用標籤來控制對 API Gateway 中的 REST API 的存取

您可以在 IAM 政策中使用屬性型存取控制，以微調對 REST API 的存取許可。

如需更多詳細資訊，請參閱 the section called “屬性型存取控制”。

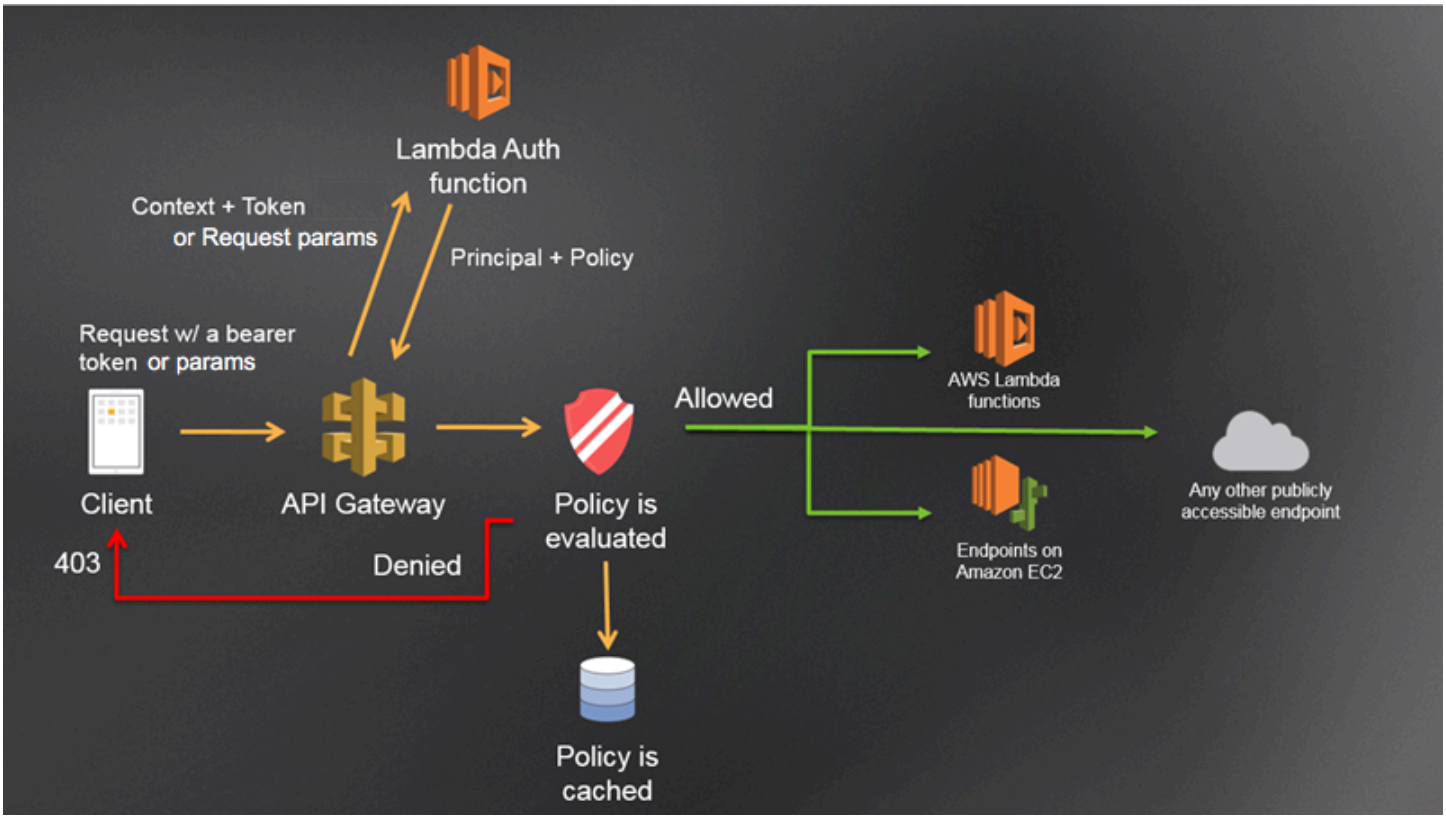
使用 API Gateway Lambda 授權方

使用 Lambda 授權者 (以前稱為自訂授權者) 來控制 API 的存取權。當用戶端向您的 API 方法提出要求時，API Gateway 會呼叫您的 Lambda 授權者。Lambda 授權者會將呼叫者的身分識別視為輸入，並傳回 IAM 政策做為輸出。

使用 Lambda 授權器實作自訂授權配置。您的配置可以使用請求參數來確定調用者的身份，或者使用承載令牌身份驗證策略，例如 OAuth 或 SAML。使用或 AWS 開發套件，在 API Gateway REST API 主控台中建立 Lambda 授權者。 AWS CLI

授 Lambda 者授權工作流程

下圖顯示 Lambda 授權者的授權工作流程。



API Gateway Lambda 授權工作流程

1. 客戶端調用 API Gateway 上的方法，傳遞承載令牌或請求參數。
2. API Gateway 會檢查方法請求是否已使用 Lambda 授權器進行設定。如果已設定，API Gateway 將會呼叫 Lambda 函式。
3. Lambda 函數會驗證呼叫者。該函數可以通過以下方式進行身份驗證：
   - 通過調用 OAuth 提供程序以獲取 OAuth 訪問令牌。
   - 呼叫 SAML 提供者以取得 SAML 宣告。
   - 根據請求參數值產生 IAM 政策。
   - 從資料庫擷取認證。
4. Lambda 函數會傳回 IAM 政策和主要識別碼。如果 Lambda 函數未傳回該資訊，則呼叫會失敗。
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 會叫用該方法。

     如果您啟用授權快取，API Gateway 會快取原則，以便不再叫用 Lambda 授權者函數。

您可以自訂`403 ACCESS_DENIED`或`401 UNAUTHORIZED`閘道回應。如需進一步了解，請參閱the section called “閘道回應”。

選擇一種 Lambda 授權者類型

Lambda 授權方有兩種類型：

請求以參數為基礎的 Lambda 授權者 (授權者) **REQUEST**

`REQUEST`授權者會以標頭、查詢字串參數和`$context`變數的組合接收呼叫者的身分。`stageVariables`您可以使用`REQUEST`授權者根據來自多個身分識別來源 (例如`$context.path`和上下`$context.httpMethod`文變數) 的資訊來建立精細的原則。

如果您為授權`REQUEST`者開啟授權快取，API Gateway 會驗證要求中是否存在所有指定的身分識別來源。如果指定的識別來源遺失、空值或空白，API Gateway 會傳回 `401 Unauthorized` HTTP 回應，而不呼叫 Lambda 授權者函數。定義多個身分識別來源時，它們都會用來衍生授權者的快取金鑰，並保留順序。您可以使用多個身分識別來源來定義精細的快取金鑰。

如果您變更任何快取金鑰部分，並重新部署 API，授權者會捨棄快取的政策文件並產生新的原則文件。

如果您關閉授權REQUEST者的授權快取，API Gateway 會直接將請求傳遞給 Lambda 函數。

基於令牌的 Lambda 授權者（授權者）**TOKEN**

TOKEN授權者在承載令牌中接收呼叫者的身份，例如 JSON Web 令牌（JWT）或 OAuth 令牌。

如果您開啟授權TOKEN者的授權快取，在權杖來源中指定的標頭名稱會成為快取金鑰。

此外，您可以使用權杖驗證來輸入 RegEx 陳述式。API Gateway 會針對此運算式執行輸入權杖的初始驗證，並在成功驗證後叫用 Lambda 授權者函數。這樣做有助於減少對 API 的呼叫。

該IdentityValidationExpression屬性僅支持TOKEN授權者。如需詳細資訊，請參閱 the section called “x-amazon-apigateway-authorizer”。

> **Note**
>
> 我們建議您使用REQUEST授權者來控制 API 的存取權。在使用授權者時，您可以根據多個身分識別來源控制 API 的存取，與使用REQUEST授權者時的單一身分識別來源相比。TOKEN此外，您可以使用REQUEST授權者的多個身分識別來源來分隔快取金鑰。

範例**REQUEST**授權者 Lambda 函數

下列範例程式碼會建立 Lambda 授權者函數，如果用戶端提供的HeaderAuth1標頭、QueryString1查詢參數和 stage 變數StageVar1分別符合指定的、和stageValue1值headerValue1queryValue1，則允許請求。

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.
    var authResponse = {};
    var condition = {};
    condition.IpAddress = {};

    if (headers.HeaderAuth1 === "headerValue1"
        && queryStringParameters.QueryString1 === "queryValue1"
        && stageVariables.StageVar1 === "stageValue1") {
        callback(null, generateAllow('me', event.methodArn));
    }  else {
        callback("Unauthorized");
    }
}

// Help function to generate an IAM policy
var generatePolicy = function(principalId, effect, resource) {
    // Required output:
    var authResponse = {};
    authResponse.principalId = principalId;
    if (effect && resource) {
        var policyDocument = {};
        policyDocument.Version = '2012-10-17'; // default version
        policyDocument.Statement = [];
        var statementOne = {};
```

```
        statementOne.Action = 'execute-api:Invoke'; // default action
        statementOne.Effect = effect;
        statementOne.Resource = resource;
        policyDocument.Statement[0] = statementOne;
        authResponse.policyDocument = policyDocument;
    }
    // Optional output with custom properties of the String, Number or Boolean type.
    authResponse.context = {
        "stringKey": "stringval",
        "numberKey": 123,
        "booleanKey": true
    };
    return authResponse;
}

var generateAllow = function(principalId, resource) {
    return generatePolicy(principalId, 'Allow', resource);
}

var generateDeny = function(principalId, resource) {
    return generatePolicy(principalId, 'Deny', resource);
}
```

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.

import json


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.

    authResponse = {}
    condition = {}
    condition['IpAddress'] = {}

    if (headers['HeaderAuth1'] == "headerValue1" and
 queryStringParameters['QueryString1'] == "queryValue1" and
 stageVariables['StageVar1'] == "stageValue1"):
        response = generateAllow('me', event['methodArn'])
        print('authorized')
        return json.loads(response)
    else:
        print('unauthorized')
        raise Exception('Unauthorized') # Return a 401 Unauthorized response
        return 'unauthorized'

    # 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)
```

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：

```
{
  "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，授權者函數返回 401 Unauthorized HTTP 響應，並且該方法請求失敗。

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。或者，它可以返回一個包`context`含可傳遞到集成後端的其他信息的對象。如需詳細資訊，請參閱 來自 API Gateway Lambda 授權者的輸出。

在生產代碼中，您可能需要在授予授權之前對用戶進行身份驗證。您可以依照該提供者的說明文件中的指示呼叫驗證提供者，在 Lambda 函數中新增驗證邏輯。

範例**TOKEN**授權者 Lambda 函數

下列範例程式碼會建立 TOKEN Lambda 授權者函數，如果用戶端提供的 Token 值為，可讓呼叫者叫用方法。`allow`如果令牌值是，則不允許調用者調用請求`deny`。如果令牌值`unauthorized`或空字符串，則授權者函數返回`401 UNAUTHORIZED`響應。

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：

```
{
  "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：

```
{
  "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，API Gateway 會傳回 `403 Forbidden` HTTP 回應，且方法要求失敗。

- 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。或者，它可以返回一個包含可傳遞到集成後端的其他信息的對象。如需詳細資訊，請參閱 來自 API Gateway Lambda 授權者的輸出。

在生產代碼中，您可能需要在授予授權之前對用戶進行身份驗證。您可以依照該提供者的說明文件中的指示呼叫驗證提供者，在 Lambda 函數中新增驗證邏輯。

Lambda 授權者函數的其他範例

下列清單顯示 Lambda 授權者函數的其他範例。您可以在建立 API 的相同帳戶或不同帳戶中建立 Lambda 函數。

對於前面的 Lambda 函數範例，您可以使用內建函數 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，則需要將 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。

其他 Lambda 授權者函數範例

- 有關示例應用程序，請參閱巴西開放銀行-授權樣本 GitHub。
- 有關 Lambda 函數的更多示例，請參閱 aws-apigateway-lambda-authorizer。 GitHub
- 您可以建立 Lambda 授權器，使用 Amazon Cognito 使用者集區對使用者進行身份驗證，並使用已驗證許可根據政策存放區授權來電者。如需詳細資訊，請參閱 Amazon 驗證許可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。
- Lambda 主控台提供 Python 藍圖，您可以選擇使用藍圖並選擇藍圖來使用該api-gateway-authorizer-python藍圖。

設定 Lambda 授權者

建立 Lambda 函數之後，您可以將 Lambda 函數設定為 API 的授權者。然後，您將方法設定為叫用 Lambda 授權器，以判斷呼叫者是否可以叫用您的方法。您可以在建立 API 的相同帳戶或不同帳戶中建立 Lambda 函數。

您可以使用 API Gateway 主控台中的內建工具或使用郵差來測試 Lambda 授權者。如需如何使用郵遞員測試 Lambda 授權者函數的指示，請參閱the section called “使用 Lambda 授權方呼叫 API”。

設定 Lambda 授權者 (主控台)

下列程序說明如何在 API Gateway REST API 主控台中建立 Lambda 授權器。若要進一步了解不同類型的 Lambda 授權者，請參閱the section called “選擇一種 Lambda 授權者類型”。

REQUEST authorizer

若要設定 **REQUEST** Lambda 授權者

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 [授權者]。
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. 對於 Lambda 函數，請選取 AWS 區域 您建立 Lambda 授權者函數的位置，然後輸入函數名稱。
7. 將 Lambda 叫用角色保持空白，讓 API Gateway REST API 主控台設定以資源為基礎的政策。該政策授予 API Gateway 許可以叫用 Lambda 授權器函數。您也可以選擇輸入 IAM 角色的名稱，以允許 API Gateway 叫用 Lambda 授權者函數。如需角色範例，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。啟用原則快取時，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，您的授權者必須傳回適用於所有 API 方法的原則。若要強制執行特定方法的原則，請使用內容變數`$context.path`和. `$context.httpMethod`
11. 選擇建立授權方。

TOKEN authorizer

若要設定 **TOKEN** Lambda 授權者

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 [授權者]。
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. 對於 Lambda 函數，請選取 AWS 區域 您建立 Lambda 授權者函數的位置，然後輸入函數名稱。
7. 將 Lambda 叫用角色保持空白，讓 API Gateway REST API 主控台設定以資源為基礎的政策。該政策授予 API Gateway 許可以叫用 Lambda 授權器函數。您也可以選擇輸入 IAM 角色的名稱，以允許 API Gateway 叫用 Lambda 授權者函數。如需角色範例，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，才能將授權權杖傳送給 Lambda 授權者。
10. (選擇性) 對於權杖驗證，請輸入 RegEx 陳述式。API Gateway 會對此表達式執行輸入字符的初始驗證，並在成功驗證時叫用授權方。
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。啟用原則快取時，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，您的授權者必須傳回適用於所有 API 方法的原則。若要強制執行特定方法的原則，您可以關閉授權快取。
12. 選擇建立授權方。

建立 Lambda 授權者之後，您就可以對其進行測試。下列程序說明如何測試 Lambda 授權者。

REQUEST authorizer

若要測試 **REQUEST** Lambda 授權者

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 選擇授權者的姓名。
3. 在「測試授權者」下，輸入身分識別來源的值。

   如果您正在使用the section called “範例REQUEST授權者 Lambda 函數”，請執行下列操作：

   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`**。

您無法修改測試叫用的內容變數，但可以修改 Lambda 函數的 API Gateway 授權者測試事件範本。然後，您可以使用修改後的內容變數來測試 Lambda 授權者函數。如需詳細資訊，請參閱AWS Lambda 開發人員指南中的主控台測試 Lambda 函數。

4. 選擇測試授權方。

TOKEN authorizer

若要測試 **TOKEN** Lambda 授權者

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 選擇授權者的姓名。
3. 在「測試授權者」下，輸入權杖的值。

   如果您正在使用the section called “範例TOKEN授權者 Lambda 函數”，請執行下列操作：

   - 針對 authorizationToken，輸入。**allow**
4. 選擇測試授權方。

   如果您的 Lambda 授權者在測試環境中成功拒絕了請求，則測試會以 `200 OK` HTTP 回應進行回應。不過，在測試環境之外，API Gateway 會傳回 `403 Forbidden` HTTP 回應，且方法要求會失敗。

設定 Lambda 授權者 ()AWS CLI

下列建立授權者命令顯示如何使用. AWS CLI

REQUEST authorizer

下列範例會建立`REQUEST`授權者，並使用`Authorizer`標頭和`accountId`上下文變數做為身分識別來源：

```
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

下列範例會建立TOKEN授權者，並使用Authorization標頭做為身分識別來源：

```
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 授權者之後，您就可以對其進行測試。下列test-invoke-authorizer命令顯示如何測試 Lambda 授權者：

```
aws apigateway test-invoke-authorizer --rest-api-id 1234123412 \
   --authorizer-id efg1234 \
   --headers Authorization='Value'
```

設定使用 Lambda 授權器 (主控台) 的方法

設定 Lambda 授權者之後，您必須將其附加至 API 的方法。

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. 選取一個 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，輸入與您為 API 建立 Lambda 授權器時指定的權杖來源名稱相符的標頭名稱。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 API 的方法以使用 Lambda 授權器 ()AWS CLI

設定 Lambda 授權者之後，您必須將其附加至 API 的方法。您可以建立新方法或使用修補程式作業將授權者附加至現有方法。

下列 put-method 命令顯示如何建立使用 Lambda 授權器的新方法：

```
aws apigateway put-method --rest-api-id 1234123412 \
  --resource-id a1b2c3 \
  --http-method PUT \
  --authorization-type CUSTOM \
  --authorizer-id efg1234
```

以下更新方法命令顯示如何更新現有方法以使用 Lambda 授權器：

```
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 Amazon API Gateway Lambda 授權方

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 可能會將新金鑰新增至地圖。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 授權者的輸出

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。

```
{
  "principalId": "yyyyyyyy", // The principal user identification associated with the
 token sent by the client.
```

```
  "policyDocument": {
    "Version": "2012-10-17",
    "Statement": [
      {
        "Action": "execute-api:Invoke",
        "Effect": "Allow|Deny",
        "Resource": "arn:aws:execute-
api:{regionId}:{accountId}:{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 $context資料模型、授權者、對應範本和 CloudWatch 存取記錄的變數。

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。範例輸出包含一個政策陳述式，用來封鎖 (Deny) 對 AWS 帳號 () `dev` 階段 API (`ymy8tbxw7b`) `GET` 方法的呼叫。`123456789012`

```
{
  "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。

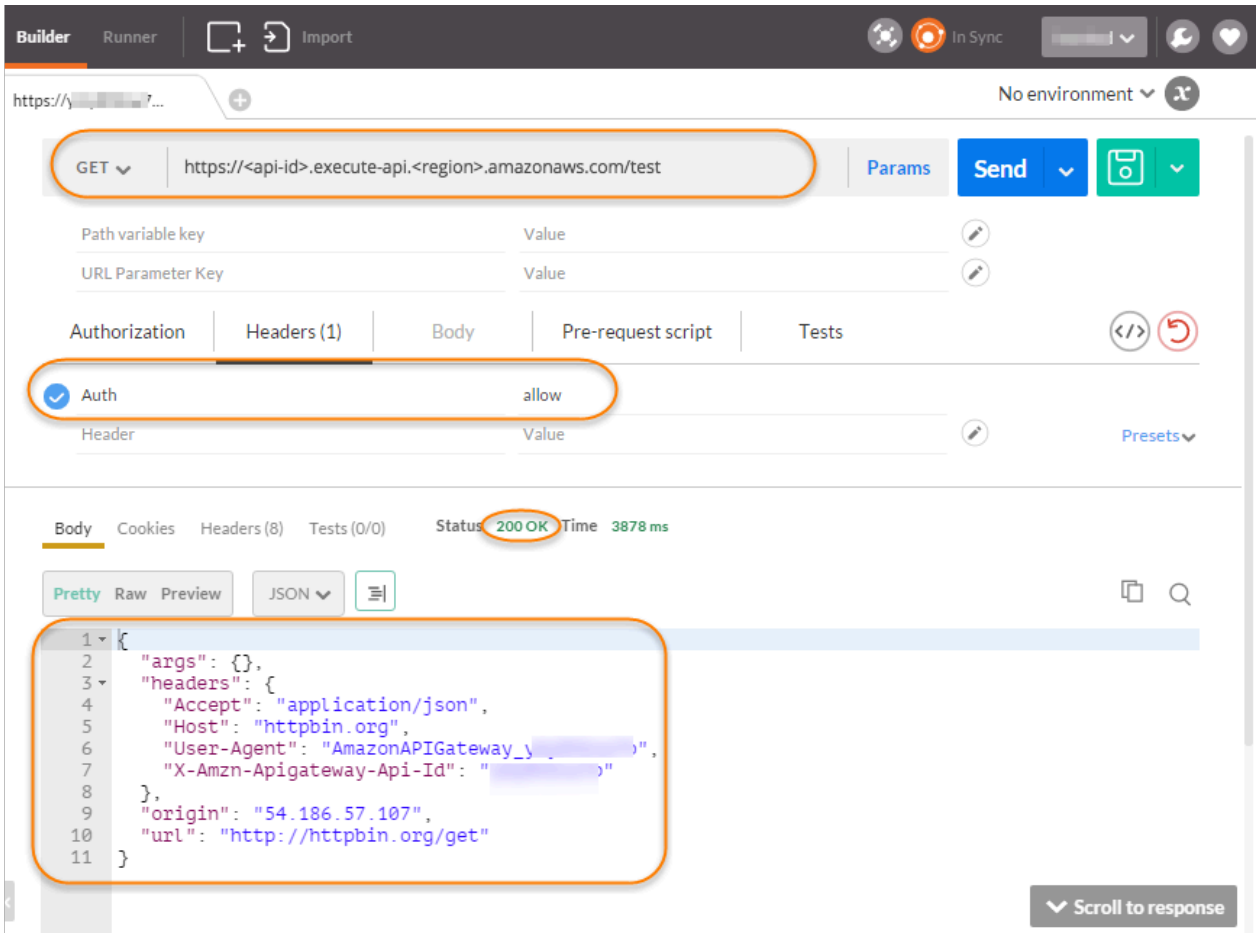
> **Note**
>
> 呼叫啟用授權者的方法時， CloudWatch 如果**`TOKEN`**授權者所需的權杖未設定、為 null 或由指定的 Token 驗證運算式無效，則 API Gateway 不會將呼叫記錄到。同樣地， CloudWatch 如果`REQUEST`授權者的任何必要身分識別來源未設定、為空值或空白，API Gateway 也不會將呼叫記錄到。

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

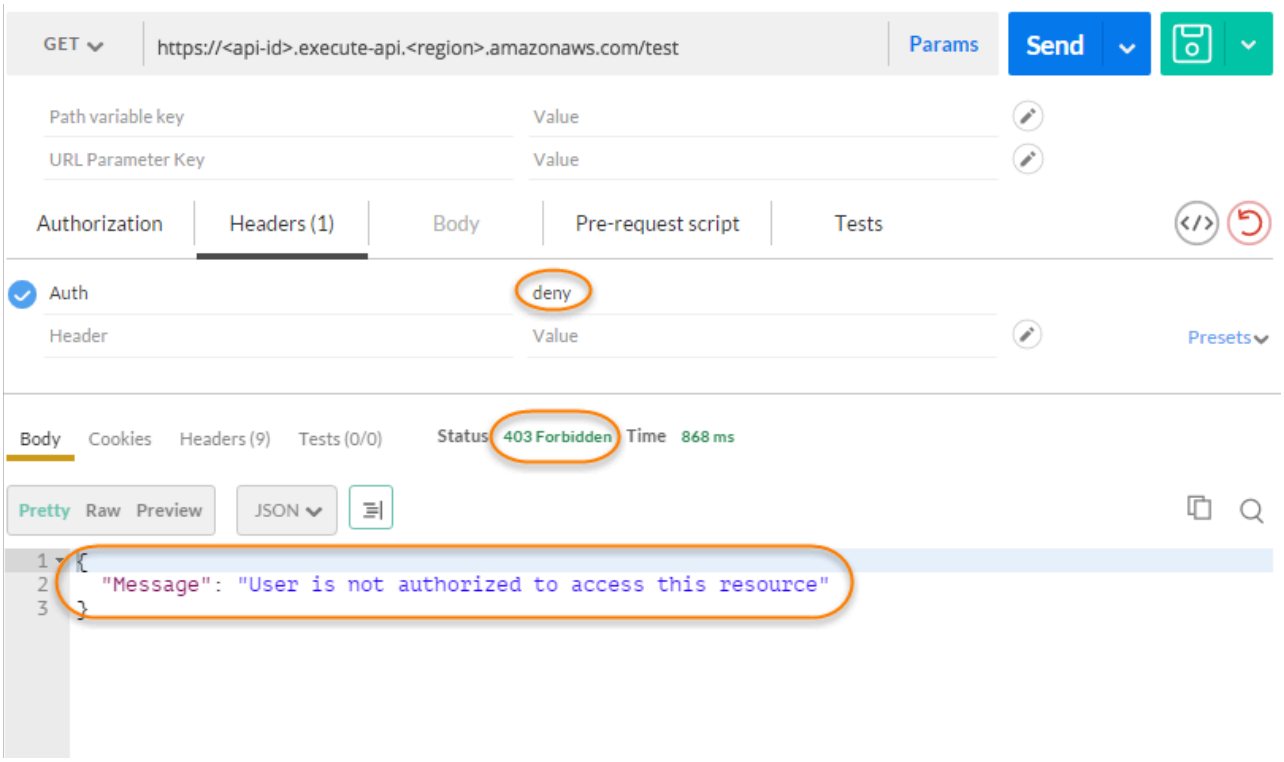
1. 開啟 Postman，選擇 GET 方法，然後將 API 的 Invoke URL (呼叫 URL) 貼到相鄰的 URL 欄位中。

   新增 Lambda 授權字符標頭，並將值設定為 `allow`。選擇 Send (傳送)。



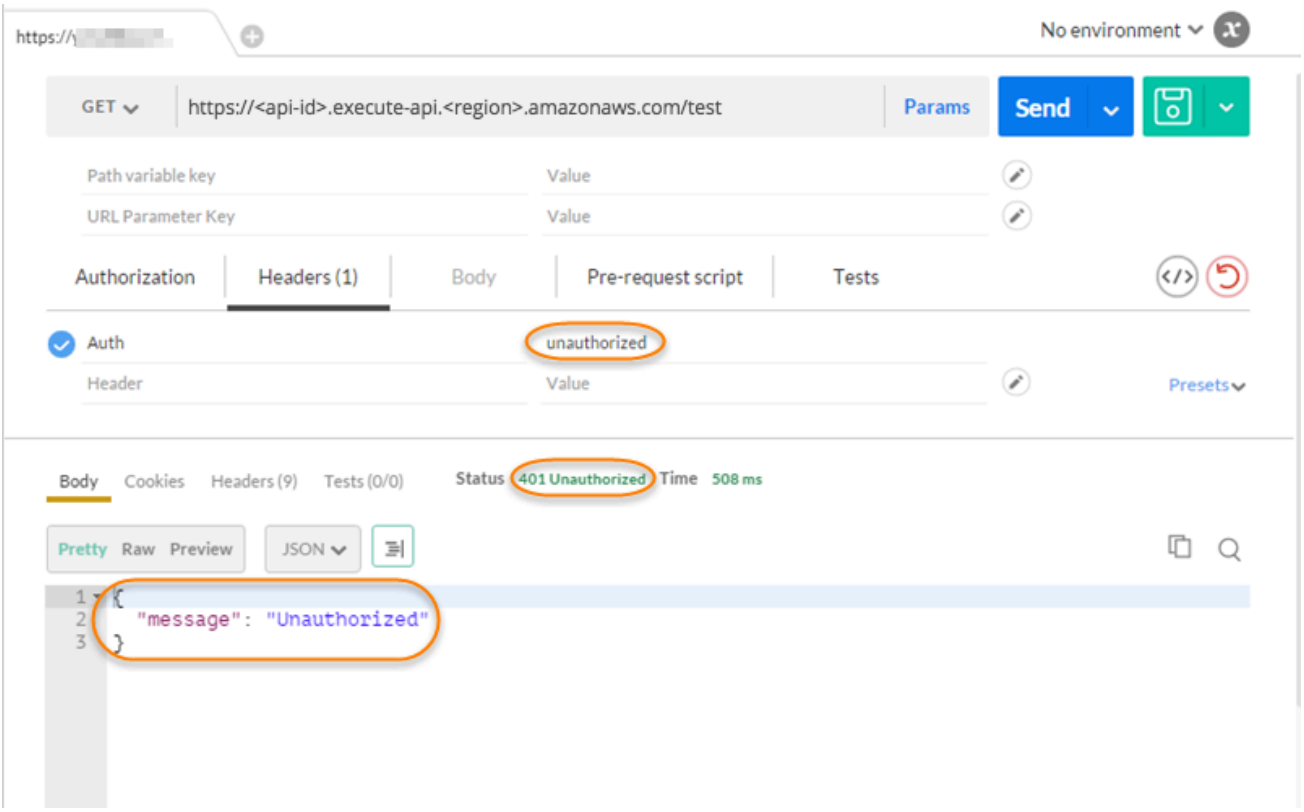
   回應顯示 API Gateway Lambda 授權方傳回 200 OK (200 OK) 回應，並成功授權呼叫存取與方法整合的 HTTP 端點 (http://httpbin.org/get)。

2. 同樣在 Postman 中，將 Lambda 授權字符標頭值變更為 `deny`。選擇 Send (傳送)。



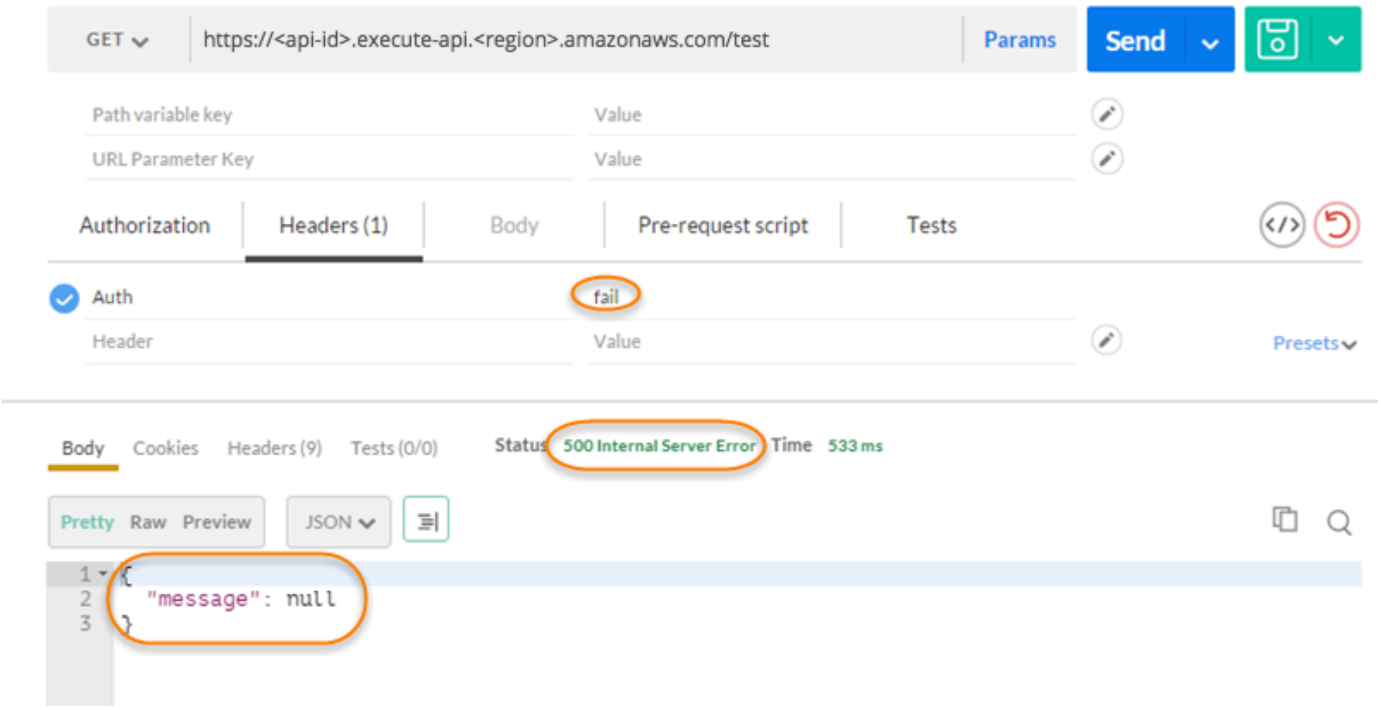
回應顯示 API Gateway Lambda 授權方傳回 403 Forbidden (403 禁止) 回應，而不授權呼叫存取 HTTP 端點。

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 Lambda 授權方

您現在也可以使用來自不同 AWS 帳戶的 AWS Lambda 函數作為 API 授權者函數。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帳戶中已有 API Gateway API，另一個 AWS 帳戶中已有 Lambda 授權者函數。

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. 將 Lambda 調用角色保留空白，以便讓 API Gateway 主控台設定資源型政策。此政策會授與 API Gateway 許可，以調用授權方 Lambda 函數。您也可以選擇輸入 IAM 角色，以允許 API Gateway 調用授權方 Lambda 函數。如需這類角色的範例，請參閱「建立可擔任的 IAM 角色」。
8. 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。
9. 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。
- 選擇性地針對變數替代字驗證，輸入對 RegEx 帳單。API Gateway 會對此表達式執行輸入字符的初始驗證，並在成功驗證時叫用授權方。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 或空白，API Gateway 會傳回 `401 Unauthorized` 回應，而不會呼叫授權方 Lambda 函數。

  定義多個身分來源時，會使用這些來源衍生授權方的快取金鑰。變更任何快取金鑰部分都會使授權方捨棄快取的政策文件，並產生新的文件。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。

10. 選擇建立授權方。

11. 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。

使用 Amazon Cognito 使用者集區作為授權方來控制對 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 建立使用者集區，或使用其他帳戶擁有的使用者集區。 AWS
- 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開發套件，請參閱使用開發套件的 Amazon Cognito 程式碼範例。 AWS
- 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

- 取得為 REST API 建立 Amazon Cognito 使用者集區授權方的許可
- 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
- 使用以下方式為 REST API 建立亞馬遜認知授權者 AWS CloudFormation

取得為 REST API 建立 Amazon Cognito 使用者集區授權方的許可

若要使用 Amazon Cognito 使用者集區建立授權方，您必須有能在所選 Amazon Cognito 使用者集區中建立或更新授權方的 `Allow` 許可。下列 IAM 政策文件顯示此類許可的範例：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "apigateway:POST"
            ],
            "Resource": "arn:aws:apigateway:*::/restapis/*/authorizers",
            "Condition": {
                "ArnLike": {
                    "apigateway:CognitoUserPoolProviderArn": [
                        "arn:aws:cognito-idp:us-east-1:123456789012:userpool/us-
east-1_aD06NQmjO",
                        "arn:aws:cognito-idp:us-east-1:234567890123:userpool/us-
east-1_xJ1MQtPEN"
```

```
                    ]
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "apigateway:PATCH"
            ],
            "Resource": "arn:aws:apigateway:*::/restapis/*/authorizers/*",
            "Condition": {
                "ArnLike": {
                    "apigateway:CognitoUserPoolProviderArn": [
                        "arn:aws:cognito-idp:us-east-1:123456789012:userpool/us-
east-1_aD06NQmjO",
                        "arn:aws:cognito-idp:us-east-1:234567890123:userpool/us-
east-1_xJ1MQtPEN"
                    ]
                }
            }
        }
    ]
}
```

確定政策連接到您所屬的 IAM 群組，或您被指派的 IAM 角色。

在前一份政策文件中，apigateway:POST 動作是用於建立新的授權方，而 apigateway:PATCH 動作是用於更新現有的授權方。您可以分別覆寫 Resource 值的前兩個萬用字元 (*)，將政策限制在特定區域或特定的 API。

Condition 子句用在這裡以限制指定使用者集區的 Allowed 許可。當有 Condition 子句時，拒絕任何不符合條件的存取使用者集區。當許可沒有 Condition 子句時，允許存取任何使用者集區。

設定 Condition 子句有下列選項：

- 您可以設定 ArnLike 或 ArnEquals 條件式表達式，只允許在指定的使用者集區中建立或更新 COGNITO_USER_POOLS 授權方。
- 您可以設定 ArnNotLike 或 ArnNotEquals 條件式表達式，允許在表達式中未指定的任何使用者集區中建立或更新 COGNITO_USER_POOLS 授權方。
- 您可以省略 Condition 子句，允許在任何區域之任何 AWS 帳戶的任何使用者集區中建立或更新 COGNITO_USER_POOLS 授權方。

如需 Amazon Resource Name (ARN) 條件表達式的詳細資訊，請參閱 Amazon Resource Name 條件運算子。如範例所示，apigateway:CognitoUserPoolProviderArn 是 COGNITO_USER_POOLS 使用者集區的 ARN 清單，可以或無法和 COGNITO_USER_POOLS 類型的 API Gateway authorizer 一起使用。

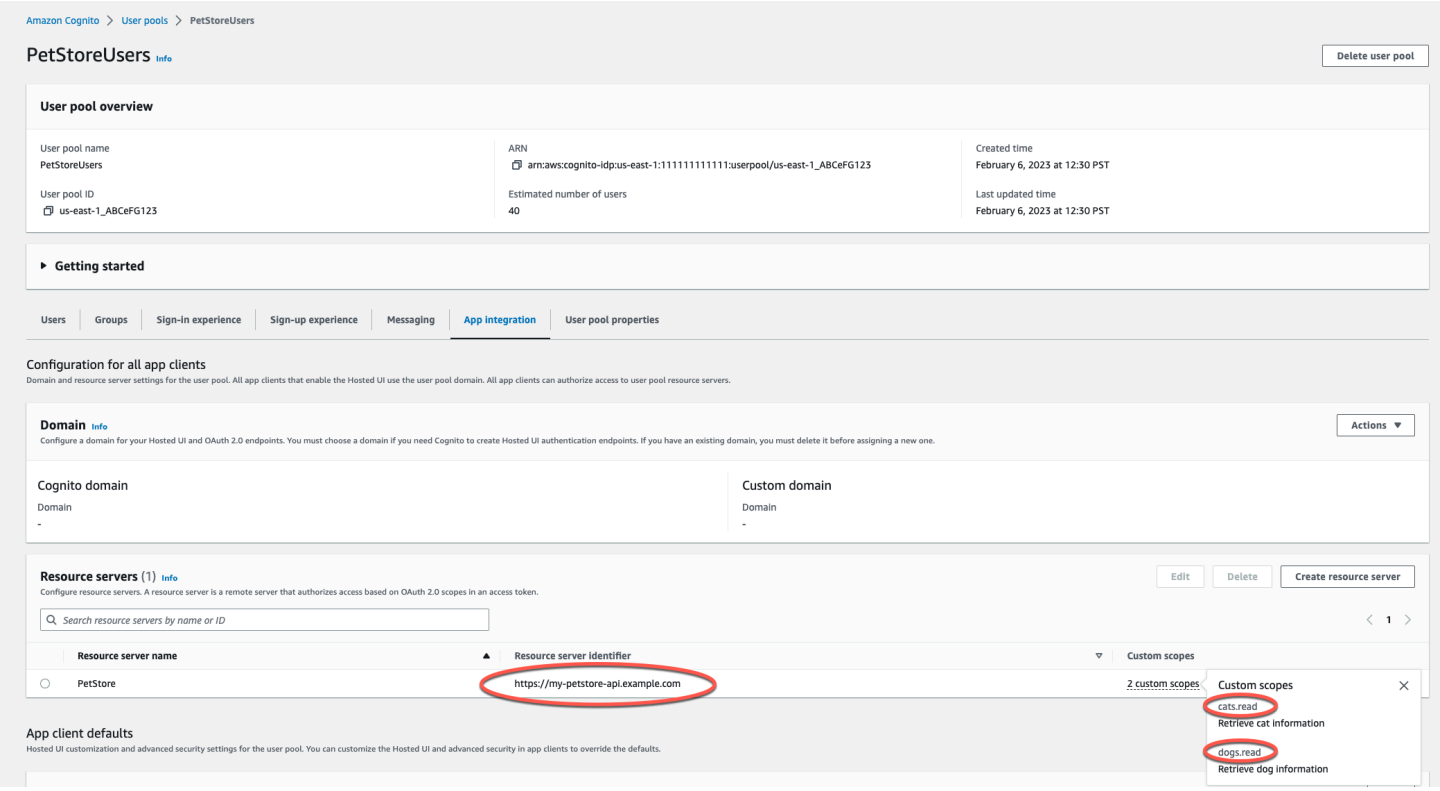
為 REST API 建立 Amazon Cognito 使用者集區

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 Amazon 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. 對於 Cognito 使用者集區，請選擇您建立 Amazon Cognito 的 AWS 區域 位置，然後選取可用的使用者集區。

      您可以使用階段變數來定義您的使用者集區。使用下列格式適用於您的使用者集區：`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，選擇性地對它進行呼叫測試。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 使用 OpenAPI 的 API Gateway 延伸。

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 Amplify 入門。

使用 API Gateway 主控台為 REST API 設定跨帳戶的 Amazon Cognito 授權方

您現在也可以使用來自不同 AWS 帳戶的 Amazon Cognito 使用者集區做為 API 授權者。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。

這些說明假設您在一個帳戶中已有 API Gateway API，而另一個 AWS 帳戶中已有 Amazon Cognito 使用者集區。

使用 API Gateway 主控台設定跨帳戶 Amazon Cognito 授權方

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。

使用以下方式為 REST API 建立亞馬遜認知授權者 AWS CloudFormation

您可以用 AWS CloudFormation 來建立 Amazon Cognito 使用者集區和 Amazon Cognito 可授權者。範例 AWS CloudFormation 範本會執行下列動作：

- 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
          - /
```

設定 REST API 整合

設定 API 方法後，您必須整合它與後端的端點。後端端點也稱為整合端點，可以是 Lambda 函數、HTTP 網頁或 AWS 服務動作。

如同使用 API 方法一樣，API 整合有整合請求和整合回應。整合請求會封裝後端收到的 HTTP 請求。它可能會，也可能不會與用戶端提交的方法不同。整合回應是封裝後端傳回輸出的 HTTP 回應。

設定整合請求包括以下內容：設定如何將用戶端提交的方法請求傳送到後端；設定如何轉換請求資料，如有必要，轉換成整合請求資料；指定呼叫哪個 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
- 在 API Gateway 中設定 Lambda 整合
- 在 API Gateway 中設定 HTTP 整合
- 設定 API Gateway 私有整合
- 在 API Gateway 中設定模擬整合

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 API Gateway 所需許可的 IAM 角色，以代表您呼叫後端。
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。「使用 API Gateway 主控台設定請求與回應資料映射」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，獲取寵物頁面的方法請求具有以下整合請求 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。

使用其他 AWS 服務動作設定整合要求時，整合請求 URI 也是 ARN，類似於與 Lambda Invoke 動作的整合。例如，對於與 Amazon S3 GetBucket動作的整合，整合請求 URI 是以下格式的 ARN：

```
arn:aws:apigateway:api-region:s3:path/{bucket}
```

整合請求 URI 是指定動作的路徑慣例，其中 *{bucket}* 是儲存貯體名稱的預留位置。或者，也可以依其名稱參照 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，您還必須設定認證，以允許 API Gateway 呼叫整合動作。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。討論過整合回應的基本設定之後，我們會在「使用 API Gateway 主控台設定請求與回應資料映射」中涵蓋進階主題，同時論及傳送承載和處理內容編碼。

選擇 API Gateway API 整合類型

您可以根據您要使用的整合端點類型以及希望資料移入移出整合端點的方式，選擇 API 整合類型。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 維修動作，您只能 AWS 整合非 Proxy 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 自訂整合是整合的特殊案例，其中 AWS 整合端點會對應至 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 函數的慣用整合類型，不適用於任何其他 AWS 服務動作，包括除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/`

$a_0/a_1/.../a_N$、/site/$b_0/b_1/.../b_M$ 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 [a_0, a_1, ..., a_N, b_0, b_1, ...b_M, ...] 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 主控台建立整合要求。

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。

若要設定 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. (選擇性) 您可以使用下列下拉式功能表來設定方法要求設定。選擇方法請求設置並配置您的方法請求。如需詳細資訊，請參閱的步驟 3 the section called “在主控台中編輯方法要求”。

   您也可以在建立方法之後設定方法要求設定。
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。

若要設定 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. 針對「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. (選擇性) 您可以使用下列下拉式功能表來設定方法要求設定。選擇方法請求設置並配置您的方法請求。如需詳細資訊，請參閱的步驟 3 the section called “在主控台中編輯方法要求”。

   您也可以在建立方法之後設定方法要求設定。
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。

若要設定 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. 針對「整合類型」選擇「AWS 服務」。
4. 在 [AWS 地區] 中，選擇您希望此方法用來呼叫動作的 [AWS 區域]。
5. 對於AWS 服務，請選擇您希望此方法調用的 AWS 服務。
6. 對於AWS 子網域，請輸入服務使用的 AWS 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。如需 HTTP 方法類型，請參閱您為服務選擇之 AWS 服務的 API 參考文件。AWS
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。如需可用動作和自訂資源路徑，請參閱您為服務選擇之 AWS 服務的 API 參考文件。AWS
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。指定以下格式的存取政策，包含所需動作數和資源陳述式：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
```

```
      "Action": [
        "action-statement"
      ],
      "Resource": [
        "resource-statement"
      ]
    },
    ...
  ]
}
```

如需動作和資源陳述式語法，請參閱您為服務選擇之 AWS 服務的AWS 說明文件。

如需 IAM 角色的信任關係，請指定以下動作，讓 API Gateway 代表您的 AWS 帳戶採取行動：

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

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. (選擇性) 您可以使用下列下拉式功能表來設定方法要求設定。選擇方法請求設置並配置您的方法請求。如需詳細資訊，請參閱的步驟 3 the section called “在主控台中編輯方法要求”。

    您也可以在建立方法之後設定方法要求設定。
13. 選擇建立方法。

設定模擬整合

如果您希望 API Gateway 充當後端以返回靜態響應，請使用模擬集成。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。

若要設定模擬整合

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. 針對「整合類型」選擇「模擬」。
4. (選擇性) 您可以使用下列下拉式功能表來設定方法要求設定。選擇方法請求設置並配置您的方法請求。如需詳細資訊，請參閱的步驟 3 the section called “在主控台中編輯方法要求”。

   您也可以在建立方法之後設定方法要求設定。
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。如需更多詳細資訊，請參閱 使用對應範本來覆寫 API 請求和回應參數和狀態碼。

在 API Gateway 中設定 Lambda 整合

您可以使用 Lambda 代理整合或 Lambda 非代理 (自訂) 整合，將 API 方法與 Lambda 函數整合。

在 Lambda 代理整合中，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

- 在 API Gateway 中設定 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

在 API Gateway 中設定 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
- 使用 Lambda 代理整合設定代理資源
- 設定 Lambda 代理伺服器整合 AWS CLI
- 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

了解 API Gateway Lambda 代理整合

Amazon API Gateway Lambda 代理整合是一個簡單、強大且靈活的機制，可透過設定單一 API 方法來建置 API。Lambda 代理整合可讓用戶端在後端呼叫單一 Lambda 函數。此函數可存取其他 AWS 服務的許多資源或功能，包括呼叫其他 Lambda 函數。

在 Lambda 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。

使用 Lambda 代理整合設定代理資源

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。

設定 Lambda 代理伺服器整合 AWS CLI

在本節中，我們將說明如 AWS CLI 何使用 Lambda 代理整合來設定 API。

> **Note**
>
> 如需使用 API Gateway 主控台來設定代理資源與 Lambda 代理整合搭配的詳細指示，請參閱教學課程：建置具有 Lambda 代理整合的 Hello World REST API。

例如，我們使用下列範例 Lambda 函數作為 API 的後端：

```
export const handler = function(event, context, callback) {
    console.log('Received event:', JSON.stringify(event, null, 2));
    var res ={
        "statusCode": 200,
        "headers": {
            "Content-Type": "*/*"
        }
    };
    var greeter = 'World';
    if (event.greeter && event.greeter!=="") {
        greeter =  event.greeter;
    } else if (event.body && event.body !== "") {
        var body = JSON.parse(event.body);
        if (body.greeter && body.greeter !== "") {
            greeter = body.greeter;
        }
    } else if (event.queryStringParameters && event.queryStringParameters.greeter &&
 event.queryStringParameters.greeter !== "") {
        greeter = event.queryStringParameters.greeter;
    } else if (event.multiValueHeaders && event.multiValueHeaders.greeter &&
 event.multiValueHeaders.greeter != "") {
        greeter = event.multiValueHeaders.greeter.join(" and ");
    } else if (event.headers && event.headers.greeter && event.headers.greeter != "") {
        greeter = event.headers.greeter;
    }
```

```
    res.body = "Hello, " + greeter + "!";
    callback(null, res);
};
```

將此項目與 Lambda 自訂整合設定進行比較，可以在請求參數與內文中表示此 Lambda 函數的輸入。您可以有更多的自由，讓用戶端傳遞相同的輸入資料。在這裡，用戶端可以傳入接待員名稱以做為查詢字串參數、標頭或內文屬性。此函數也可以支援 Lambda 自訂整合。API 設定較為簡單。您根本不需要設定方法回應或整合回應。

若要設定 Lambda 代理伺服器整合 AWS CLI

1. 呼叫 `create-rest-api` 命令來建立 API：

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

   請記下回應中所產生 API 的 `id` 值 (`te6si5ach7`)：

   ```
   {
       "name": "HelloWorldProxy (AWS CLI)",
       "id": "te6si5ach7",
       "createdDate": 1508461860
   }
   ```

   在本節中，您需要 API `id`。

2. 呼叫 `get-resources` 命令來取得根資源 `id`：

   ```
   aws apigateway get-resources --rest-api-id te6si5ach7 --region us-west-2
   ```

   成功回應顯示如下：

   ```
   {
       "items": [
           {
               "path": "/",
               "id": "krznpq9xpg"
           }
       ]
   }
   ```

記下根資源 id 值 (krznpq9xpg)。下一個步驟和之後的步驟都需要它。

3. 呼叫 create-resource 以建立 /greeting 的 API Gateway 資源：

```
aws apigateway create-resource --rest-api-id te6si5ach7 \
      --region us-west-2 \
      --parent-id krznpq9xpg \
      --path-part {proxy+}
```

成功回應類似如下：

```
{
    "path": "/{proxy+}",
    "pathPart": "{proxy+}",
    "id": "2jf6xt",
    "parentId": "krznpq9xpg"
}
```

記下所產生 {proxy+} 資源的 id 值 (2jf6xt)。在下一個步驟中，您需要它才能在 /{proxy+} 資源上建立方法。

4. 呼叫 put-method 來建立 ANY 方法請求 ANY /{proxy+}：

```
aws apigateway put-method --rest-api-id te6si5ach7 \
       --region us-west-2 \
       --resource-id 2jf6xt \
       --http-method ANY \
       --authorization-type "NONE"
```

成功回應類似如下：

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

此 API 方法可讓用戶端從後端的 Lambda 函數接收或傳送問候語。

5. 呼叫 `put-integration` 來設定具有 Lambda 函數 (即 `ANY /{proxy+}`) 之 `HelloWorld` 方法的整合。如果提供 `"Hello, {name}!"` 參數，則此函數會以 `greeter` 訊息來回應請求，如果未設定查詢字串參數，則會以 `"Hello, World!"` 來回應請求。

```
aws apigateway put-integration \
        --region us-west-2 \
        --rest-api-id 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”。

成功輸出類似如下：

```
{
    "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"
}
```

您可以呼叫 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。

6. 呼叫 `create-deployment` 來將 API 部署至 `test` 階段：

```
aws apigateway create-deployment --rest-api-id te6si5ach7 --stage-name test --
region us-west-2
```

7. 在終端機中使用下列 cURL 命令，以測試 API。

   使用查詢字串參數 `?greeter=jane` 呼叫 API：

   ```
   curl -X GET 'https://te6si5ach7.execute-api.us-west-2.amazonaws.com/test/greeting?
   greeter=jane'
   ```

   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!
   ```

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。

```
{
  "resource": "/my/path",
  "path": "/my/path",
  "httpMethod": "GET",
  "headers": {
    "header1": "value1",
    "header2": "value1,value2"
  },
```

```
  "multiValueHeaders": {
    "header1": [
      "value1"
    ],
    "header2": [
      "value1",
      "value2"
    ]
  },
  "queryStringParameters": {
    "parameter1": "value1,value2",
    "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": "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 可能會將新金鑰新增至地圖。

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。

在 API Gateway 中設定 Lambda 自訂整合

為了示範如何設定 Lambda 自訂整合，我們建立 API Gateway API 來公開 `GET /greeting?greeter={name}` 方法以呼叫 Lambda 函數。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

```
export const handler = function(event, context, callback) {
    var res ={
        "statusCode": 200,
        "headers": {
            "Content-Type": "*/*"
        }
    };
```

```
    if (event.greeter==null) {
        callback(new Error('Missing the required greeter parameter.'));
    } else if (event.greeter === "") {
        res.body = "Hello, World";
        callback(null, res);
    } else {
        res.body = "Hello, " + event.greeter +"!";
        callback(null, res);
    }
};
```

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。

若要設定 Lambda 自訂整合 AWS CLI

1. 呼叫 `create-rest-api` 命令來建立 API：

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

   請記下回應中所產生 API 的 `id` 值 (`te6si5ach7`)：

   ```
   {
       "name": "HelloWorld (AWS CLI)",
       "id": "te6si5ach7",
       "createdDate": 1508461860
   }
   ```

   在本節中，您需要 API `id`。
2. 呼叫 `get-resources` 命令來取得根資源 `id`：

   ```
   aws apigateway get-resources --rest-api-id te6si5ach7 --region us-west-2
   ```

   成功回應如下：

   ```
   {
       "items": [
           {
               "path": "/",
               "id": "krznpq9xpg"
           }
       ]
   }
   ```

   記下根資源 `id` 值 (`krznpq9xpg`)。下一個步驟和之後的步驟都需要它。
3. 呼叫 `create-resource` 以建立 `/greeting` 的 API Gateway 資源：

   ```
   aws apigateway create-resource --rest-api-id te6si5ach7 \
         --region us-west-2 \
   ```

```
      --parent-id krznpq9xpg \
      --path-part greeting
```

成功回應類似如下：

```
{
    "path": "/greeting",
    "pathPart": "greeting",
    "id": "2jf6xt",
    "parentId": "krznpq9xpg"
}
```

記下所產生 greeting 資源的 id 值 (2jf6xt)。在下一個步驟中，您需要它才能在 /greeting 資源上建立方法。

4. 呼叫 put-method 來建立 API 方法請求 GET /greeting?greeter={name}：

```
aws apigateway put-method --rest-api-id te6si5ach7 \
       --region us-west-2 \
       --resource-id 2jf6xt \
       --http-method GET \
       --authorization-type "NONE" \
       --request-parameters method.request.querystring.greeter=false
```

成功回應類似如下：

```
{
    "apiKeyRequired": false,
    "httpMethod": "GET",
    "authorizationType": "NONE",
    "requestParameters": {
        "method.request.querystring.greeter": false
    }
}
```

這個 API 方法可讓用戶端從後端的 Lambda 函數接收問候語。greeter 是選用參數，因為後端應該處理匿名發起人或自我識別發起人。

5. 呼叫 put-method-response 來設定 200 OK 方法請求的 GET /greeting?greeter={name} 回應：

```
aws apigateway put-method-response \
        --region us-west-2 \
        --rest-api-id te6si5ach7 \
        --resource-id 2jf6xt \
        --http-method GET \
        --status-code 200
```

6. 呼叫 put-integration 來設定具有 Lambda 函數 (即 GET /greeting?greeter={name}) 之 HelloWorld 方法的整合。如果提供 "Hello, {name}!" 參數，則此函數會以 greeter 訊息來回應請求，如果未設定查詢字串參數，則會以 "Hello, World!" 來回應請求。

```
aws apigateway put-integration \
        --region us-west-2 \
        --rest-api-id 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叫用之 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。
> 成功輸出與下列輸出類似：

```
{
    "passthroughBehavior": "WHEN_NO_MATCH",
    "cacheKeyParameters": [],
    "uri": "arn:aws:apigateway:us-east-1:lambda:path/2015-03-31/functions/
arn:aws:lambda:us-east-1:123456789012:function:HelloWorld/invocations",
```

```
    "httpMethod": "POST",
    "requestTemplates": {
        "application/json": "{\"greeter\":\"$input.params('greeter')\"}"
    },
    "cacheNamespace": "krznpq9xpg",
    "credentials": "arn:aws:iam::123456789012:role/apigAwsProxyRole",
    "type": "AWS"
}
```

IAM 角色 apigAwsProxyRole 的政策必須允許 apigateway 服務叫用 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。

7. 呼叫 put-integration-response 來設定整合回應，以將 Lambda 函數輸出當作 200 OK 方法回應傳遞至用戶端。

```
aws apigateway put-integration-response \
       --region us-west-2 \
       --rest-api-id te6si5ach7 \
       --resource-id 2jf6xt \
       --http-method GET \
       --status-code 200 \
       --selection-pattern ""
```

透過將選取模式設定為空白字串，200 OK 回應是預設值。

成功回應應該類似如下：

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

8. 呼叫 create-deployment 來將 API 部署至 test 階段：

```
aws apigateway create-deployment --rest-api-id te6si5ach7 --stage-name test --
region us-west-2
```

9. 在終端機中使用下列 cURL 命令，以測試 API：

```
curl -X GET 'https://te6si5ach7.execute-api.us-west-2.amazonaws.com/test/greeting?
greeter=me' \
  -H 'authorization: AWS4-HMAC-SHA256 Credential={access_key}/20171020/us-
west-2/execute-api/aws4_request, SignedHeaders=content-type;host;x-amz-date,
 Signature=f327...5751'
```

設定後端 Lambda 函數的非同步叫用

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：

- 將標`X-Amz-Invocation-Type`題新增至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. 將標X-Amz-Invocation-Type題新增至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 非同步叫用 AWS CloudFormation

下列 AWS CloudFormation 範本顯示如何設定`AWS::ApiGateway::Method`非同步呼叫。

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 錯誤的格式如下：

```
{
  "errorMessage": "<replaceable>string</replaceable>",
  "errorType": "<replaceable>string</replaceable>",
  "stackTrace": [
    "<replaceable>string</replaceable>",
    ...
  ]
}
```

這裡的 `errorMessage` 是錯誤的字串表達式。`errorType` 是語言相關錯誤或例外狀況類型。`stackTrace` 是字串表達式清單，顯示造成錯誤出現的堆疊追蹤。

例如，請考慮下列 JavaScript (Node.js) Lambda 函數。

```
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，若要使用 `selectionPattern` 設定新的 AWS CLI 表達式，請呼叫下列 put-integration-response 命令：

```
aws apigateway put-integration-response --rest-api-id z0vprf0mdh --resource-id x3o5ih
 --http-method GET --status-code 400 --selection-pattern "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。

若要使用 `selectionPattern` 更新現有的 AWS CLI 值，請呼叫 update-integration-response 操作，將 `/selectionPattern` 路徑值替換成 `Malformed*` 模式的指定 regex 表達式。

若要使用 API Gateway 主控台設定 `selectionPattern` 表達式，當設定或更新指定的 HTTP 狀態碼的整合回應時，請在 Lambda 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 可讓您傳回自訂錯誤物件當做 JSON 字串，不是上節所述的標準錯誤。此錯誤可以是任何有效的 JSON 物件。例如，下列 JavaScript (Node.js) Lambda 函數會傳回自訂錯誤：

```
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。

在 API Gateway 中設定 HTTP 整合

您可以使用 HTTP 代理整合或 HTTP 自訂整合，將 API 方法與 HTTP 端點進行整合。

API Gateway 支援以下端點連接埠：80、443 及 1024-65535。

使用代理整合時，設定非常簡單。如果您不在乎內容編碼或快取，您只需要根據後端需求來設定 HTTP 方法與 HTTP 端點 URI。

使用自訂整合時，需要進行更多的設定。除了代理整合設定步驟，您還需要指定傳入請求資料如何對應到整合請求，以及產生的整合回應資料如何對應到方法回應。

主題

- 在 API Gateway 中設定 HTTP 代理整合
- 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，請參閱教學課程：建置具有 HTTP 代理整合的 REST API。

以下 OpenAPI 定義文件顯示了一個 API 的示例，其中包含與PetStore網站集成的代理資源。

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 的基本路徑 (`/test`對應於階段) 會對應至網站

的 PetStore 頁面 (`/petstore`)。單一整合要求會使用 API 的貪婪路徑變數和 Catch-All ANY 方法來反映整個 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

有了 HTTP 自訂整合，您可以更精細地控制要在 API 方法與 API 整合之間傳遞哪些資料，以及如何傳遞這些資料。您會透過資料對應來執行此作業。

在方法請求設定過程中，您可以在 Method 資源上設定 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，作為整合要求設定的一部分，您可以在對應的整合資源上設定 requestParameters 屬性，以指定 parameter-to-parameter 對應。您也可以設定 requestTemplates 屬性，針對每個支援的內容類型各指定一個映射範本。對應範本會將方法請求參數或內文對應到整合請求內文。

同樣地，作為方法回應設定的一部分，您可以在資源上設定 responseParameters 屬性。MethodResponse這會宣告哪些要發送到用戶端的方法回應參數，是要從後端傳回之整合回應參數或特定適用的內文屬性對應而來。然後，作為整合回應設定的一部分，您可以在對應的IntegrationResponse資源上設定 responseParameters 屬性，以指定對 parameter-to-parameter 應。您也可以設定 responseTemplates 映射，針對每個支援的內容類型各指定一個映射範本。對應範本會將整合回應參數或整合回應內文屬性對應到方法回應內文。

如需設定映射範本的詳細資訊，請參閱設定 REST API 的資料轉換。

設定 API Gateway 私有整合

API Gateway 私有整合可以輕鬆地公開 Amazon VPC 內的 HTTP/HTTPS 資源，以供 VPC 外部的用戶端存取。若要將私有 VPC 資源的存取擴展到 VPC 邊界外部，您可以建立具有私有整合的 API。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 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。

> **Note**
>
> Network Load Balancer 和 API 必須擁有相同 AWS 帳戶。

使用 API Gateway 私有整合，您可以啟用存取 VPC 內的 HTTP/HTTPS 資源，而不需要深入了解私有網路組態或技術特定設備。

主題

- 設定 API Gateway 私有整合的 Network Load Balancer
- 授予建立 VPC 連結的許可
- 使用 API Gateway 主控台設定具有私有整合的 API Gateway API

- 設定具有私有整合的 API Gateway API AWS CLI
- 使用 OpenAPI 設定具有私有整合的 API
- 用於私有整合的 API Gateway 帳戶

設定 API Gateway 私有整合的 Network Load Balancer

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：建置具有私有整合的 API”。

> **Note**
>
> Network Load Balancer 和 API 必須擁有相同 AWS 帳戶。

使用 API Gateway 主控台建立私有整合的 Network Load Balancer

1. 登入 AWS Management Console 並開啟 Amazon EC2 主控台，網址為 https://console.aws.amazon.com/ec2/。
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. 關閉的安全性群組評估 PrivateLink。

    - 若要使用主控台關閉 PrivateLink 流量的安全群組評估，您可以選擇 [安全性] 索引標籤，然後選擇 [編輯]。在 [安全性] 設定中，清除 [對 PrivateLink 流量執行輸入規則]。
    - 若要使用關閉 PrivateLink 流量的安全群組評估 AWS CLI，請使用下列命令：

```
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。

授予建立 VPC 連結的許可

若要讓您或您帳戶中的使用者建立和維護 VPC 連結，您或使用者必須有權建立、刪除和檢視 VPC 端點服務組態、變更 VPC 端點服務許可，以及檢查負載平衡器。若要授予這類許可，請使用下列步驟。

授予建立、更新和刪除 VPC 連結的許可

1. 建立與下列類似的 IAM 政策：

```
{
    "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": "*"
        }
    ]
}
```

2. 建立或選擇 IAM 角色，並將先前的政策連接至角色。
3. 將 IAM 角色指派給您或您帳戶中建立 VPC 連結的使用者。

使用 API Gateway 主控台設定具有私有整合的 API Gateway API

如需使用 API Gateway 主控台設定具有私有整合 API 的指示，請參閱教學：建置具有 API Gateway 私有整合的 REST API。

設定具有私有整合的 API Gateway API AWS CLI

建立具有私有整合的 API 前，您必須設定 VPC 資源、使用 VPC 來源建立 Network Load Balancer，並將其設定為目標。如果需求皆不符合，請按照 設定 API Gateway 私有整合的 Network Load Balancer 來安裝 VPC 資源、建立 Network Load Balancer，並將 VPC 資源設定為 Network Load Balancer 的目標。

> **Note**
>
> Network Load Balancer 和 API 必須擁有相同 AWS 帳戶。

為了能夠建立和管理 `VpcLink`，您還必須設定適當的許可。如需詳細資訊，請參閱 授予建立 VPC 連結的許可。

> **Note**
>
> 您只需要在 API 中建立 VpcLink 的許可。您不需要使用 VpcLink 的許可。

建立 Network Load Balancer 之後，請記下其 ARN。您需要它才能建立私有整合的 VPC 連結。

若要使用私有整合設定 API AWS CLI

1. 建立目標設為所指定 Network Load Balancer 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。驗證方式是呼叫下列 CLI 命令：

```
aws apigateway get-vpc-link --vpc-link-id gim7c3
```

如果操作失敗，您會取得 FAILED 狀態，以及包含錯誤訊息的 statusMessage。例如，如果您嘗試建立具有已與 VPC 端點建立關聯之 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。

請記下新建立 id 的 VpcLink 值 (先前輸出中的 *gim7c3*)。您需要它才能設定私有整合。

2. 透過建立 API Gateway RestApi 資源來設定 API：

```
aws apigateway create-rest-api --name 'My VPC Link Test'
```

請記下所傳回結果中 RestApi 的 id 值。您需要此值才能對 API 執行進一步操作。

基於說明，我們將建立根資源 (GET) 上只有 / 方法的 API，並整合此方法與 VpcLink。

3. 設定 GET / 方法。先取得根資源的識別符 (/)：

```
aws apigateway get-resources --rest-api-id  abcdef123
```

在輸出中，記下 id 路徑的 / 值。在此範例中，假設它是 *skpp60rab7*。

設定 API 方法 GET / 的方法請求：

```
aws apigateway put-method \
       --rest-api-id  abcdef123 \
       --resource-id skpp60rab7 \
       --http-method GET \
       --authorization-type "NONE"
```

如果您未搭配使用代理整合與 VpcLink，則也必須至少設定 200 狀態碼的方法回應。我們將在這裡使用代理整合。

4. 設定 HTTP_PROXY 類型的私有整合，並呼叫 put-integration 命令，如下所示：

```
aws apigateway put-integration \
    --rest-api-id 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
```

針對私有整合，將 connection-type 設定為 VPC_LINK，並將 connection-id 設定為您 VpcLink 的識別符或參考您 VpcLink ID 的階段變數。uri 參數不是用於將請求路由至端點，但用於設定 Host 標頭以及進行憑證驗證。

命令會傳回下列輸出：

```
{
    "passthroughBehavior": "WHEN_NO_MATCH",
    "timeoutInMillis": 29000,
    "connectionId": "gim7c3",
    "uri": "http://my-vpclink-test-nlb-1234567890abcdef.us-east-2.amazonaws.com",
    "connectionType": "VPC_LINK",
    "httpMethod": "GET",
    "cacheNamespace": "skpp60rab7",
    "type": "HTTP_PROXY",
    "cacheKeyParameters": []
}
```

使用階段變數，您可以在建立整合時設定 connectionId 屬性：

```
aws apigateway put-integration \
    --rest-api-id 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。

或者，您可以更新整合，以使用階段變數來重設 connectionId 值：

```
 aws apigateway update-integration \
    --rest-api-id 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。

您可以透過重設 VpcLinks 階段變數值，使用階段變數將 API 與其他 VPC 或 Network Load Balancer 整合。

因為我們使用私有代理整合，所以 API 現在已準備好進行部署和測試執行。使用非代理整合，您也必須設定方法回應和整合回應，就像設定具有 HTTP 自訂整合的 API 一樣。

5. 若要測試 API，請部署 API。如果您已使用階段變數做為 VpcLink ID 的預留位置，則這是必要的。若要使用階段變數來部署 API，請呼叫 create-deployment 命令，如下所示：

```
aws apigateway create-deployment \
    --rest-api-id abcdef123 \
    --stage-name test \
    --variables vpcLinkId=gim7c3
```

若要更新具有不同 VpcLink ID 的階段變數 (例如，*asf9d7*)，請呼叫 update-stage 命令：

```
aws apigateway update-stage \
    --rest-api-id abcdef123 \
    --stage-name test \
    --patch-operations op=replace,path='/variables/vpcLinkId',value='asf9d7'
```

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 invoke-URL 來檢視結果。

當您硬式編碼具有 connection-id ID 常值的 VpcLink 屬性時，也可以呼叫 test-invoke-method 來測試在部署 API 之前呼叫 API。

使用 OpenAPI 設定具有私有整合的 API

您可以匯入 API OpenAPI 檔案來設定具有私有整合的 API。這些設定類似具有 HTTP 整合之 API 的 OpenAPI 定義，但例外如下：

- 您必須將 connectionType 明確地設定為 VPC_LINK。
- 您必須將 connectionId 明確地設定為 VpcLink 的 ID 或參考 VpcLink ID 的階段變數。

- 私有整合中的 uri 參數指向 VPC 中的 HTTP/HTTPS 端點，但改為用來設定整合請求的 Host 標頭。
- 具有 VPC 中 HTTPS 端點之私有整合中的 uri 參數用來針對 VPC 端點上所安裝憑證中的網域名稱來驗證所指出的網域名稱。

您可以使用階段變數來參考 VpcLink ID。或者，您可以將 ID 值直接指派給 connectionId。

下列 JSON 格式化 OpenAPI 檔案會顯示 API 的範例，具有階段變數 (${stageVariables.vpcLinkId}) 所參考的 VPC 連結：

OpenAPI 2.0

```
{
  "swagger": "2.0",
  "info": {
    "version": "2017-11-17T04:40:23Z",
    "title": "MyApiWithVpcLink"
  },
  "host": "p3wocvip9a.execute-api.us-west-2.amazonaws.com",
  "basePath": "/test",
  "schemes": [
    "https"
  ],
  "paths": {
    "/": {
      "get": {
        "produces": [
          "application/json"
        ],
        "responses": {
          "200": {
            "description": "200 response",
            "schema": {
              "$ref": "#/definitions/Empty"
            }
          }
        },
        "x-amazon-apigateway-integration": {
          "responses": {
            "default": {
              "statusCode": "200"
            }
```

```
          },
          "uri": "http://my-vpclink-test-nlb-1234567890abcdef.us-
east-2.amazonaws.com",
          "passthroughBehavior": "when_no_match",
          "connectionType": "VPC_LINK",
          "connectionId": "${stageVariables.vpcLinkId}",
          "httpMethod": "GET",
          "type": "http_proxy"
        }
      }
    }
  },
  "definitions": {
    "Empty": {
      "type": "object",
      "title": "Empty Schema"
    }
  }
}
```

用於私有整合的 API Gateway 帳戶

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

區域	帳戶 ID
eu-west-3	061510835048
eu-central-1	474240146802
eu-central-2	166639821150
eu-north-1	394634713161
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

在 API Gateway 中設定模擬整合

Amazon API Gateway 支援 API 方法的模擬整合。此功能可讓 API 開發人員直接從 API Gateway 產生 API 回應，而不需要整合後端。您是 API 開發人員，在專案開發完成之前，可以使用此功能來解鎖需要使用 API 的相依團隊。您也可以使用此功能來佈建您 API 的登錄頁面，以提供 API 的概觀，並導覽至 API。如需這類登錄頁面的範例，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課程：建置具有非 HTTP 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. 選擇新增對應範本。
   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。

在 API Gateway 中使用請求驗證

您可以先設定 API Gateway 執行 API 請求的基本驗證，再繼續進行整合請求。驗證失敗時，API Gateway 會立即失敗請求，將 400 錯誤回應傳回給呼叫者，並在 CloudWatch 記錄中發佈驗證結果。這樣可減少不必要的後端呼叫。更重要的是，它可讓您專注於應用程式特定的驗證努力。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
- 了解資料模型
- 在 API Gateway 中設定基本請求驗證
- 具有基本請求驗證之範例 API 的 OpenAPI 定義
- AWS CloudFormation 具有基本請求驗證的示例 API 模板

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。
- 適用的請求承載是否遵守在方法中設定的 JSON 結構描述請求。

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 整合傳遞行為 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 整合傳遞行為。

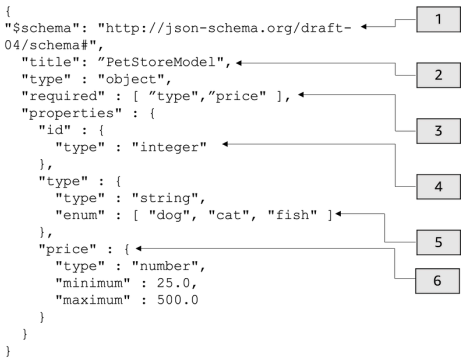
了解資料模型

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 了解對應範本。

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
  }
]
}
```

輸出模型

```
{
"$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
  - 設定基本要求驗證 AWS CLI
  - 使用 OpenAPI 定義來設定基本請求驗證
- 您可以取得有關資料轉換和對應範本 了解對應範本 的詳細資訊。

- 您還可以看到更複雜的數據模型。請參閱API Gateway 的範例資料模型和對應範本。

在 API Gateway 中設定基本請求驗證

本節說明如何使用主控台和 OpenAPI 定義設定 API Gateway 的要求驗證。 AWS CLI

主題

- 使用 API Gateway 主控台設定請求驗證
- 設定基本要求驗證 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 的對應。RequestValidators

若要遵循本教學課程，您將使用 AWS CloudFormation 範本建立不完整的 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。 AWS 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 AWS CloudFormation 堆疊的步驟

1. 請在以下位置開啟 AWS CloudFormation 主控台。 https://console.aws.amazon.com/cloudformation
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，請確認 AWS CloudFormation 可以在您的帳戶中建立 IAM 資源。
9. 選擇提交。

AWS CloudFormation 規定範本中指定的資源。完成資源佈建可能需要幾分鐘的時間。當 AWS CloudFormation 堆疊的狀態為「建立 _ 完成」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. 針對名稱，輸入 **PetStoreModel**。
4. 對於內容類型，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，請參閱了解資料模型。

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. 針對名稱，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. 針對名稱，輸入 **petType**。

   c. 對於對應來源，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**，然後選取做為「模型」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 AWS CloudFormation 堆疊的步驟

1. 請在以下位置開啟 AWS CloudFormation 主控台。 https://console.aws.amazon.com/cloudformation
2. 選取您的 AWS 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。

設定基本要求驗證 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 AWS CloudFormation 範本建立不完整的 API Gateway API。

> **Note**
>
> 這與控制台教程不同的 AWS CloudFormation 模板。

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 AWS CloudFormation 堆疊的步驟

下載並解壓縮的應用程式建立範本。 AWS 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. 使用下面的命令來創建 AWS CloudFormation 堆棧。

```
aws cloudformation create-stack --stack-name request-validation-tutorial-cli
 --template-body file://request-validation-tutorial-cli.zip --capabilities
 CAPABILITY_NAMED_IAM
```

2. AWS CloudFormation 規定範本中指定的資源。完成資源佈建可能需要幾分鐘的時間。使用以下命令查看 AWS CloudFormation 堆棧的狀態。

```
aws cloudformation describe-stacks --stack-name request-validation-tutorial-cli
```

3. 當 AWS CloudFormation 堆棧的狀態為時`StackStatus: "CREATE_COMPLETE"`，請使用以下命令檢索 future 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，也就是 API 的識別碼。對於本教學課程，API ID 為 `abc123`。
- ResourceId，這是顯示`GET`和`POST`方法之驗證器資源的 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 AWS CloudFormation 堆疊的步驟

- 使用以下命令刪除資 AWS CloudFormation 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-驗證對象 對應中指定一組 x-amazon-apigateway-request-驗證器. 請求驗證器對象 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。
- `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-驗證器屬性 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，將區域性 API 匯入 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
    }
  }
}
```

具有基本請求驗證之範例 API 的 OpenAPI 定義

下列 OpenAPI 定義會定義已啟用請求驗證的範例 API。該 API 是 API 的一個子PetStore集。它會公開 POST 方法以將寵物新增至 pets 集合，並公開 GET 方法以依指定的類型來查詢寵物。

在 API 層級的 x-amazon-apigateway-request-validators 對應中，已宣告兩種請求驗證程式。params-only 驗證程式已在 API 上予以啟用，並且透過 GET 方法繼承。此驗證程式可讓 API Gateway 驗證傳入請求中已包含不是空白的必要查詢參數 (q1)。all 驗證程式已在 POST 方法上予以啟用。此驗證程式會驗證已設定不是空白的必要標頭參數 (h1)。另外還會驗證承載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。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。

此模型需要輸入 JSON 物件包含 name、type 和 price 屬性。name 屬性可以是任意字串、type 必須是其中一個指定的列舉欄位 (["dog", "cat", "fish"])，而 price 的範圍必須是 25 到 500。id 不是必要參數。

OpenAPI 2.0

```
{
  "swagger": "2.0",
  "info": {
    "title": "ReqValidators Sample",
    "version": "1.0.0"
  },
  "schemes": [
    "https"
  ],
  "basePath": "/v1",
  "produces": [
    "application/json"
  ],
  "x-amazon-apigateway-request-validators" : {
    "all" : {
      "validateRequestBody" : true,
      "validateRequestParameters" : true
    },
    "params-only" : {
      "validateRequestBody" : false,
      "validateRequestParameters" : true
    }
  },
  "x-amazon-apigateway-request-validator" : "params-only",
  "paths": {
    "/validation": {
      "post": {
        "x-amazon-apigateway-request-validator" : "all",
```

```
        "parameters": [
          {
            "in": "header",
            "name": "h1",
            "required": true
          },
          {
            "in": "body",
            "name": "RequestBodyModel",
            "required": true,
            "schema": {
              "$ref": "#/definitions/RequestBodyModel"
            }
          }
        ],
        "responses": {
          "200": {
            "schema": {
              "type": "array",
              "items": {
                "$ref": "#/definitions/Error"
              }
            },
            "headers" : {
              "test-method-response-header" : {
                "type" : "string"
              }
            }
          }
        },
        "security" : [{
          "api_key" : []
        }],
        "x-amazon-apigateway-auth" : {
          "type" : "none"
        },
        "x-amazon-apigateway-integration" : {
          "type" : "http",
          "uri" : "http://petstore-demo-endpoint.execute-api.com/petstore/pets",
          "httpMethod" : "POST",
          "requestParameters": {
            "integration.request.header.custom_h1": "method.request.header.h1"
          },
          "responses" : {
```

```
            "2\\d{2}" : {
              "statusCode" : "200"
            },
            "default" : {
              "statusCode" : "400",
              "responseParameters" : {
                "method.response.header.test-method-response-header" : "'static
value'"
              },
              "responseTemplates" : {
                "application/json" : "json 400 response template",
                "application/xml" : "xml 400 response template"
              }
            }
          }
        }
      },
      "get": {
        "parameters": [
          {
            "name": "q1",
            "in": "query",
            "required": true
          }
        ],
        "responses": {
          "200": {
            "schema": {
              "type": "array",
              "items": {
                "$ref": "#/definitions/Error"
              }
            },
            "headers" : {
              "test-method-response-header" : {
                "type" : "string"
              }
            }
          }
        },
        "security" : [{
          "api_key" : []
        }],
        "x-amazon-apigateway-auth" : {
```

```
          "type" : "none"
        },
        "x-amazon-apigateway-integration" : {
          "type" : "http",
          "uri" : "http://petstore-demo-endpoint.execute-api.com/petstore/pets",
          "httpMethod" : "GET",
          "requestParameters": {
            "integration.request.querystring.type": "method.request.querystring.q1"
          },
          "responses" : {
            "2\\d{2}" : {
              "statusCode" : "200"
            },
            "default" : {
              "statusCode" : "400",
              "responseParameters" : {
                "method.response.header.test-method-response-header" : "'static
value'"
              },
              "responseTemplates" : {
                "application/json" : "json 400 response template",
                "application/xml" : "xml 400 response template"
              }
            }
          }
        }
      }
    }
  },
  "definitions": {
    "RequestBodyModel": {
      "type": "object",
      "properties": {
        "id":    { "type": "integer" },
        "type":  { "type": "string", "enum": ["dog", "cat", "fish"] },
        "name":  { "type": "string" },
        "price": { "type": "number", "minimum": 25, "maximum": 500 }
      },
      "required": ["type", "name", "price"]
    },
    "Error": {
      "type": "object",
      "properties": {
```

```
      }
    }
  }
}
```

AWS CloudFormation 具有基本請求驗證的示例 API 模板

下列範 AWS CloudFormation 例範本定義會定義啟用要求驗證的範例 API。該 API 是 API 的一個子 PetStore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}'
```

設定 REST API 的資料轉換

在 API Gateway 中，API 的方法請求可從整合請求承載中取得不同格式的承載。同樣地，後端可能會傳回與方法回應承載不同的整合回應承載。您可以使用對應範本，跨 API Gateway 對應 URL 路徑參數、URL 查詢字串參數、HTTP 標頭和請求內文。

對應範本是以 Velocity 範本語言 (VTL) 表示，並使用 JSONPath 表達式套用至承載的指令碼。

根據 JSON 結構描述草稿第 4 版，承載可以有一個資料模型。若要進一步了解模型，請參閱 了解資料模型。

> **Note**
>
> 您不必定義任何模型來建立對應範本，但必須定義模型，才能讓 API Gateway 產生 SDK，或為您的 API 開啟請求內文驗證。

主題

- 了解對應範本
- 在 API Gateway 中設定資料轉換
- 使用對應範本來覆寫 API 請求和回應參數和狀態碼
- 使用 API Gateway 主控台設定請求與回應資料映射
- API Gateway 的範例資料模型和對應範本
- Amazon API Gateway API 請求和回應資料映射參考

- API Gateway 映射範本和存取記錄變數參考

了解對應範本

在 API Gateway 中，API 的方法請求或回應可從整合請求或回應中取得不同格式的承載。

您可以轉換資料：

- 使承載符合 API 指定的格式。
- 覆寫 API 請求與回應參數和狀態碼。
- 傳回用戶端選取的回應標頭。
- 在 HTTP Proxy 或 Proxy 的方法要求中，建立路徑參數、查詢字串參數或 AWS 服務 標頭參數的關聯。
- 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。

您可以使用對應範本轉換資料。對應範本是以 Velocity 範本語言 (VTL) 表示，並使用 JSONPath 套用至承載的指令碼。

下列範例顯示資料轉換的輸入資料、對應範本以及輸出PetStore 資料。

輸入資料

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

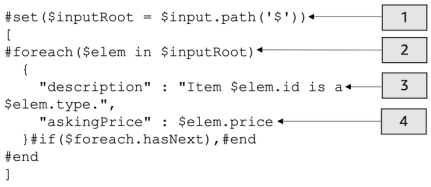
對應範本

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

#end
]
```

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
  }
]
```

下圖顯示此對應範本的詳細資訊。



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. `foreach` 迴圈逐一處理原始 JSON 資料中的每個物件。
3. 描述是來自原始 JSON 資料的寵物 `id` 和 `type` 串連。
4. `askingPrice` 是 `price` (來自原始 JSON 資料的價格)。

PetStore 對映樣板

```
1 #set($inputRoot = $input.path('$'))
2 [
3 #foreach($elem in $inputRoot)
4  {
5    "description" : "Item $elem.id is a $elem.type.",
6    "askingPrice" : $elem.price
7  }#if($foreach.hasNext),#end
8 #end
9 ]
```

在此對應範本中：

1. 在第 1 行，`$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. 在第 3 行，`foreach` 迴圈逐一處理原始 JSON 資料中的每個物件。
3. 在第 5 行，`description` 是來自原始 JSON 資料的寵物 `id` 和 `type` 串連。
4. 在第 6 行，`askingPrice` 是 `price` (來自原始 JSON 資料的價格)。

如需 Velocity 範本語言的詳細資訊，請參閱 Apache Velocity - VTL Reference。如需 JSONPath 的詳細資訊，請參閱 JSONPath - XPath for JSON。

對應範本假設基礎資料屬於 JSON 物件。它不需要定義資料模型。不過，輸出資料的模型允許以語言特定物件的形式傳回先前的資料。如需詳細資訊，請參閱 了解資料模型。

複雜的對應範本

您也可以建立更複雜的對應範本。下列範例說明用來判斷是否養得起寵物的參考串連和取捨點 (100)。

輸入資料

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

```
  {
    "id": 3,
    "type": "fish",
    "price": 0.99
  }
]
```

對應範本

```
#set($inputRoot = $input.path('$'))
#set($cheap = 100)
[
#foreach($elem in $inputRoot)
  {
#set($name = "${elem.type}number$elem.id")
    "name" : $name,
    "description" : "Item $elem.id is a $elem.type.",
    #if($elem.price > $cheap )#set ($afford = 'too much!') #{else}#set
 ($afford = $elem.price)#end
"askingPrice" : $afford
  }#if($foreach.hasNext),#end

#end
]
```

輸出資料

```
[
  {
    "name" : dognumber1,
    "description" : "Item 1 is a dog.",
    "askingPrice" : too much!
  },
  {
    "name" : catnumber2,
    "description" : "Item 2 is a cat.",
    "askingPrice" : too much!
  },
  {
    "name" : fishnumber3,
    "description" : "Item 3 is a fish.",
    "askingPrice" : 0.99
  }
]
```

您還可以看到更複雜的數據模型。請參閱API Gateway 的範例資料模型和對應範本。

在 API Gateway 中設定資料轉換

本節說明如何設定對應範本，以使用主控台和 AWS CLI 轉換整合要求和回應。

主題

- 使用 API Gateway 主控台設定資料轉換
- 使用 AWS CLI 設定資料轉換
- 完成的資料轉換 AWS CloudFormation 範本
- 後續步驟

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 檔案建立不完整的 API 和 DynamoDB 資料表。data-transformation-tutorial-console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 PetStore 對映樣板 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。 AWS 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 AWS CloudFormation 堆疊的步驟

1. 請在以下位置開啟 AWS CloudFormation 主控台。 https://console.aws.amazon.com/cloudformation
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. 對於功能，請確認 AWS CloudFormation 可以在您的帳戶中建立 IAM 資源。

9. 選擇提交。

AWS CloudFormation 規定範本中指定的資源。完成資源佈建可能需要幾分鐘的時間。當 AWS CloudFormation 堆疊的狀態為「建立 _ 完成」時，您就可以繼續進行下一個步驟。

測試 GET 整合回應

1. 在的 AWS 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 PetStore 對映樣板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. 選擇新增對應範本。

   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。

   AWS CloudFormation 範本已填入部分整合要求欄位。

   - 整合類型為 AWS 服務。
   - 就 AWS 服務 是 DynamoDB。
   - HTTP 方法為 POST。
   - 動作為 PutItem。
   - 允許 API Gateway 將項目放入 DynamoDB 表格的執行角色為。data-transformation-tutorial-console-APIGatewayRole AWS CloudFormation 建立此角色以允許 API Gateway 具有與 DynamoDB 互動的最低權限。

   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 整合傳遞行為。
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 在 API Gateway 中使用請求驗證。

7. 選擇儲存以儲存您的對應範本。

從 **POST** 方法新增方法和整合回應

AWS CloudFormation 創建了一個空白的方法和集成響應。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 Amazon API Gateway API 請求和回應資料映射參考。

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 `$context`資料模型、授權者、對應範本和 CloudWatch 存取記錄的變數。

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 AWS CloudFormation 堆疊的步驟

1. 請在以下位置開啟 AWS CloudFormation 主控台。 https://console.aws.amazon.com/cloudformation
2. 選取您的 AWS CloudFormation 堆疊。
3. 選擇刪除，然後確認您的選擇。

使用 AWS CLI 設定資料轉換

在本教學課程中，您將使用下列 .zip 檔案建立不完整的 API 和 DynamoDB 資料表。data-transformation-tutorial-cli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 PetStore 對映樣板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 AWS CloudFormation 堆疊的步驟

下載並解壓縮的應用程式建立範本。 AWS 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. 使用下面的命令來創建 AWS CloudFormation 堆棧。

```
aws cloudformation create-stack --stack-name data-transformation-tutorial-cli
 --template-body file://data-transformation-tutorial-cli.zip --capabilities
 CAPABILITY_NAMED_IAM
```

2. AWS CloudFormation 規定範本中指定的資源。完成資源佈建可能需要幾分鐘的時間。使用以下命令查看 AWS CloudFormation 堆棧的狀態。

```
aws cloudformation describe-stacks --stack-name data-transformation-tutorial-cli
```

3. 當 AWS CloudFormation 堆棧的狀態為時`StackStatus: "CREATE_COMPLETE"`，請使用以下命令檢索 future 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 表格中的角色名稱。對於本教學課程，角色名稱為 `data-transformation-tutorial-cli-APIGatewayRole-ABCDEFG`。
   - DDBTableName，也就是動 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，這是公開GET和POST方法之寵物資源的 ID。對於本教學課程，資源 ID 為 `efg456`
   - ApiId，也就是 API 的識別碼。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 PetStore 對映樣板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。用您的值替換*rest-api-id*和*## ID*。

  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堆疊中建立的 DynamoDB 表格。 AWS CloudFormation

2. 使用以下命令在POST方法上創建 AWS 服務 集成。

```
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 AWS CloudFormation 堆疊的步驟

- 使用以下命令刪除資 AWS CloudFormation 源。

```
aws cloudformation delete-stack  --stack-name data-transformation-tutorial-cli
```

完成的資料轉換 AWS CloudFormation 範本

下列範例是完整的 AWS CloudFormation 範本，它會建立包含和方法/pets資源的 API GET 和 POST DynamoDB 表格。

- 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 PetStore 對映樣板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。

```
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
```

後續步驟

若要探索更複雜的對應範本，請參閱下列範例：

- 查看更複雜的模型和對應範本，其中具有範例相簿 相簿範例。
- 若要取得關於模型的詳細資訊，請參閱 了解資料模型。
- 如需如何對應不同回應代碼輸出的相關資訊，請參閱 使用 API Gateway 主控台設定請求與回應資料映射。
- 如需如何從 API 的方法請求資料設定資料對應的相關資訊，請參閱 API Gateway 映射範本和存取記錄變數參考。

使用對應範本來覆寫 API 請求和回應參數和狀態碼

標準 API Gateway 參數和回應碼對應範本可讓您將參數對應， one-to-one 並將一系列整合回應狀態碼 (由規則運算式比對) 對應至單一回應狀態碼。對應範本覆寫可讓您彈性執行 many-to-one參數對映；套用標準 API Gateway 對映後覆寫參數；根據主體內容或其他參數值有條件地對應參數；以程式設計方式即時建立新參數；以及覆寫整合端點傳回的狀態碼。可以覆寫任何類型的請求參數、回應標頭，或回應狀態碼。

以下是用於對應範本覆寫的範例：

- 若要建立新的標頭 (或覆寫現有標頭) 做為兩個參數的連接
- 若要根據內文內容將回應程式碼覆寫為成功或失敗程式碼
- 若要有條件地根據它的內容或一些其他參數的內容重新對應參數
- 若要反覆查看 json 內文的內容和將金鑰值對重新對應為標頭或查詢字串

若要建立映射範本覆寫，請使用映射範本中下列一個或多個 `$context` 變數：

請求內文對應範本	回應內文對應範本。
`$context.requestOverride.header. header_name`	`$context.responseOverride.header. header_name`

請求內文對應範本	回應內文對應範本。
`$context.requestOverride.path. path_name`	`$context.responseOverride.status`
`$context.requestOverride.querystring. querystring_name`	

> **Note**
>
> 對應範本覆寫無法用於 Proxy 整合端點，其缺乏資料對應。如需整合類型的詳細資訊，請參閱選擇 API Gateway API 整合類型。

> **Important**
>
> 覆寫是最終。覆寫只能套用到每個參數一次。多次嘗試覆寫相同的參數會導致來自 Amazon API Gateway 的 5XX 回應。如果您必須多次在整個範本中覆寫相同的參數，我們建議您建立變數與在範本結尾套用覆寫。請注意，只會在整個範本剖析後才會套用範本。請參閱教學課程：使用 API Gateway 主控台覆寫 API 請求參數和標頭。

以下教學課程示範如何建立和測試在 API Gateway 主控台中的映射範本覆寫。這些教學課程使用PetStore 範例 API 做為起點。這兩個教學課程都假設您已經建立了PetStore 範例 API。

主題

- 教學課程：使用 API Gateway 主控台來覆寫 API 回應狀態程式碼
- 教學課程：使用 API Gateway 主控台覆寫 API 請求參數和標頭
- 範例：使用 API Gateway CLI 覆寫 API 請求參數和標頭
- 範例：使用 SDK 覆寫 API 的要求參數和標頭 JavaScript

教學課程：使用 API Gateway 主控台來覆寫 API 回應狀態程式碼

若要使用 PetStore 範例 API 擷取寵物，請使用的 API 方法要求`GET /pets/{petId}`，其中`{petId}`是可以在執行階段取得數字的 path 參數。

在此教學課程中，您將覆寫此 GET 方法的回應程式碼，方法是在偵測到錯誤條件時，建立將 `$context.responseOverride.status` 對應到 `400` 的對應範本。

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 在 [API] 下，選擇 [PetStore API]，然後選擇 [資源]。
3. 在資源樹狀結構中的 `/{petId}` 下，選擇 GET 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。
5. 針對 petId，輸入 **-1**，然後選擇測試。

   您會在結果中注意到兩件事：

   首先，響應主體指示一個 out-of-range錯誤：

   ```
   {
     "errors": [
       {
         "key": "GetPetRequest.petId",
         "message": "The value is out of range."
       }
     ]
   }
   ```

   其次，在日誌方塊下的最後一列結尾為：`Method completed with status: 200`。
6. 在整合回應索引標籤上，針對預設 - 回應，選擇編輯。
7. 選擇對應範本。
8. 選擇新增對應範本。
9. 針對內容類型，輸入 **application/json**。
10. 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
    ```
11. 選擇儲存。
12. 選擇測試標籤。

13. 針對 petId，輸入 **-1**。
14. 在結果中，響應主體指示錯 out-of-range 誤：

```
{
  "errors": [
    {
      "key": "GetPetRequest.petId",
      "message": "The value is out of range."
    }
  ]
}
```

然而，在 Logs (日誌) 方塊下的最後一列目前結尾為：`Method completed with status: 400`。

教學課程：使用 API Gateway 主控台覆寫 API 請求參數和標頭

在此教學課程中，您將覆寫 GET 方法的請求標頭程式碼，方法是建立會將 `$context.requestOverride.header.`*`header_name`* 對應到結合兩個其他標頭之新標頭的對應範本。

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 在 [API] 下方，選擇 PetStore API。
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. 針對名稱，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。
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. 選擇新增對應範本。

    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($header1Override = "foo")
#set($header3Value = "$input.params('header1')$input.params('header2')")
$input.json("$")
#set($context.requestOverride.header.header3 = $header3Value)
#set($context.requestOverride.header.header1 = $header1Override)
#set($context.requestOverride.header.multivalueheader=[$header1Override,
 $header3Value])
```

12. 選擇儲存。
13. 選擇測試標籤。
14. 在 {pets} ({pets}) 的 Headers (標頭) 下，複製下列程式碼：

```
header1:header1Val
header2:header2Val
```

15. 選擇 Test (測試)。

    在日誌中，您應該會看到一個包括此文字的項目：

```
Endpoint request headers: {header3=header1Valheader2Val,
header2=header2Val, header1=foo, x-amzn-apigateway-api-id=<api-id>,
Accept=application/json, multivalueheader=foo,header1Valheader2Val}
```

範例：使用 API Gateway CLI 覆寫 API 請求參數和標頭

以下 CLI 範例示範如何使用 `put-integration` 命令來覆寫回應程式碼：

```
aws apigateway put-integration --rest-api-id <API_ID> --resource-
id <PATH_TO_RESOURCE_ID> --http-method <METHOD>
            --type <INTEGRATION_TYPE> --request-templates <REQUEST_TEMPLATE_MAP>
```

其中 *<REQUEST_TEMPLATE_MAP>* 是從內容類型到要套用之範本字串的對應。對應的結構如下所示：

```
Content_type1=template_string,Content_type2=template_string
```

或者，在 JSON 語法中：

```
{"content_type1": "template_string"
...}
```

以下範例示範如何使用 put-integration-response 命令來覆寫 API 回應程式碼：

```
aws apigateway put-integration-response --rest-api-id <API_ID> --resource-
id <PATH_TO_RESOURCE_ID> --http-method <METHOD>
            --status-code <STATUS_CODE> --response-templates <RESPONSE_TEMPLATE_MAP>
```

其中 *<RESPONSE_TEMPLATE_MAP>* 具有與上述 *<REQUEST_TEMPLATE_MAP>* 相同的格式。

範例：使用 SDK 覆寫 API 的要求參數和標頭 JavaScript

以下範例示範如何使用 put-integration 命令來覆寫回應程式碼：

要求:

```
var params = {
  httpMethod: 'STRING_VALUE', /* required */
  resourceId: 'STRING_VALUE', /* required */
  restApiId: 'STRING_VALUE', /* required */
  type: HTTP | AWS | MOCK | HTTP_PROXY | AWS_PROXY, /* required */
  requestTemplates: {
    '<Content_type>': 'TEMPLATE_STRING',
    /* '<String>': ... */
  },
};
apigateway.putIntegration(params, function(err, data) {
  if (err) console.log(err, err.stack); // an error occurred
  else     console.log(data);           // successful response
});
```

回應：

```
var params = {
  httpMethod: 'STRING_VALUE', /* required */
  resourceId: 'STRING_VALUE', /* required */
  restApiId: 'STRING_VALUE', /* required */
  statusCode: 'STRING_VALUE', /* required */
  responseTemplates: {
    '<Content_type>': 'TEMPLATE_STRING',
    /* '<String>': ... */
  },
};
apigateway.putIntegrationResponse(params, function(err, data) {
```

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

使用 API Gateway 主控台設定請求與回應資料映射

若要使用 API Gateway 主控台來定義 API 的整合請求/回應，請遵循這些說明進行。

> **Note**
>
> 這些說明假設您已完成使用 API Gateway 主控台設定 API 整合請求中的步驟。

1. 在 [資源] 窗格中，選擇您的方法。
2. 在整合請求索引標籤上，於整合請求設定下，選擇編輯。
3. 選擇要求主體傳遞的選項，以設定未對應內容類型的方法要求主體如何透過整合要求傳遞，而不需轉換至 Lambda 函數、HTTP Proxy 或 AWS 服務 Proxy。共有三個選項：

   - 當方法請求內容類型不符合與下一個步驟中定義之映射範本相關聯的任何內容類型時，如果您想要讓方法請求內文透過整合請求傳遞到後端而不進行轉換，請選擇沒有範本符合請求內容類型標頭時。

     > **Note**
     >
     > 呼叫 API Gateway API 時，您可以透過在整合資源上將 `WHEN_NO_MATCH` 設定為 `passthroughBehavior` 屬性值，以選擇此選項。

   - 當整合請求中未定義任何映射範本時，如果您想要讓方法請求內文透過整合請求傳遞到後端而不進行轉換，請選擇 When there are no templates defined (recommended) (未定義範本時 (建議))。如果選取此選項時已定義範本，未映射內容類型的方法請求會遭到拒絕，並顯示 HTTP 415 Unsupported Media Type (不支援的媒體類型) 回應。

     > **Note**
     >
     > 呼叫 API Gateway API 時，您可以透過在整合資源上將 `WHEN_NO_TEMPLATE` 設定為 `passthroughBehavior` 屬性值，以選擇此選項。

   - 當方法請求內容類型不符合與整合請求中定義之映射範本相關聯的任何內容類型時，或是當整合請求中未定義任何映射範本時，如果您不想要讓方法請求傳遞，請選擇 Never (永不)。未映射

內容類型的方法請求會遭到拒絕，並顯示 HTTP 415 Unsupported Media Type (不支援的媒體類型) 回應。

> **Note**
>
> 呼叫 API Gateway API 時，您可以透過在整合資源上將 NEVER 設定為 `passthroughBehavior` 屬性值，以選擇此選項。

如需整合傳遞行為的詳細資訊，請參閱「整合傳遞行為」。

4. 對於 HTTP Proxy 或 AWS 服務 Proxy，若要將路徑參數、查詢字串參數或標頭參數與 HTTP Proxy 或服 AWS 務 Proxy 方法要求中的對應路徑參數、查詢字串參數或標頭參數產生關聯，請執行下列動作：

   a. 分別選擇 URL 路徑參數、URL 查詢字串參數或 HTTP 請求標頭，然後分別選擇 新增路徑、新增查詢字串或新增標頭。

   b. 在「名稱」中，輸入 HTTP 代理主機或服務代理伺 AWS 服器中的 path 參數、查詢字串參數或標頭參數的名稱。

   c. 針對映射自，輸入路徑參數、查詢字串參數或標頭參數的映射值。請使用下列其中一個格式：

      - **`method.request.path.`*`parameter-name`***，代表方法請求頁面中所定義之名為 *`parameter-name`* 的路徑參數。
      - **`method.request.querystring.`*`parameter-name`***，代表方法請求頁面中所定義之名為 *`parameter-name`* 的查詢字串參數。
      - **`method.request.multivaluequerystring.`*`parameter-name`***，代表方法請求頁面中所定義之名為 *`parameter-name`* 的多值查詢字串參數。
      - **`method.request.header.`*`parameter-name`***，代表方法請求頁面中所定義之名為 *`parameter-name`* 的標頭參數。

        或者，您可以將字串常值 (以一對單引號括住) 設定為整合標頭。
      - **`method.request.multivalueheader.`*`parameter-name`***，代表方法請求頁面中所定義之名為 *`parameter-name`* 的多值標頭參數。

   d. 若要新增其他參數，請選擇新增按鈕。

5. 若要新增對應範本，請選擇對應範本。

6. 若要定義傳入請求的對應範本，請選擇新增對應範本。針對內容類型，輸入內容類型 (例如 **`application/json`**)。然後，輸入對映樣板。如需詳細資訊，請參閱 了解對應範本。

7. 選擇 Save (儲存)。
8. 您可以將後端整合回應映射到傳回給呼叫應用程式的 API 方法回應。這包括將從後端可用回應標頭中選取的回應標頭傳回給用戶端，並將後端回應承載的資料格式轉換成 API 指定的格式。您可以設定方法回應與整合回應，來指定這類對應。

   若要讓方法根據 Lambda 函數、HTTP 代理或 AWS 服務代理所傳回的 HTTP 狀態碼接收自訂回應資料格式，請執行下列動作：

   a. 選擇整合回應。選擇預設 - 回應上的編輯，以指定從方法設定 200 HTTP 回應碼，或選擇建立回應，以指定從方法設定任何其他 HTTP 回應狀態碼。

   b. 對於 Lambda 錯誤正則表達式（用於 Lambda 函數）或 HTTP 狀態正則表達式（適用於 HTTP 代理或 AWS 服務代理），請輸入一般表示式以指定哪些 Lambda 函數錯誤字串（針對 Lambda 函數）或 HTTP 回應狀態碼（適用於 HTTP 代理或 AWS 服務代理）對應至此輸出對應。例如，若要將所有 2xx HTTP 回應狀態碼從 HTTP 代理對應到此輸出對應，請針對 HTTP status regex (HTTP 狀態 regex) 輸入「**`2\d{2}`**」。若要從 Lambda 函數傳回包含「無效請求」的錯誤訊息給 `400 Bad Request` 回應，請輸入「**`.*Invalid request.*`**」作為 Lambda 錯誤 regex 表達式。另一方面，若要對來自 Lambda 的所有未映射錯誤訊息傳回 `400 Bad Request`，請在 Lambda 錯誤 regex 中輸入「**`(\n|.)+`**」。最後一個規則表達式可作為 API 的預設錯誤回應使用。

      > **Note**
      >
      > API Gateway 使用 Java 模式 regex 進行回應映射。如需詳細資訊，請參閱 Oracle 文件中的模式一節。
      > 錯誤模式會與 Lambda 回應中 `errorMessage` 屬性的整個字串進行比對，該屬性在 Node.js 中是由 `callback(errorMessage)` 填入，在 Java 中是由 `throw new MyException(errorMessage)` 填入。此外，逸出字元在套用規則表達式之前為未逸出。
      > 如果您使用「.+」做為選取模式來篩選回應，請注意它可能不會比對含有新行（「\n」）字元的回應。

   c. 如果啟用，請針對方法回應狀態，選取您在方法回應頁面上定義的 HTTP 回應狀態碼。

   d. 針對您在方法回應頁面中為 HTTP 回應狀態碼定義之每個標頭的標頭對應，指定對應值。對於 Mapping value (對應值)，請使用以下其中一個格式：

      - **`integration.response.multivalueheaders.`*`header-name`***，其中 *`header-name`* 是後端多值回應標頭的名稱。

例如，若要傳回後端回應的 `Date` 標頭做為 API 方法回應的 `Timestamp` 標頭，Response header (回應標頭) 欄會包含 Timestamp (時間戳記) 項目，且相關聯的 Mapping value (對應值) 應設為 integration.response.multivalueheaders.Date。

- **`integration.response.header.`*`header-name`***，其中 *`header-name`* 是後端單一值回應標頭的名稱。

  例如，若要傳回後端回應的 `Date` 標頭做為 API 方法回應的 `Timestamp` 標頭，Response header (回應標頭) 欄會包含 Timestamp (時間戳記) 項目，且相關聯的 Mapping value (對應值) 應設為 integration.response.header.Date。

e. 選擇對應範本，然後選擇新增對應範本。在 [內容類型] 方塊中，輸入將從 Lambda 函數、HTTP 代理或 AWS 服務代理傳送至方法之資料的內容類型。然後，輸入對映樣板。如需詳細資訊，請參閱 了解對應範本。

f. 選擇 Save (儲存)。

API Gateway 的範例資料模型和對應範本

下列章節提供的模型與映射範本範例，可在 API Gateway 中作為 API 的起點。如需資料轉型的詳細資訊，請參閱 了解對應範本。如需資料模型的詳細資訊，請參閱the section called “了解資料模型”。

主題

- 相簿範例
- 新聞文章範例

相簿範例

下列範例顯示 API Gateway 中的相簿 API。我們提供了範例資料轉換、額外模型和對應範本。

主題

- 範例資料轉換
- 相片資料的輸入模型
- 相片資料的輸出模型
- 照片資料的輸入對應範本

範例資料轉換

下列範例說明如何使用 Velocity 範本語言 (VTL) 對應範本，轉換有關相片的輸入資料。如需 Velocity 範本語言的詳細資訊，請參閱 Apache Velocity - VTL Reference。

輸入資料

```
{
  "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
      }
    ]
  }
}
```

輸出對應範本

```
#set($inputRoot = $input.path('$'))
{
  "photos": [
#foreach($elem in $inputRoot.photos.photo)
    {
      "id": "$elem.id",
      "photographedBy": "$elem.photographer_first_name $elem.pho
tographer_last_name",
      "title": "$elem.title",
      "ispublic": $elem.ispublic,
      "isfriend": $elem.isfriend,
      "isfamily": $elem.isfamily
    }#if($foreach.hasNext),#end

#end
  ]
}
```

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
    }
  ]
}
```

相片資料的輸入模型

您可以定義輸入資料的模型。此輸入模型要求您上傳一張照片，並為每頁指定至少 10 張照片。您可以使用此輸入模型，產生 SDK 或針對您的 API 開啟請求驗證。

```
{
  "$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" }
            }
          }
        }
      }
    }
  }
}
```

相片資料的輸出模型

您可以定義輸出資料的模型。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" }
        }
      }
    }
  }
}
```

照片資料的輸入對應範本

您可以定義對應範本來修改輸入資料。您可以修改輸入資料，取得進一步的函數整合或整合回應。

```
#set($inputRoot = $input.path('$'))
{
  "photos": {
    "page": $inputRoot.photos.page,
    "pages": "$inputRoot.photos.pages",
    "perpage": $inputRoot.photos.perpage,
    "total": "$inputRoot.photos.total",
    "photo": [
#foreach($elem in $inputRoot.photos.photo)
      {
        "id": "$elem.id",
        "owner": "$elem.owner",
```

```
        "photographer_first_name" : "$elem.photographer_first_name",
        "photographer_last_name" : "$elem.photographer_last_name",
        "secret": "$elem.secret",
        "server": "$elem.server",
        "farm": $elem.farm,
        "title": "$elem.title",
        "ispublic": $elem.ispublic,
        "isfriend": $elem.isfriend,
        "isfamily": $elem.isfamily
      }#if($foreach.hasNext),#end

#end
    ]
  }
}
```

新聞文章範例

下列範例顯示 API Gateway 中的新聞文章 API。我們提供了範例資料轉換、額外模型和對應範本。

主題

- 範例資料轉換
- 新聞資料輸入模型
- 新聞數據的輸出模型
- 新聞數據的輸入映射模板

範例資料轉換

下列範例顯示如何使用 Velocity 範本語言 (VTL) 對應範本，轉換有關新聞文章的輸入資料。如需 Velocity 範本語言的詳細資訊，請參閱 Apache Velocity - VTL Reference。

輸入資料

```
{
  "count": 1,
  "items": [
    {
      "last_updated_date": "2015-04-24",
      "expire_date": "2016-04-25",
      "author_first_name": "John",
      "description": "Sample Description",
      "creation_date": "2015-04-20",
```

```
      "title": "Sample Title",
      "allow_comment": true,
      "author": {
        "last_name": "Doe",
        "email": "johndoe@example.com",
        "first_name": "John"
      },
      "body": "Sample Body",
      "publish_date": "2015-04-25",
      "version": "1",
      "author_last_name": "Doe",
      "parent_id": 2345678901,
      "article_url": "http://www.example.com/articles/3456789012"
    }
  ],
  "version": 1
}
```

輸出對應範本

```
#set($inputRoot = $input.path('$'))
{
  "count": $inputRoot.count,
  "items": [
#foreach($elem in $inputRoot.items)
    {
      "creation_date": "$elem.creation_date",
      "title": "$elem.title",
      "author": "$elem.author.first_name $elem.author.last_name",
      "body": "$elem.body",
      "publish_date": "$elem.publish_date",
      "article_url": "$elem.article_url"
    }#if($foreach.hasNext),#end

#end
  ],
  "version": $inputRoot.version
}
```

輸出資料

```
{
  "count": 1,
  "items": [
    {
      "creation_date": "2015-04-20",
      "title": "Sample Title",
      "author": "John Doe",
      "body": "Sample Body",
      "publish_date": "2015-04-25",
      "article_url": "http://www.example.com/articles/3456789012"
    }
  ],
  "version": 1
}
```

新聞資料輸入模型

您可以定義輸入資料的模型。此輸入模型要求新聞文章包含 URL、標題和正文。您可以使用此輸入模型，產生 SDK 或針對您的 API 開啟請求驗證。

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "title": "NewsArticleInputModel",
  "type": "object",
  "properties": {
    "count": { "type": "integer" },
    "items": {
      "type": "array",
      "items": {
        "type": "object",
        "required": [
      "article_url",
      "title",
      "body"
      ],
        "properties": {
          "last_updated_date": { "type": "string" },
          "expire_date": { "type": "string" },
          "author_first_name": { "type": "string" },
          "description": { "type": "string" },
          "creation_date": { "type": "string" },
```

```
          "title": { "type": "string" },
          "allow_comment": { "type": "boolean" },
          "author": {
            "type": "object",
            "properties": {
              "last_name": { "type": "string" },
              "email": { "type": "string" },
              "first_name": { "type": "string" }
            }
          },
          "body": { "type": "string" },
          "publish_date": { "type": "string" },
          "version": { "type": "string" },
          "author_last_name": { "type": "string" },
          "parent_id": { "type": "integer" },
          "article_url": { "type": "string" }
        }
      }
    },
    "version": { "type": "integer" }
  }
}
```

新聞數據的輸出模型

您可以定義輸出資料的模型。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" }
        }
      }
    }
  }
}
```

新聞數據的輸入映射模板

您可以定義對應範本來修改輸入資料。您可以修改輸入資料，取得進一步的函數整合或整合回應。

```
#set($inputRoot = $input.path('$'))
{
  "count": $inputRoot.count,
  "items": [
#foreach($elem in $inputRoot.items)
    {
      "last_updated_date": "$elem.last_updated_date",
      "expire_date": "$elem.expire_date",
      "author_first_name": "$elem.author_first_name",
      "description": "$elem.description",
      "creation_date": "$elem.creation_date",
      "title": "$elem.title",
      "allow_comment": "$elem.allow_comment",
      "author": {
        "last_name": "$elem.author.last_name",
        "email": "$elem.author.email",
        "first_name": "$elem.author.first_name"
      },
      "body": "$elem.body",
      "publish_date": "$elem.publish_date",
      "version": "$elem.version",
      "author_last_name": "$elem.author_last_name",
      "parent_id": $elem.parent_id,
      "article_url": "$elem.article_url"
    }#if($foreach.hasNext),#end

#end
  ],
  "version": $inputRoot.version
}
```

Amazon API Gateway API 請求和回應資料映射參考

本節會說明如何設定將資料 (包含 `context`、`stage` 或 `util` 變數中存放的其他資料) 從 API 方法請求資料映射到對應的整合請求參數；以及如何設定將資料 (包含其他資料) 從整合回應資料映射到方法回應參數。方法請求資料包含請求參數 (路徑、查詢字串、標頭) 和內文。整合回應資料包含回應參數 (標頭) 和內文。如需使用階段變數的詳細資訊，請參閱「Amazon API Gateway 階段變數參考」。

主題

- 將方法請求資料對應到整合請求參數
- 將整合回應資料對應到方法回應標頭
- 對應方法和整合之間的請求和回應承載
- 整合傳遞行為

將方法請求資料對應到整合請求參數

格式為路徑變數、查詢字串或標頭的整合請求參數，可以從任何已定義的方法請求參數和承載對應。

在下表中，*PARAM_NAME* 是指定參數類型的方法請求參數名稱。它必須符合規則表達式 `'^[a-zA-Z0-9._$-]+$]'`。它必須已先定義，您才能參考它。*JSONPath_EXPRESSION* 是請求或回應內文之 JSON 欄位的 JSONPath 表達式。

> **Note**
>
> "$" 字首在這個語法中予以省略。

整合請求資料對應表達式

對應的資料來源	對應表達式
方法請求路徑	`method.request.path.` *PARAM_NAME*
方法請求查詢字串	`method.request.querystring.` *PARAM_NAME*
多值方法請求查詢字串	`method.request.multivaluequerystring.` *PARAM_NAME*

對應的資料來源	對應表達式
方法請求標頭	`method.request.header.` *PARAM_NAME*
多值方法請求標頭	`method.request.multivalueheader.` *PARAM_NAME*
方法請求內文	`method.request.body`
方法請求主體（JsonPath）	`method.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。

Example 從 OpenAPI 的方法請求參數對應

下列範例顯示的 OpenAPI 程式碼片段會：

- 將名為 `methodRequestHeaderParam` 的方法請求標頭對應到名為 `integrationPathParam` 的整合請求路徑參數
- 將名為 `methodRequestQueryParam` 的多值方法請求查詢字串對應到名為 `integrationQueryParam` 的整合請求查詢字串

```
...
"requestParameters" : {

    "integration.request.path.integrationPathParam" :
 "method.request.header.methodRequestHeaderParam",
    "integration.request.querystring.integrationQueryParam" :
 "method.request.multivaluequerystring.methodRequestQueryParam"

}
```

```
...
```

您也可以使用 JSONPath 表達式從 JSON 請求內文中的欄位，對應整合請求參數。下表顯示方法請求內文的對應表達式及其 JSON 欄位。

Example 從 OpenAPI 的方法請求內文對應

下列範例顯示的 OpenAPI 程式碼片段將 1) 方法請求內文對應到名為 body-header 的整合請求標頭；以及對應 2) 內文的 JSON 欄位，如 JSON 表達式所表示 (petstore.pets[0].name，無 $. 字首)。

```
...
"requestParameters" : {

    "integration.request.header.body-header" : "method.request.body",
    "integration.request.path.pet-name" : "method.request.body.petstore.pets[0].name",

}
...
```

將整合回應資料對應到方法回應標頭

您可以從任何整合回應標頭或整合回應內文、$context 變數或靜態值，映射方法回應標頭參數。

方法回應標頭對應表達式

對應的資料來源	對應表達式
整合回應標頭	integration.response.header. *PARAM_NAME*
整合回應標頭	integration.response.multivalueheader. *PARAM_NAME*
整合回應內文	integration.response.body

對應的資料來源	對應表達式
整合回應主體 (JsonPath)	`integration.response.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。

Example 從 OpenAPI 整合回應對應的資料

下列範例顯示的 OpenAPI 程式碼片段將 1) 整合回應的 `redirect.url`，即 JSONPath 欄位對應到請求回應的 `location` 標頭；將 2) 整合回應的 `x-app-id` 標頭對應到方法回應的 `id` 標頭。

```
...
"responseParameters" : {

    "method.response.header.location" : "integration.response.body.redirect.url",
    "method.response.header.id" : "integration.response.header.x-app-id",
    "method.response.header.items" : "integration.response.multivalueheader.item",

}
...
```

對應方法和整合之間的請求和回應承載

API Gateway 使用 Velocity 範本語言 (VTL) 引擎來處理整合請求和整合回應的內文映射範本。對應範本會將方法請求承載轉譯為對應的整合請求承載，並將整合回應內文轉譯成方法回應內文。

VTL 範本使用 JSONPath 表達式、其他參數 (例如呼叫的環境與階段變數) 和公用程式函數來處理 JSON 資料。

如果模型定義成說明承載的資料結構，則 API Gateway 可以使用模型來產生整合請求或整合回應的骨架映射範本。您可以使用骨架範本協助自訂和擴展對應的 VTL 指令碼。不過，您可以從頭開始建立對應範本，且無須定義承載資料結構的模型。

選取 VTL 對應範本

API Gateway 使用下列邏輯來選取使用 Velocity 範本語言 (VTL) 的映射範本，將承載從方法請求映射到對應的整合請求，或將承載從整合回應映射到對應的方法回應。

針對請求承載，API Gateway 會將請求的 `Content-Type` 標頭值用作選取請求承載映射範本的金鑰。針對回應承載，API Gateway 會將傳入請求的 `Accept` 標頭值用作選取映射範本的金鑰。

當請求中沒有 `Content-Type` 標頭時，API Gateway 會假設其預設值為 `application/json`。對於這種請求，API Gateway 會將 `application/json` 用作預設金鑰來選取映射範本 (如果有已定義的映射範本)。當沒有符合此金鑰的範本時，如果 passthroughBehavior 屬性設定為 `WHEN_NO_MATCH` 或 `WHEN_NO_TEMPLATES`，則 API Gateway 會傳送未映射的承載。

當請求中未指定 `Accept` 標頭時，API Gateway 會假設其預設值為 `application/json`。在此情況下，API Gateway 會選取現有的 `application/json` 映射範本來映射回應承載。若未針對 `application/json` 定義任何範本，則 API Gateway 會選取第一個現有的範本並使用它作為預設值來映射回應承載。同樣地，當指定的 `Accept` 標頭值不符合任何現有的範本金鑰時，API Gateway 會使用第一個現有的範本。若未定義任何範本，API Gateway 只會傳遞未映射的回應承載。

例如，假設 API 針對請求承載定義了 `application/json` 範本，且針對回應承載定義了 `application/xml` 範本。如果用戶端在請求中設定 `"Content-Type : application/json"` 和 `"Accept : application/xml"` 標頭，則請求和回應承載都會使用對應的對應範本來處理。如果沒有 `Accept:application/xml` 標頭，則會使用 `application/xml` 對應範本來對應回應承載。若要改傳回未對應的回應承載，您必須為 `application/json` 設定空的範本。

選取對應範本時，只有 MIME 類型是從 `Accept` 和 `Content-Type` 標頭使用。例如，`"Content-Type: application/json; charset=UTF-8"` 的標頭會有已選取 `application/json` 金鑰的請求範本。

整合傳遞行為

使用非代理整合，當方法請求傳送承載，但 `Content-Type` 標頭不符合任何指定的對應範本，或未定義任何對應範本時，您可以選擇將用戶端提供的請求承載透過整合請求傳送到後端，且不須轉換。這個過程稱為整合傳遞。

關於代理整合，API Gateway 會將整個請求傳遞到您的後端，而您並無法修改傳遞行為。

傳入請求的實際傳遞行為，是由您在整合請求設定期間針對指定對應範本所選擇的選項，以及用戶端在傳入請求中設定的內容類型標頭而決定。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，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 `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 標頭\選取的傳遞選項	**WHEN_NO_MATCH**	**WHEN_NO_TEMPLATES**	**NEVER**
無 (預設為 `application/json`)	使用此範本轉換請求承載。	使用此範本轉換請求承載。	使用此範本轉換請求承載。

Content-Type 標頭\選取的傳遞選項	WHEN_NO_MATCH	WHEN_NO_TEMPLATES	NEVER
application/json	使用此範本轉換請求承載。	使用此範本轉換請求承載。	使用此範本轉換請求承載。
application/xml	請求承載未轉換，而且會依現狀傳送到後端。	請求遭到拒絕，回應為 HTTP 415 Unsupported Media Type。	請求遭到拒絕，回應為 HTTP 415 Unsupported Media Type。

範例 2：針對 application/xml 內容類型在整合請求中定義的一個對應範本。

Content-Type 標頭\選取的傳遞選項	WHEN_NO_MATCH	WHEN_NO_TEMPLATES	NEVER
無 (預設為 application/json)	請求承載未轉換，而且會依現狀傳送到後端。	請求遭到拒絕，回應為 HTTP 415 Unsupported Media Type。	請求遭到拒絕，回應為 HTTP 415 Unsupported Media Type。
application/json	請求承載未轉換，而且會依現狀傳送到後端。	請求遭到拒絕，回應為 HTTP 415 Unsupported Media Type。	請求遭到拒絕，回應為 HTTP 415 Unsupported Media Type。
application/xml	使用此範本轉換請求承載。	使用此範本轉換請求承載。	使用此範本轉換請求承載。

API Gateway 映射範本和存取記錄變數參考

本節提供 Amazon API Gateway 為與資料模型、授權者、對應範本和 CloudWatch 存取記錄搭配使用而定義的變數和函數的參考資訊。如需如何使用這些變數和函數的詳細資訊，請參閱了解對應範本。如需 Velocity 範本語言 (VTL) 的詳細資訊，請參閱 VTL 參考。

主題

- $context資料模型、授權者、對應範本和 CloudWatch 存取記錄的變數
- $context 變數範本範例
- 僅適用於存取記錄的 $context 變數
- $input 變數
- $input 變數範本範例
- $stageVariables
- $util 變數

> **Note**
>
> 若為 $method 和 $integration 變數，請參閱 the section called “請求和回應資料對應參考”。

$context資料模型、授權者、對應範本和 CloudWatch 存取記錄的變數

下列$context變數可用於資料模型、授權者、對應範本和 CloudWatch 存取記錄。API Gateway 可能會新增其他上下文變數。

如需只能在 CloudWatch 存取記錄中使用的$context變數，請參閱the section called “僅適用於存取記錄的 $context 變數”。

參數	描述
$context.accountId	API 擁有者的 AWS 帳戶識別碼。
$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”。

參數	描述
	Note 呼叫 `$context.authorizer.claims` 會傳回 null。
`$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”。
`$context.authorizer.`*property*	API Gateway Lambda 授權方函數所傳回 `context` 映射之指定索引鍵/值組的字串化值。例如，如果授權方傳回下列 `context` 映射： `"context" : { "key": "value", "numKey": 1, "boolKey": true }` 呼叫 `$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。 如需詳細資訊，請參閱 the section called “使用 Lambda 授權方”。
`$context.awsEndpointRequestId`	AWS 端點的要求識別碼。
`$context.deploymentId`	API 部署的識別碼。
`$context.domainName`	用來叫用 API 的完整網域名稱。這應該與傳入的 `Host` 標頭相同。
`$context.domainPrefix`	`$context.domainName` 的第一個標籤。

參數	描述
$context.error.message	包含 API Gateway 錯誤訊息的字串。此變數只能用於內文GatewayResponse對應範本中的簡單變數替換，Velocity 範本語言引擎不會處理，也不會在存取記錄中進行處理。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	一種類型GatewayResponse。此變數只能用於內文GatewayResponse對應範本中的簡單變數替換，Velocity 範本語言引擎不會處理，也不會在存取記錄中進行處理。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	描述
`$context.identity.apiKeyId`	與需要 API 金鑰之 API 請求相關聯的 API 金鑰 ID。
`$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。
`$context.identity.cognitoAuthenticationProvider`	提出請求的發起人所使用的 Amazon Cognito 身分驗證提供者清單 (以逗號分隔)。僅在使用 Amazon Cognito 登入資料簽署請求時才可使用。 例如，適用於 Amazon Cognito 使用者集區的身分，`cognito-idp.`*`region`*`.amazonaws.com/`*`user_pool_id`*`,cognito-idp.`*`region`*`.amazonaws.com/`*`user_pool_id`*`:CognitoSignIn:`*`token subject claim`* 如需詳細資訊，請參閱 Amazon Cognito 開發人員指南中的使用聯合身分。
`$context.identity.cognitoAuthenticationType`	提出請求的發起人的 Amazon Cognito 身分驗證類型。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	描述
$context.identity.sourceIp	向 API Gateway 端點發出要求之立即 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	向 API Gateway 端點發出要求的 VPC 人雲端識別碼。

參數	描述
$context.identity.vpceId	向 API Gateway 端點發出要求之 VPC 端點的 VPC 端點識別碼。只有當您擁有私有 API 時才會出現。
$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	true如果請求被定向到初期測試，並且false請求未定向到初期測試，則返回。僅當您啟用了初期測試時才會出現。
$context.path	請求路徑。例如，對於 https://{rest-api-id}.execute-api.{region}.amazonaws.com/{stage}/root/child 的非代理請求 URL，$context.path 值是 /{stage}/root/child 。
$context.protocol	請求通訊協定，例如 HTTP/1.1。 **Note** 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。

參數	描述
`$context.requestId`	請求的 ID。用戶端可以覆寫此請求 ID。使用 `$context.extendedRequestId` 即可取得 API Gateway 產生的唯一請求 ID。
`$context.requestOverride.header.` *`header_name`*	請求標題會覆寫。如果此參數已經定義，它包含要使用的標頭 (而不是在 Integration Request (整合請求) 窗格中定義的 HTTP Headers (HTTP 標頭))。如需詳細資訊，請參閱 使用對應範本來覆寫 API 請求和回應參數和狀態碼。
`$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 使用對應範本來覆寫 API 請求和回應參數和狀態碼。
`$context.requestOverride.querystring.` *`querystring_name`*	請求查詢字串會覆寫。如果此參數已經定義，它包含要使用的請求查詢字串 (而不是在 Integration Request (整合請求) 窗格中定義的 URL Query String (URL 查詢字串))。如需詳細資訊，請參閱 使用對應範本來覆寫 API 請求和回應參數和狀態碼。
`$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 使用對應範本來覆寫 API 請求和回應參數和狀態碼。

參數	描述
`$context.responseOverride.status`	回應狀態碼會覆寫。如果此參數已經定義，它包含要傳回的狀態碼 (而不是在 Integration Response (整合回應) 窗格中定義為 Default mapping (預設映射) 的 Method response status (方法回應狀態))。如需詳細資訊，請參閱 使用對應範本來覆寫 API 請求和回應參數和狀態碼。
`$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。
`$context.resourcePath`	您資源的路徑。例如，對於非代理請求 URI `https://{rest-api-id}.execute-api.{region}.amazonaws.com/{stage}/root/child` ，`$context.resourcePath` 值是 `/root/child` 。如需詳細資訊，請參閱 教學課程：建置具有非 HTTP 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”。

`$context` 變數範本範例

如果您的 API 方法將結構化資料傳遞到需要資料採用特定格式的後端，建議您使用 `$context` 變數。

以下範例顯示一個映射範本，其會將傳入的 `$context` 變數映射至整合請求承載中名稱稍有不同的後端變數：

> **Note**
>
> 其中一個變數是 API 金鑰。此範例假設該方法需要 API 金鑰。

```
{
    "stage" : "$context.stage",
    "request_id" : "$context.requestId",
    "api_id" : "$context.apiId",
    "resource_path" : "$context.resourcePath",
    "resource_id" : "$context.resourceId",
    "http_method" : "$context.httpMethod",
    "source_ip" : "$context.identity.sourceIp",
    "user-agent" : "$context.identity.userAgent",
    "account_id" : "$context.identity.accountId",
    "api_key" : "$context.identity.apiKey",
    "caller" : "$context.identity.caller",
    "user" : "$context.identity.user",
    "user_arn" : "$context.identity.userArn"
}
```

此對應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'
}
```

僅適用於存取記錄的 **$context** 變數

以下 $context 變數僅適用於存取記錄。如需詳細資訊，請參閱 the section called “CloudWatch 日誌”。(如需 WebSocket API，請參閱the section called “指標”。)

參數	描述
$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	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 端點的要求識別碼。
$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.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

參數	描述
	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.endpointType	API 的端點類型。
$context.integration.error	從整合傳回的錯誤訊息。
$context.integration.integrationStatus	對於 Lambda 代理整合，狀態碼會從後端 Lambda 函數程式碼傳回 AWS Lambda，而不是從後端 Lambda 函數
$context.integration.latency	整合延遲 (以毫秒為單位)。等同於 $context.integrationLatency 。
$context.integration.requestId	AWS 端點的要求識別碼。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.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.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.status	方法回應狀態。
$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。

參數	描述
`$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”。

`$input` 變數

`$input` 變數代表映射範本要處理的方法請求承載和參數。它提供了以下功能：

變數和函數	描述
`$input.body`	傳回原始請求承載做為字串。
`$input.json(x)`	此函數會評估 JSONPath 表達式，並傳回結果作為 JSON 字串。 例如，`$input.json('$.pets')` 會傳回代表 `pets` 結構的 JSON 字串。 如需 JSONPath 的詳細資訊，請參閱 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。
`$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。

變數和函數	描述
	例如，如果表達式 `$input.path('$.pets')` 傳回如下物件： `[ { "id": 1, "type": "dog", "price": 249.99 }, { "id": 2, "type": "cat", "price": 124.99 }, { "id": 3, "type": "fish", "price": 0.99 } ]` `$input.path('$.pets').count()` 會傳回 "3"。 如需 JSONPath 的詳細資訊，請參閱 JSONPath 或適用於 Java 的 JSONPath。

$input 變數範本範例

下列範例顯示如何在對應範本中使用$input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 API Gateway 來嘗試這些範例。

參數對應範本範例

下列範例會透過 JSON 承載，將所有要求參數 (包括pathheader、和) 傳送至整合端點：querystring

```
#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
  }
}
```

針對包含下列輸入參數的要求：

- 名為的路徑參數 myparam
- 查詢字串參數 querystring1=value1,value2&querystring2=value3
- 標頭"header1" : "value1"、"header2" : "value2"、"header3" : "value3"。

此對應範本的輸出應如下所示：

```
{
  "params" : {
        "path" : {
            "path" : "myparam"
                }
    ,        "querystring" : {
            "querystring1" : "value1,value2"
      ,            "querystring2" : "value3"
                }
    ,        "header" : {
            "header3" : "value3"
      ,            "header2" : "value2"
      ,            "header1" : "value1"
                }
          }
}
```

JSON 對應範本範例

建議您使用 $input 變數來取得查詢字串和請求內文，而不論是否使用模型。您可能還想要獲取參數和有效負載，或有效負載的子部分。以下三個範例顯示如何執行此操作。

下列範例會使用對應範本取得承載的子區段。這個例子獲取輸入參數，name然後獲取整個 POST 主體：

```
{
    "name" : "$input.params('name')",
    "body" : $input.json('$')
}
```

對於包含查詢字串參數name=Bella&type=dog和下列主體的要求：

```
{
    "Price" : "249.99",
    "Age": "6"
}
```

此對應範本的輸出應如下所示：

```
{
    "name" : "Bella",
    "body" : {"Price":"249.99","Age":"6"}
}
```

如果 JSON 輸入包含無法剖析的未逸出字元 JavaScript，API Gateway 可能會傳回 400 回應。套用$util.escapeJavaScript($input.json('$'))以確保可正確剖析 JSON 輸入。

上一個$util.escapeJavaScript($input.json('$'))應用的示例如下：

```
{
    "name" : "$input.params('name')",
    "body" : $util.escapeJavaScript($input.json('$'))
}
```

在這種情況下，此映射模板的輸出應如下所示：

```
{
    "name" : "Bella",
    "body": {\"Price\":\"249.99\",\"Age\":\"6\"}
```

```
}
```

JSONPath 運算式範例

下列範例示範如何將 JSONPath 表達式傳遞給 `json()` 方法。您也可以使用句點來讀取請求主體物件的子區段`.`，以指定屬性：

```
{
    "name" : "$input.params('name')",
    "body" : $input.json('$.Age')
}
```

對於包含查詢字串參數`name=Bella&type=dog`和下列主體的要求：

```
{
    "Price" : "249.99",
    "Age": "6"
}
```

此對應範本的輸出應如下所示：

```
{
    "name" : "Bella",
    "body" : "6"
}
```

如果方法請求有效負載包含無法解析的未轉義字符 JavaScript，則 API Gateway 可能會返回`400`響應。套用`$util.escapeJavaScript()`以確保可正確剖析 JSON 輸入。

上一個`$util.escapeJavaScript($input.json('$.Age'))`應用的示例如下：

```
{
    "name" : "$input.params('name')",
    "body" : "$util.escapeJavaScript($input.json('$.Age'))"
}
```

在這種情況下，此映射模板的輸出應如下所示：

```
{
    "name" : "Bella",
    "body": "\"6\""
```

```
}
```

請求和響應示例

下列範例會$input.params()針對具有路徑$input.json()的資源使用$input.path()、和/things/{id}：

```
{
    "id" : "$input.params('id')",
    "count" : "$input.path('$.things').size()",
    "things" : $input.json('$.things')"
}
```

對於包含 path 參數123和下列主體的要求：

```
{
      "things": {
            "1": {},
            "2": {},
            "3": {}
      }
}
```

此對應範本的輸出應如下所示：

```
{"id":"123","count":"3","things":{"1":{},"2":{},"3":{}}}
```

如果方法請求有效負載包含無法解析的未轉義字符 JavaScript，則 API Gateway 可能會返回響應。400套用$util.escapeJavaScript()以確保可正確剖析 JSON 輸入。

上一個$util.escapeJavaScript($input.json('$.things'))應用的示例如下：

```
{
     "id" : "$input.params('id')",
     "count" : "$input.path('$.things').size()",
     "things" : "$util.escapeJavaScript($input.json('$.things'))"
}
```

此對應範本的輸出應如下所示：

```
{"id":"123","count":"3","things":"{\"1\":{},\"2\":{},\"3\":{}}"}
```

如需其他映射範例，請參閱「了解對應範本」。

$stageVariables

階段變數可以用於參數映射和映射範本，並做為方法整合中使用之 ARN 和 URL 的預留位置。如需詳細資訊，請參閱 the section called “設定階段變數”。

語法	描述
$stageVariables. *<variable_name>* 、$stageVariables[' *<variable_name>* '] 或 ${stageVariables[' *<variable_name>* ']}	*<variable_name>* 代表階段變數名稱。

$util 變數

$util 變數包含要在映射範本中使用的公用程式函數。

> **Note**
>
> 除非特別指定，否則預設字元集為 UTF-8。

函數	描述
$util.escapeJavaScript()	轉義使用字符串規則的字符串中的 JavaScript 字符。 **Note** 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：

函數	描述
	<pre>"input" : "$util.escapeJavaScript(data).replaceAll("\\'","'")"</pre>

$util.parseJson()	採用「字串化」JSON，並傳回結果的物件呈現。您可以使用此函數的結果，來存取和運用 Apache Velocity 範本語言 (VTL) 中原生承載的元素。例如，如果您有下列承載：
	<pre>{"errorMessage":"{\"key1\":\"var1\",\"key2\":{\"arr\":[1,2,3]}}"}</pre>並使用下列映射範本
	<pre>#set ($errorMessageObj = $util.parseJson($input.path('$.errorMessage')))
	{
	   "errorMessageObjKey2ArrVal" : $errorMessageObj.key2.arr[0]
	}</pre>您將會收到下列輸出：
	<pre>{
	   "errorMessageObjKey2ArrVal" : 1
	}</pre>

$util.urlEncode()	將字符串轉換為「應用程序/x-www-form-urlencoded」格式。
$util.urlDecode()	解碼「應用程序/x-www-form-urlencoded」字符串。
$util.base64Encode()	將資料編碼為 base64 編碼字串。

函數	描述
`$util.base64Decode()`	解碼 base64 編碼字串中的資料。

API Gateway 中的閘道回應

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，閘道回應會以 GatewayResponse. 在 OpenAPI 中，一個`GatewayResponse`實例由 x-amazon-apigateway-gateway-響應. 網關響應擴展來描述。

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 在外部交換與 AWS 雲端之間進行調解時，您可以使用 VTL 對應範本進行整合要求或整合回應，將承載從一種格式對應到另一種格式。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 $context 變數的詳細資訊，請參閱「$context資料模型、授權者、對應範本和 CloudWatch 存取記錄的變數」。如需有關 $stageVariables 的詳細資訊，請參閱 $stageVariables。如需有關方法請求參數的詳細資訊，請參閱 the section called “$input 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
- 閘道回應類型

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，並假設對應 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，因此會傳回錯誤來觸發指定的閘道回應。您應該取得類似如下的閘道回應：

```
> GET /custErr/pets/5?q=1 HTTP/1.1
Host: o81lxisefl.execute-api.us-east-1.amazonaws.com
User-Agent: curl/7.51.0
Accept: */*

HTTP/1.1 404 Not Found
Content-Type: application/json
Content-Length: 334
Connection: keep-alive
Date: Tue, 02 May 2017 03:15:47 GMT
x-amzn-RequestId: 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，請呼叫閘道回應：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 定義會顯示自訂MISSING_AUTHENTICATION_TOKEN類型GatewayResponse的範例。

```
  "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。

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。

閘道回應類型	預設狀態碼	描述
AUTHORIZER_FAILURE	500	自訂或 Amazon Cognito 授權方無法驗證發起人時的閘道回應。如果未指定回應類型，此回應預設為 DEFAULT_5XX 類型。
BAD_REQUEST_PARAMETERS	400	根據啟用的請求驗證程式無法驗證請求參數時的閘道回應。如果未指定回應類型，此回應預設為 DEFAULT_4XX 類型。
BAD_REQUEST_BODY	400	根據啟用的請求驗證程式無法驗證請求內文時的閘道回應。如果未指定回應類型，此回應預設為 DEFAULT_4XX 類型。
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。 **Note** AWS WAF 自訂回應的優先順序高於自訂閘道回應。

閘道回應類型	預設狀態碼	描述
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 驗證 Token 的閘道回應已過期錯誤。如果未指定回應類型，此回應預設為 DEFAULT_4XX 類型。
INTEGRATION_FAILURE	504	整合失敗錯誤的閘道回應。如果未指定回應類型，此回應預設為 DEFAULT_5XX 類型。
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。

閘道回應類型	預設狀態碼	描述
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。如果未指定響應類型，則此響應默認為：HTTP content length exceeded 10485760 bytes。
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。

閘道回應類型	預設狀態碼	描述
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。 **Note** AWS WAF 自訂回應的優先順序高於自訂閘道回應。

為 REST API 資源啟用 CORS

跨來源資源共享 (CORS) 是一種瀏覽器安全功能，限制從瀏覽器中執行之指令碼啟動的跨來源 HTTP 請求。如需詳細資訊，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。在這種情況下，未映射內容類型的方法請求將被拒絕 HTTP 415 不支持的媒體類型響應。如需詳細資訊，請參閱 整合傳遞行為。
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: '*'`
- `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。

為非代理整合啟用 CORS AWS Management Console

您可以使用 AWS Management Console 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。

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 CORS

使用 API Gateway 主控台在資源上啟用 CORS

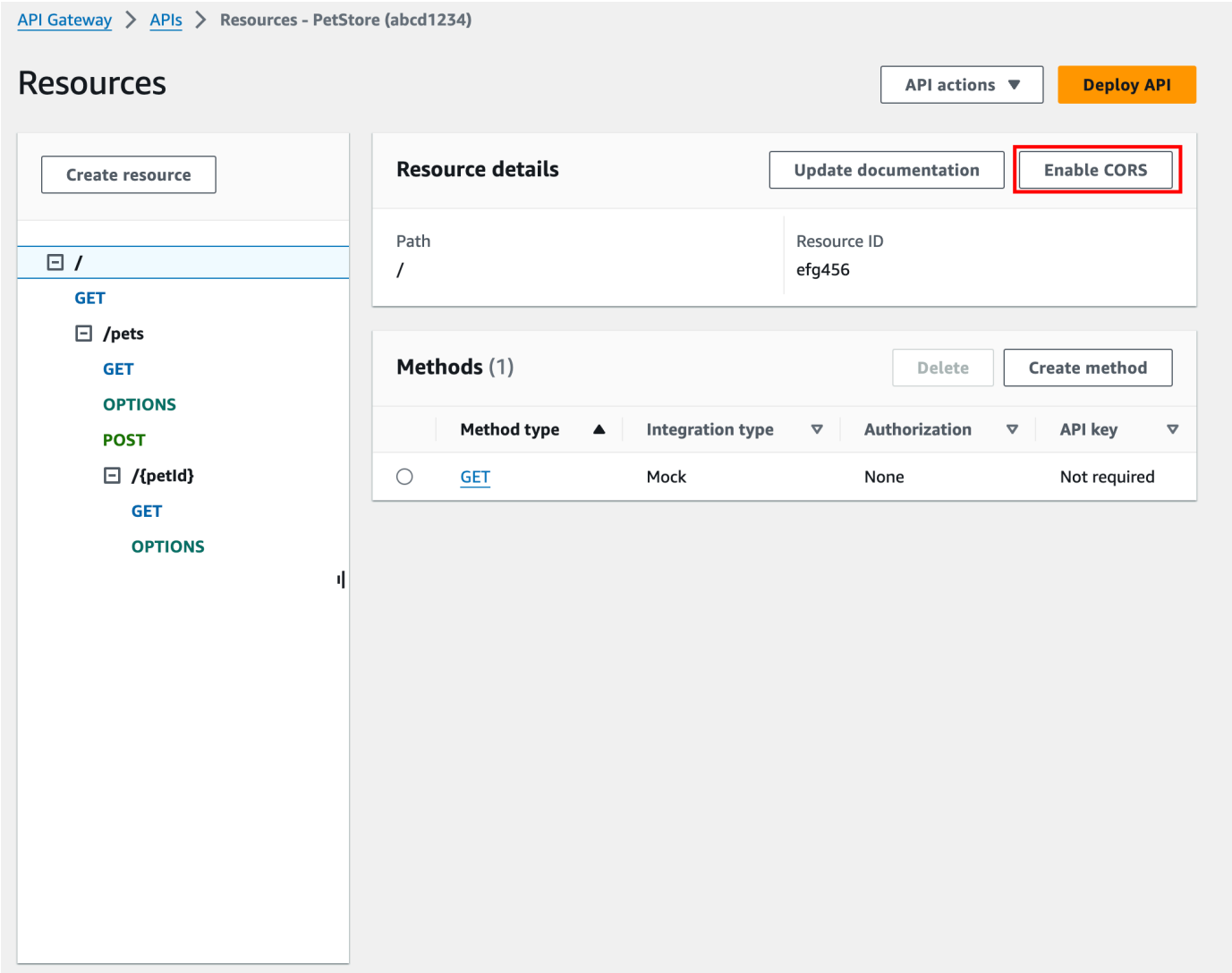
您可以使用 API Gateway 主控台，為您已建立的 REST API 資源上的一個或所有方法啟用 CORS 支援。啟用 COR 支援之後，請將整合傳遞行為設定為NEVER。在這種情況下，未映射內容類型的方法請求將被拒絕 HTTP 415 不支持的媒體類型響應。如需更多資訊，請參閱整合傳遞行為

> **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。

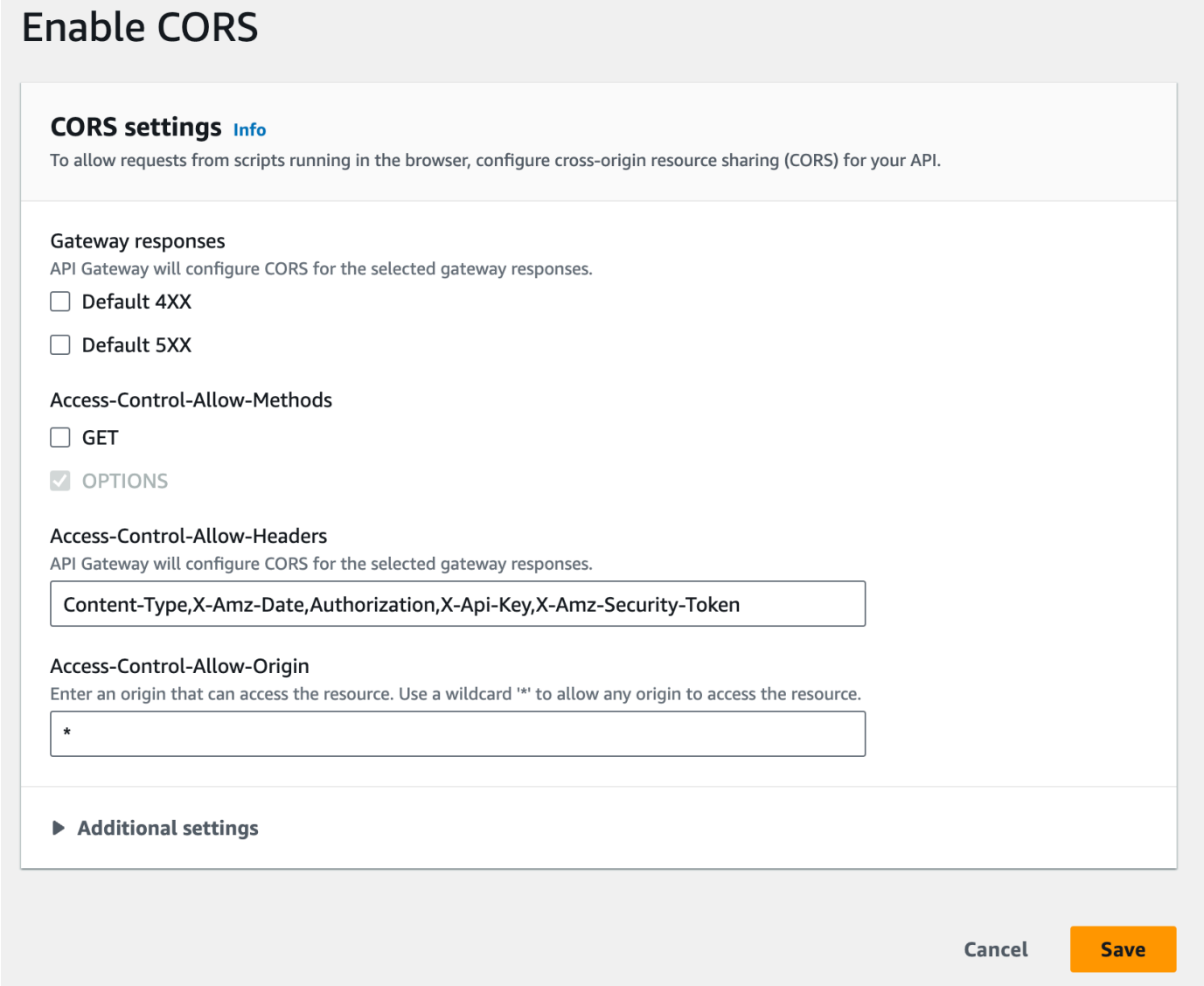
   b. 選取每一種方法以啟用 CORS 支援。OPTION 方法必須啟用 CORS。

      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，做為 Access-Control-Allow-Origin 標頭值，以允許所有來源中的存取請求，或指定允許其存取資源的來源。

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 “部署 REST API (主控台)”。

> **Note**
>
> 如果您在執行此程序之後無法在資源上啟用 CORS 支援，建議您將 CORS 設定與範例 API /pets 資源進行比較。若要瞭解如何建立範例 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 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。如需更多詳細資訊，請參閱 為 REST API 資源啟用 CORS。

使用 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。

AWS Lambda 代理整合

要處理 AWS Lambda 代理集成的二進制有效載荷，您必須對函數的響應進行 base64 編碼。您還必須binaryMediaTypes為您的 API 配置。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。例如，`*/*` 包含所有內容類型。

如需範例程式碼，請參閱 the section called “從 Lambda 代理整合傳回二進位媒體”。

非代理伺服器整合

若要處理非 Proxy 整合的二進位裝載，請將媒體類型新增至資源binaryMediaTypes清單。`RestApi`您的 API `binaryMediaTypes` 組態是 API 視為二進位資料的內容類型列表。或者，您可以在整合和資源上設定 contentHandling 屬性。IntegrationResponse`contentHandling` 值可以是 `CONVERT_TO_BINARY`、`CONVERT_TO_TEXT` 或未定義。

根據 `contentHandling` 值，以及回應的 `Content-Type` 標頭或傳入請求的 `Accept` 標頭是否符合 `binaryMediaTypes` 清單中的項目，API Gateway 可以將原始二進位位元組編碼為 Base64 編碼字串、將 Base64 編碼字串解碼回其原始位元組，或傳遞本文而不進行任何修改。

您必須遵循下列方式設定 API，以在 API Gateway 中支援 API 的二進位承載：

- 將所需的二進位媒體類型新增至資RestApi源`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 如何針對要求Content-Type標頭的特定組態、資源binaryMediaTypes清單和整合RestApi資源的contentHandling屬性值，轉換要求承載。

API Gateway 中的 API 請求內容類型轉換

方法請求承載	請求 **Content-Type** 標頭	**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
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

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_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 如何針對請求`Accept`標頭的特定配置，資源`binaryMediaTypes`列表和RestApi資源的`contentHandling`屬性值轉換響應有效負載。IntegrationResponse

> **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。

API Gateway 回應內容類型轉換

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，請參閱教學課程：在 API Gateway 中建立 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。

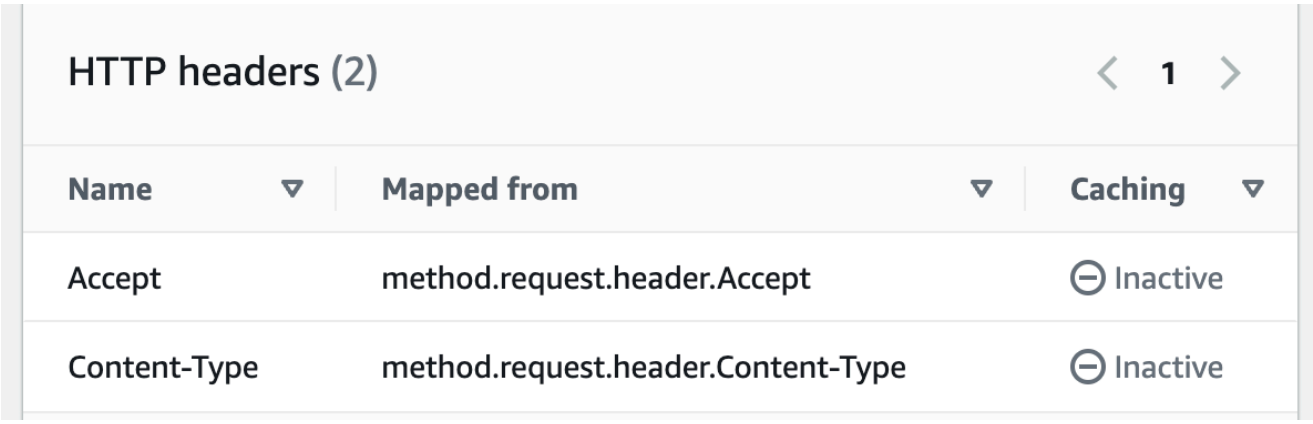
c. 針對內容處理，選擇一個選項。

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 將二進位資料以 JSON 屬性的形式傳送至 AWS Lambda 或 Kinesis，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"
    }
  ]
}
```

匯入與匯出內容編碼

若要匯入上的`binaryMediaTypes`清單 RestApi，請使用下列 API Gateway 副檔名至 API 的 OpenAPI 定義檔案。此延伸也可用來匯出 API 設定。

- x-amazon-apigateway-binary-媒體類型屬性

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. 回應物件

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。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，請參閱使用 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，後面接著所提供的對應範本。PUT 方法接受原始二進位 Blob 作為輸入，並傳回「200 OK」回應與空的承載。

OpenAPI 3.0

```
{
   "openapi": "3.0.0",
   "info": {
      "version": "2016-10-21T17:26:28Z",
      "title": "ApiName"
   },
   "paths": {
      "/s3": {
         "get": {
            "parameters": [
               {
                  "name": "key",
                  "in": "query",
                  "required": false,
                  "schema": {
                     "type": "string"
                  }
               }
            ],
            "responses": {
               "200": {
                  "description": "200 response",
                  "content": {
                     "application/json": {
                        "schema": {
                           "$ref": "#/components/schemas/Empty"
```

```
                    }
                 }
              }
           },
           "500": {
              "description": "500 response"
           }
        },
        "x-amazon-apigateway-integration": {
           "credentials": "arn:aws:iam::123456789012:role/binarySupportRole",
           "responses": {
              "default": {
                 "statusCode": "500"
              },
              "2\\d{2}": {
                 "statusCode": "200"
              }
           },
           "requestParameters": {
              "integration.request.path.key": "method.request.querystring.key"
           },
           "uri": "arn:aws:apigateway:us-west-2:s3:path/{key}",
           "passthroughBehavior": "when_no_match",
           "httpMethod": "GET",
           "type": "aws"
        }
     },
     "put": {
        "parameters": [
           {
              "name": "key",
              "in": "query",
              "required": false,
              "schema": {
                 "type": "string"
              }
           }
        ],
        "responses": {
           "200": {
              "description": "200 response",
              "content": {
                 "application/json": {
                    "schema": {
```

```
                        "$ref": "#/components/schemas/Empty"
                     }
                  },
                  "application/octet-stream": {
                     "schema": {
                        "$ref": "#/components/schemas/Empty"
                     }
                  }
               }
            },
            "500": {
               "description": "500 response"
            }
         },
         "x-amazon-apigateway-integration": {
            "credentials": "arn:aws:iam::123456789012:role/binarySupportRole",
            "responses": {
               "default": {
                  "statusCode": "500"
               },
               "2\\d{2}": {
                  "statusCode": "200"
               }
            },
            "requestParameters": {
               "integration.request.path.key": "method.request.querystring.key"
            },
            "uri": "arn:aws:apigateway:us-west-2:s3:path/{key}",
            "passthroughBehavior": "when_no_match",
            "httpMethod": "PUT",
            "type": "aws",
            "contentHandling": "CONVERT_TO_BINARY"
         }
      }
   },
   "x-amazon-apigateway-binary-media-types": [
      "application/octet-stream",
      "image/jpeg"
   ],
   "servers": [
      {
         "url": "https://abcdefghi.execute-api.us-east-1.amazonaws.com/{basePath}",
         "variables": {
```

```
            "basePath": {
              "default": "/v1"
            }
         }
      }
   ],
   "components": {
      "schemas": {
         "Empty": {
            "type": "object",
            "title": "Empty Schema"
         }
      }
   }
}
```

OpenAPI 2.0

```
{
  "swagger": "2.0",
  "info": {
    "version": "2016-10-21T17:26:28Z",
    "title": "ApiName"
  },
  "host": "abcdefghi.execute-api.us-east-1.amazonaws.com",
  "basePath": "/v1",
  "schemes": [
    "https"
  ],
  "paths": {
    "/s3": {
      "get": {
        "produces": [
          "application/json"
        ],
        "parameters": [
          {
            "name": "key",
            "in": "query",
            "required": false,
            "type": "string"
          }
        ],
```

```
        "responses": {
          "200": {
            "description": "200 response",
            "schema": {
              "$ref": "#/definitions/Empty"
            }
          },
          "500": {
            "description": "500 response"
          }
        },
        "x-amazon-apigateway-integration": {
          "credentials": "arn:aws:iam::123456789012:role/binarySupportRole",
          "responses": {
            "default": {
              "statusCode": "500"
            },
            "2\\d{2}": {
              "statusCode": "200"            }
          },
          "requestParameters": {
            "integration.request.path.key": "method.request.querystring.key"
          },
          "uri": "arn:aws:apigateway:us-west-2:s3:path/{key}",
          "passthroughBehavior": "when_no_match",
          "httpMethod": "GET",
          "type": "aws"
        }
      },
      "put": {
        "produces": [
          "application/json", "application/octet-stream"
        ],
        "parameters": [
          {
            "name": "key",
            "in": "query",
            "required": false,
            "type": "string"
          }
        ],
        "responses": {
          "200": {
            "description": "200 response",
```

```
            "schema": {
              "$ref": "#/definitions/Empty"
            }
          },
          "500": {
            "description": "500 response"
          }
        },
        "x-amazon-apigateway-integration": {
          "credentials": "arn:aws:iam::123456789012:role/binarySupportRole",
          "responses": {
            "default": {
              "statusCode": "500"
            },
            "2\\d{2}": {
              "statusCode": "200"
            }
          },
          "requestParameters": {
            "integration.request.path.key": "method.request.querystring.key"
          },
          "uri": "arn:aws:apigateway:us-west-2:s3:path/{key}",
          "passthroughBehavior": "when_no_match",
          "httpMethod": "PUT",
          "type": "aws",
          "contentHandling" : "CONVERT_TO_BINARY"
        }
      }
    }
  },
  "x-amazon-apigateway-binary-media-types" : ["application/octet-stream", "image/
jpeg"],
  "definitions": {
    "Empty": {
      "type": "object",
      "title": "Empty Schema"
    }
  }
}
```

從 Amazon S3 下載影像

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 會公開下載`GET /lambda?key={file-name}`和上傳指定影像檔案的和`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，後面接著所提供的對應範本。PUT 方法接受原始二進位 Blob 作為輸入，並傳回「200 OK」回應與空的承載。

您可以建立 API 呼叫的 Lambda 函數，而且它必須傳回`Content-Type`標頭為的 base64 編碼字串。`application/json`

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 Amazon API Gateway 中叫用 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://restapi_id.execute-api.region.amazonaws.com/stage_name/
```

其中 *`restapi_id`* 是 API 標識符，*`#####`*，而 *`stage_name # API #######`*。 AWS

> **Important**
>
> 在您可以叫用 API 之前，您必須先在 API Gateway 中部署 API。如需部署 API 的指示，請參閱在 Amazon API Gateway 中部署 REST API。

主題

- 獲取 API 的調用網址
- 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
- 針對 REST API 使用 API 由 API 產生的 API 所產生的 JavaScript SDK
- 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

獲取 API 的調用網址

您可以使用主控台 AWS CLI、或匯出的 OpenAPI 定義來取得 API 的叫用 URL。

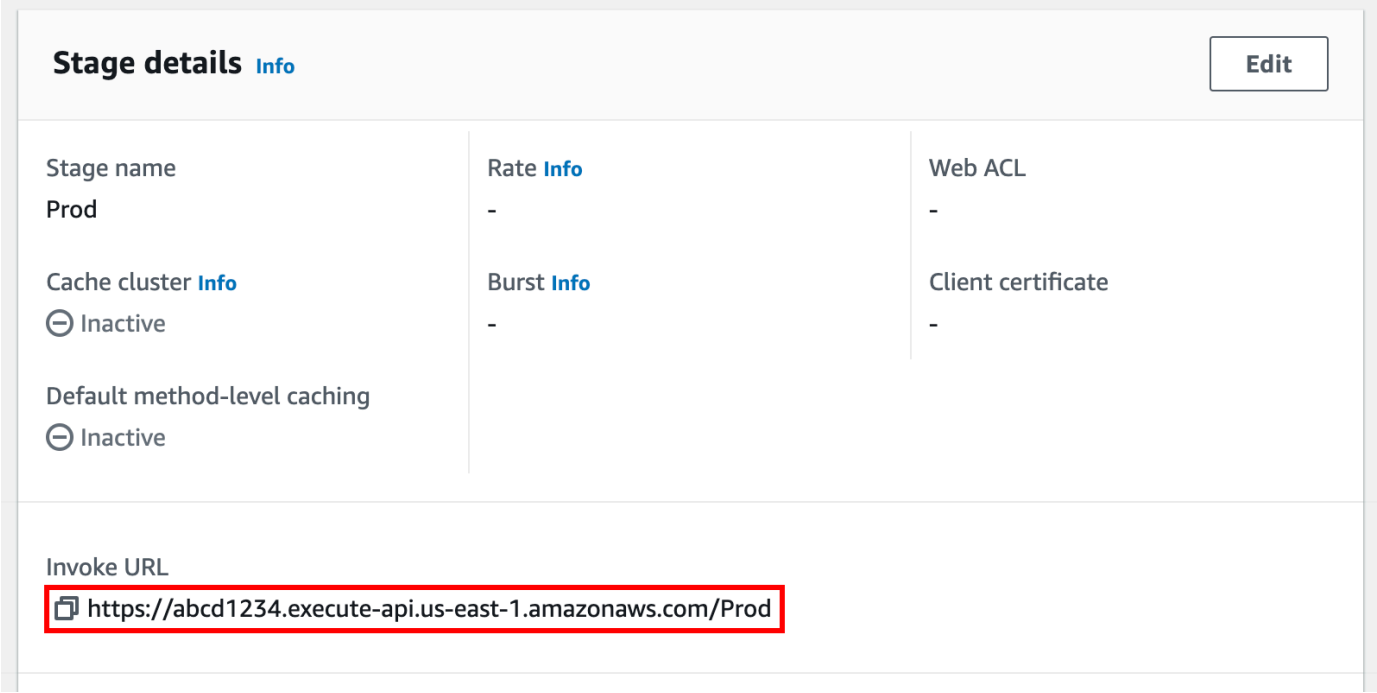
使用控制台獲取 API 的調用 URL

下列程序顯示如何在 REST API 主控台中取得 API 的呼叫網址。

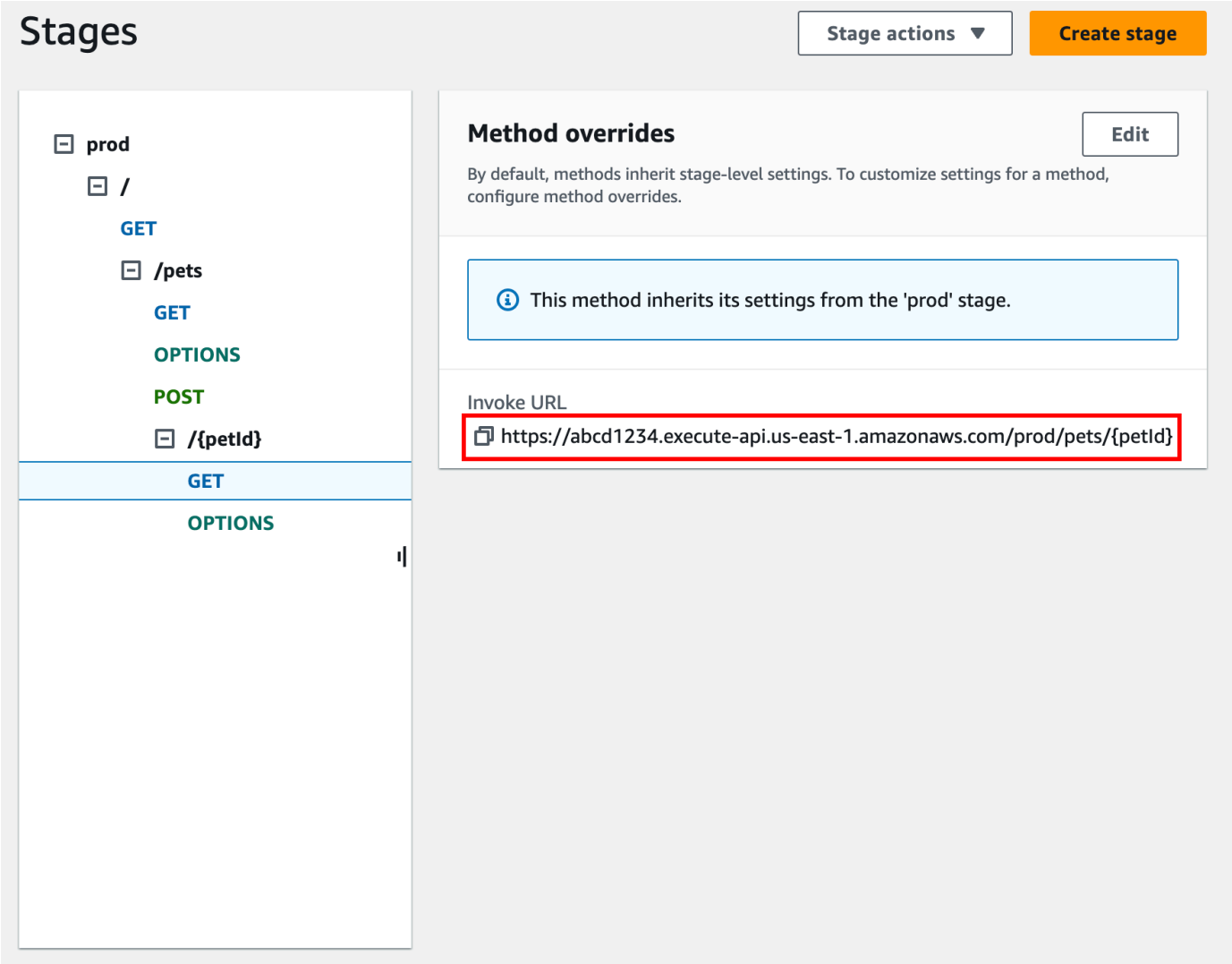
若要使用 REST API 主控台取得 API 的呼叫網址

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 選擇已部署的 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。

   此網址適用於 API 的根資源。



5. 若要取得 API 中其他資源的 API 叫用 URL，請展開次要導覽窗格下的階段，然後選擇方法。
6. 選擇複製圖示以複製 API 的資源層級叫用 URL。



取得 API 的叫用網址，使用 AWS CLI

下列程序顯示如何使用取得 API 的呼叫 URL AWS CLI。

若要取得 API 的叫用網址，請使用 AWS CLI

1. 使用下列命令來取得`rest-api-id`. 此指令會傳回您「地區」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. 將範例取代為您`rest-api-id`的範例`rest-api-id`，以 *{####}* 取代範例 *{####}*，並以您的區域取代 *{region}*。

```
https://{restapi_id}.execute-api.{region}.amazonaws.com/{stage_name}/
```

使用 API 匯出的 OpenAPI 定義檔案取得 API 的呼叫網址

您也可以結合 API 的已匯出 OpenAPI 定義檔案的`host`和`basePath`欄位來建構根 URL。如需如何匯出 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) 來呼叫已部署的 API。

此外，您可以使用 API Gateway 主控台來測試 API 呼叫。測試使用 API 閘道的`TestInvoke`功能，該功能允許在部署 API 之前進行 API 測試。如需詳細資訊，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 %%。

使用網頁瀏覽器叫用 API

如果您的 API 允許匿名訪問，則可以使用任何 Web 瀏覽器調用任何`GET`方法。在瀏覽器的地址欄中輸入完整的調用 URL。

對於其他方法或任何需要驗證的呼叫，您必須指定有效負載或簽署要求。您可以在 HTML 頁面後面的指令碼或使用其中一個 AWS SDK 的用戶端應用程式中處理這些問題。

使用捲曲調用 API

您可以在終端中使用 curl 之類的工具來調用 API。下列範例 curl 命令會在 API `prod` 階段的`getUsers`資源上叫用 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

- 必要條件
- 使用 API Gateway 主控台測試方法

必要條件

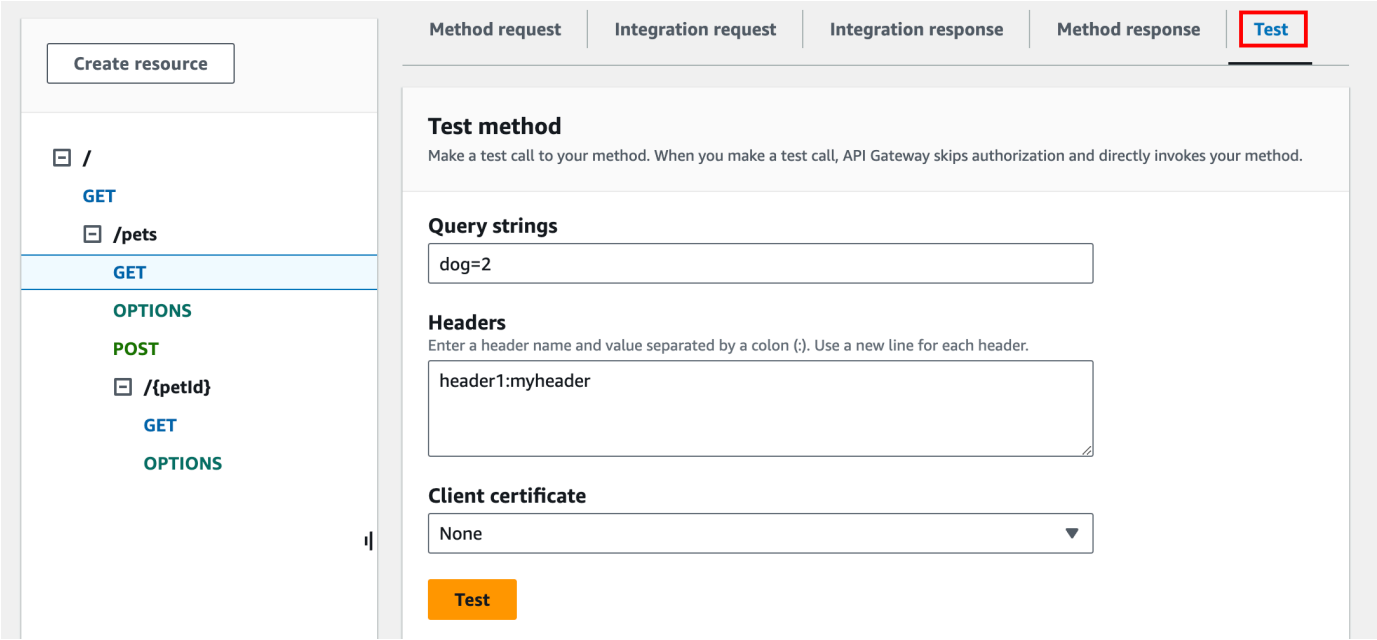
- 您必須指定所要測試之方法的設定。請遵循中的說明進行API Gateway 中其餘 API 的方法

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（毫秒）是從調用者接收請求和返回響應之間的時間。
   - 回應內文是 HTTP 回應內文。
   - 回應標頭是 HTTP 回應標頭。

   > **Tip**
   >
   > 根據對應的不同，HTTP 狀態碼、回應主體和回應標頭可能與從 Lambda 函數、HTTP 代理或 AWS 服務代理傳送的標頭不同。

   - 日誌是在 API Gateway 主控台外部呼叫此方法時，會寫入的模擬 Amazon CloudWatch 日誌項目。

> **Note**
>
> 雖然 CloudWatch 日誌條目是模擬的，但方法調用的結果是真實的。

除了使用 API Gateway 主控台之外，您還可以使用 AWS CLI 或 API Gateway 的 AWS SDK 來測試叫用方法。若要使用這項操作 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 中針對 REST API 產生軟體開發套件中的步驟。

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。應用程序存根包含一個`pom.xml`文件和項目根目錄下的文件`src`夾（前面的命令中的 *`SimpleCalc-SDKClient`*）。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 中針對 REST API 產生軟體開發套件中的步驟，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 登入資料提供者授權對 API 的呼叫，請使用該`ApiClientFactory`類別透過使用 API Gateway 產生的 SDK 傳遞一組 AWS 登入資料，如下列範例所示。

```
// Use CognitoCachingCredentialsProvider to provide AWS credentials
// for the ApiClientFactory
AWSCredentialsProvider credentialsProvider = new CognitoCachingCredentialsProvider(
        context,          // activity context
        "identityPoolId", // Cognito identity pool id
        Regions.US_EAST_1 // region of Cognito identity pool
);

ApiClientFactory factory = new ApiClientFactory()
  .credentialsProvider(credentialsProvider);
```

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");
```

針對 REST API 使用 API 由 API 產生的 API 所產生的 JavaScript SDK

> **Note**
>
> 這些說明假設您已完成在 API Gateway 中針對 REST API 產生軟體開發套件中的指示。

> **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。

若要安裝、啟動和呼叫由 REST API 的 API Gateway 產生的 JavaScript SDK

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，請參閱為 REST API 資源啟用 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 認證初始化 API Gateway 產生的 SDK，請使用類似下列內容的程式碼。如果您使用 AWS 認證，則對 API 的所有請求都將被簽署。

```
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 而言，適用於 API 方法的 SDK 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：

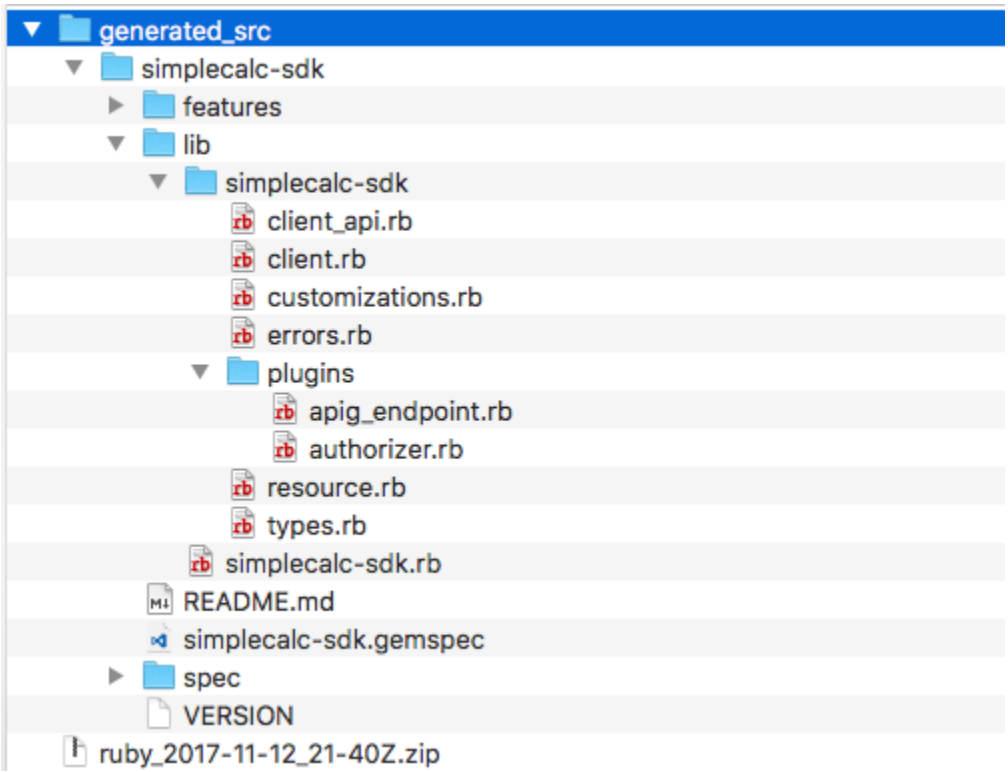
```
1. rootGet(params);      // where params={"a": ..., "b": ..., "op": ...} is
 resolved to the query parameters
2. rootPost(null, body); // where body={"a": ..., "b": ..., "op": ...}
3. aBOpGet(params);      // where params={"a": ..., "b": ..., "op": ...} is
 resolved to the path parameters
```

使用由 API Gateway 為 REST API 產生的 Ruby 軟體開發套件

> **Note**
>
> 這些說明假設您已完成「在 API Gateway 中針對 REST API 產生軟體開發套件」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 已配置AWS_IAM類型的授權，則可以通過提供accessKey和初始化secretKey期間包含調用者的 AWS 憑據：

```
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。我們將使用 SimpleCalc API 作為示例來說明以下主題：

- 如何將所需的 AWS 移動 SDK 組件安裝到您的 Xcode 項目中
- 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

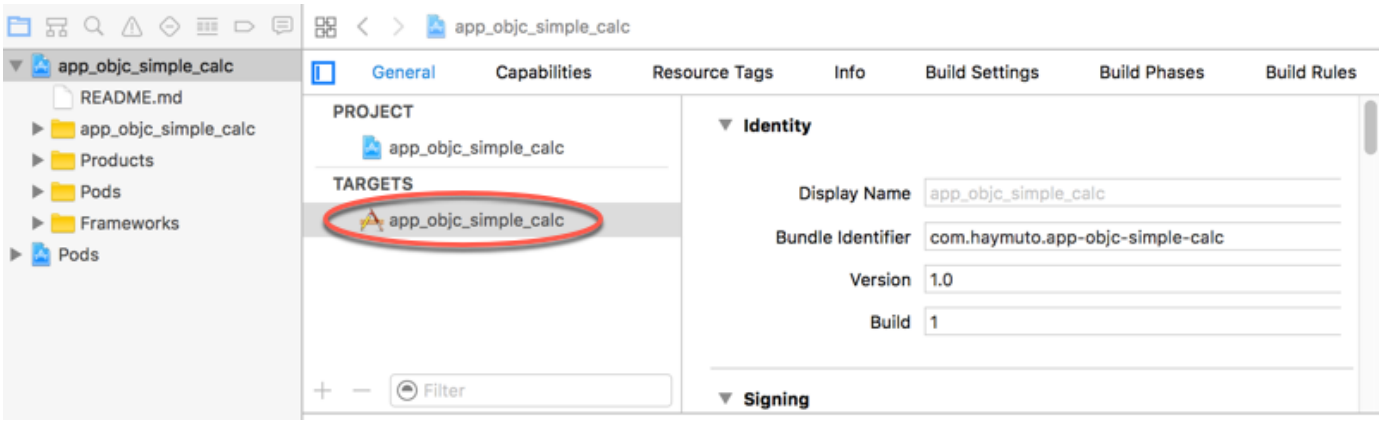
開始下列程序之前，您必須先在 Objective-C 中完成在 API Gateway 中針對 REST API 產生軟體開發套件中的 iOS 步驟，並下載已產生之軟體開發套件的 .zip 檔案。

在目標 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，您可能需要將解壓縮的 SDK 文件夾重命名為類似**`sdk_objc_simple_calc`**的內容。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匯入必要的 API Gateway 程式庫和其他相依的 AWS 行動 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. 要使用 AWS Mobile SDK for iOS 將其導入到 Xcode 項目中 CocoaPods，請執行以下操作：

   a. CocoaPods 透過在終端機視窗中執行下列命令來進行安裝：

   ```
   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'
   ```

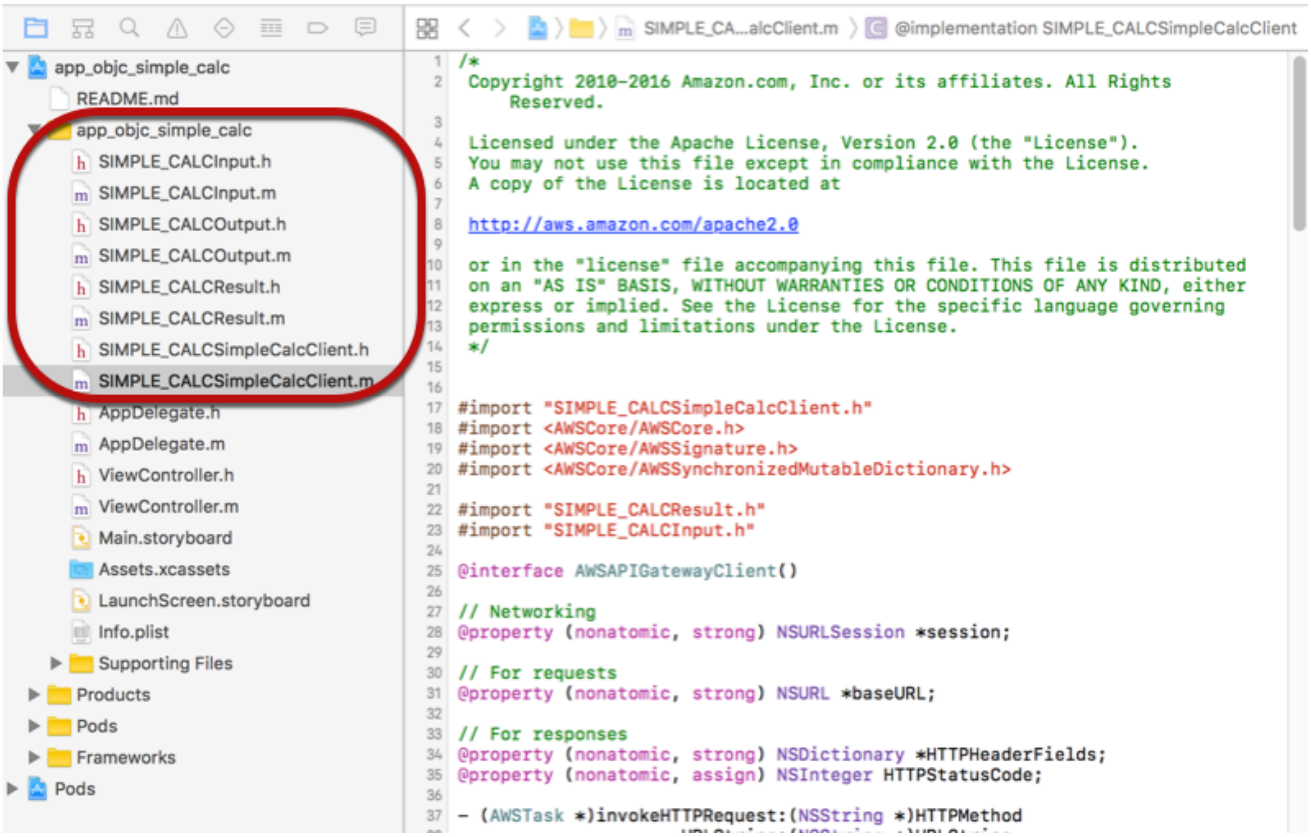
   c. 開啟終端機視窗並執行下列命令：

   ```
   pod install
   ```

   這會安裝 API Gateway 元件和其他相依的 AWS 行動 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。



要通過明確下載 AWS 移動 SDK 或使用迦太基將 AWS Mobile SDK for iOS Objective-C 導入到您的項目中，請按照 README.md 文件中的說明進行操作。請務必只使用其中一個選項來匯入 AWS 行動 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

當您使用這個 SimpleCalc API 的`SIMPLE_CALC`前置詞產生 SDK 時，其中包含兩個用於輸入 (`Input`) 和 output (`Result`) 方法的模型時，在 SDK 中，產生的 API 用戶端類別會變成 `SIMPLE_CALCSimpleCalcClient`和`SIMPLE_CALCResult`，而對應的資料類別分別為`SIMPLE_CALCInput`和。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 中針對 REST API 產生軟體開發套件中的 iOS 步驟，並下載已產生之軟體開發套件的 .zip 檔案。

主題

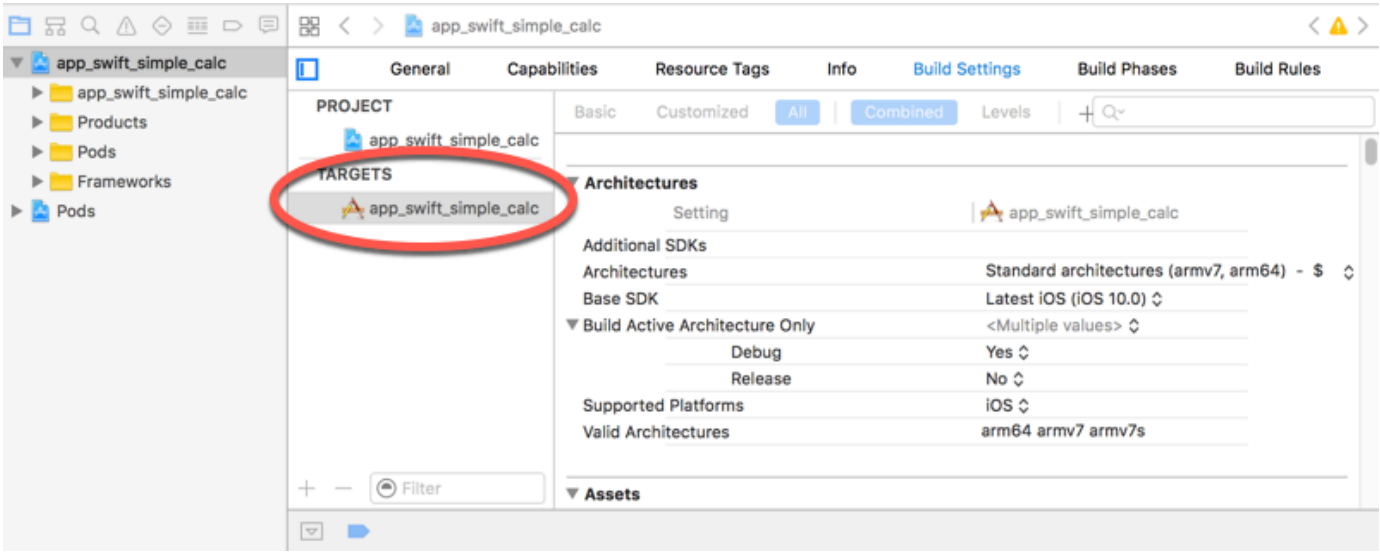
- 在 Swift 專案中安裝 AWS 行動 SDK 和 API 閘道產生的 SDK
- 在 Swift 專案中透過 API Gateway 所產生的 iOS 軟體開發套件來呼叫 API 方法

在 Swift 專案中安裝 AWS 行動 SDK 和 API 閘道產生的 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，您可能需要將解壓縮的 SDK 文件夾重命名為類似**sdk_swift_simple_calc**的內容。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匯入必要的 AWS 行動 SDK 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. 要使用將所需的 AWS 移動 SDK 組件導入到 Xcode 項目中 CocoaPods，請執行以下操作：

   a. 如果未安裝，請在終端機視窗中執行下列命令 CocoaPods 來進行安裝：

   ```
   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'
```

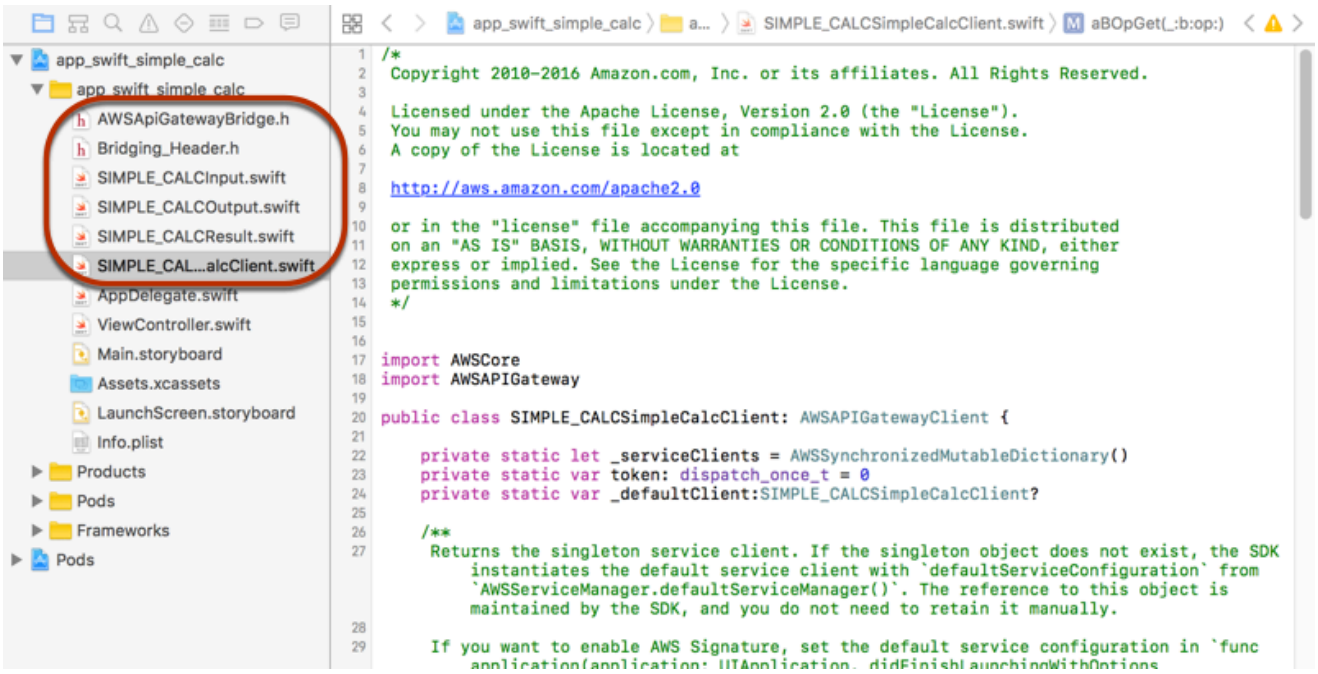
c. 開啟終端機視窗，並在應用程式目錄中執行下列命令：

```
pod install
```

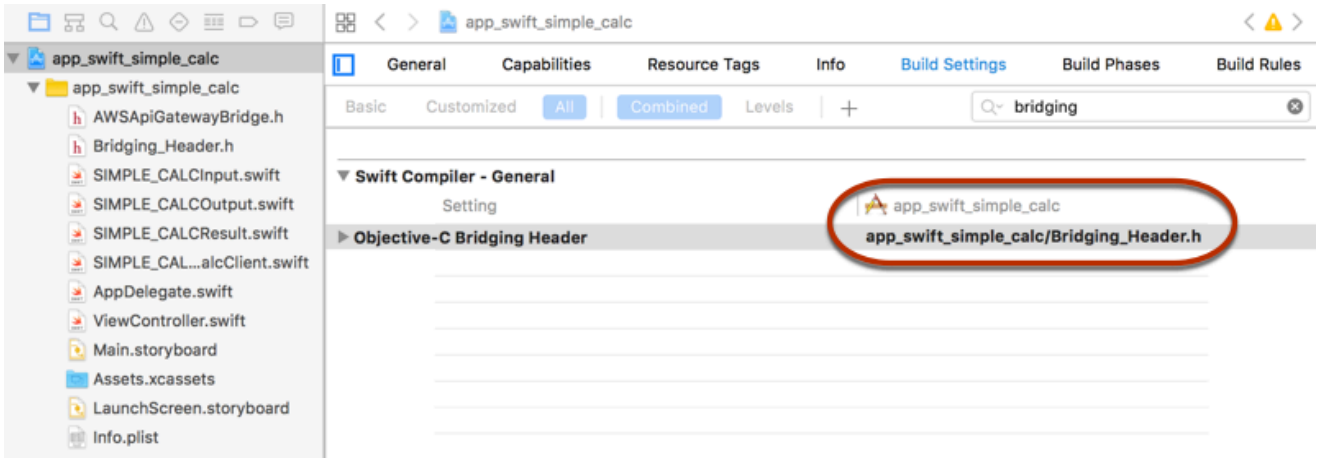
這會將 API Gateway 元件和任何相依的 AWS 行動 SDK 元件安裝到應用程式的專案中。

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。

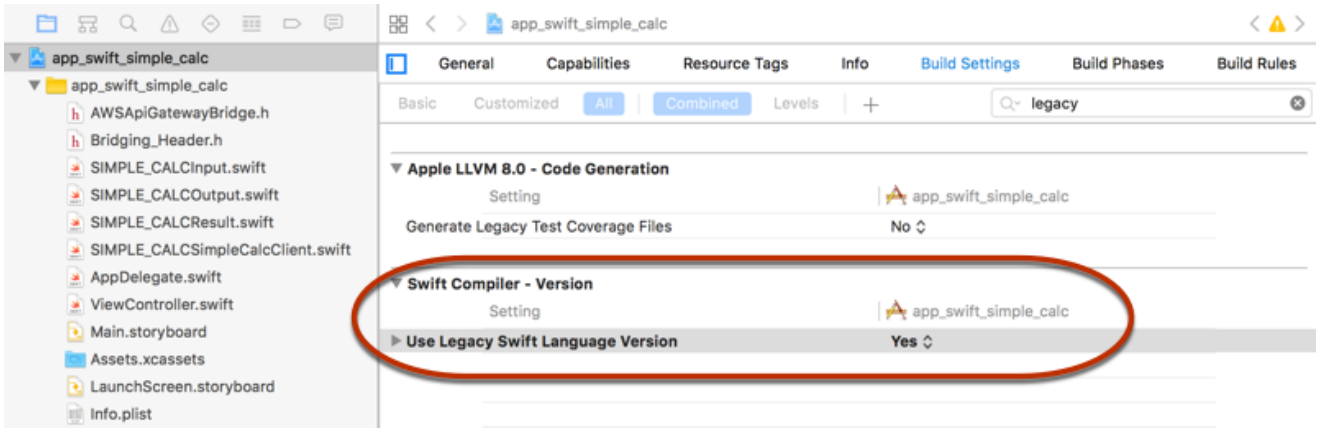


f. 要啟用從 Swift 代碼項目調用 AWS 移動 SDK 的 Objective-C 庫，請在 Swift 編譯器-Xcode 項目配置的常規設置下的 Objective-C 橋接頭屬性上設置`Bridging_Header.h`文件路徑：



> **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 使用的是比 AWS 移動 SDK 支持的更新版本的 Swift，則會出現 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)：



要通過明確下載 AWS 移動 SDK 或使用迦太基將適用於 iOS 的 AWS 移動 SDK 導入到您的項目中，請按照 SDK 包附帶的`README.md`文件中的說明進行操作。請務必只使用其中一個選項來匯入 AWS 行動 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

當您使用兩個模型來描述 SimpleCalc API 要求和回應的 input (Input) 和 output (Result) 產生 SDK 時，在 SDK 中，產生的 API 用戶端類別會變成SIMPLE_CALCSimpleCalcClient，而對應的資料類別分別為SIMPLE_CALCInput和SIMPLE_CALCResult。SIMPLE_CALCAPI 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)，再取得方`default`法 (先前所示)：

```
let credentialsProvider =
 AWSCognitoCredentialsProvider(regionType: AWSRegionType.USEast1, identityPoolId:
 "my_pool_id")
let configuration = AWSServiceConfiguration(region: AWSRegionType.USEast1,
 credentialsProvider: credentialsProvider)
AWSServiceManager.defaultServiceManager().defaultServiceConfiguration =
 configuration
```

2. 呼叫 `GET /?a=1&b=2&op=+` 方法來執行 1+2：

```
client.rootGet("+", a: "1", b:"2").continueWithBlock {(task: AWSTask) -> AnyObject?
 in
    self.showResult(task)
    return nil
}
```

其中，helper 函數 `self.showResult(task)` 會將結果或錯誤列印至主控台，例如：

```
func showResult(task: AWSTask) {
    if let error = task.error {
        print("Error: \(error)")
    } else if let result = task.result {
        if result is SIMPLE_CALCResult {
            let res = result as! SIMPLE_CALCResult
            print(String(format:"%@ %@ %@ = %@", res.input!.a!, res.input!.op!,
 res.input!.b!, res.output!.c!))
        } else if result is NSDictionary {
            let res = result as! NSDictionary
            print("NSDictionary: \(res)")
```

```
        }
    }
}
```

在生產應用程式中，您可以在文字欄位中顯示結果或錯誤。產生的顯示畫面為 `1 + 2 = 3`。

3. 呼叫具有承載的 `POST /` 來執行 `1-2`：

```
let body = SIMPLE_CALCInput()
body.a=1
body.b=2
body.op="-"
client.rootPost(body).continueWithBlock {(task: AWSTask) -> AnyObject? in
    self.showResult(task)
    return nil
}
```

產生的顯示畫面為 `1 - 2 = -1`。

4. 呼叫 `GET /{a}/{b}/{op}` 來執行 `1/2`：

```
client.aBOpGet("1", b:"2", op:"div").continueWithBlock {(task: AWSTask) ->
 AnyObject? in
    self.showResult(task)
    return nil
}
```

產生的顯示畫面為 `1 div 2 = 0.5`。在此，`div` 用來代替 `/`，因為後端的簡單 Lambda 函數不會處理 URL 編碼的路徑變數。

使用 OpenAPI 設定 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
- 將區域性 API 匯入 API Gateway
- 匯入 OpenAPI 檔案以更新現有的 API 定義
- 設定 OpenAPI basePath 屬性
- AWS 匯入應用程式介面的變數
- 匯入期間的錯誤與警告
- 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 控制台或 AWS SDK 來執行此操作。 AWS CLI

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

若要從 OpenAPI 定義檔案匯入 API，以使用建立新的邊緣最佳化 API AWS CLI，請使用下`import-rest-api`列命令：

```
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'
```

將區域性 API 匯入 API Gateway

當您匯入 API 時，可以選擇 API 的區域端點組態。您可以使用 API Gateway 主控台 AWS CLI、或 AWS 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 AWS CLI匯入區域 API

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 AWS CLI匯入區域 API

若要使用從 OpenAPI 定義檔案匯入 API AWS CLI，請使用以下`import-rest-api`指令：

```
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。

該 import-to-update 操作可以在兩種模式下發生：合併或覆蓋。

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。

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 匯入應用程式介面的變數

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*}。

AWS 變數

變數名稱	描述	
AWS::AccountId	匯入 API 的 AWS 帳戶識別碼，例如 123456789012。	
AWS::Partition	匯入 API 所在的 AWS 分割區。對於標準 AWS 區域，分區是aws。	
AWS::Region	將 API 匯入的 AWS 區域 — 例如，. us-east-2	

AWS 變量示例

下列範例會使用 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
```

匯入期間的錯誤與警告

匯入期間的錯誤

匯入期間，OpenAPI 文件無效等重大問題可能會產生錯誤。這些錯誤會在失敗回應中以例外狀況 (例如 BadRequestException) 傳回。發生錯誤時，會捨棄新的 API 定義，而且不會對現有的 API 進行變更。

匯入期間的警告

匯入期間，遺失模型參考等次要問題可能會產生警告。如果發生警告，並將 `failonwarnings=false` 查詢表達式附加至請求 URL，操作將會繼續。否則會復原更新。`failonwarnings` 預設會設定為 `false`。在這種情況下，警告會作為結果RestApi資源中的欄位傳回。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。如需簽署要求的詳細資訊，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，請務必包含擴充功能參數，如下列範例所示，以確保包含`x-amazon-apigateway-request-validator`擴充功能：
>
> ```
> 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，您可以透過提交 GET 要求來匯出現有的 REST API，並將 to-be-exported API 指定為 URL 路徑的一部分。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 匯出 API 提交 GET 要求的詳細資訊，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`。有關 API Gateway 可用的所有區域，請參閱區域和端點。

下載 YAML 格式的 REST API OpenAPI 定義

以 YAML 格式的 OpenAPI 定義匯出並下載 REST API：

OpenAPI 3.0

```
GET /restapis/<restapi_id>/stages/<stage_name>/exports/oas30
Host: apigateway.<region>.amazonaws.com
Accept: application/yaml
```

OpenAPI 2.0

```
GET /restapis/<restapi_id>/stages/<stage_name>/exports/swagger
Host: apigateway.<region>.amazonaws.com
Accept: application/yaml
```

下載 JSON 格式且具有 Postman 延伸的 REST API OpenAPI 定義

以 JSON 格式的 OpenAPI 定義搭配 Postman 匯出並下載 REST API：

OpenAPI 3.0

```
GET /restapis/<restapi_id>/stages/<stage_name>/exports/oas30?extensions=postman
Host: apigateway.<region>.amazonaws.com
Accept: application/json
```

OpenAPI 2.0

```
GET /restapis/<restapi_id>/stages/<stage_name>/exports/swagger?extensions=postman
Host: apigateway.<region>.amazonaws.com
Accept: application/json
```

下載 YAML 格式且具有 API Gateway 整合的 REST API OpenAPI 定義

以 YAML 格式的 OpenAPI 定義搭配 API Gateway 整合匯出並下載 REST API：

OpenAPI 3.0

```
GET /restapis/<restapi_id>/stages/<stage_name>/exports/oas30?extensions=integrations
Host: apigateway.<region>.amazonaws.com
Accept: application/yaml
```

OpenAPI 2.0

```
GET /restapis/<restapi_id>/stages/<stage_name>/exports/swagger?
extensions=integrations
Host: apigateway.<region>.amazonaws.com
Accept: application/yaml
```

使用 API Gateway 主控台匯出 REST API

將 REST API 部署至階段之後，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。

發佈 REST API 以供客戶叫用

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 Amazon API Gateway 中部署 REST API
- 設定 REST API 的自訂網域名稱

在 Amazon API Gateway 中部署 REST API

API 在建立之後必須進行部署，這樣您的使用者才能呼叫該 API。

若要部署 API，請建立 API 部署，並建立它與階段的關聯。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，集成，授權者，資源策略以及階段設置以外的其他任何內容。

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。您也可以為 CloudTrail 或的 API 呼叫啟用記錄 CloudWatch，並且可以為後端選取用戶端憑證來驗證 API 要求。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，或將階段變數值從階段名稱 `test` 更新為階段名稱 `prod`。

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 中部署 REST API。
- 設定 REST API 的階段
- 設定 Amazon API Gateway Canary Release 部署
- 需要重新部署的 REST API 更新

在 API Gateway 中部署 REST API。

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

- 使用 AWS CLI 建立部署
- 從 API Gateway 主控台部署 REST API

使用 AWS CLI 建立部署

當您建立部署時，您可以執行個體化 Deployment 資源。您可以使用 API Gateway 主控台、AWS CLI、AWS 軟體開發套件或 API Gateway REST API 來建立部署。

若要使用 CLI 建立部署，請使用 create-deployment 命令：

```
aws apigateway create-deployment --rest-api-id <rest-api-id> --region <region>
```

在您將此部署與階段建立關聯前，都無法呼叫 API。若現已有階段，您可使用新建立的部署 ID (deploymentId) 更新階段的 <deployment-id> 屬性，來完成此操作。

```
aws apigateway update-stage --region <region> \
   --rest-api-id <rest-api-id> \
   --stage-name <stage-name> \
   --patch-operations op='replace',path='/deploymentId',value='<deployment-id>'
```

當您第一次部署 API 時，可以同時建立階段和部署：

```
aws apigateway create-deployment --region <region> \
   --rest-api-id <rest-api-id> \
   --stage-name <stage-name>
```

這就是當您第一次部署 API 或將 API 重新部署到新階段時，API Gateway 主控台後端完成的操作。

從 API Gateway 主控台部署 REST API

第一次部署 REST API 之前，您必須先進行建立。如需詳細資訊，請參閱 在 API Gateway 中開發 REST API。

主題

- 將 REST API 部署至階段
- 將 REST API 重新部署至階段
- 更新 REST API 部署的階段組態
- 設定 REST API 部署的階段變數
- 建立階段與 REST API 不同部署的關聯

將 REST API 部署至階段

API Gateway 主控台可讓您透過建立部署並將它與全新或現有階段建立關聯來部署 API。

> **Note**
>
> 若要將 API Gateway 中的階段與不同的部署建立關聯，請參閱建立階段與 REST API 不同部署的關聯。

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。

將 REST API 重新部署至階段

若要重新部署 API，請按照the section called “將 REST API 部署至階段”中的步驟執行。您可視需要重複使用相同階段，次數不限。

更新 REST API 部署的階段組態

部署 API 之後，您可以修改階段設定來啟用或停用 API 快取、記錄或請求調節。您也可以選擇後端的用戶端憑證來驗證 API Gateway，並設定階段變數以在執行時間將部署內容傳遞至 API 整合。如需更多詳細資訊，請參閱 更新階段設定。

> **Important**
>
> 修改階段設定之後，您必須重新部署 API，以讓變更生效。

> **Note**
>
> 如果已更新的設定 (例如啟用記錄功能) 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。

設定 REST API 部署的階段變數

針對部署，您可以設定或修改階段變數，以在執行時間將部署特定資料傳遞至 API 整合。您可以在 Stage Editor (階段編輯器) 的 Stage Variables (階段變數) 標籤上執行這項操作。如需詳細資訊，請參閱「設定 REST API 部署的階段變數」中的說明。

建立階段與 REST API 不同部署的關聯

因為部署代表 API 快照，而階段定義快照的路徑，所以您可以選擇不同的部署階段組合，以控制使用者如何呼叫不同版本的 API。例如，如果您想要將 API 狀態復原到先前的部署，或將 API 的「私有分支」合併至公有分支，則這十分有用。

下列程序示範如何在 API Gateway 主控台中使用 Stage Editor (階段編輯器) 來執行此操作。假設您必須部署 API 多次。

1. 如果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。

設定 REST API 的階段

階段是部署的具名參考，而這是 API 快照。您使用階段來管理和最佳化特定部署。例如，您可以設定階段設定來啟用快取、自訂請求調節、設定記錄、定義階段變數，或連線 Canary 版本以進行測試。

主題

- 使用 API Gateway 主控台設定階段
- 在 API Gateway 中為 API 階段設定標籤
- 設定 REST API 部署的階段變數

使用 API Gateway 主控台設定階段

主題

- 建立新的階段
- 更新階段設定
- 覆寫階段層級設定
- 刪除階段

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。

更新階段設定

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。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。

使用 API Gateway 主控台更新階段設定

這些步驟假設您已經將 API 部署到階段。

1. 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

   若要為階段啟用 API 快取，請開啟佈建 API 快取。然後配置默認方法級緩存，緩存容量，加密緩存數據，緩存 time-to-live （ TTL ），以及每個密鑰緩存失效的任何要求。

   除非您開啟預設的方法層級快取或針對特定方法開啟方法層級快取，快取才會處於作用中狀態。

   如需快取設定的詳細資訊，請參閱 啟用 API 快取以提升回應能力。

> **Note**
>
> 如果您為 API 階段啟用 API 快取，您的 AWS 帳戶可能需要支付 API 快取費用。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 設定和執行 REST API 的 API Gateway 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 設定和執行 REST API 的 API Gateway 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

若要建立 AWS WAF Web ACL 與階段的關聯，請從「Web ACL」下拉式清單中選取網頁 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. 選擇儲存。
9. 若要為與此 API Gateway API 這個階段相關聯的所有方法啟用 Amazon 日 CloudWatch 誌，請在日誌和追蹤區段中選擇編輯。

Note

若要啟用 CloudWatch 記錄，您還必須指定 IAM 角色的 ARN，讓 API Gateway 代表您的使用者將資訊寫入 CloudWatch 記錄。若要執行此作業，請從 APIs 主導覽窗格選擇 Settings (設定)。然後，針對CloudWatch 記錄角色，輸入 IAM 角色的 ARN。
對於常見的應用程式案例，IAM 角色可以連接 `AmazonAPIGatewayPushToCloudWatchLogs` 的受管政策，它包含下列存取政策陳述式：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:DescribeLogGroups",
                "logs:DescribeLogStreams",
                "logs:PutLogEvents",
                "logs:GetLogEvents",
                "logs:FilterLogEvents"
            ],
            "Resource": "*"
        }
    ]
}
```

IAM 角色也必須包含下列信任關係陳述式：

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

```
  ]
}
```

如需有關的詳細資訊 CloudWatch，請參閱 Amazon CloudWatch 使用者指南。

10. 從 [記錄CloudWatch 檔] 下拉式功能表中選取記錄層級。記錄層級如下：

    - 關閉 — 此階段未開啟記錄功能。
    - 僅限錯誤 — 僅針對錯誤啟用記錄功能。
    - 錯誤和資訊記錄 — 已啟用所有事件的記錄功能。
    - 完整的請求和回應記錄 — 已啟用所有事件的詳細記錄。這對於故障排除 API 很有用，但可能會導致記錄機密資料。

    > **Note**
    >
    > 我們建議您不要對生產 API 使用完整的請求和回應日誌。

11. 選取詳細指標，讓 API Gateway 報告 CloudWatch 顯示`API calls`、、`Latency`、`Integration latency`、`400 errors`、和的 API 指標`500 errors`。如需詳細資訊 CloudWatch，請參閱 Amazon CloudWatch 使用者指南中的基本監控和詳細監控。

    > **Important**
    >
    > 您的帳戶需支付存取方法層級 CloudWatch 指標的費用，但不會收取 API 層級或階段層級指標的費用。

12. 若要啟用目的地的存取記錄，請開啟自訂存取記錄。
13. 對於存取記錄目的地 ARN，請輸入記錄群組或 Firehose 串流的 ARN。

    「Firehose」的 ARN 格式為。`arn:aws:firehose:`*`{region}`*`:`*`{account-id}`*`:deliverystream/amazon-apigateway-`*`{your-stream-name}`*您的「Firehose」串流的名稱必須是`amazon-apigateway-`*`{your-stream-name}`*。
14. 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 “CloudWatch API Gateway 的記錄檔格式”。
15. 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 使用 X-Ray 追蹤使用者對 REST API 的請求。
16. 選擇 Save changes (儲存變更)。重新部署 API 以使新設定生效。

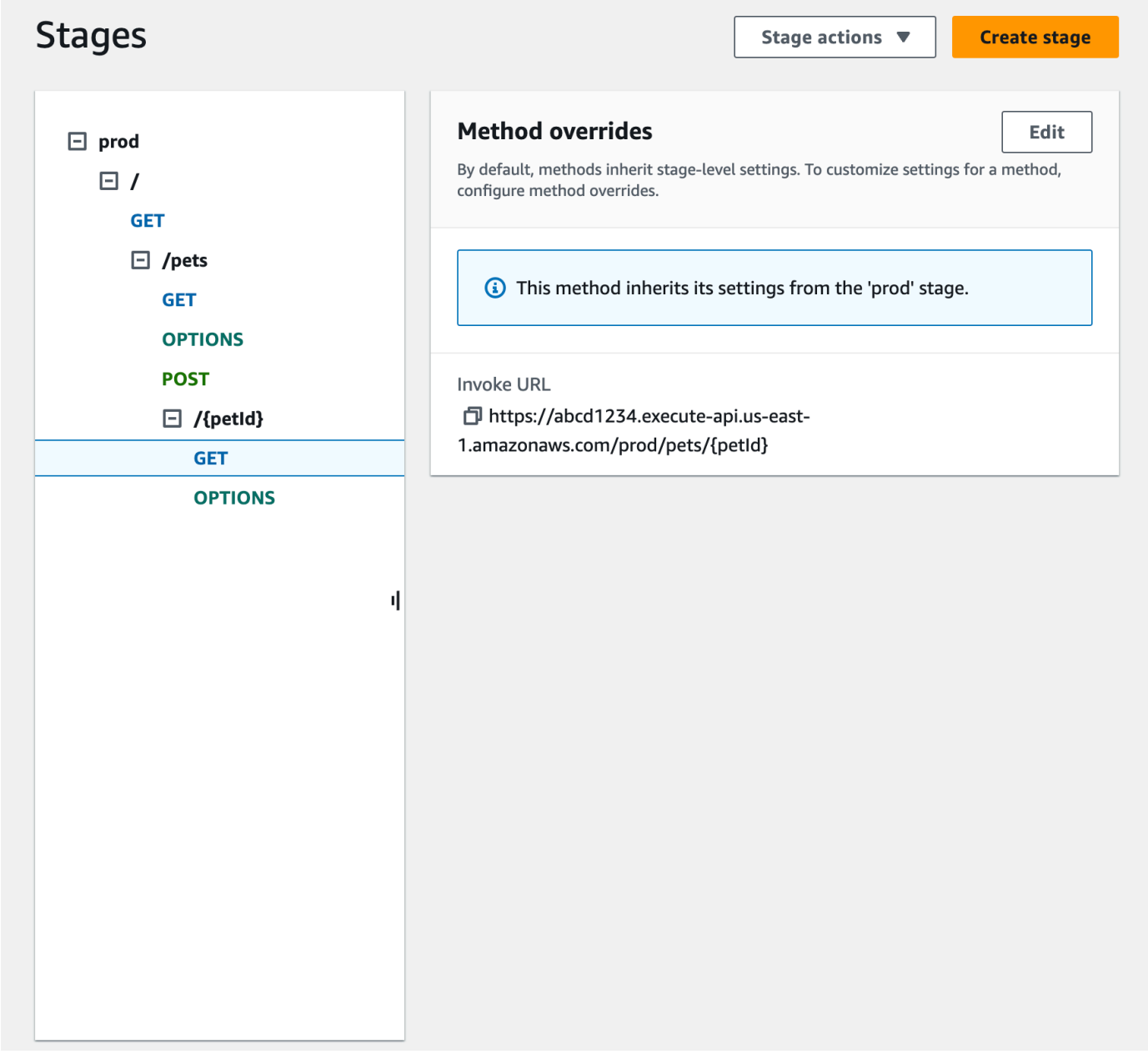
覆寫階段層級設定

您可以覆寫下列已啟用的階段層級設定。其中一些選項可能會向您收取額外費用 AWS 帳戶。

使用 API Gateway 主控台覆寫階段層級設定

使用 API Gateway 主控台覆寫階段層級設定

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，請針對記CloudWatch 錄選取記錄層級。

4. 若要開啟方法層級詳細指標，請選取詳細指標。您的帳戶需支付存取方法層級 CloudWatch 指標的費用，但不會收取 API 層級或階段層級指標的費用。
5. 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 “調節”。
6. 若要設定方法層級快取，請選取啟用方法快取。如果您變更「階段」詳細資料中的預設方法層級快取設定，則不會影響此設定。
7. 選擇儲存。

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。

使用 API Gateway 主控台刪除階段

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。若要這麼做，您可以使用 API Gateway 主控台、 AWS CLI/SDK 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
- 使用設定 API 階段的標籤 AWS CLI
- 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 Manage tags (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。

使用設定 API 階段的標籤 AWS CLI

您可以使用建立階段命令或標籤資 AWS CLI 源命令來設定 API 階段的標籤。您可以使用無標籤資源命令從 API 階段刪除一個或多個標籤。

下列範例會在建立`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
```

下列範例會將標籤新增至`prod`舞台：

```
aws apigateway tag-resource --resource-arn arn:aws:apigateway:us-east-2::/
restapis/abc123/stages/prod --tags Department=Sales
```

下列範例會從test舞台移除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"
    }
}
```

設定 REST API 部署的階段變數

階段變數是名稱/值對，您可以定義為與 REST API 部署階段相關聯的組態屬性。它們的作用如同環境變數，可用於 API 設定和對應範本。

例如，您可以在階段組態中定義階段變數，然後將其值設為您 REST API 方法之 HTTP 整合的 URL 字串。之後，您可以從 API 設定使用相關聯的階段變數名稱參考 URL 字串。如此一來，您可以將階段變數值重設為對應的 URL，在每個階段的不同端點使用相同的 API 設定。

您也可以存取對應範本中的階段變數，或將組態參數傳送到 AWS Lambda 或 HTTP 後端。

如需映射範本的詳細資訊，請參閱「API Gateway 映射範本和存取記錄變數參考」。

> **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給整合，請使用 AWS Lambda 授權者。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 授權者的輸出”。

使用案例

透過 API Gateway 中的部署階段，您可以管理每個 API 的多個發行階段，像是 alpha、beta 和生產。使用階段變數，您可以設定 API 部署階段與不同的後端端點互動。

例如，您的 API 可將 GET 請求當做 HTTP 代理傳送到後端 Web 主機 (例如，`http://example.com`)。在這種情況下，後端 Web 主機是以階段變數設定，所以當開發人員呼叫您的生產端點時，API Gateway 會呼叫 example.com。當您呼叫您的 beta 端點時，API Gateway 會在 beta 階段使用階段變數中設定的值，並呼叫不同的 Web 主機 (例如 `beta.example.com`)。同樣地，階段變數也可用來為 API 中的每個階段指定不同的 AWS Lambda 函數名稱。

您也可以使用階段變數，透過您的對應範本將組態參數傳送到 Lambda 函數。例如，您可能希望在您 API 的多個階段中重複使用相同的 Lambda 函數，但此函數應該從不同的 Amazon DynamoDB 資料表讀取資料，視呼叫的階段而定。在產生 Lambda 函數請求的對應範本中，您可以使用階段變數將資料表名稱傳送給 Lambda。

範例

若要使用階段變數來自訂 HTTP 整合端點，您必須先設定指定名稱的階段變數 (例如 **`url`**)，然後為它指派值 (例如 **`example.com`**)。之後，從您的方法組態設定一個 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 執行的階段而定。

您可以使用類似的方式參考階段變數，在認證欄位中指定 Lambda 函數名稱、 AWS 服務代理路徑或 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。當您在 API Gateway 主控台中指定 Lambda 函數時，會彈出一個指 AWS CLI 令以設定適當的權限。您也可以使用 AWS Command Line Interface （ AWS CLI ）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 Amazon API Gateway 主控台設定階段變數

在此教學課程中，您會了解如何使用 Amazon API Gateway 主控台，為範例 API 的兩個部署階段設定階段變數。開始之前，請務必先達成以下必要條件：

- 您必須擁有 API Gateway 中可用的 API。請遵循中的說明進行在 API 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 Amazon 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

使用 API Gateway 主控台宣告階段變數

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. 針對名稱，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：

   針對名稱，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：

   針對名稱，輸入 **`function`**。針對值，輸入 **`HelloWorld`**。

   > **Note**
   >
   > 將 Lambda 函數設定為階段變數值時，請使用函數的本機名稱，這可能會包括其別名或版本規格，如 **`HelloWorld`**、**`HelloWorld:1`** 或 **`HelloWorld:alpha`**。請勿使用函數的 ARN (例如，**`arn:aws:lambda:us-east-1:123456789012:function:HelloWorld`**)。API Gateway 主控台假設 Lambda 函數的階段變數值為不完整的函數名稱，會將指定的階段變數擴充為 ARN。
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`**)。

若要了解如何使用階段變數，請參閱 the section called “使用階段變數 ”。

使用 Amazon API Gateway 階段變數

您可以使用 API Gateway 階段變數來存取不同 API 部署階段的 HTTP 和 Lambda 後端。您可以使用階段變數，將階段特定組態中繼資料傳送到 HTTP 後端作為查詢參數，以及傳送到 Lambda 函數作為輸入對應範本中產生的承載。

必要條件

您必須建立兩個階段，並將 url 階段變數設定為兩個不同的 HTTP 端點：將 function 階段變數指派給兩個不同 Lambda 函數，以及包含階段特定中繼資料的 stageName 階段變數。

透過有階段變數的 API 存取 HTTP 端點

1. 在 Stages (階段) 導覽窗格中，選擇 beta。在階段詳細資訊下，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。

2. 您從 beta 階段 GET 請求取得的回應，如旁所示。您也可以使用瀏覽器導覽至 http://httpbin.org/get 驗證結果。此值已在 beta 階段指派給 `url` 變數。兩個回應完全相同。
3. 在 Stages (階段) 導覽窗格中，選擇 prod 階段。在階段詳細資訊下，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。
4. 您從 prod 階段 GET 請求取得的回應，如旁所示。您可以通過使用瀏覽器導航到驗證結果 HTTP：//petstore-demo-endpoint執行-api.com/寵物店/寵物。此值已在 prod 階段指派給 `url` 變數。兩個回應完全相同。

在查詢參數表達式中透過階段變數將階段特定中繼資料傳送到 HTTP 後端

此程序說明如何使用查詢參數表達式中的階段變數值，將階段特定中繼資料傳送到 HTTP 後端。我們會使用在 `stageName` 中宣告的 使用 Amazon API Gateway 主控台設定階段變數 階段變數。

1. 在 Resource (資源) 導覽窗格中選擇 GET 方法。

   若要將查詢字串參數新增至方法的 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. 針對名稱，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 使用階段變數呼叫 Lambda 函數

此程序說明如何使用階段變數呼叫 Lambda 函數做為 API 的後端。我們會使用之前宣告的 `function` 階段變數。如需詳細資訊，請參閱 使用 Amazon API Gateway 主控台設定階段變數。

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]。

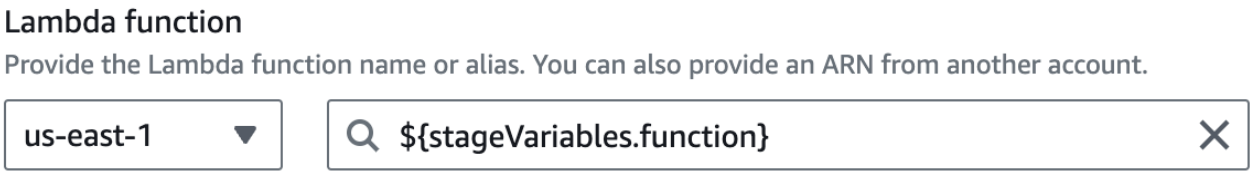
   然後，執行下列動作：

   a. 針對方法類型，選取 GET。

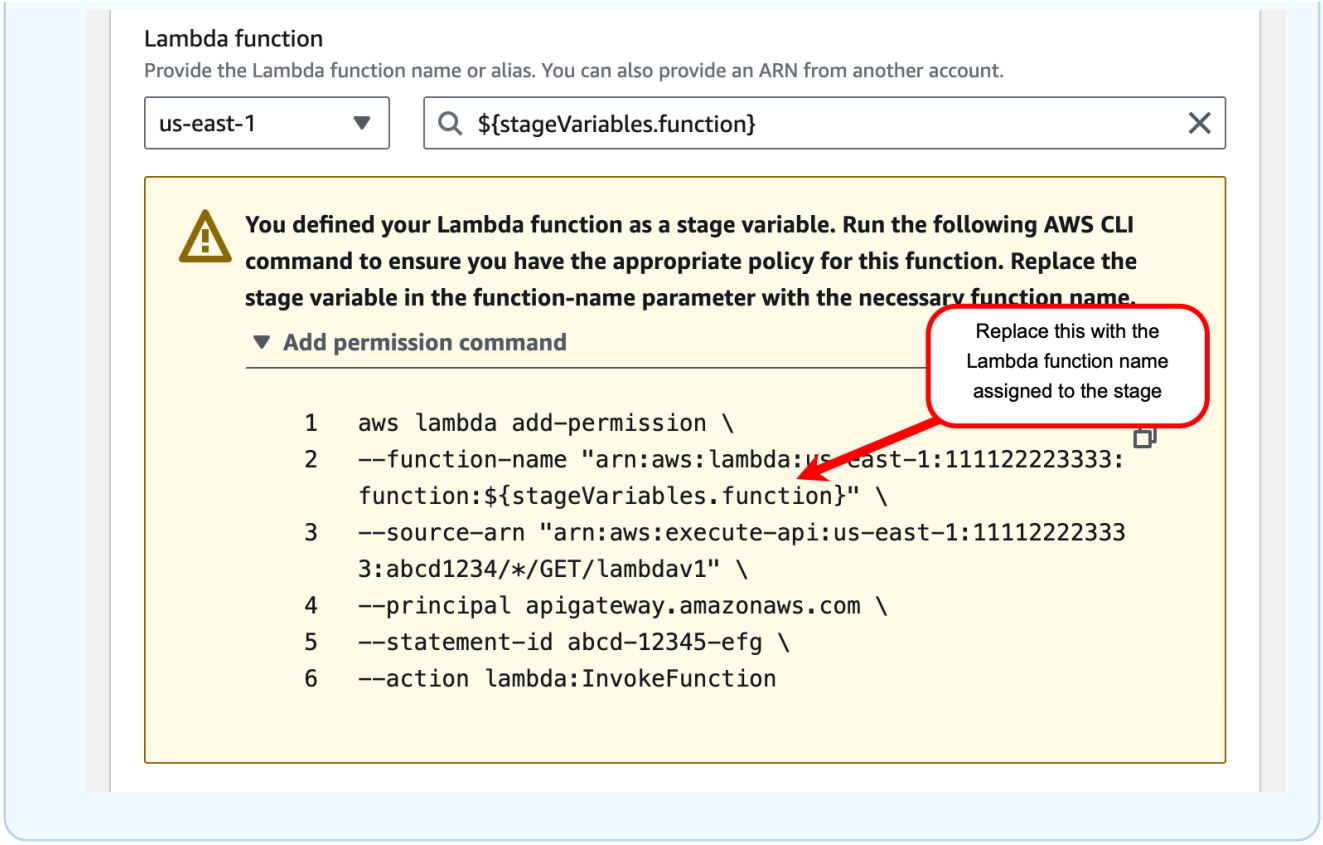
   b. 針對整合類型，選取 Lambda 函數。

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**
>
> 出現輸入新增權限命令的提示時，複製 AWS CLI 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`$stageVariables.function`值為`HelloWorld`，請執行下列 AWS CLI 命令：
>
> ```
> aws lambda add-permission --function-name arn:aws:lambda:us-
> east-1:account-id:function:HelloWorld --source-arn arn:aws:execute-
> api:us-east-1:account-id:api-id/*/GET/lambdav1 --principal
>  apigateway.amazonaws.com --statement-id statement-id-guid --action
>  lambda:InvokeFunction
> ```
>
> 如果不執行此作業，會在呼叫方法時造成 `500 Internal Server Error` 回應。將 `${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。

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. 出現輸入新增權限命令的提示時，複製 AWS CLI 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."
```

Amazon API Gateway 階段變數參考

在下列情況下，您可以使用 API Gateway 階段變數。

參數對應表達式

階段變數可用在 API 方法請求或回應標頭參數的參數映射表達式中，沒有任何部分替換。在下列範例中，階段變數參考不使用 $ 與包圍的 {...}。

- `stageVariables.<variable_name>`

對應範本

階段變數可以用在對應範本的任何位置，如下例所示。

- { "name" : "$stageVariables.<variable_name>"}
- { "name" : "${stageVariables.<variable_name>}"}

HTTP 整合 URI

階段變數可用作 HTTP 整合 URL 的一部分，如下例所示。

- 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 URI

階段變數可用來做為 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 URI (Lambda 函數)

階段變數可用來取代 Lambda 函數名稱或版本/別名，如下例所示。

- arn:aws:apigateway:<region>:lambda:path/2015-03-31/functions/arn:aws:lambda:<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授權者的 Amazon Cognito 使用者集區。

- `arn:aws:cognito-idp:<region>:<account_id>:userpool/${stageVariables.<variable_name>}`

AWS 整合認證

階段變 AWS 數可以作為使用者/角色認證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 軟體開發套件，在 API Gateway 中建立 Canary 版本部署。

主題

- 使用 API Gateway 主控台建立 Canary 部署
- 使用 AWS CLI 建立 Canary 部署

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 網頁 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。如需更多詳細資訊，請參閱 在 API Gateway 中設定 REST API 的 CloudWatch 記錄。
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。如需更多詳細資訊，請參閱 the section called “設定階段變數”。
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 AWS CLI 建立 Canary 部署

先建立具有兩個階段變數的基準部署，但不需要任何 Canary：

```
aws apigateway create-deployment \
    --variables sv0=val0,sv1=val1 \
    --rest-api-id abcd1234 \
    --stage-name 'prod' \
```

此命令會傳回所產生 Deployment 的呈現，與下列類似：

```
{
    "id": "du4ot1",
    "createdDate": 1511379050
}
```

產生的部署 id 識別 API 的快照 (或版本)。

現在在 prod 階段上建立 Canary 部署：

```
aws apigateway create-deployment  --rest-api-id abcd1234 \
    --canary-settings \
    '{
        "percentTraffic":10.5,
        "useStageCache":false,
        "stageVariableOverrides":{
            "sv1":"val2",
            "sv2":"val3"
        }
    }' \
    --stage-name 'prod'
```

如果指定的階段 (prod) 不存在，上述命令會傳回錯誤。否則，它會傳回新建的部署資源呈現，與下列類似：

```
{
    "id": "a6rox0",
    "createdDate": 1511379433
}
```

產生的部署 id 識別 Canary Release 的 API 測試版本。因此，相關聯的階段是已啟用 Canary。您可以呼叫 get-stage 命令來檢視此階段呈現，與下列類似：

```
aws apigateway get-stage --rest-api-id 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。

您可以更新階段以啟用 Canary，以從現有一般部署建立 Canary Release 部署。為了示範這項操作，請先建立一般部署：

```
aws apigateway create-deployment \
    --variables sv0=val0,sv1=val1 \
    --rest-api-id abcd1234 \
    --stage-name 'beta'
```

此命令會傳回基本版本部署呈現：

```
{
    "id": "cifeiw",
    "createdDate": 1511380879
}
```

相關聯的 Beta 階段沒有任何 Canary 設定：

```
{
    "stageName": "beta",
    "variables": {
        "sv0": "val0",
        "sv1": "val1"
    },
    "cacheClusterEnabled": false,
```

```
    "cacheClusterStatus": "NOT_AVAILABLE",
    "deploymentId": "cifeiw",
    "lastUpdatedDate": 1511380879,
    "createdDate": 1511380879,
    "methodSettings": {}
}
```

現在，在階段上連接 Canary，以建立新的 Canary Release 部署：

```
aws apigateway update-stage \
    --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"
        }]'
```

更新過的階段呈現如下：

```
{
    "stageName": "beta",
    "variables": {
        "sv0": "val0",
        "sv1": "val1"
    },
    "cacheClusterEnabled": false,
    "cacheClusterStatus": "NOT_AVAILABLE",
    "deploymentId": "cifeiw",
    "lastUpdatedDate": 1511381930,
    "createdDate": 1511380879,
    "canarySettings": {
        "percentTraffic": 10.5,
        "deploymentId": "cifeiw",
        "useStageCache": false,
```

```
        "stageVariableOverrides": {
            "sv2": "val3",
            "sv1": "val2"
        }
    },
    "methodSettings": {}
}
```

因為我們只會在現有 API 版本上啟用 Canary，所以生產版本 (`Stage`) 和 Canary Release (`canarySettings`) 指向相同的部署，即 API 的相同版本 (`deploymentId`)。在您變更 API 並將它再次部署至此階段之後，新的版本將會在 Canary Release 中，而基本版本仍然在生產版本中。Canary Release 中的 `deploymentId` 更新為新部署 `id` 而生產版本中的 `deploymentId` 保持不變時，這是在階段發展中體現。

更新 Canary Release

部署 Canary Release 之後，建議您調整 Canary 流量百分比，或者啟用或停用階段快取來最佳化測試效能。更新執行內容時，您也可以修改 Canary Release 中所使用的階段變數。若要進行這類更新，請呼叫 canarySettings 上具有新值的 stage:update 操作。

您可以使用 API Gateway 主控台、AWS CLI update-stage 命令或 AWS 軟體開發套件來更新 Canary 版本。

主題

- 使用 API Gateway 主控台更新 Canary Release
- 使用 AWS CLI 更新 Canary Release

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 Save (儲存)。

使用 AWS CLI 更新 Canary Release

若要使用 AWS CLI 更新 Canary，請呼叫 update-stage 命令。

若要啟用或停用 Canary 的階段快取，請呼叫 update-stage 命令，如下所示：

```
aws apigateway update-stage \
    --rest-api-id {rest-api-id} \
    --stage-name '{stage-name}'  \
    --patch-operations op=replace,path=/canarySettings/useStageCache,value=true
```

若要調整 Canary 流量百分比，請呼叫 update-stage 來取代階段上的 /canarySettings/percentTraffic 值。

```
aws apigateway update-stage \
    --rest-api-id {rest-api-id} \
    --stage-name '{stage-name}' \
    --patch-operations op=replace,path=/canarySettings/percentTraffic,value=25.0
```

更新 Canary 階段變數，包含新增、取代或移除 Canary 階段變數：

```
aws apigateway update-stage  \
    --rest-api-id {rest-api-id} \
    --stage-name '{stage-name}'  \
    --patch-operations '[{
        "op": "replace",
        "path": "/canarySettings/stageVariableOverrides/newVar",
        "value": "newVal"
      }, {
        "op": "replace",
        "path": "/canarySettings/stageVariableOverrides/var2",
        "value": "val4"
      }, {
        "op": "remove",
        "path": "/canarySettings/stageVariableOverrides/var1"
      }]'
```

您可以將操作結合為單一 patch-operations 值，來更新上述所有項目：

```
aws apigateway update-stage  \
    --rest-api-id {rest-api-id} \
    --stage-name '{stage-name}' \
    --patch-operations '[{
        "op": "replace",
        "path": "/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

若要提升 Canary Release，請讓進行測試的 API 版本可在生產階段中使用。此操作包含下列任務：

- 重設具有 Canary 部署 ID 設定之階段的部署 ID。這會更新具有 Canary 快照之階段的 API 快照，同時讓測試版本成為生產版本。
- 使用 Canary 階段變數更新階段變數 (如果有的話)。這會更新具有 Canary API 執行內容之階段的 API 執行內容。沒有這項更新時，如果測試版本使用不同的階段變數或不同的現有階段變數值，則新的 API 版本可能會產生意外的結果。
- 將 Canary 流量百分比設定為 0.0%。

提升 Canary Release 並不會停用階段上的 Canary。若要停用 Canary，您必須移除階段上的 Canary 設定。

主題

- 使用 API Gateway 主控台提升 Canary Release

- 使用 AWS CLI 提升 Canary Release

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 AWS CLI 進行驗證。如需範例，請參閱 the section called “使用 AWS CLI 提升 Canary Release”。

使用 AWS CLI 提升 Canary Release

若要使用 AWS CLI 命令將 Canary Release 提升為生產版本，請呼叫 `update-stage` 命令將 Canary 相關聯的 `deploymentId` 複製至與階段相關聯的 `deploymentId`、將 Canary 流量百分比重設為零 (`0.0`)，以及將任何 Canary 繫結階段變數複製至對應的階段繫結階段變數。

假設我們有 Canary Release 部署，如與以下類似的階段所描述：

```
{
    "_links": {
        ...
    },
    "accessLogSettings": {
        ...
    },
    "cacheClusterEnabled": false,
    "cacheClusterStatus": "NOT_AVAILABLE",
    "canarySettings": {
        "deploymentId": "eh1sby",
        "useStageCache": false,
        "stageVariableOverrides": {
            "sv2": "val3",
            "sv1": "val2"
```

```
        },
        "percentTraffic": 10.5
    },
    "createdDate": "2017-11-20T04:42:19Z",
    "deploymentId": "nfcn0x",
    "lastUpdatedDate": "2017-11-22T00:54:28Z",
    "methodSettings": {
        ...
    },
    "stageName": "prod",
    "variables": {
        "sv1": "val1"
    }
}
```

我們呼叫下列 update-stage 請求將其提升：

```
aws apigateway update-stage  \
    --rest-api-id {rest-api-id}  \
    --stage-name '{stage-name}'  \
    --patch-operations '[{
        "op": "replace",
        "value": "0.0",
        "path": "/canarySettings/percentTraffic"
      }, {
        "op": "copy",
        "from": "/canarySettings/stageVariableOverrides",
        "path": "/variables"
      }, {
        "op": "copy",
        "from": "/canarySettings/deploymentId",
        "path": "/deploymentId"
      }]'
```

提升後，該階段現如下所示：

```
{
    "_links": {
        ...
    },
    "accessLogSettings": {
        ...
    },
```

```
    "cacheClusterEnabled": false,
    "cacheClusterStatus": "NOT_AVAILABLE",
    "canarySettings": {
        "deploymentId": "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"
    }
}
```

如您所見，將 Canary Release 提升到該階段不會停用 Canary，而部署仍然是 Canary Release 部署。為了讓該部署成為一般生產版本部署，您必須停用 Canary 設定。如需有關如何停用 Canary Release 部署的詳細資訊，請參閱 the section called “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 軟體開發套件來停用 Canary 版本部署。

主題

- 使用 API Gateway 主控台關閉 Canary 版本
- 使用 AWS CLI 關閉 Canary 版本

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 Delete (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 AWS CLI 進行驗證。如需範例，請參閱 the section called “使用 AWS CLI 關閉 Canary 版本”。

使用 AWS CLI 關閉 Canary 版本

若要使用 AWS CLI 關閉 Canary 版本部署，請呼叫 `update-stage` 命令，如下所示：

```
aws apigateway update-stage \
    --rest-api-id abcd1234 \
    --stage-name canary \
    --patch-operations '[{"op":"remove", "path":"/canarySettings"}]'
```

成功回應會傳回與下列類似的承載：

```
{
    "stageName": "prod",
    "accessLogSettings": {
        ...
    },
    "cacheClusterEnabled": false,
    "cacheClusterStatus": "NOT_AVAILABLE",
    "deploymentId": "nfcn0x",
    "lastUpdatedDate": 1511309280,
    "createdDate": 1511152939,
    "methodSettings": {
        ...
    }
}
```

如輸出中所示，`canarySettings` 屬性於已停用 Canary 部署的階段中不再存在。

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 或 API Gateway REST API 來維護 API。更新 API 需要修改 API 的特定資源屬性或組態設定。資源更新必須重新部署 API，組態更新則不必。

下表詳列了可以更新的 API 資源。

API 資源更新需要重新部署 API

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。
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。
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。

資源	備註
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。
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。

下表詳列了可以更新的 API 組態。

API 組態更新無須重新部署 API

組態	備註
帳戶	關於適用的屬性和支援的操作，請參閱 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。
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。
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。

設定 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`api-id`中由 API Gateway 產生，`region`(AWS 區域) 由您在建立 API 時指定，並`stage`由您在部署 API 時指定。

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 端點很難取回，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 對應至您的 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**
>
> 區域性自訂網域可與 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。
> 私有 API 並不支援自訂網域名稱。
> 您可以選擇 REST API 支援的最低 TLS 版本。如果是 REST API，您可選擇 TLS 1.2 版或 TLS 1.0 版。

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。您的網域名稱必須遵循 RFC 1035 規範，且每個標籤最多可有 63 個八位元組，總共 255 個八位元組。需要時，您可以使用 Amazon Route 53 或使用您選擇的第三方網域註冊機構來註冊網際網路網域。API 的自訂網域名稱可以是已註冊網際網路網域的子網域或根網域 (也稱為 "zone apex") 的名稱。

在 API Gateway 中建立自訂網域名稱，您必須建立或更新 DNS 提供者的資源記錄，以對應至您的 API 端點。如果沒有這種對應，送往自訂網域名稱的 API 請求無法到達 API Gateway。

> **Note**
>
> 自訂網域名稱在所有 AWS 帳戶的區域中必須是唯一的。
> 若要在區域或 AWS 帳戶之間移動邊緣最佳化的自訂網域名稱，您必須刪除現有的 CloudFront 分發，然後建立新的分發。此程序可能需要大約 30 分鐘才能使用新的自訂網域名稱。如需詳細資訊，請參閱更新 CloudFront 分佈。

邊緣最佳化的自訂網域名稱

當您部署邊緣最佳化 API 時，API Gateway 會設定 Amazon CloudFront 分發和 DNS 記錄，以將 API 網域名稱對應至 CloudFront 分發網域名稱。接著，API 的要求會透過對應的 CloudFront 分發路由至 API Gateway。

當您為邊緣最佳化 API 建立自訂網域名稱時，API Gateway 會設定分發 CloudFront。但是您必須設定 DNS 記錄，才能將自訂網域名稱對應至 CloudFront發佈網域名稱。此對應適用於綁定自訂網域名稱的 API 請求，以透過對應的 CloudFront 散佈路由至 API Gateway。您也必須提供自訂網域名稱的憑證。

> **Note**
>
> API Gateway 建立的 CloudFront 散佈是由與 API Gateway 關聯的區域特定帳戶所擁有。追蹤作業以在 CloudWatch 記錄檔中建立和更新此類 CloudFront散發時，您必須使用此 API Gateway 帳戶 ID。如需詳細資訊，請參閱 記錄自定義域名創建 CloudTrail。

若要設定邊緣最佳化的自訂網域名稱或更新其憑證，您必須擁有更新 CloudFront 發佈的權限。

若要提供存取權，請新增權限至您的使用者、群組或角色：

- 使用者和群組位於 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。請按照 IAM 使用者指南 的 為第三方身分提供者 (聯合) 建立角色 中的指示進行操作。
- 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 中的指示進行操作。

- (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。

更新 CloudFront 發行版需要下列權限。

```
{
    "Version": "2012-10-17",
    "Statement": [
         {
            "Sid": "AllowCloudFrontUpdateDistribution",
            "Effect": "Allow",
            "Action": [
                "cloudfront:updateDistribution"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

API Gateway 透過在散發上利用伺服器名稱指示 (SNI)，來支援邊緣最佳化的自訂網域名稱。CloudFront 如需有關在 CloudFront分發上使用自訂網域名稱的詳細資訊，包括所需的憑證格式和憑證金鑰長度的大小上限，請參閱 Amazon CloudFront 開發人員指南中的使用替代網域名稱和 HTTPS。

若要設定自訂網域名稱作為 API 的主機名稱，您身為 API 擁有者所以必須提供自訂網域名稱的 SSL/TLS 憑證。

若要提供邊緣最佳化自訂網域名稱的憑證，您可以請求 AWS Certificate Manager (ACM) 在 ACM 中產生新的憑證，或將 `us-east-1` 區域 (美國東部 (維吉尼亞北部)) 中第三方憑證授權機構所發出的憑證匯入 ACM。

區域性自訂網域名稱

當您建立區域 API 的自訂網域名稱時，API Gateway 會建立 API 的區域網域名稱。您必須設定 DNS 記錄，將自訂網域名稱對應至區域性網域名稱。您也必須提供自訂網域名稱的憑證。

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，在單一 AWS 帳戶中，您可以設定`*.example.com`和行`a.example.com`為不同。

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 API Gateway 映射範本和存取記錄變數參考。

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 帳戶建立了與萬用字元自訂網域名稱衝突的自訂網域名稱，您就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。

自訂網域名稱的憑證

> **Important**
>
> 您可以指定自訂網域名稱所用的憑證。如果您的應用程式使用憑證釘選 (有時稱為 SSL 釘選) 來釘選 ACM 憑證，則應用程式在續訂憑證後 AWS 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。

若要在支援 ACM 的區域中提供自訂網域名稱的憑證，您必須向 ACM 請求憑證。若要在不支援 ACM 的區域中提供區域自訂網域名稱的憑證，您必須將憑證匯入至該區域中的 API Gateway。

若要匯入 SSL/TLS 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。若要將受 AWS 管理憑證用於網域名稱，您只需參考其 ARN 即可。

ACM 可讓您直覺式地設定和使用 API 的自訂網域名稱。您可以建立所指定網域名稱的憑證 (或匯入憑證)、使用 ACM 所提供的憑證 ARN 在 API Gateway 中設定網域名稱，以及將自訂網域名稱下的基本路徑對應至 API 的已部署階段。使用 ACM 所發出的憑證，就不需要擔心公開任何敏感的憑證詳細資訊，例如私有金鑰。

主題

- 在 中備妥憑證 AWS Certificate Manager
- 在 API Gateway 中為您的自訂網域選擇安全性原則
- 建立邊緣最佳化自訂網域名稱
- 在 API Gateway 中設定區域性自訂網域名稱
- 將自訂網域名稱遷移至不同的 API 端點
- 使用適用於 REST API 的 API 映射
- 停用 REST API 的預設端點
- 為了控制 DNS 備援功能，請設定自訂運作狀態檢查

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下列步驟說明如何完成這項操作。如需詳細資訊，請參閱《AWS Certificate Manager 使用者指南》。

> **Note**
>
> 若要使用 ACM 憑證搭配 API Gateway 邊緣最佳化自訂網域名稱，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。針對 API Gateway 區域自訂網域名稱，您必須在與 API 相同的區域中請求或匯入憑證。憑證必須由公開信任的憑證授權機構簽署，並涵蓋自訂網域名稱。

首先，註冊您的網際網路網域，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。

若要在 ACM 中建立網域名稱的 SSL/TLS 憑證或將其匯入至 ACM，請執行下列其中一項操作：

請求 ACM 針對網域名稱所提供的憑證

1. 登入 AWS Certificate Manager 主控台。

2. 選擇 Request a certificate (請求憑證)。
3. 在 Domain name (網域名稱) 中，輸入 API 的自訂網域名稱 (例如 `api.example.com`)。
4. (選擇性) 選擇 Add another name to this certificate (將其他名稱新增至此憑證)。
5. 選擇 Review and request (檢閱和請求)。
6. 選擇 Confirm and request (確認和請求)。
7. 針對有效的請求，在 ACM 發出憑證之前，網際網路網域的註冊擁有者必須先同意請求。

將網域名稱的憑證匯入至 ACM

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 的局部清單，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
   ```

   > **Note**
   >
   > Amazon API Gateway 利用 Amazon CloudFront 來支援自訂網域名稱的憑證。因此，自訂網域名稱 SSL/TLS 憑證的需求和條件約束是由。CloudFront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。

   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. 將憑證上傳至 AWS Certificate Manager：

   a. 登入 AWS Certificate Manager 主控台。

   b. 選擇 Import a certificate (匯入憑證)。

   c. 針對 Certificate body (憑證內文)，輸入或貼上您憑證授權機構中的 PEM 格式化伺服器憑證內文。以下示範這類憑證的縮寫範例。

```
-----BEGIN CERTIFICATE-----
EXAMPLECA+KgAwIBAgIQJ1XxJ8Pl++gOfQtj0IBoqDANBgkqhkiG9w0BAQUFADBB
...
az8Cg1aicxLBQ7EaWIhhgEXAMPLE
-----END CERTIFICATE-----
```

   d. 針對 Certificate private key (憑證私有金鑰)，輸入或貼上您 PEM 格式化憑證的私有金鑰。以下示範這類金鑰的縮寫範例。

```
-----BEGIN RSA PRIVATE KEY-----
EXAMPLEBAAKCAQEA2Qb3LDHD7StY7Wj6U2/opV6Xu37qUCCkeDWhwpZMYJ9/nETO
...
1qGvJ3u04vdnzaYN5WoyN5LFckrlA71+CszD1CGSqbVDWEXAMPLE
-----END RSA PRIVATE KEY-----
```

   e. 針對 Certificate chain (憑證鏈)，輸入或貼上 PEM 格式化中繼憑證，以及選擇性地逐一輸入或貼上根憑證，不能有任何空白行。如果您包含根憑證，則憑證鏈的開頭必須是中繼憑證，而結尾必須是根憑證。使用憑證授權機構所提供的中繼憑證。請不要包含不在信任路徑鏈中的任何中介。以下示範縮寫範例。

```
-----BEGIN CERTIFICATE-----
EXAMPLECA4ugAwIBAgIQWrYdrB5NogYUx1U9Pamy3DANBgkqhkiG9w0BAQUFADCB
...
8/ifBlIK3se2e4/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 Review and import (檢閱和匯入)。

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 中為您的自訂網域選擇安全性原則

為了提高 Amazon API Gateway 自訂網域的安全性，您可以在 API Gateway 主控台 AWS CLI、或 AWS 開發套件中選擇安全政策。

安全策略是 API Gateway 提供的最低 TLS 版本和密碼套件的預先定義組合。您可以選擇 TLS 1.2 版或 TLS 1.0 版安全政策。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。

在自訂網域設定中，安全政策會判斷以下兩個設定：

- API Gateway 用來與 API 用戶端通訊的最低 TLS 版本
- API Gateway 用來加密要傳回 API 用戶端內容的密碼

如果您選擇 TLS 1.0 安全性原則，安全性原則會接受 TLS 1.0、TLS 1.2 和 TLS 1.3 流量。如果您選擇 TLS 1.2 安全性原則，安全性原則會接受 TLS 1.2 和 TLS 1.3 流量，並拒絕 TLS 1.0 流量。

> **Note**
>
> 您只能指定自訂網域的安全性原則。對於使用預設端點的 API，API Gateway 會使用下列安全性原則：
>
> - 針對邊緣最佳化 API：`TLS-1-0`
> - 對於區域性 API：`TLS-1-0`
> - 對於私有 API：`TLS-1-2`

主題

- 如何為自訂網域指定安全性原則
- 針對邊緣最佳化自訂網域所支援的安全性原則、TLS 通訊協定版本和加密
- 地區自訂網域支援的安全性原則、TLS 通訊協定版本和密碼
- 私有 API 支援的 TLS 通訊協定版本和密碼
- OpenSSL 和 RFC 密碼名稱
- 有關 HTTP API 和 WebSocket API 的資訊

如何為自訂網域指定安全性原則

當您建立自訂網域名稱時，您可以為其指定安全性原則。若要瞭解如何建立自訂網域，請參閱the section called “建立邊緣最佳化自訂網域名稱”或the section called “設定區域性自訂網域名稱”。

若要變更自訂網域名稱的安全性原則，請更新自訂網域設定。您可以使用 AWS Management Console、或 AWS SDK 更新您的自訂網域名稱設定。 AWS CLI

當您使用 API Gateway REST API 時 AWS CLI，或指定新的 TLS 版本，`TLS_1_0`或`TLS_1_2`在`securityPolicy`參數中指定。如需詳細資訊，請參閱 Amazon API 閘道 REST API 參考或參考資料update-domain-name中的網域名稱:更新。AWS CLI

更新作業可能需要幾分鐘才能完成。

針對邊緣最佳化自訂網域所支援的安全性原則、TLS 通訊協定版本和加密

下表說明可針對邊緣最佳化自訂網域名稱指定的安全性原則。

安全政策	TLS_1_0	TLS_1
TLS 通訊協定		
TLSv1.3	◆	◆
TLSv1.2	◆	◆
TLSv1.1	◆	
TLSv1	◆	
TLS 密碼		

安全政策	TLS_1_0	TLS_1
TLS_AES_128_GCM_SHA256	♦	♦
TLS_AES_256_GCM_SHA384	♦	♦
TLS_CHACHA20_POLY1305_SHA256	♦	♦
ECDHE-ECDSA-AES128-GCM-SHA256	♦	♦
ECDHE-ECDSA-AES128-SHA256	♦	♦
ECDHE-ECDSA-AES128-SHA	♦	
ECDHE-ECDSA-AES256-GCM-SHA384	♦	♦
ECDHE-ECDSA-CHACHA20-POLY1305	♦	♦
ECDHE-ECDSA-AES256-SHA384	♦	♦
ECDHE-ECDSA-AES256-SHA	♦	
ECDHE-RSA-AES128-GCM-SHA256	♦	♦
ECDHE-RSA-AES128-SHA256	♦	♦
ECDHE-RSA-AES128-SHA	♦	
ECDHE-RSA-AES256-GCM-SHA384	♦	♦
ECDHE-RSA-CHACHA20-POLY1305	♦	♦

安全政策	TLS_1_0	TLS_1
ECDHE-RSA-AES256-SHA384	♦	♦
ECDHE-RSA-AES256-SHA	♦	
AES128-GCM-SHA256	♦	
AES256-GCM-SHA384	♦	♦
AES128-SHA256	♦	♦
AES256-SHA	♦	
AES128-SHA	♦	
DES-CBC3-SHA	♦	

地區自訂網域支援的安全性原則、TLS 通訊協定版本和密碼

下表說明可以為地區自訂網域名稱指定的安全性原則。

安全政策	TLS_1_0	TLS_1
TLS 通訊協定		
TLSv1.3	♦	♦
TLSv1.2	♦	♦
TLSv1.1	♦	
TLSv1	♦	
TLS 密碼		
TLS_AES_128_GCM_SHA256	♦	♦
TLS_AES_256_GCM_SHA384	♦	♦

安全政策	TLS_1_0	TLS_1
TLS_CHACHA20_POLY1305_SHA256	♦	♦
ECDHE-ECDSA-AES128-GCM-SHA256	♦	♦
ECDHE-RSA-AES128-GCM-SHA256	♦	♦
ECDHE-ECDSA-AES128-SHA256	♦	♦
ECDHE-RSA-AES128-SHA256	♦	♦
ECDHE-ECDSA-AES128-SHA	♦	
ECDHE-RSA-AES128-SHA	♦	
ECDHE-ECDSA-AES256-GCM-SHA384	♦	♦
ECDHE-RSA-AES256-GCM-SHA384	♦	♦
ECDHE-ECDSA-AES256-SHA384	♦	♦
ECDHE-RSA-AES256-SHA384	♦	♦
ECDHE-RSA-AES256-SHA	♦	
ECDHE-ECDSA-AES256-SHA	♦	
AES128-GCM-SHA256	♦	♦
AES128-SHA256	♦	♦

安全政策	TLS_1_0	TLS_1
AES128-SHA	♦	
AES256-GCM-SHA384	♦	♦
AES256-SHA256	♦	♦
AES256-SHA	♦	

私有 API 支援的 TLS 通訊協定版本和密碼

下表說明私有 API 支援的 TLS 通訊協定和密碼。不支援為私有 API 指定安全性原則。

安全政策	TLS_1
TLS 通訊協定	
TLSv1.2	♦
TLS 密碼	
ECDHE-ECDSA-AES128-GCM-SHA256	♦
ECDHE-RSA-AES128-GCM-SHA256	♦
ECDHE-ECDSA-AES128-SHA256	♦
ECDHE-RSA-AES128-SHA256	♦
ECDHE-ECDSA-AES256-GCM-SHA384	♦
ECDHE-RSA-AES256-GCM-SHA384	♦
ECDHE-ECDSA-AES256-SHA384	♦
ECDHE-RSA-AES256-SHA384	♦
AES128-GCM-SHA256	♦
AES128-SHA256	♦

安全政策	TLS_1
AES256-GCM-SHA384	♦
AES256-SHA256	♦

OpenSSL 和 RFC 密碼名稱

對於相同的密碼，OpenSSL 和 IETF RFC 5246 使用不同的名稱。下表將 OpenSSL 名稱對應到每個密碼的 RFC 名稱。

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

有關 HTTP API 和 WebSocket API 的資訊

如需 HTTP API 和 WebSocket API 的詳細資訊，請參閱the section called “適用於 API 的安全性原則”和the section called “ WebSocket API 的安全性原則”。

建立邊緣最佳化自訂網域名稱

主題

- 設定 API Gateway API 的邊緣最佳化自訂網域名稱
- 記錄自定義域名創建 CloudTrail
- 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

- 輪換匯入至 ACM 的憑證
- 呼叫具有自訂網域名稱的 API

設定 API Gateway API 的邊緣最佳化自訂網域名稱

下列程序描述了如何使用 API Gateway 主控台來建立 API 的自訂網域名稱。

使用 API Gateway 主控台建立自訂網域名稱

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 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. 在 Configuration (組態) 下，選擇 Edge-optimized (邊緣最佳化)。
6. 選擇最低版本的 TLS。
7. 選擇 ACM 憑證。

   > **Note**
   >
   > 若要使用 ACM 憑證搭配 API Gateway 邊緣最佳化自訂網域名稱，您必須在 `us-east-1` 區域 (美國東部 (維吉尼亞北部)) 中請求或匯入憑證。

8. 選擇 Create domain name (建立網域名稱)。
9. 建立自訂網域名稱之後，主控台會以的形式顯示相關聯的 CloudFront 通訊網域名稱*`distribution-id`*`.cloudfront.net`，以及憑證 ARN。請注意輸出中顯示的 CloudFront 分發網域名稱。在後續步驟中，您需要它來設定自訂網域之 DNS 中的 CNAME 值或 A 記錄別名目標。

   > **Note**
   >
   > 新建立的自訂網域名稱大約需要 40 分鐘的時間準備。同時，您可以設定 DNS 記錄別名，將自訂網域名稱對應至關聯的 CloudFront 通訊網域名稱，並在初始化自訂網域名稱時為自訂網域名稱設定基本路徑對應。

10. 接下來，您可以使用 DNS 提供者設定 DNS 記錄，將自訂網域名稱對應至相關聯的 CloudFront 散發。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。

針對大多數的 DNS 提供者，自訂網域名稱會新增至託管區域作為 CNAME 資源記錄集。CNAME 記錄名稱指定您先前在 Domain Name (網域名稱) 中輸入的自訂網域名稱 (例如，`api.example.com`)。CNAME 記錄值會指定 CloudFront 散發的網域名稱。不過，如果您的自訂網域是 Zone Apex (即 `example.com`，而不是 `api.example.com`)，則無法使用 CNAME 記錄。Zone Apex 通常也稱為您組織的根網域。針對 Zone Apex，您需要使用 A 記錄別名，但前提是 DNS 提供者予以支援。

使用 Route 53，您可以為自訂網域名稱建立 A 記錄別名，並將 CloudFront 發佈網域名稱指定為別名目標。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。

使用 A 記錄別名也可以消除基礎 CloudFront發佈網域名稱的曝光，因為網域名稱對應僅在 Route 53 內進行。基於這些原因，建議您盡可能使用 Route 53 A 記錄別名。

除了使用 API Gateway 主控台之外，您還可以使用 API Gateway REST API、 AWS CLI 或其中一個 AWS 開發套件來設定 API 的自訂網域名稱。作為說明，下列程序概述使用 REST API 呼叫進行這項操作的步驟。

使用 API Gateway REST API 設定自訂網域名稱

1. 呼叫 domainname:create，並指定 AWS Certificate Manager中所存放憑證的自訂網域名稱和 ARN。

   成功的 API 呼叫會傳回回`201 Created`應，其中包含憑證 ARN 以及其承載中相關聯的 CloudFront 散發名稱。
2. 請注意輸出中顯示的 CloudFront 分發網域名稱。在後續步驟中，您需要它來設定自訂網域之 DNS 中的 CNAME 值或 A 記錄別名目標。
3. 依照上述程序設定 A 記錄別名，將自訂網域名稱對應至其 CloudFront 發佈名稱。

如需此 REST API 呼叫的程式碼範例，請參閱 domainname:create。

記錄自定義域名創建 CloudTrail

啟 CloudTrail 用記錄您帳戶發出的 API Gateway 呼叫時，API Gateway 會在針對 API 建立或更新自訂網域名稱時記錄相關的 CloudFront 分發更新。由於這些 CloudFront 發行版由 API Gateway 擁有，因此每個報告的 CloudFront 發行版都會透過下列其中一個特定區域的 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
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

區域	帳戶 ID
ap-south-1	507069717855
ap-south-2	644042651268
ap-east-1	174803364771
sa-east-1	287228555773
me-south-1	855739686837
me-central-1	614065512851

使用自訂網域名稱作為其主機名稱來設定 API 的基本路徑對應

您可以使用單一自訂網域名稱作為多個 API 的主機名稱。做法是在自訂網域名稱上設定基本路徑對應。使用基本路徑對應，自訂網域下的 API 可以透過自訂網域名稱與相關聯基本路徑的組合進行存取。

例如，如果您已建立名為 `PetStore` 的 API 和名為 `PetShop` 的另一個 API，並在 API Gateway 中設定自訂網域名稱 `api.example.com`，則可以將 `PetStore` API 的 URL 設定為 `https://api.example.com` 或 `https://api.example.com/myPetStore`。`PetStore`API 是與自訂網域名稱 `myPetStore` 下具有空字串或 `api.example.com` 的基本路徑建立關聯。同樣地，您可以指派 `yourPetShop` API 的基本路徑 `PetShop`。`https://api.example.com/yourPetShop` URL 接著會是 `PetShop` API 的根 URL。

設定 API 的基本路徑之前，請完成中的步驟設定 API Gateway API 的邊緣最佳化自訂網域名稱

下列程序會設定 API 對應，以將自訂網域名稱的路徑對應至 API 階段。

若要使用 API Gateway 主控台建立 API 對應

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. 選擇 Configure API mappings (設定 API 對應)。
4. 選擇 Add new mapping (新增對應)。
5. 指定對應的 API、Stage (階段) 及 Path (路徑) (選擇性)。

6. 選擇 Save (儲存)。

此外，您可以呼叫 API Gateway REST API、 AWS CLI 或其中一個 AWS 開發套件，以自訂網域名稱作為主機名稱來設定 API 的基本路徑對應。作為說明，下列程序概述使用 REST API 呼叫進行這項操作的步驟。

使用 API Gateway REST API 設定 API 的基本路徑對應

- 在特定自訂網域名稱上呼叫 basepathmapping:create，並在請求承載中指定 `basePath`、`restApiId` 和部署 `stage` 屬性。

  成功 API 呼叫會傳回 `201 Created` 回應。

如需 REST API 呼叫的程式碼範例，請參閱 basepathmapping:create。

輪換匯入至 ACM 的憑證

ACM 會自動處理它所發出之憑證的續約。您不需要為自訂網域名稱輪換任何 ACM 頒發的憑證。CloudFront 代表您處理它。

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，您可以離開 ACM 並 CloudFront 自動為您處理憑證輪換。若要建立或匯入新的 ACM 憑證，請遵循所指定網域名稱的請求或匯入新 ACM 憑證步驟。

若要輪替網域名稱的憑證，您可以使用 API Gateway 主控台、API Gateway REST API、 AWS CLI 或其中一個 AWS SDK。

使用 API Gateway 主控台輪換匯入至 ACM 的過期憑證

1. 在 ACM 中請求或匯入憑證。
2. 回到 API Gateway 主控台。
3. 從 API Gateway 主控台主要導覽窗格中，選擇 Custom Domain Names (自訂網域名稱)。
4. 選擇自訂網域名稱。
5. 選擇 Edit (編輯)。
6. 從 ACM Certificate (ACM 憑證) 下拉式清單中，選擇所要的憑證。
7. 選擇 Save (儲存)，開始輪換自訂網域名稱的憑證。

> **Note**
>
> 大約需要 40 分鐘的時間，程序才能完成。輪換完成之後，您可以選擇 ACM Certificate (ACM 憑證) 旁邊的雙向箭頭圖示來復原為原始憑證。

為了說明如何以程式設計方式輪換自訂網域名稱的已匯入憑證，我們概述使用 API Gateway REST API 的步驟。

使用 API Gateway REST API 輪換已匯入的憑證

- 呼叫 domainname:update 動作，並指定所指定網域名稱之新 ACM 憑證的 ARN。

呼叫具有自訂網域名稱的 API

呼叫具有自訂網域名稱的 API 與呼叫具有其預設網域名稱的 API 相同，但前提是使用正確的 URL。

下列範例會比較和對比所指定區域 (`udxjef`) 中兩個 API (`qf3duz` 和 `us-east-1`) 以及所指定自訂網域名稱 (`api.example.com`) 的一組預設 URL 和對應自訂 URL。

具有預設和自訂網域名稱之 API 的根 URL

API ID	階段	預設 URL	基本路徑	自訂 URL
udxjef	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

API ID	階段	預設 URL	基本路徑	自訂 URL
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

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 CloudFront 在發行版上強制執行 SNI。如需如何 CloudFront 使用自訂網域名稱的相關資訊，請參閱 Amazon CloudFront 自訂 SSL。

在 API Gateway 中設定區域性自訂網域名稱

您可以為 AWS 區域 API 端點 (針對區域) 建立自訂網域名稱。若要建立自訂網域名稱，您必須提供具有區域限制的 ACM 憑證。如需建立或上傳自訂網域名稱憑證的詳細資訊，請參閱「在 中備妥憑證 AWS Certificate Manager」。

> **Important**
>
> 針對 API Gateway 區域自訂網域名稱，您必須在與 API 相同的區域中請求或匯入憑證。

當您使用 ACM 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 (如果該角色尚不存在)。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。角色已命名，AWSServiceRoleForAPIGateway並且會附加 API GatewayServiceRolePolicy 管理政策。如需使用服務連結角色的詳細資訊，請參閱使用服務連結角色。

> **Important**
>
> 您必須建立 DNS 記錄，將自訂網域名稱指向區域性網域名稱。如此可讓繫結至自訂網域名稱的流量路由至 API 的區域性主機名稱。DNS 記錄可以是 CNAME 或 "A" 類型。

主題

- 使用 API Gateway 主控台設定具有 ACM 憑證的區域性自訂網域名稱
- 使用 ACM 憑證設定地區自訂網域名稱 AWS CLI

使用 API Gateway 主控台設定具有 ACM 憑證的區域性自訂網域名稱

若要使用 API Gateway 主控台來設定區域性自訂網域名稱，請使用下列程序。

使用 API Gateway 主控台設定區域性自訂網域名稱

1. 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. 在 Configuration (組態) 下方，選擇 Regional (區域性)。
6. 選擇最低版本的 TLS。
7. 選擇 ACM 憑證。憑證必須與 API 位於相同的區域。
8. 選擇 Create (建立)。
9. 遵循設定 Route 53 以將流量路由到 API Gateway 上的 Route 53 文件。

下列程序會設定 API 對應，以將自訂網域名稱的路徑對應至 API 階段。

若要使用 API Gateway 主控台建立 API 對應

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. 選擇 Configure API mappings (設定 API 對應)。
4. 選擇 Add new mapping (新增對應)。
5. 指定對應的 API、Stage (階段) 及 Path (路徑)。

6. 選擇 Save (儲存)。

若要了解如何設定自訂網域的基礎路徑對應，請參閱 使用自訂網域名稱作為其主機名稱來設定 API 的基本路徑對應。

使用 ACM 憑證設定地區自訂網域名稱 AWS CLI

若要使用 AWS CLI 為地區 API 設定自訂網域名稱，請使用下列程序。

1. 呼叫 `create-domain-name`，並指定自訂網域名稱與區域性憑證的 ARN。

```
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
```

請注意，指定的憑證是來自 `us-west-2` 區域，在此範例中，我們假設基礎 API 是來自相同的區域。

如果成功，呼叫會傳回類似如下的結果：

```
{
    "ApiMappingSelectionExpression": "$request.basepath",
    "DomainName": "regional.example.com",
    "DomainNameConfigurations": [
        {
            "ApiGatewayDomainName": "d-id.execute-api.us-west-2.amazonaws.com",
            "CertificateArn": "arn:aws:acm:us-west-2:123456789012:certificate/id",
            "DomainNameStatus": "AVAILABLE",
            "EndpointType": "REGIONAL",
            "HostedZoneId": "id",
            "SecurityPolicy": "TLS_1_2"
        }
    ]
}
```

`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。

2. 建立 DNS 記錄，將自訂網域名稱與區域性網域名稱相關聯。如此可讓繫結至自訂網域名稱的請求路由至 API 的區域性主機名稱。
3. 新增基底路徑對應，以根據指定的自訂網域名稱 (如 `0qzs2sy7bh`) 在部署階段 (如 `test`) 中公開指定的 API (例如 `regional.example.com`)。

```
aws apigatewayv2 create-api-mapping \
    --domain-name 'regional.example.com' \
    --api-mapping-key 'myApi' \
    --api-id 0qzs2sy7bh \
    --stage 'test'
```

因此，使用階段中所部署 API 之自訂網域名稱的基底 URL 會變成 `https://regional.example.com/myAPI`。

4. 請設定 DNS 記錄，將區域性自訂網域名稱對應到其指定託管區域 ID 的主機名稱。首先建立 JSON 檔案，其中包含用來設定區域性網域名稱 DNS 記錄的組態。下列範例說明如何建立 DNS A 記錄，以將區域性自訂網域名稱 (`regional.example.com`) 對應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
        }
      }
    }
  ]
}
```

5. 執行以下 CLI 命令：

```
aws route53 change-resource-record-sets \
    --hosted-zone-id {your-hosted-zone-id} \
    --change-batch file://path/to/your/setup-dns-record.json
```

其中 *{your-hosted-zone-id}* 是您帳戶中設定之 DNS 記錄的路由 53 託管區域識別碼。change-batch參數值指向文件夾中的 JSON 文件（*setup-dns-record.json*）（*##/*到您的）。

將自訂網域名稱遷移至不同的 API 端點

您可以在邊緣最佳化與區域端點之間遷移自訂網域名稱。您會先將新的端點組態類型新增至自訂網域名稱的現有 endpointConfiguration.types 清單。接著，您設定 DNS 記錄，以將自訂網域名稱指向新佈建的端點。選用的最後一個步驟是移除過時自訂網域名稱組態資料。

規劃遷移時，請記住，針對邊緣最佳化 API 的自訂網域名稱，ACM 所提供的必要憑證必須來自美國東部 (維吉尼亞北部) (us-east-1)。此憑證會分發至所有地理位置。不過，針對區域 API，區域網域名稱的 ACM 憑證必須來自託管 API 的相同區域。您可以將不在 us-east-1 區域中的邊緣最佳化自訂網域名稱遷移至區域自訂網域名稱，方法是先請求來自 API 之本機區域的新 ACM 憑證。

這可能需要最多 60 秒的時間才能完成邊緣最佳化自訂網域名稱與 API Gateway 中區域自訂網域名稱之間的遷移。若要讓新建立的端點準備好接受流量，遷移時間也取決於何時更新 DNS 記錄。

主題

- 使用移轉自訂網域名稱 AWS CLI

使用移轉自訂網域名稱 AWS CLI

若要使用將自訂網域名稱從邊緣最佳化端點移轉至區域端點，反之亦然，請呼叫update-domain-name指令以新增新端點類型，並選擇性地呼叫update-domain-name指令以移除舊的端點類型。AWS CLI

主題

- 將邊緣最佳化自訂網域名稱遷移至區域
- 將區域自訂網域名稱遷移至邊緣最佳化

將邊緣最佳化自訂網域名稱遷移至區域

若要將邊緣最佳化自訂網域名稱遷移至區域自訂網域名稱，請呼叫 update-domain-name CLI 命令，如下所示：

```
aws apigateway update-domain-name \
    --domain-name 'api.example.com' \
    --patch-operations  [ \
        { op:'add', path: '/endpointConfiguration/types',value: 'REGIONAL' }, \
        { op:'add', path: '/regionalCertificateArn', value: 'arn:aws:acm:us-
west-2:123456789012:certificate/cd833b28-58d2-407e-83e9-dce3fd852149' } \
      ]
```

區域憑證必須與區域 API 位在相同的區域。

成功回應會有 200 OK 狀態碼以及與下列類似的內文：

```
{
    "certificateArn": "arn:aws:acm:us-
east-1:123456789012:certificate/34a95aa1-77fa-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，您可以呼叫下update-domain-name列命令來移除邊緣最佳化的自訂網域名稱： AWS CLI

```
aws apigateway update-domain-name \
    --domain-name api.example.com \
    --patch-operations [ \
            {op:'remove', path:'/endpointConfiguration/types', value:'EDGE'}, \
            {op:'remove', path:'certificateName'}, \
            {op:'remove', path:'certificateArn'} \
        ]
```

將區域自訂網域名稱遷移至邊緣最佳化

若要將區域自訂網域名稱移轉至邊緣最佳化的自訂網域名稱，請呼叫的update-domain-name指令 AWS CLI，如下所示：

```
aws apigateway update-domain-name \
    --domain-name 'api.example.com' \
    --patch-operations  [ \
            { op:'add', path:'/endpointConfiguration/types',value: 'EDGE' }, \
            { op:'add', path:'/certificateName', value:'edge-cert'}, \
            { op:'add', path:'/certificateArn', value: 'arn:aws:acm:us-
east-1:123456789012:certificate/34a95aa1-77fa-427c-aa07-3a88bd9f3c0a' } \
        ]
```

在 us-east-1 區域中，必須建立邊緣最佳化網域憑證。

成功回應會有 200 OK 狀態碼以及與下列類似的內文：

```
{
    "certificateArn": "arn:aws:acm:us-
east-1:738575810317:certificate/34a95aa1-77fa-427c-aa07-3a88bd9f3c0a",
    "certificateName": "edge-cert",
    "certificateUploadDate": "2017-10-16T23:22:57Z",
    "distributionDomainName": "d1frvgze7vy1bf.cloudfront.net",
    "domainName": "api.example.com",
    "endpointConfiguration": {
        "types": [
            "EDGE",
            "REGIONAL"
        ]
    },
    "regionalCertificateArn": "arn:aws:acm:us-
east-1:123456789012:certificate/3d881b54-851a-478a-a887-f6502760461d",
    "regionalDomainName": "d-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：

```
aws apigateway update-domain-name \
    --domain-name api.example.com \
    --patch-operations [ \
        {op:'remove', path:'/endpointConfiguration/types', value:'REGIONAL'}, \
        {op:'remove', path:'regionalCertificateArn'} \
      ]
```

此命令的結果類似下列輸出，而且只有邊緣最佳化網域名稱組態資料：

```
{
    "certificateArn": "arn:aws:acm:us-
east-1:738575810317:certificate/34a95aa1-77fa-427c-aa07-3a88bd9f3c0a",
    "certificateName": "edge-cert",
    "certificateUploadDate": "2017-10-16T23:22:57Z",
    "distributionDomainName": "d1frvgze7vy1bf.cloudfront.net",
    "domainName": "regional.haymuto.com",
    "endpointConfiguration": {
        "types": "EDGE"
    }
}
```

使用適用於 REST API 的 API 映射

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`。

> **Note**
>
> 若要設定具有多個層級的 API 映射，您的自訂網域名稱必須為區域，並使用 TLS 1.2 安全政策。

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，以選取要叫用的 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。

要求	選取的 API	說明
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。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 對應中，自訂網域名稱和對應的 API 必須位於相同的 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。
- 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 對應至與 HTTP API 或 REST API 相同的自訂網域名稱。

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”。

如需建立所有資源的 AWS Serverless Application Model 範例範本，請參閱開啟 SAM 的工作階段 GitHub。

AWS Management Console

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 對應)。
6. 選擇 Add new mapping (新增對應)。
7. 輸入 API、Stage (階段)，以及選擇性地輸入 Path (路徑)。
8. 選擇 Save (儲存)。

AWS CLI

下列 AWS CLI 命令會建立 API 對應。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
```

AWS CloudFormation

下列 AWS CloudFormation 範例會建立 API 對應。

> **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
```

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` 端點來叫用 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。

> **Note**
>
> 當您停用預設端點時，它會影響 API 的所有階段。

下列 AWS CLI 命令會停用 REST API 的預設端點。

```
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。

下列 AWS CLI 指令會建立部署。

```
aws apigateway create-deployment \
    --rest-api-id abcdef123 \
    --stage-name dev
```

為了控制 DNS 備援功能，請設定自訂運作狀態檢查

您可以使用 Amazon Route 53 運作狀態檢查來控制次要區域中主要 AWS 區域 到一個 API Gateway API 的 DNS 容錯移轉。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，您會使用可協助操作員控制 DNS 容錯移轉的 CloudWatch 警示。如需設定容錯移轉時的詳細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和對 VPC 中的私有 AWS Lambda 資源執行 Route 53 健康狀態檢查。 CloudWatch

主題

- 必要條件
- 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

必要條件

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，共兩 AWS 區域份。如需更多詳細資訊，請參閱the section called “在 中備妥憑證 AWS Certificate Manager”。
- 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 警示的詳細資訊，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：

- 在兩個 API Gateway AWS 區域
- 兩個相同名稱的 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 警報
- 根據次要區域的 CloudWatch 警報進行 53 號路線健康檢查

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 AWS 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 測量結果並起始容錯移轉。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
```

將測量結果資料取代為您設定之 CloudWatch 警示的對應資料。

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

若要返回主要區域，請傳送使健康狀態檢查通過的 CloudWatch 量度。

```
aws cloudwatch put-metric-data \
    --metric-name Failover \
    --namespace HealthCheck \
    --unit Count \
    --value 0 \
    --region us-west-1
```

將測量結果資料取代為您設定之 CloudWatch 警示的對應資料。

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健全狀況檢查。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

- 啟用 API 快取以提升回應能力
- 啟用 API 的承載壓縮功能

啟用 API 快取以提升回應能力

您可以在 Amazon API Gateway 中啟用 API 快取，來快取您端點的回應。透過快取，您可以減少對端點進行的呼叫數目，並改善 API 請求的延遲。

當您啟用階段的快取時，API Gateway 會在指定的 time-to-live (TTL) 期間內快取來自端點的回應，以秒為單位。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。您可以使用 Amazon 中的`CacheHitCount`和`CacheMissCount`指標 CloudWatch 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。如需有關 AWS《1996 年美國 Health 保險流通與責任法案》(HIPAA)，以及使用 AWS 服務來處理、儲存和傳輸受保護的健康資訊 (PHI) 的詳細資訊，請參閱 HIP AA 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cacheClusterSize中的。

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？。

在 API Gateway 主控台中，您可以在「階段」頁面上設定快取。您可以佈建階段快取，並指定預設的方法層級快取設定。如果您開啟預設的方法層級快取，則舞台上所有方法的方法層級快取都會開啟，除非該GET方法具有方法覆寫。您部署到階段的任何其他GET方法都會有方法層級快取。若要針對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更多資訊，請參閱the section called “覆寫方法快取的階段快取”。

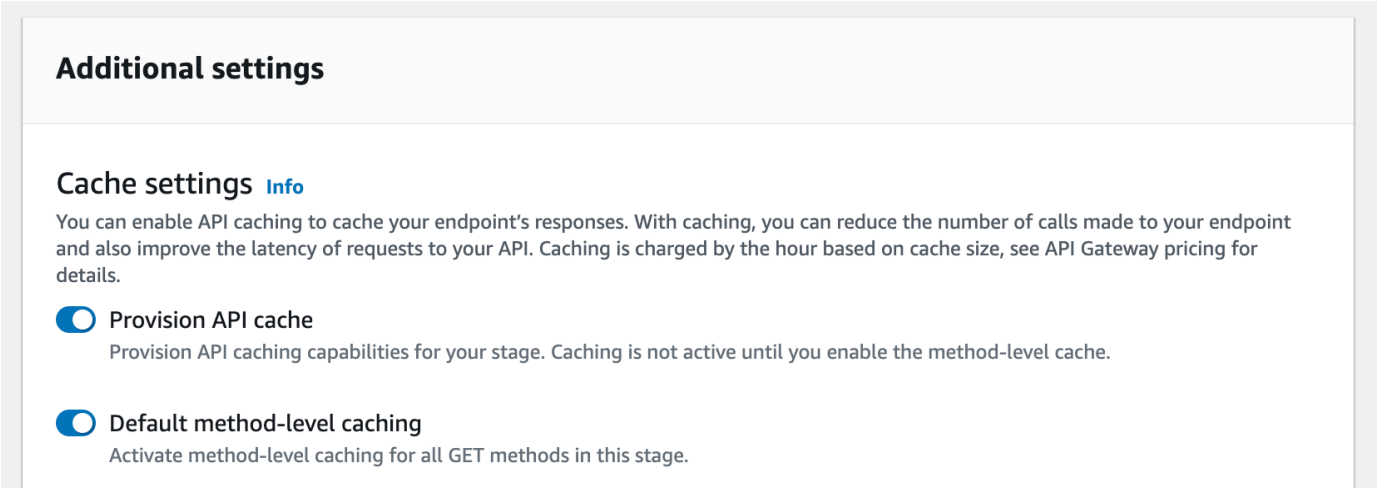
設定指定階段的 API 快取：

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，對於 [快取設定]，開啟 [佈建 API 快取]。

   這會為您的階段佈建快取叢集。
6. 若要啟用舞台的快取，請開啟「預設方法層級快取」。

這會開啟舞台上所有GET方法的方法層級快取。您部署到此階段的任何其他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不會影響該現有設定。



7. 選擇儲存變更。

> **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，快取叢集值會從變更`Create in progress`為`Active`。當快取刪除完成時，快取叢集值會從變更`Delete in progress`為`Inactive`。
> 當您為舞台上的所有方法開啟方法層級快取時，Default 方法層級快取值會變更為。`Active`如果您關閉舞台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 CloudWatch API CacheHitCount和階段CacheMissCount的指標和。

- 將時間戳記放在回應中。

> **Note**
>
> 您不應使用 CloudFront 回應中的`X-Cache`標頭來判斷您的 API 是否從 API Gateway 快取執行個體提供服務。

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，來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。

如果您變更「階段」詳細資料中的預設方法層級快取設定，則不會影響具有覆寫項目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**
   >
   > 在您為階段佈建快取叢集之前，快取才會處於作用中狀態。

7. 選擇儲存。

使用方法或整合參數作為快取金鑰來編製快取回應的索引

當快取方法或整合具有參數時 (其格式可能是自訂標頭、URL 路徑或查詢字串)，您可以使用部分或所有參數來形成快取金鑰。API Gateway 可以根據所使用的參數值來快取方法的回應。

> **Note**
>
> 設定資源的快取時必須提供快取金鑰。

例如，假設您有一個請求，其格式如下：

```
GET /users?type=... HTTP/1.1
host: example.com
...
```

在此請求中，type 可接受 admin 或 regular 值。如果您包含 type 參數作為快取金鑰的一部分，則會分別快取 GET /users?type=admin 的回應與 GET /users?type=regular 的回應。

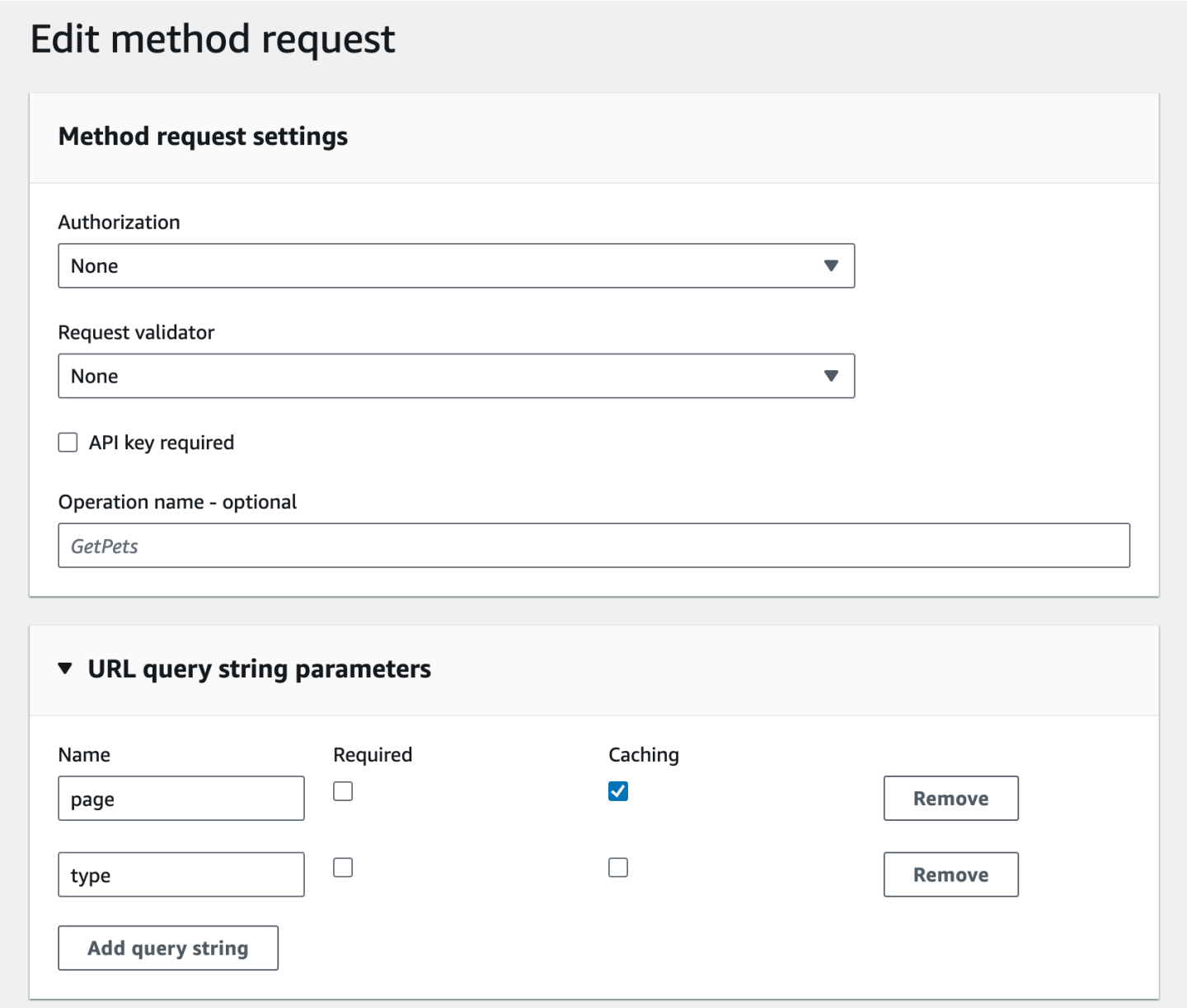
當方法或整合請求接受多個參數時，您可以選擇包含部分或所有參數來建立快取金鑰。例如，您可以針對下列請求 (依 TTL 期間所列的順序提出)，僅在快取金鑰中包含 type 參數：

```
GET /users?type=admin&department=A HTTP/1.1
host: example.com
...
```

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
...
```

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)。



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。

若要排清 API 階段快取，請選擇階段動作選單，然後選取排清階段快取。

> **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。

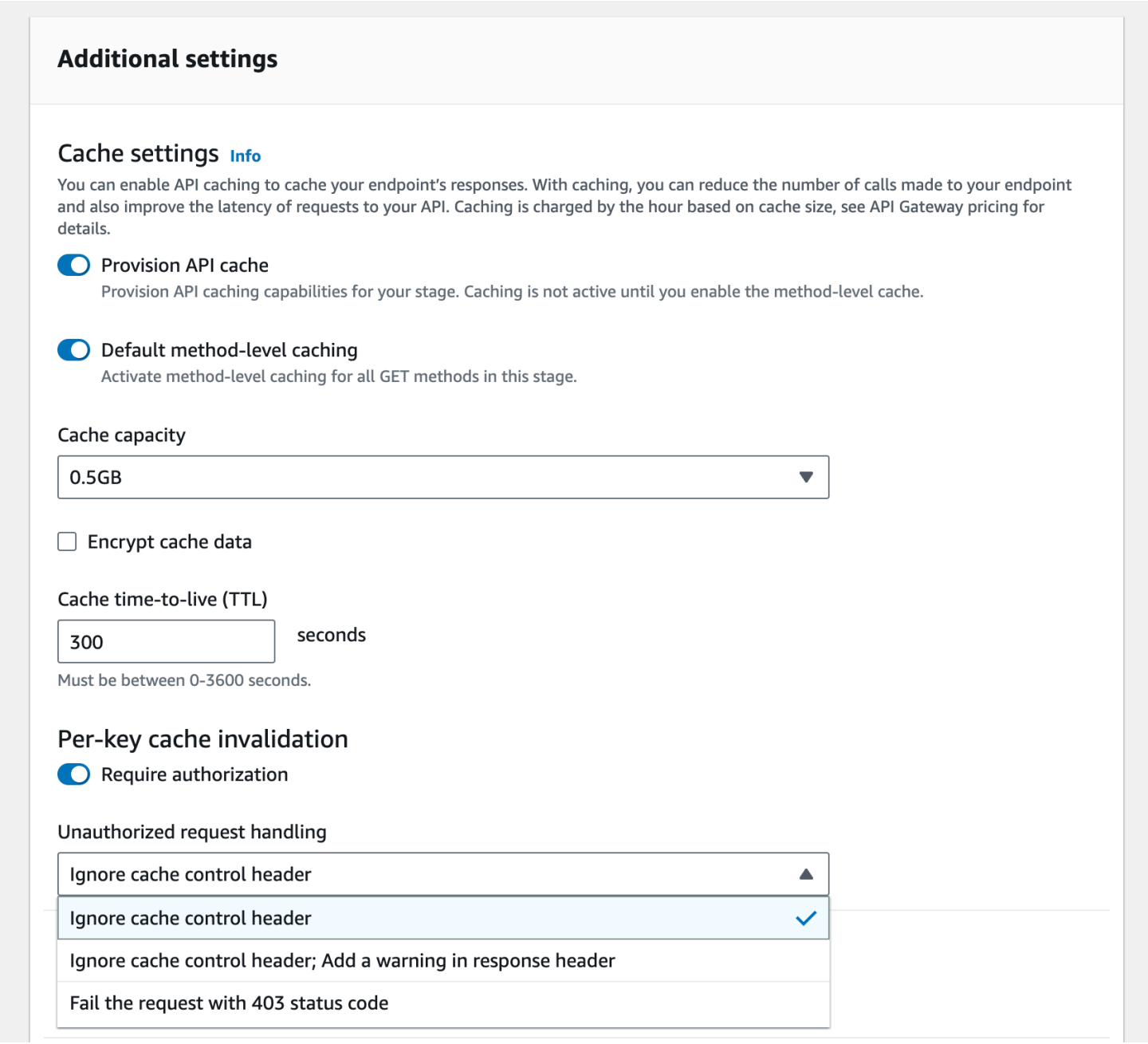
```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "execute-api:InvalidateCache"
      ],
      "Resource": [
        "arn:aws:execute-api:region:account-id:api-id/stage-name/GET/resource-path-specifier"
      ]
    }
  ]
}
```

此政策可讓 API Gateway 執行服務使一或多項指定資源請求的快取失效。若要指定一組目標資源，請針對 `account-id`、`api-id` 與 `Resource` 之 ARN 值中的其他項目使用萬用字元 (*)。如需如何設定 API Gateway 執行服務許可的詳細資訊，請參閱 使用 IAM 許可控制 API 的存取。

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。

您可以在 API Gateway 主控台中選擇未經授權的要求處理選項，以控制未經授權要求的處理方式。



這三個選項會導致下列行為：

- Fail the request with 403 status code (請求失敗並顯示 403 狀態碼)：傳回 403 未授權回應。

  若要使用 API 來設定此選項，請使用 `FAIL_WITH_403`。

- Ignore cache control header; Add a warning in response header (忽略快取控制標頭；在回應標頭中新增警告)：處理請求並在回應中新增警告標頭。

  若要使用 API 來設定此選項，請使用 `SUCCEED_WITH_RESPONSE_HEADER`。

- Ignore cache control header (忽略快取控制標頭)：處理請求但不會在回應中新增警告標頭。

  若要使用 API 來設定此選項，請使用 SUCCEED_WITHOUT_RESPONSE_HEADER。

啟用 API 的承載壓縮功能

API Gateway 可讓您的用戶端對使用其中一個支援的內容編碼壓縮的承載呼叫 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 來啟用或停用 API 的壓縮。 AWS CLI

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

- 啟用 API 的承載壓縮功能
- 對壓縮的承載呼叫 API 方法
- 接收已壓縮承載的 API 回應

啟用 API 的承載壓縮功能

您可以使用 API Gateway 主控台 AWS CLI、或 AWS SDK 為 API 啟用壓縮。

針對現有的 API，您必須在啟用壓縮後部署 API，以使變更生效。針對新的 API，您可以在 API 設定完成後部署 API。

> **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
- 使用以下方式啟用 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。

使用以下方式啟用 API 的承載壓縮 AWS CLI

若要使用 AWS CLI 建立新 API 並啟用壓縮，請依照下列方式呼叫`create-rest-api`命令：

```
aws apigateway create-rest-api \
    --name "My test API" \
    --minimum-compression-size 0
```

若要使用在 AWS CLI 現有 API 上啟用壓縮，請依照下列方式呼叫`update-rest-api`命令：

```
aws apigateway update-rest-api \
```

```
    --rest-api-id 1234567890 \
    --patch-operations op=replace,path=/minimumCompressionSize,value=0
```

minimumCompressionSize 屬性擁有介於 0 和 10485760 的非負數整數值 (1 千萬個位元組)。它可衡量壓縮閾值。如果承載大小小於這個值，則不會對承載進行壓縮或解壓縮。因此，請將此值設為零以允許任何承載大小使用壓縮功能。

若要使用 AWS CLI 來停用壓縮，請依照下列方式呼叫update-rest-api命令：

```
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

對壓縮的承載呼叫 API 方法

若要提出已壓縮承載的 API 請求，用戶端必須使用其中一個支援的內容編碼來設定 Content-Encoding 標頭。

假設您是 API 用戶端，並且想要呼叫 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_gzip_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 和階段之前發生在解壓縮的早期階段，則不會將錯誤 CloudWatch 記錄到記錄中。

接收已壓縮承載的 API 回應

對已啟用壓縮功能的 API 提出請求時，用戶端可以透過指定 `Accept-Encoding` 標頭與支援的內容編碼，選擇接收特定格式的壓縮回應承載。

只有符合下列條件時，API Gateway 才會壓縮回應承載：

- 傳入請求具有 `Accept-Encoding` 標頭以及支援的內容編碼與格式。

> **Note**
>
> 如果未設定標頭，預設值為 `*`，如 RFC 7231 中所定義。在此情況下，API Gateway 不會壓縮承載。某些瀏覽器或用戶端可能會自動將 `Accept-Encoding` (例如 `Accept-Encoding:gzip, deflate, br`) 新增至已啟用壓縮功能的請求。這會觸發 API Gateway 的承載壓縮功能。若未明確指定支援的 `Accept-Encoding` 標頭值，API Gateway 就不會壓縮承載。

- 在 API 上設定 minimumCompressionSize 以啟用壓縮功能。
- 整合回應沒有 Content-Encoding 標頭。
- 整合回應承載的大小在套用適用的映射範本之後，大於或等於指定的 minimumCompressionSize 值。

API Gateway 會套用針對整合回應設定的任何對應範本，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。

PetStore API 示例和以下請求是一個示例：

```
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
...

[binary]

J[binary])JV
[binary]:P^IeA*[binary]+([binary]L [binary]X[binary]YZ[binary]ku0L0B7!9[binary][binary]C#[binary]&[binary][binary]Y[binary][binary]a[binary][binary]^[binary]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 分配給用戶端

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

- 使用 API 金鑰建立及使用用量計劃
- 記載 REST API
- 在 API Gateway 中針對 REST API 產生軟體開發套件
- 透過銷售您的 API Gateway API AWS Marketplace

使用 API 金鑰建立及使用用量計劃

在您建立、測試和部署 API 之後，您可以使用 API Gateway 用量計劃讓它們可做為產品優惠供給客戶使用。您可以設定用量計劃和 API 金鑰，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，您會將其發佈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 “使用 API Gateway 主控台設定 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。

- 如果您使用開發人員入口網站發佈 API，請注意，客戶可以訂閱指定使用方案中的所有 API，即使您尚未讓客戶看到這些 API 也是如此。
- 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 金鑰新增至用量計劃後，更新作業可能需要幾分鐘才能完成。

設定用量計劃的步驟？

下列步驟概述身為 API 擁有者的您，如何為客戶建立和設定用量計劃。

設定用量計劃

1. 建立一或多個 API、設定方法要求 API 金鑰，以及將 API 部署至階段。
2. 產生或匯入 API 金鑰來發佈到將使用您 API 的應用程式開發人員 (您的客戶)。
3. 建立具有所需調節和配額限制的用量計劃。
4. 將 API 階段和 API 金鑰與用量計劃建立關聯。

API 發起人必須在請求 API 的 `x-api-key` 標頭中提供已指派的 API 金鑰。

> **Note**
>
> 若要在用量計劃中包含 API 方法，您必須設定個別的 API 方法要求 API 金鑰。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”。

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 金鑰發佈至客戶，並要求他們傳遞 API 金鑰，做為每個傳入請求的 `X-API-Key` 標頭。

`AUTHORIZER`

您可以讓 Lambda 授權方傳回 API 金鑰，當做授權回應的一部分。如需授權回應的詳細資訊，請參閱the section called “來自 API Gateway Lambda 授權者的輸出”。

> **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”。

使用 API Gateway 主控台選擇 API 的 API 金鑰來源

1. 登入 API Gateway 主控台。
2. 選擇現有的 API 或建立新的 API。
3. 在主導覽窗格中，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。

若要使用選擇 API 的 API 金鑰來源 AWS CLI，請依照下列方式呼叫`update-rest-api`命令：

```
aws apigateway update-rest-api --rest-api-id 1234123412 --patch-operations
 op=replace,path=/apiKeySource,value=AUTHORIZER
```

若要讓用戶端提交 API 金鑰，請在上述 CLI 命令中將 `value` 設為 HEADER。

若要使用 API Gateway REST API 選擇 API 的 API 金鑰來源，請呼叫 restapi:update，如下所示：

```
PATCH /restapis/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。

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。

用戶端現在可以在呼叫 API 方法的同時，提供以所選 API 金鑰為標頭值的 x-api-key 標頭。

若要使用授權方來源 API 金鑰：

1. 使用所需的 API 方法建立 API，然後將 API 部署至階段。

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。如需建立權杖型授權者的指示，請參閱。the section called “範例TOKEN授權者 Lambda 函數”
4. 設定 API 方法請求 API 金鑰，也在方法中啟用 Lambda 授權方。
5. 將 API 重新部署到同一個階段。如果您將 API 部署到新的階段，請務必更新用量計劃以連接新的 API 階段。

用戶端現在可以不用明確提供任何 API 金鑰，呼叫需要 API 金鑰的方法。自動使用授權方傳回的 API 金鑰。

使用 API Gateway 主控台設定 API 金鑰

若要設定 API 金鑰，請執行下列步驟：

- 設定 API 方法以要求 API 金鑰。
- 為區域中的 API 建立或匯入 API 金鑰。

在設定 API 金鑰之前，您必須已建立 API 並將它部署至階段。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 API Gateway 中開發 REST API 和在 Amazon API Gateway 中部署 REST API。

建立 API 金鑰之後，您必須將其與使用方案產生關聯。如需詳細資訊，請參閱 使用 API Gateway 主控台建立、設定和測試用量計劃。

> **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

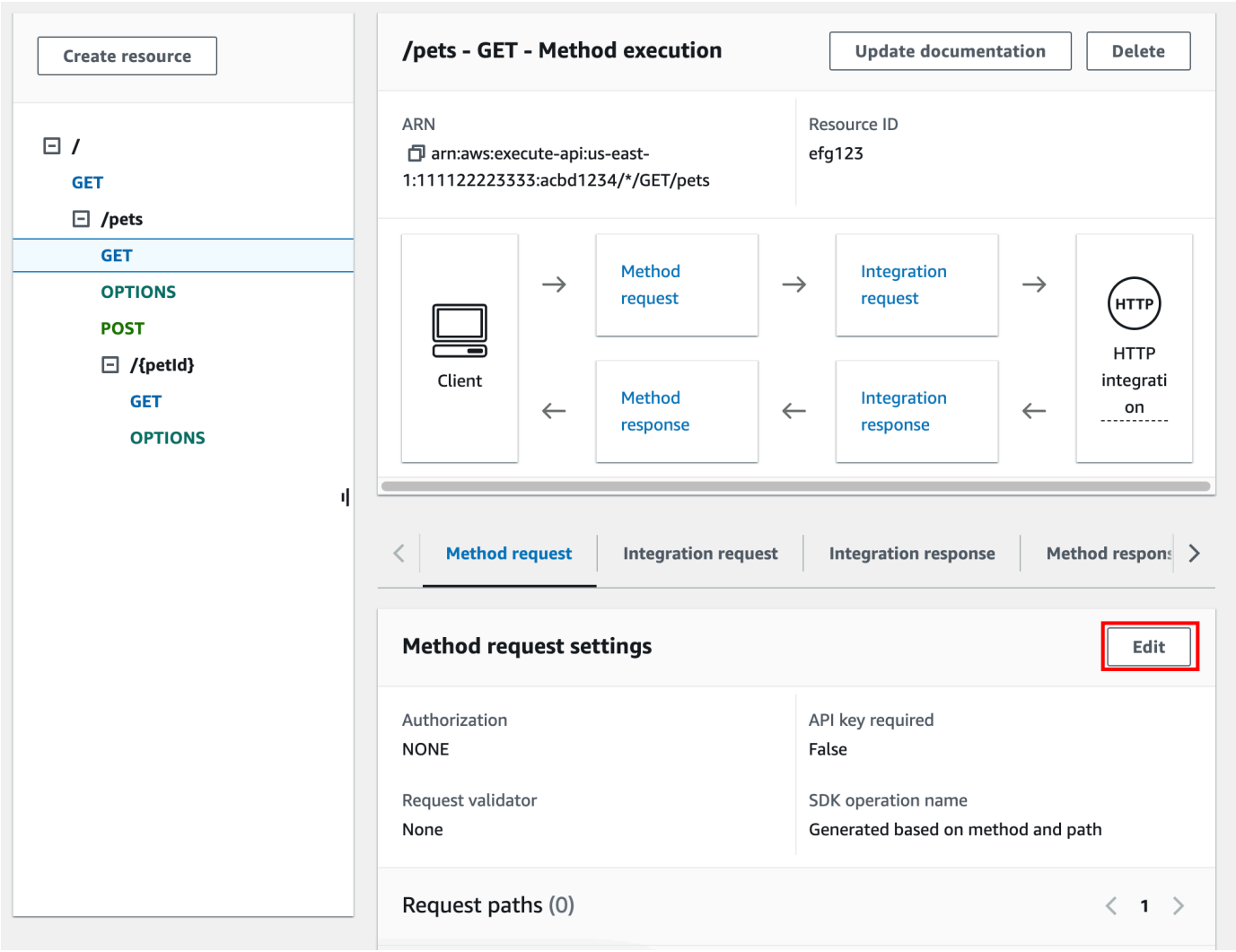
- 方法中需要 API 金鑰
- 建立 API 金鑰
- 匯入 API 金鑰

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。

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。

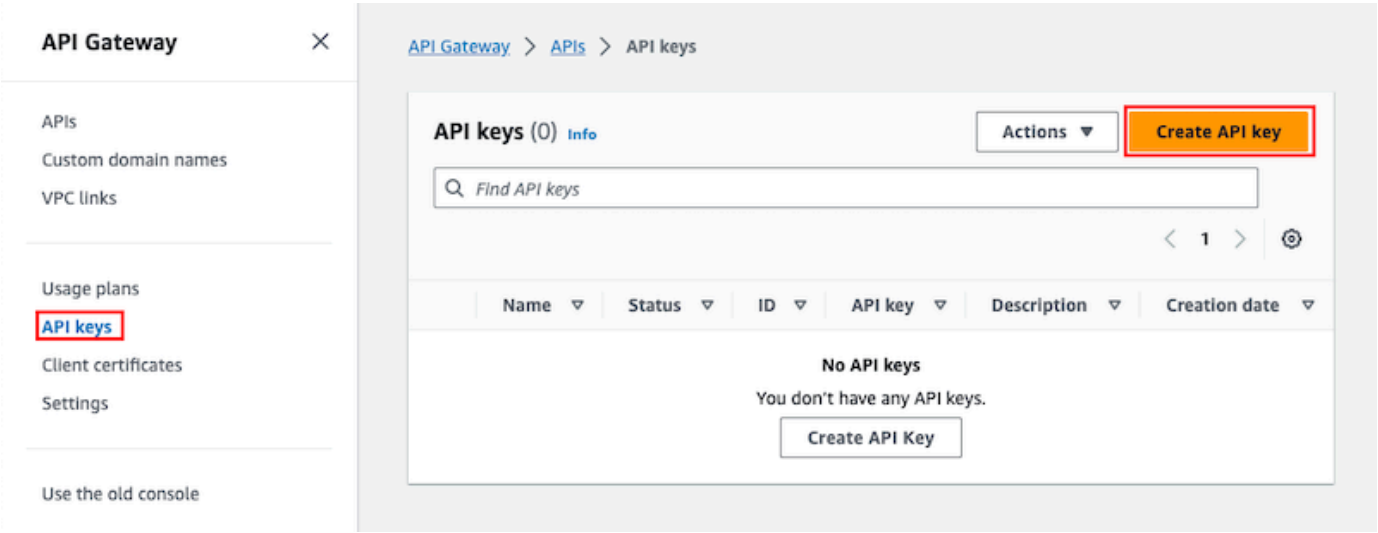
如果需要 API 金鑰選項設為 `false`，而且您不執行之前的各項步驟，則方法不會使用任何與 API 階段相關聯的 API 金鑰。

建立 API 金鑰

如果您已建立或匯入 API 金鑰供用量計劃使用，您可以略過此操作和下一個程序。

建立 API 金鑰

1. 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。

匯入 API 金鑰

下列程序說明如何匯入 API 金鑰供用量計劃使用。

匯入 API 金鑰

1. 在以下網址登入 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。

使用 API Gateway 主控台建立、設定和測試用量計劃

建立用量計劃之前，請確保您已設定所需的 API 金鑰。如需詳細資訊，請參閱使用 API Gateway 主控台設定 API 金鑰。

本節說明如何使用 API Gateway 主控台建立和使用用量計劃。

主題

- 遷移您的 API 到預設用量計劃 (如需要)
- 建立用量計劃
- 測試用量計劃
- 維護計劃用量

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。但是，您必須使用該UsagePlanapiStages屬性將指定的 API 階段值（apiId和stage）與包含的 API 密鑰（via UsagePlanKey）相關聯，而不是使用ApiKeystageKeys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。

您也可以使用下列方式將 API 移轉至預設使 AWS CLI 用方案：

若要使用移轉至預設使用方案 AWS CLI

1. 呼叫 CLI 命令：update-account。
2. 針對 cli-input-json 參數，請使用下列 JSON：

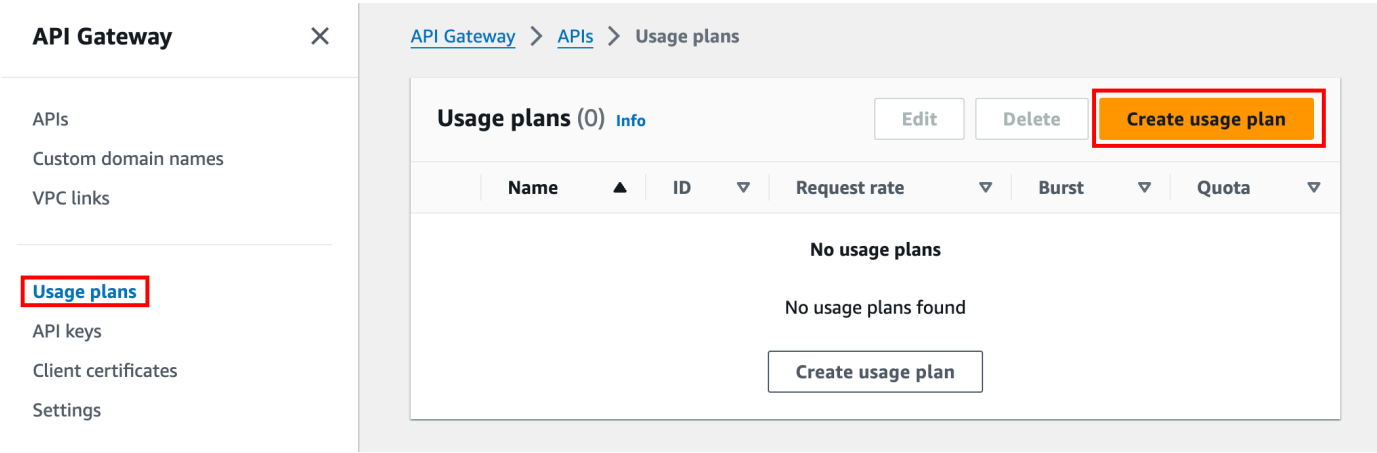
```
[
    {
        "op": "add",
        "path": "/features",
        "value": "UsagePlans"
    }
]
```

建立用量計劃

下列程序說明如何建立用量計劃。

建立用量計劃

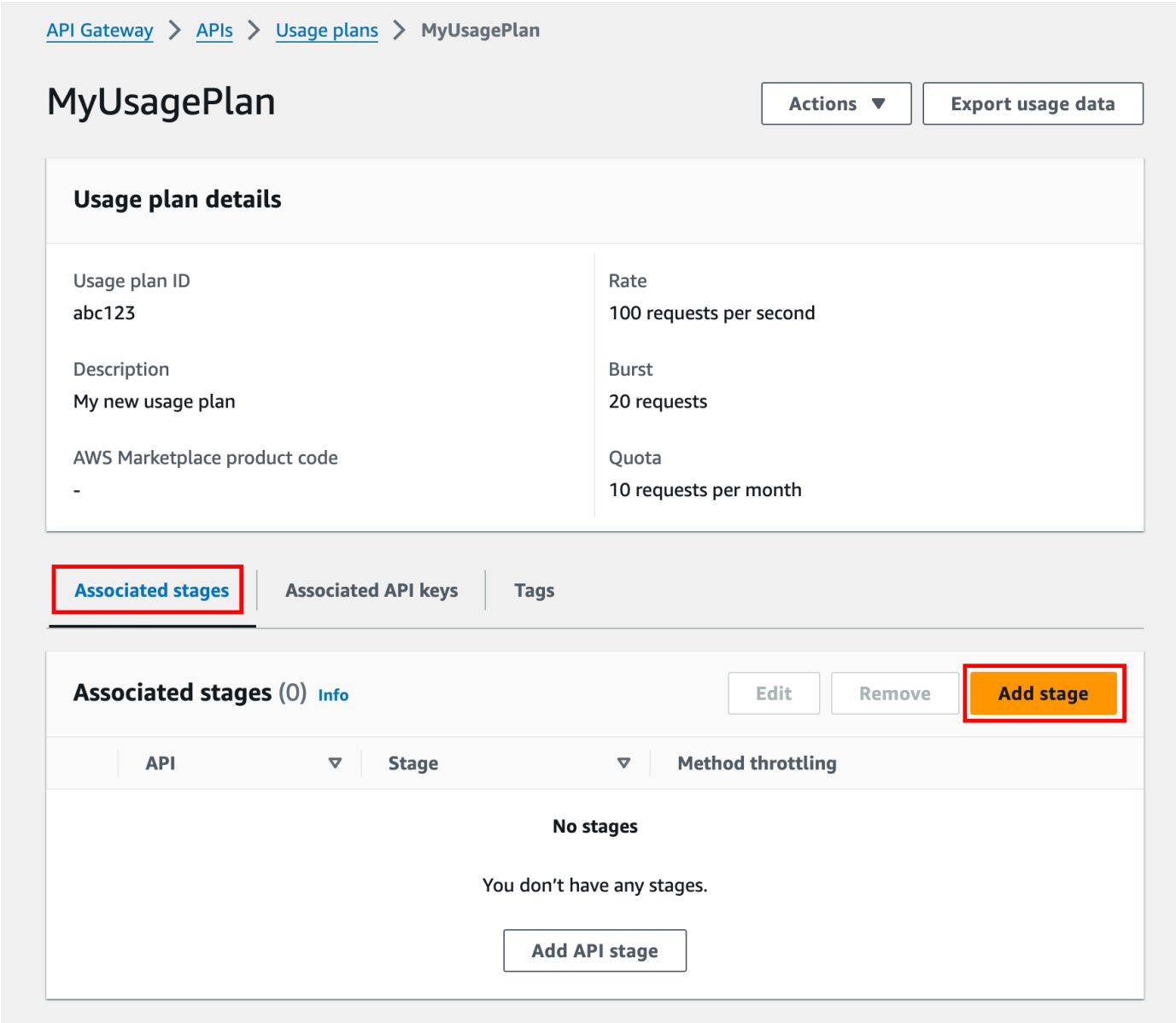
1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 在 API Gateway 主導覽窗格中，選擇用量計劃，然後選擇建立用量計劃。



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。

若要將階段新增至用量計劃

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：

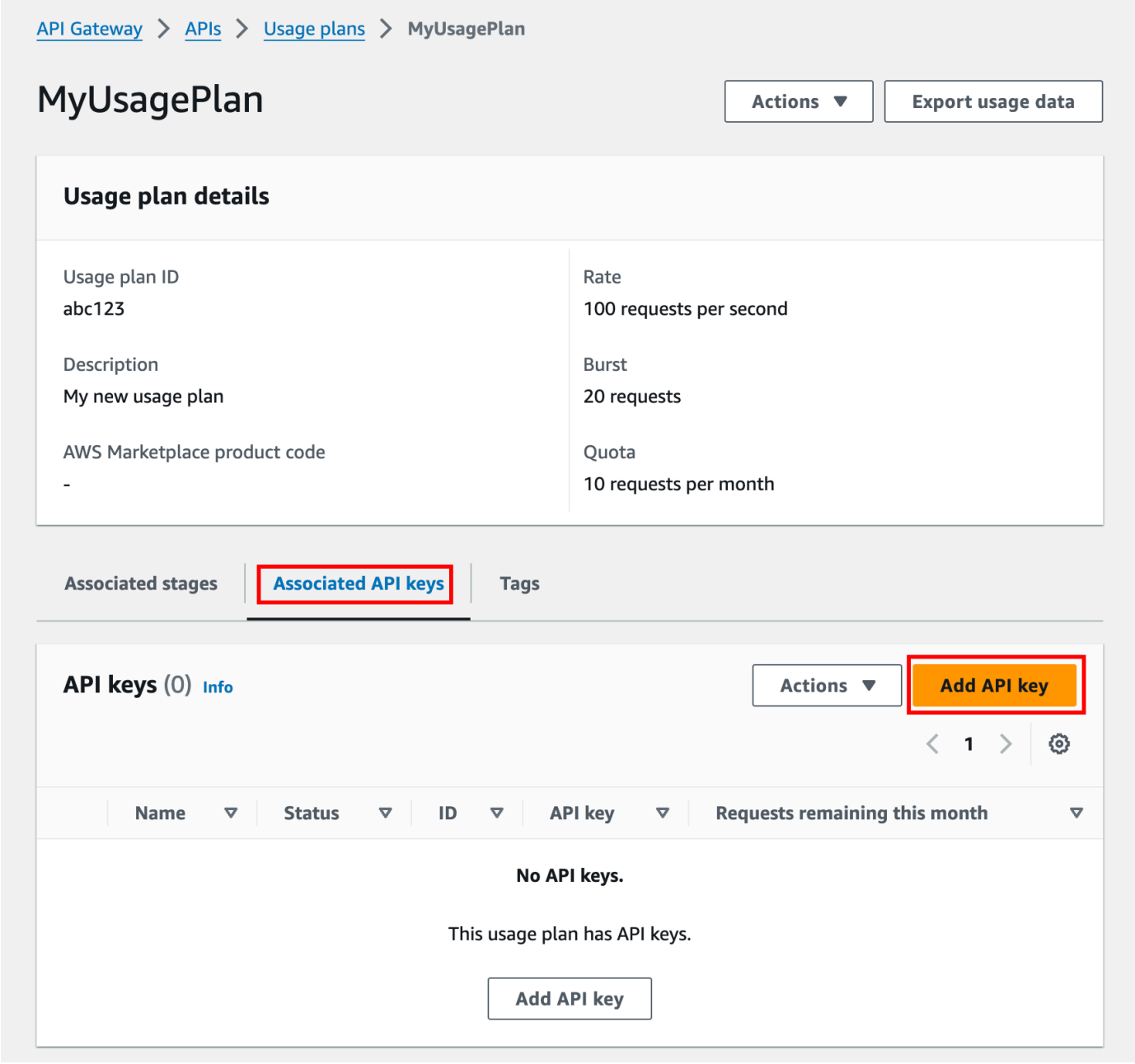
   a. 選擇方法層級限流，然後選擇新增方法。
   b. 針對資源，從 API 中選取資源。
   c. 針對方法，從 API 中選取方法。

d. 輸入用量計劃的費率和暴增。

6. 選擇新增至用量計劃。

若要將金鑰新增至用量計劃

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。

測試用量計劃

要測試使用計劃，您可以使用 AWS SDK AWS CLI，或像郵遞員這樣的 REST API 客戶端。如需使用 Postman 測試用量計劃的範例，請參閱測試用量計劃。

維護計劃用量

維護用量計劃涉及監控指定時段內已使用和剩餘的配額，以及如有需要，依指定的量擴充剩餘配額。下列程序說明如何監控配額。

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。

```
{
  "thisPeriod": {
    "px1KW6...qBazOJH": [
      [
        0,
        5000
      ],
      [
        0,
        5000
      ],
      [
        0,
        10
```

```
      ]
    ]
  },
  "startDate": "2016-08-01",
  "endDate": "2016-08-03"
}
```

範例中的用量資料顯示某 API 用戶端自 2016 年 8 月 1 日到 2016 年 8 月 3 日的每日用量資料，依 API 金鑰 (px1KW6...qBaz0JH) 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。

使用 API Gateway REST API 設定 API 金鑰

若要設定 API 金鑰，請執行下列步驟：

- 設定 API 方法以要求 API 金鑰。
- 為區域中的 API 建立或匯入 API 金鑰。

在設定 API 金鑰之前，您必須已建立 API 並將它部署至階段。在您建立 API 金鑰值之後便無法變更。

如需建立和部署 API 的 REST API 呼叫，請分別參閱 `restapi:create` 和 `deployment:create`。

> **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
- 建立或匯入 API 金鑰

方法中需要 API 金鑰

方法中如需要 API 金鑰，請執行下列其中一項操作：

- 呼叫 `method:put` 來建立方法。在請求承載中將 `apiKeyRequired` 設為 `true`。
- 呼叫 `method:update`，將 `apiKeyRequired` 設定為 `true`。

建立或匯入 API 金鑰

若要建立或匯入 API 金鑰，請執行下列其中一項操作：

- 呼叫 `apikey:create` 來建立 API 金鑰。
- 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。若要了解如何設定用量計劃，請參閱 使用 API Gateway CLI 和 REST API 建立、設定和測試用量計劃。

使用 API Gateway CLI 和 REST API 建立、設定和測試用量計劃

設定用量計劃之前，您必須已經完成下列操作：設定所選 API 的方法要求 API 金鑰、已將 API 部署或重新部署到階段，以及已建立或匯入一或多個 API 金鑰。如需詳細資訊，請參閱使用 API Gateway REST API 設定 API 金鑰。

若要使用 API Gateway REST API 設定用量計劃，請使用下列指示，假設您已建立可新增至用量計劃的 API。

主題

- 遷移到預設用量計劃

- 建立用量計劃
- 使用 AWS CLI 管理使用量計劃
- 測試用量計劃

遷移到預設用量計劃

第一次建立用量計劃時，您可以下列內文呼叫 `account:update`，將與所選 API 金鑰相關聯的現有 API 階段遷移到用量計劃：

```
{
  "patchOperations" : [ {
    "op" : "add",
    "path" : "/features",
    "value" : "UsagePlans"
  } ]
}
```

如需遷移與 API 金鑰相關聯之 API 階段的詳細資訊，請參閱在 API Gateway 主控台中遷移到預設用量計劃。

建立用量計劃

下列程序說明如何建立用量計劃。

使用 REST API 建立用量計劃

1. 呼叫 `usageplan:create` 來建立用量計劃。在承載中，指定計劃的名稱和說、相關聯的 API 階段、速率限制和配額。

   記下產生的用量計劃識別符。下一個步驟需要此值。
2. 請執行下列其中一項：

   a. 呼叫 `usageplankey:create` 將 API 金鑰新增至用量計劃。在承載中指定 `keyId` 和 `keyType`。

      若要將更多的 API 金鑰新增至用量計劃，請重複之前的呼叫，一次一個 API 金鑰。

   b. 呼叫 `apikey:import` 將一或多個 API 金鑰直接新增至指定的用量計劃。請求承載應該包含 API 金鑰值、相關聯的用量計劃識別符、指出用量計劃已啟用金鑰的布林值旗標，以及 API 金鑰名稱和說明 (如可能)。

以下 apikey:import 請求範例會將三個 API 金鑰 (識別為 key、name 和 description) 新增至一個用量計劃 (識別為 usageplanIds)：

```
POST /apikeys?mode=import&format=csv&failonwarnings=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。

使用 AWS CLI 管理使用量計劃

以下程式碼範例說明如何透過呼叫 update-usage-plan 命令來在用量計劃中新增、移除或修改方法層級調節設定。

> **Note**
>
> 請務必將 us-east-1 變更為您 API 的適當區域值。

若要為調節個別的資源和方法新增或取代速率限制：

```
aws apigateway --region us-east-1 update-usage-plan --usage-plan-id <planId> --patch-
operations
```

```
                        op="replace",path="/apiStages/<apiId>:<stage>/
throttle/<resourcePath>/<httpMethod>/rateLimit",value="0.1"
```

若要為調節個別的資源和方法新增或取代爆量限制：

```
aws apigateway --region us-east-1 update-usage-plan --usage-plan-id <planId>
 --patch-operations op="replace",path="/apiStages/<apiId>:<stage>/
throttle/<resourcePath>/<httpMethod>/burstLimit",value="1"
```

若要為個別的資源和方法移除方法層級調節設定：

```
aws apigateway --region us-east-1 update-usage-plan --usage-plan-id <planId>
 --patch-operations op="remove",path="/apiStages/<apiId>:<stage>/
throttle/<resourcePath>/<httpMethod>",value=""
```

要移除 API 的所有方法層級調節設定：

```
aws apigateway --region us-east-1 update-usage-plan --usage-plan-id <planId> --patch-
operations op="remove",path="/apiStages/<apiId>:<stage>/throttle ",value=""
```

此處為使用 Pet Store 範例 API 的範例：

```
aws apigateway --region us-east-1 update-usage-plan --usage-plan-id <planId> --patch-
operations
                op="replace",path="/apiStages/<apiId>:<stage>/throttle",value='"{\"/
pets/GET\":{\"rateLimit\":1.0,\"burstLimit\":1},\"//GET\":{\"rateLimit\":1.0,
\"burstLimit\":1}}"'
```

測試用量計劃

作為一個例子，讓我們使用 PetStore API，這是在教學課程：匯入範例來建立 REST API. 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，它會記錄`Usage Plan check failed due to an internal error`中的錯誤消息 CloudWatch。您可以忽略這類偶發的錯誤。

使用以下方式建立和設定 API 金鑰和使用計劃 AWS CloudFormation

您可以使 AWS CloudFormation 用在 API 方法上要求 API 金鑰，並建立 API 的使用計劃。範例 AWS CloudFormation 範本會執行下列動作：

- 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 金鑰的方法

您可以使用 OpenAPI 定義來要求方法上的 API 金鑰。

針對每個方法，建立安全性需求物件，以要求 API 金鑰來叫用該方法。然後，`api_key`在安全性定義中定義。建立 API 之後，將新的 API 階段新增至您的使用方案。

下列範例會建立 API，並需要`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"
      }
    }
  }
}
```

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”。

記載 REST API

為了協助客戶了解及使用您的 API，您應該記錄 API。為了協助您記錄 API，API Gateway 可讓您新增及更新個別 API 實體的說明內容，這是 API 開發程序不可或缺的一部分。API Gateway 可存放來源內容，並讓您封存不同版本的文件。您可以建立文件版本與 API 階段的關聯、將特定階段的文件快照匯出到外部 OpenAPI 檔案，並在發佈文件時分發檔案。

若要記錄您的 API，您可以呼叫 API Gateway REST API、使用其中一個 AWS SDK、使AWS CLI用 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，請參閱合 AWS 作夥伴網路 (APN) 部落格上的 ReadMe 與 API Gateway 整合，讓您的開發人員中心保持在最新狀態。

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 主控台發佈 API 文件
- 使用 API Gateway REST API 記載 API
- 使用 API Gateway REST API 發佈 API 文件

- 匯入 API 文件
- 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，文檔部分由DocumentationPart資源表示。作為一個整體的 API 文檔由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資源是儲存適用於個別 API 實體的文件內容的 JSON 物件。其 properties 欄位包含對應鍵值對的文件內容。其 location 屬性識別相關聯的 API 實體。

內容對應的成形取決於身為 API 開發人員的您。金鑰/值對的值可以是字串、數值、布林值、物件或陣列。location 物件的成形取決於目標實體類型。

DocumentationPart 資源支援內容繼承：API 實體的文件內容適用於 API 實體的子系。如需子實體與內容繼承定義的詳細資訊，請參閱從較一般規格的 API 實體繼承內容。

文件組件的位置

DocumentationPart執行個體的位置屬性可識別套用相關內容的 API 實體。API 實體可以是 API Gateway REST API 資源，例如資源RestApi、方法MethodResponse、授權者或模型。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	statusCode	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	statusCode	name	備註	
				文件會由符合位置模式的實體繼承。	

案例	path	statusCode	name	備註
3	/pets/petId	*	id	當案例 1、2 與 3 相符時，由於案例 3 的優先順序比案例 2 高且比案例 1 更明確，因此與此位置模式相關聯的文件會由符合位置模式的實體繼承。

以下是較泛型的 DocumentationPart 執行個體與較明確的執行個體的另一個對比範例。下列一般錯誤訊息 "Invalid request error" 會插入 400 錯誤回應的 OpenAPI 定義，除非遭到覆寫。

```
{
    "location" : {
        "type" : "RESPONSE",
        "statusCode": "400"
    },
    "properties" : {
        "description" : "Invalid request error."
    }"
}
```

透過下列覆寫，/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

下表顯示有效和必填欄位，以及與指定類型 API 實體相關聯之DocumentationPart資源的適用預設值。

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
Resource	`{ "location": { "type": "RESOURCE", "path": "resource_path " }, ... }`	type	path 的預設值為 /。	否
方法	`{ "location": { "type": "METHOD", "path": "resource_path ", "method": "http_verb " }, ... }`	type	path 與 method 的預設值分別為 / 與 *。	是，符合 path 的字首，且符合任何值的 method。
查詢參數	`{ "location": { "type": "QUERY_PARAMETER", "path": "resource_path ", "method": "HTTP_verb ", "name": "query_parameter_name " }, ... }`	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

請求內文	{
	"location": {
	"type": "REQUEST_BODY",
	"path": "*resource_path* ",
	"method": "*http_verb* "
	},
	...
	}
		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。

請求標頭參數	{
	"location": {
	"type": "REQUEST_HEADER",
	"path": "*resource_path* ",
	"method": "*HTTP_verb* ",
	"name": "*header_name* "
	},
	...
	}
		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

請求路徑參數	{
	"location": {
	"type": "PATH_PARAMETER",
	"path": "*resource*/{*path_parameter_name* }",
	"method": "*HTTP_verb* ",
	"name": "*path_parameter_name* "
	},
	...
	}
		type, name	path 與 method 的預設值分別為 / 與 *。	是，符合 path 的字首，且符合 method 的確切值。

回應	{
	"location": {
	"type": "RESPONSE",
	"path": "*resource_path* ",
	"method": "*http_verb* ",
	"statusCode": "*status_code* "
	},
	...
	}
		type	path、method 與 statusCode 的預設值分別為 /、* 與 *。	是，符合 path 的字首，且符合 method 與 statusCode 的確切值。

API 實體	有效的位置欄位	必要的位置欄位	預設欄位值	是否可繼承內容

回應標頭	{
	"location": {
	"type": "RESPONSE_HEADER",
	"path": "*resource_path* ",
	"method": "*http_verb* ",
	"statusCode": "*status_code* ",
	"name": "*header_name* "
	},
	...
	}
		type, name	path、method 與 statusCode 的預設值分別為 /、* 與 *。	是，符合 path 的字首，且符合 method 與 statusCode 的確切值。

回應內文	{
	"location": {
	"type": "RESPONSE_BODY",
	"path": "*resource_path* ",
	"method": "*http_verb* ",
	"statusCode": "*status_code* "
	},
	...
	}
		type	path、method 與 statusCode 的預設值分別為 /、* 與 *。	是，符合 path 的字首，且符合 method 與 statusCode 的確切值。

API 實體	有效的位置欄位	必要的位置欄位	預設欄位值	是否可繼承內容
授權方	`{ "location": { "type": "AUTHORIZER", "name": "authorizer_name " }, ... }`	type	不適用	否
模型	`{ "location": { "type": "MODEL", "name": "model_name " }, ... }`	type	不適用	否

文件版本

文件版本是 API DocumentationParts集合的快照，並以版本識別碼標記。發佈 API 的文件需要建立文件版本、將其與 API 階段建立關聯，並將特定階段版本的 API 文件匯出到外部 OpenAPI 檔案。在 API Gateway 中，文件快照會表示為DocumentationVersion資源。

當您更新 API 時，您會建立新版的 API。在 API Gateway 中，您可以使用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 PetStoreAPI 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。

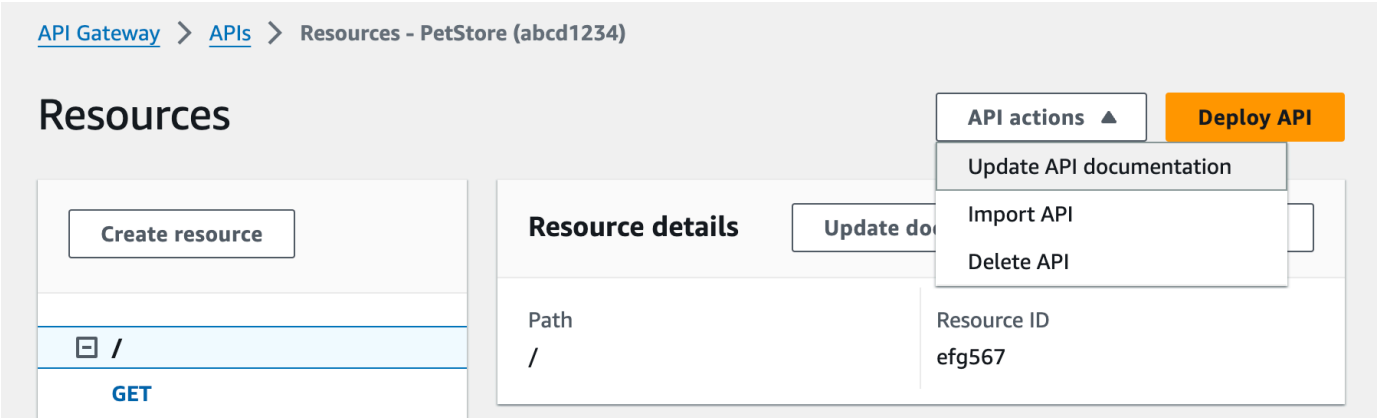
```
{
  "info": {
    "description": "Your first API Gateway API.",
    "contact": {
        "name": "John Doe",
        "email": "john.doe@api.com"
    }
  }
}
```

> **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：

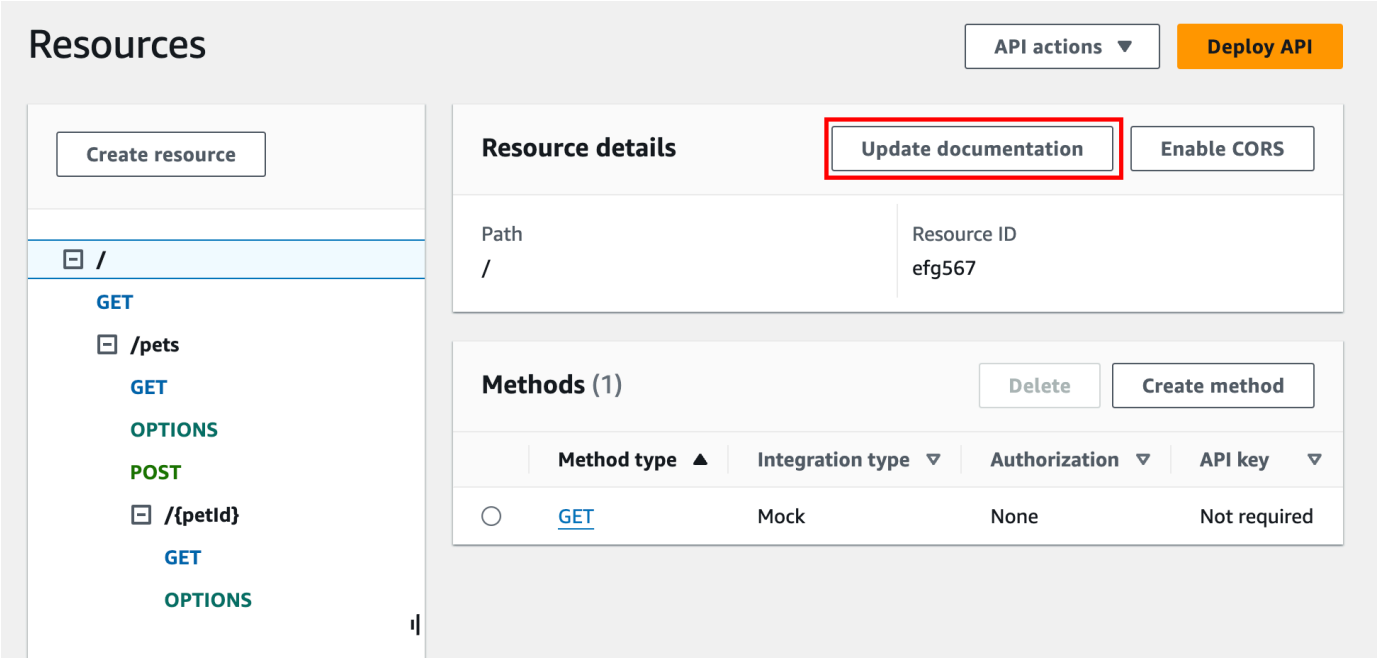
1. 在主導覽窗格中，選擇文件，然後選擇建立文件組件。
2. 針對文件類型，選取資源。
3. 在路徑中輸入路徑。
4. 在文字編輯器中輸入說明，例如：

```
{
    "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：

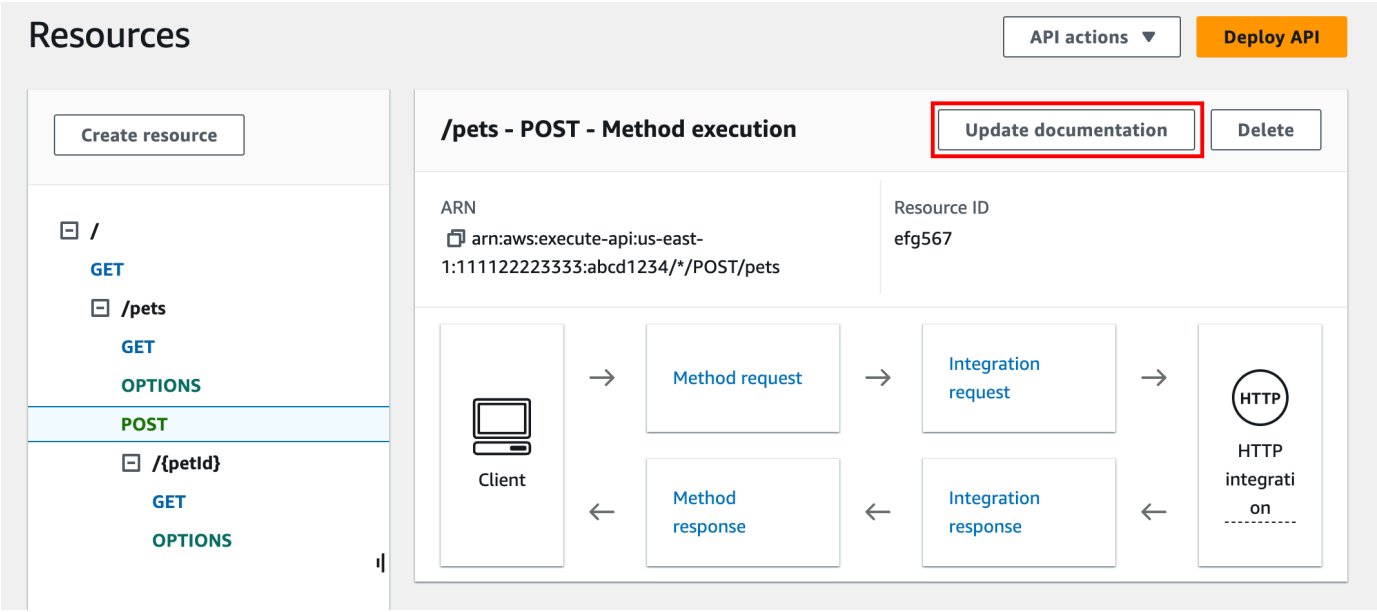
1. 在主導覽窗格中，選擇文件，然後選擇建立文件組件。
2. 針對文件類型，選取方法。
3. 在路徑中輸入路徑。
4. 針對方法，選取 HTTP 動詞。
5. 在文字編輯器中輸入說明，例如：

```
{
  "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."
  }
}
```

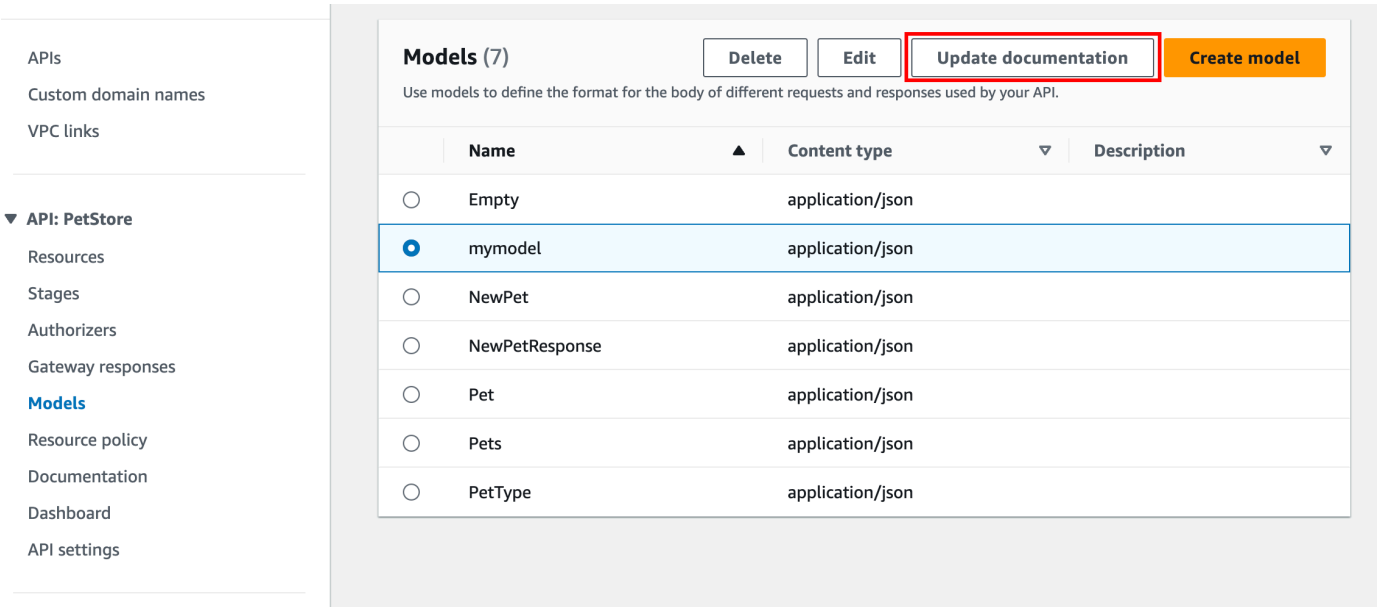
匯出 API 之後，DocumentationPart 的屬性會覆寫原始結構描述中的值。

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 主控台發佈 API 文件

下列程序說明如何發佈文件版本。

使用 API Gateway 主控台發佈文件版本

1. 在主導覽窗格中，選擇文件。
2. 選擇發佈文件。
3. 設定發佈：
   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 (發佈)。

您現在可以將文件匯出到外部 OpenAPI 檔案，繼續下載已發佈的文件。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 PetStoreAPI 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

要為 API 添加文檔，請為 API 實體添加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

要為 API 的根資源添加文檔，請添加針對相應DocumentationPart資源資源的目標資源的資源：

```
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。

要為 API 的子資源創建文檔，請添加針對相應DocumentationPart資源資源的目標資源的資源：

```
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}"
}
```

若要為 path 參數指定的子資源新增說明文件，請新增DocumentationPart資源資源目標的資源：

```
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**
>
> 實體的DocumentationPart執行個RESOURCE體無法由其任何子系資源繼承。

記錄 METHOD 實體

要為 API 的方法添加文檔，請添加針對相應 M ethod DocumentationPart資源的目標資源的資源：

```
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

若要新增要求查詢參數的文件，請新增針對該QUERY_PARAMETER類型的目標DocumentationPart資源，其中包含path和的有效欄位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說明文件，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

若要新增要求主體的文件，請新增要求主體的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" : "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

若要新增要求標頭的文件，請新增要求標頭的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

若要新增狀態碼回應的文件，請新增針對對應DocumentationPart資源的目標MethodResponse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，請為回應標頭新增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 授權者的文件，請新增指定授權者的目標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**
>
> 實體的DocumentationPart執行個AUTHORIZER體無法由其任何子系資源繼承。

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，請新增指定模型的目標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}"
}
```

重複相同步驟，為模型的任何屬性建立 DocumentationPart 例證。

> **Note**
>
> 實體的DocumentationPart執行個MODEL體無法由其任何子系資源繼承。

更新文件組件

若要更新任何類型 API 實體的文件集部分，請在指定零件識別元的DocumentationPart執行個體上提交 PATCH 要求，以新的properties對應取代現有對應。

```
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

若要列出任何類型 API 實體的文件部分，請在DocumentationParts集合上提交 GET 要求。

```
GET /restapis/restapi_id/documentation/parts HTTP/1.1
```

```
Host: apigateway.region.amazonaws.com
Content-Type: application/json
X-Amz-Date: YYYYMMDDTttttttZ
Authorization: AWS4-HMAC-SHA256 Credential=access_key_id/YYYYMMDD/region/
apigateway/aws4_request, SignedHeaders=content-length;content-type;host;x-amz-date,
 Signature=sigv4_secret
```

成功的回應會傳回 200 OK 狀態碼與包含可用的 DocumentationPart 執行個體的承載。

使用 API Gateway REST API 發佈 API 文件

若要發佈 API 的文件，請建立、更新或取得文件快照，然後建立文件快照與 API 階段的關聯。建立文件快照時，您也可以同時將其與 API 階段建立關聯。

主題

- 建立文件快照並與 API 階段建立關聯。
- 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

要創建 API 文檔部分的快照，請創建一個新DocumentationVersion資源並將其添加到 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集。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

若要取得文件快照集，請針對指定的DocumentationVersion資源提交GET要求。下列範例示範如何取得指定版本識別符 1.0.0 的文件快照。

```
GET /restapis/<restapi_id>/documentation/versions/1.0.0 HTTP/1.1
Host: apigateway.region.amazonaws.com
Content-Type: application/json
X-Amz-Date: YYYYMMDDTttttttZ
Authorization: AWS4-HMAC-SHA256 Credential=access_key_id/YYYYMMDD/region/
apigateway/aws4_request, SignedHeaders=content-length;content-type;host;x-amz-date,
 Signature=sigv4_secret
```

建立文件快照與 API 階段的關聯

若要發佈 API 文件，請建立文件快照與 API 階段的關聯。您必須已建立 API 階段，才能建立文件版本與該階段的關聯。

若要使用 API Gateway REST API 建立文件快照與 API 階段的關聯，請呼叫 stage:update 操作在 stage.documentationVersion 屬性上設定所需的文件版本：

```
PATCH /restapis/RESTAPI_ID/stages/STAGE_NAME
Host: apigateway.region.amazonaws.com
Content-Type: application/json
X-Amz-Date: YYYYMMDDTttttttZ
Authorization: AWS4-HMAC-SHA256 Credential=access_key_id/YYYYMMDD/region/
apigateway/aws4_request, SignedHeaders=content-length;content-type;host;x-amz-date,
 Signature=sigv4_secret

{
    "patchOperations": [{
        "op": "replace",
        "path": "/documentationVersion",
        "value": "VERSION_IDENTIFIER"
```

```
    }]
}
```

下載與階段相關聯的文件快照

將文件組件版本與階段建立關聯之後，您可以使用 API Gateway 主控台、API Gateway REST API、其中一個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佈的 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。您可以在有效的 OpenAPI 定義檔案中指定x-amazon-apigateway-documentation 物件副檔名內的 to-be-imported 文件部分。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 "匯入期間的錯誤與警告"。
6. 選擇 Import (匯入)。

控制 API 文件的存取

如果您有專屬文件團隊負責撰寫及編輯 API 文件，您可以為開發人員 (針對 API 開發) 與撰寫人員或編輯人員 (針對內容開發) 設定不同的存取許可。這特別適用於有第三方廠商為您建立文件時。

若要授予您的文件團隊建立、更新和發佈 API 文件的存取權，您可以透過下列 IAM 政策為文件小組指派 IAM 角色，其中 *account_id* 是文件團隊的 AWS 帳戶 ID。

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

    {
      "Sid": "StmtDocPartsAddEditViewDelete",
      "Effect": "Allow",
      "Action": [
        "apigateway:GET",
        "apigateway:PUT",
        "apigateway:POST",
        "apigateway:PATCH",
        "apigateway:DELETE"
      ],
      "Resource": [
        "arn:aws:apigateway::account_id:/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 中針對 REST API 產生軟體開發套件

若要以特定平台或特定語言的方式來呼叫您的 REST API，您必須產生特定平台或特定語言的 API 開發套件。您可以在建立、測試 API 並將其部署到階段之後產生 SDK。目前，API Gateway 支持生成一個 SDK 在 Java 中的 API JavaScript，Java 的安卓系統，目標 C 或斯威夫特為 iOS，和紅寶石。

本節說明如何產 API Gateway API 的軟體開發套件。它還演示了如何在 Java 應用程序中使用生成的 SDK，適用於 Android 應用程序的 Java，Objective-C 和斯威夫特為 iOS 應用程序，以及一個 JavaScript 應用程序。

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 Amazon 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
- 產生 API 的 Java 開發套件
- 產生 API 的 Android 開發套件
- 產生 API 的 iOS 開發套件
- 生成一個休息 JavaScript API 的開發套件
- 產生 API 的 Ruby 開發套件
- 使 AWS CLI 用命令為 API 生成 SDK

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：


Resources
Create resource
/
GET
POST
/{a}
ANY
/{b}
ANY
/{op}
GET

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 API 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. 針對名稱，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. 針對名稱，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"
```

```
            }
          }
        }
      }
    }
}
```

產生 API 的 Java 開發套件

在 API Gateway 中產生 API 的 Java 軟體開發套件

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。

產生 API 的 Android 開發套件

在 API Gateway 中產生 API 的 Android 軟體開發套件

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。

產生 API 的 iOS 開發套件

在 API Gateway 中產生 API 的 iOS 軟體開發套件

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。

     前綴的效果如下：例如，如果將 API 的前綴指定**`SIMPLE_CALC`**為、和`result`模型的 SimpleCalcAPI 的前綴`input`，`output`則生成的 SDK 將包含封裝 API 的`SIMPLE_CALCSimpleCalcClient`類，包括方法請求/響應。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。

生成一個休息 JavaScript API 的開發套件

若要在 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. 對於「平台」，請選擇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 針對 REST API 使用 API 由 API 產生的 API 所產生的 JavaScript SDK 中的說明使用所產生的開發套件。

每次更新 API，都必須重新部署 API 並重新產生開發套件來包含更新。

產生 API 的 Ruby 開發套件

在 API Gateway 中產生 API 的 Ruby 軟體開發套件

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 AWS CLI 用命令為 API 生成 SDK

您可以使 AWS CLI 用呼叫 get-sdk 命令，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

- 生成並下載 Java 的安卓軟件開發套件 AWS CLI
- 產生及下載 JavaScript SDK AWS CLI
- 產生並下載使用 AWS CLI

生成並下載 Java 的安卓軟件開發套件 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。

產生及下載 JavaScript SDK AWS CLI

若要在指定階段 (udpuvvzbkc) 產生並下載 API Gateway 所產生的 JavaScript SDK (test)，請依照下列方式呼叫命令：

```
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。

產生並下載使用 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。如需更多詳細資訊，請參閱 在 Amazon API Gateway 中叫用 REST API。

透過銷售您的 API Gateway API AWS Marketplace

建置、測試和部署 API 之後，您可以將它們封裝在 API Gateway 使用計劃中，並透過以軟體即服務 (SaaS) 產品形式銷售計劃 AWS Marketplace。訂閱您產品的 API 購買者會 AWS Marketplace 根據對使用方案的要求數量計費。

若要在上銷售 API AWS Marketplace，您必須設定銷售管道以 AWS Marketplace 與 API Gateway 整合。一般而言，這包括列出您的產品 AWS Marketplace、設定 IAM 角色與適當的政策，以允許 API Gateway 傳送使用指標 AWS Marketplace、將 AWS Marketplace 產品與 API Gateway 使用計劃產生關聯，以及將 AWS Marketplace 購買者與 API Gateway API 金鑰建立關聯。下列各節將會詳細討論。

有關將 API 作為 SaaS 產品銷售的更多信息 AWS Marketplace，請參閱用AWS Marketplace 戶指南。

主題

- 初始化 AWS Marketplace 與 API Gateway 的整合
- 處理客戶的用量計劃訂閱

初始化 AWS Marketplace 與 API Gateway 的整合

下列工作適用於與 API Gateway AWS Marketplace 整合的一次性初始化，可讓您將 API 當做 SaaS 產品銷售。

在以下位置列出產品 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`。此維度定義了 API Gateway 向您的 API 計量請求的 price-per-request 和使用。

建立計量角色

您可以使用下列執行政策與信任政策來建立名為 `ApiGatewayMarketplaceMeteringRole` 的 IAM 角色。此角色可讓 API Gateway 代表您傳送使用 AWS Marketplace 量度量給您。

計量角色的執行政策

```
{
  "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

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

將使用計劃與 AWS Marketplace 產品相關聯

當您在上刊登產品時 AWS Marketplace，您會收到 AWS Marketplace 產品代碼。若要將 API Gateway 與整合 AWS Marketplace，請將您的使用計劃與 AWS Marketplace 產品代碼建立關聯。您可以使用 API Gateway 主控台、API Gateway `UsagePlan` REST API、API Gateway REST API、用於 API Gateway 或 API Gateway 的 AWS SDK，將 API 閘道的`productCode`欄位設定 AWS CLI 為 AWS Marketplace 產品程式碼，以啟用關聯。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，會將`POST`要求 AWS Marketplace 轉寄至您在刊登產品時註冊的 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 將`subscribe-success`通知傳送至您可以訂閱的 Amazon SNS 主題。(請參閱 SaaS 客戶上線)。若要接受客戶訂閱請求，您可以透過為客戶建立或擷取 API Gateway API 金鑰，將客戶 AWS Marketplace佈建的 API 金鑰`customerId`與 API 金鑰建立關聯，然後將 API 金鑰新增至您的使用方案來處理`subscribe-success`通知。

當客戶的訂閱請求完成時，開發人員入口網站應用程式應該將相關聯的 API 金鑰提供給客戶，並通知客戶必須在 API 請求的 `x-api-key` 標頭中包含此 API 金鑰。

當客戶取消使用方案的訂閱時， 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。

########## API Gateway REST API######## AWS Marketplace customerId### API # # (

```
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"
}
```

下面的示例演示了如何獲取具有特定 AWS Marketplace `customerId`值 (*#### ID## API ###*

```
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customerId欄位更新為 AWS Marketplace 客戶的客戶 ID。ApiKey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"
    }]
}
```

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

- 配置 REST API 的交互 TLS 驗證
- 產生和設定後端身分驗證的 SSL 憑證
- 用 AWS WAF 來保護您的 API
- 調節 API 請求以獲得較佳輸送
- 亞馬遜 API 網關中的私有 REST API

配置 REST API 的交互 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。相互 TLS 是物聯網 (IoT) 和 business-to-business 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
- 故障診斷憑證警告
- 疑難排解網域名稱衝突
- 疑難排解網域名稱狀態訊息

交互 TLS 的先決條件

若要設定交互 TLS，您需要：

- 自訂網域名稱
- 至少 AWS Certificate Manager 為您的自訂網域名稱設定一個憑證
- 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 核發的憑證

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 Private Certificate Authority 憑證

若要使用匯入 ACM 的憑證或來自 AWS Private Certificate Authority 相互 TLS 的憑證，API Gateway 需要 ACM 所`ownershipVerificationCertificate`核發的憑證。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，請建立信任存取 API 的 X.509 憑證的信任庫。

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。信任庫支持以下算法：

- 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。

驗證	描述
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

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 AWS CLI 命令會上傳`certificates.pem`到您的 Amazon S3 儲存貯體。

```
aws s3 cp certificates.pem s3://bucket-name
```

您的 Amazon S3 儲存貯體必須具有 API Gateway 的讀取權限，才能允許 API Gateway 存取您的信任存放區。

設定自訂網域名稱的交互 TLS

若要為 REST API 設定相互 TLS，您必須針對 API 使用區域自訂網域名稱，並具有`TLS_1_2`安全性原則。如需有關選擇安全性原則的詳細資訊，請參閱the section called “選擇安全性原則”。

> **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。將以下內容 (包含斜線) 貼到終端機中：

```
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

若要叫用啟用交互 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，才會叫用您的 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。

下列 AWS CLI 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

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。

```
aws apigateway update-domain-name \
    --domain-name api.example.com \
    --patch-operations op='replace',path='/mutualTlsAuthentication/
truststoreUri',value=''
```

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 AWS Support 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。

產生和設定後端身分驗證的 SSL 憑證

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”」。

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
- API Gateway 支援的 HTTP 與 HTTP 代理整合憑證授權機構

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 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 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。

   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。如需詳細資訊，請參閱 the section called “將 REST API 重新部署至階段”。

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 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);
```

對於節點快速應用程式，您可以使用這些client-certificate-auth模組透過 PEM 編碼的憑證來驗證用戶端要求。

針對其他 HTTPS 伺服器，請參閱伺服器的文件。

輪換到期的用戶端憑證

由 API Gateway 產生的用戶端憑證有效時間為 365 天。您必須在 API 階段的用戶端憑證過期之前輪換憑證，以避免 API 停機。您可以呼叫用戶端憑證：API Gateway REST API 的識別碼或 AWS CLI 指令get-client-certificate並檢查傳回的過期日期屬性，以檢查憑證的 expirationDate 期。

若要輪換用戶端憑證，請執行下列作業：

1. 呼叫用戶端憑證來產生新的用戶端憑證：產生 API Gate way REST API 或的命令。 AWS CLI generate-client-certificate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"
       }
     ]
   }
   ```

   或者，您也可以呼叫 update-stage CLI 命令。
4. 更新後端伺服器以移除舊的用戶端憑證。

5. 通過調用客戶端證書從 API Gateway 刪除舊證書：刪除 API Gate way REST API，指定舊證書的 clientCertificateId（a1b2c3）：

```
DELETE /clientcertificates/a1b2c3
```

或者通過調用以下命令的 CLI 命令 delete-client-certificate：

```
aws apigateway delete-client-certificate --client-certificate-id a1b2c3
```

若要在主控台中針對先前部署的 API 輪換用戶端憑證，請執行下列作業：

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。

您需要重新部署 API，變更才會生效。如需詳細資訊，請參閱 the section called “將 REST API 重新部署至階段”。

API Gateway 支援的 HTTP 與 HTTP 代理整合憑證授權機構

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
```

用 AWS WAF 來保護您的 API

AWS WAF 是一種網絡應用程序防火牆，可幫助保護 Web 應用程序和 API 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，請參閱如何 AWS WAF 運作。

您可以使用 AWS WAF 來保護您的 API Gateway REST API 不受常見的 Web 入侵攻擊，例如 SQL 插入和跨網站指令碼 (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。

您也可以在 HTTP 標頭、方法、查詢字串、URI 和要求主體 (限制在前 64 KB) 中建立符合指定字串或規則運算式模式的規則。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 是您針對網絡漏洞利用的第一道防線。在 API 上啟 AWS WAF 用時， AWS WAF 規則會先評估其他存取控制功能 (例如資源政策、IAM 政策、Lambda 授權者和 Amazon Cognito 授權者)。例如，如果 AWS WAF 封鎖來自資源原則允許之 CIDR 區塊的存取，則優 AWS WAF 先順序且不會評估資源原則。

要 AWS WAF 為您的 API 啟用，您需要執行以下操作：

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 受管規則和您自訂規則的組合。如需詳細資訊，請參閱開始使用 AWS WAF和 Web 存取控制清單 (Web ACL)。

   > **Important**
   >
   > API Gateway 需要區域應用程式或 AWS WAFV2 網路 ACL 的 AWS WAF Classic 區域性網頁 ACL。

2. 建立 AWS WAF 網頁 ACL 與 API 階段的關聯。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 網頁 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 與現有的 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。

   如果您正在使用 AWS WAFV2，請為區域應用程式選取 AWS WAFV2 Web ACL。Web ACL 及其使用的任何其他 AWS WAFV2 資源必須與您的 API 位於相同的區域中。

   如果您正在使用 AWS WAF Classic 區域性，請選取區域網頁 ACL。
6. 選擇儲存變更。

使用「」將 AWS WAF 網頁 ACL 與 API Gateway API 階段建立關聯 AWS CLI

若要使用 AWS CLI 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，請呼叫associate-web-acl命令，如下列範例所示：

```
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
```

若要使用 AWS CLI 將 AWS WAF Classic 區域性 Web ACL 與現有 API Gateway API 階段產生關聯，請呼叫associate-web-acl命令，如下列範例所示：

```
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 網路 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 階段相關聯，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 來建立 AWS WAF Classic 區域性 網頁 ACL 與現有 API Gateway API 階段的關聯，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 API 請求以獲得較佳輸送

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

在設定 API 的調節和配額前，先了解 Amazon API Gateway 如何套用設定會有所幫助。

Amazon API Gateway 提供四種基本類型的調節相關設定：

- AWS 節流限制適用於一個區域中的所有帳戶和客戶。這些限制設定的存在是為防止 API 和帳戶接收的請求過多。這些限制由客戶設定， AWS 且無法變更。
- 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 AWS 節流限制。
- 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 AWS 節流限制。
- 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。

系統會以如下順序套用 API Gateway 調節相關設定：

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 區域節流

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、SDK 和建立 AWS Management Console 使用方案。如需如何建立使用量計劃的詳細資訊，請參閱???。

設定階段層級限流目標

您可以使用 AWS CLI、SDK 和建立階段層級節流目標。 AWS Management Console

如需如何使用建立階段層級節流目 AWS Management Console 標的相關資訊，請參閱。???如需如何使用 AWS CLI 建立階段層級節流目標的相關資訊，請參閱建立階段。

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，例PetStore如，您可以指定`Resource=/pets`、`Method=GET`。

亞馬遜 API 網關中的私有 REST API

私有 API 是只能從 Amazon VPC 內調用的 REST API。您可以使用介面 VPC 端點存取 API，這是您在 VPC 中建立的端點網路介面。介面端點採用技術AWS PrivateLink，可讓您使用私有 IP 位址私有存取 AWS 服務。

您也可以使 AWS Direct Connect 用建立從現場部署網路到 Amazon VPC 的連線，然後透過該連線存取您的私有 API。在任何情況下，流量到您的私有 API 都使用安全連接，並與公共互聯網隔離。流量不會離開 Amazon 網絡。

私有 API 的最佳做法

建議您在建立私有 API 時使用下列最佳做法：

- 使用單一 VPC 端點存取多個私有 API。這樣可以減少您可能需要的 VPC 端點數量。
- 將您的 VPC 端端點與 API 建立關聯。這會建立 Route 53 別名 DNS 記錄，並簡化呼叫私有 API 的程序。

- 為您的虛擬私人雲端開啟私人 DNS。這樣，您可以在 VPC 中調用 API，而無需傳遞主機或`x-apigw-api-id`標頭。如果您選擇不要啟用私有 DNS，您將僅能透過公有 DNS 存取您的 API。
- 限制對特定 VPC 或 VPC 端點對私有 API 的存取。在 API 的資源策略中添加`aws:SourceVpc`或`aws:SourceVpce`條件以限制訪問。
- 對於最安全的資料周邊，您可以建立 VPC 端點原則。這會控制對可呼叫私有 API 之 VPC 端點的存取。

私有 API 的注意事項

下列考量可能會影響您使用私有 API：

- 僅支援其他 API。
- 私有 API 不支援自訂網域名稱。
- 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。
- 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

要了解如何創建私有 API 並關聯 VPC 端點，請參閱，the section called “創建一個私有的 API”。若要遵循您在中建立相依性的教學課程， AWS CloudFormation 並在中建立私有 API AWS Management Console，請參閱the section called “教學課程：建置私有的 REST API”。

創建一個私有的 API

在建立私有 API 之前，請先為 API Gateway 建立 VPC 端點。接下來，您可以創建私有 API 並將資源策略附加到其中。或者，您可以將 VPC 端點與私有 API 建立關聯，以簡化呼叫 API 的方式。最後，您部署您的 API。

下列程序說明如何完成此作業。您可以使用 AWS CLI 或 AWS SDK 建立私有 REST API。 AWS Management Console

必要條件

若要遵循這些步驟，您必須擁有完全設定的 VPC。若要了解如何建立 VPC，請參閱 Amazon VPC 使用者指南中的「僅建立 VPC」。若要在建立 VPC 時遵循所有建議步驟，請啟用私有 DNS。這樣，您可以在 VPC 中調用 API，而無需傳遞主機或`x-apigw-api-id`標頭。

若要啟用私有 DNS`enableDnsSupport`，必須將 VPC 的和`enableDnsHostnames`屬性設定為`true`。如需詳細資訊，請參閱 VPC 中的 DNS Support 和更新 VPC 的 DNS 支援。

步驟 1：在 VPC 中為 API Gateway 建立 VPC 端點

下列程序顯示如何建立 API Gateway 的 VPC 端點。若要為 API Gateway 建立 VPC 端點，請指定建立私有 API 的`execute-api`網域。 AWS 區域 `execute-api`網域是用於 API 執行的 API Gateway 元件服務。

當您為 API Gateway 建立 VPC 端點時，您可以指定 DNS 設定。如果您關閉私有 DNS，則只能使用公共 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 端點連接到我的公用 API”。

AWS Management Console

若要建立 API Gateway 的介面 VPC 人雲端端點

1. 登入 AWS Management Console 並開啟 Amazon VPC 主控台，網址為 https://console.aws.amazon.com/vpc/。
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. (選擇性) 在名稱標籤中，輸入名稱以協助識別您的 VPC 端點。
5. 在 Service category (服務類別) 中，選擇 AWS services。
6. 在「服務」下方的搜尋列中，輸入**`execute-api`**。然後，在您要建立 API 的 AWS 區域 位置中選擇 API Gateway 服務端點。服務名稱應該是類似的`com.amazonaws.us-east-1.execute-api`，類型應該是接口。
7. 針對 VPC，選擇要在其中建立端點的 VPC。
8. (選擇性) 若要關閉「啟用私人 DNS 名稱」，請選擇「其他設定」，然後清除「啟用私人 DNS 名稱」。
9. 對於子網路，請選擇您在其中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. 針對「原則」，執行下列其中一個動作：
    - 如果您尚未建立私有 API，或不想設定自訂 VPC 端點原則，請選擇 [完整存取]。
    - 如果您已建立私有 API 並想要設定自訂 VPC 端點原則，則可以輸入自訂 VPC 端點原則。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 端點之後更新 VPC 端點原則。如需詳細資訊，請參閱更新 VPC 端點原則。
12. 選擇建立端點。
13. 複製產生的 VPC 端點 ID，因為您可能會在以 future 的步驟中使用它。

AWS CLI

下列create-vpc-endpoint命令可用於建立 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，因為您可能會在以 future 的步驟中使用它。

步驟 2：建立私有 API

建立虛擬私人雲端端點之後，您可以建立私有 REST API。下列程序顯示如何建立私有 API。

AWS Management Console

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. (選擇性) 對於 VPC 端點識別碼，請輸入 VPC 端點識別碼。

   如果將 VPC 端點 ID 與私有 API 建立關聯，則可以從 VPC 中叫用 API，而不必覆寫`Host`標頭或傳遞如需`x-apigw-api-id header`詳細資訊，請參閱。the section called “(選擇性) 將 VPC 端點與私有 API 建立關聯或取消關聯”
8. 選擇建立 API。

完成上述步驟後，您可以按照中the section called “REST API 主控台入門”的說明設定此 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，並將資源政策附加至您的 API。

AWS CLI

下面的update-rest-api命令顯示了如何創建一個私有 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"] }'
```

成功的呼叫會傳回類似如下的輸出：

```
{
    "createdDate": "2017-10-13T18:41:39Z",
    "description": "Simple private PetStore API",
    "endpointConfiguration": {
        "types": "PRIVATE"
    },
    "id": "0qzs2sy7bh",
    "name": "Simple PetStore (AWS CLI, Private)"
}
```

完成上述步驟後，您可以按照中the section called “教學課程：使用 SDK 或建立邊緣最佳化 API AWSAWS CLI”的說明設定此 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，並將資源政策附加至您的 API。

SDK JavaScript v3

下列範例會示範如何使用適用於 JavaScript v3 的 AWS SDK 來建立私有 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'
}
```

完成上述步驟後，您可以按照中the section called “教學課程：使用 SDK 或建立邊緣最佳化 API AWSAWS CLI”的說明設定此 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，並將資源政策附加至您的 API。

Python SDK

下列範例會示範如何使用適用於 Python 的 AWS SDK 來建立私有 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\"}"
```

完成上述步驟後，您可以按照中the section called “教學課程：使用 SDK 或建立邊緣最佳化 API AWSAWS CLI”的說明設定此 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，並將資源政策附加至您的 API。

步驟 3：設定私有 API 的資源策略

所有 VPC 都無法存取您目前的私有 API。使用資源策略授與您的 VPC 和 VPC 端點對私有 API 的存取權。您可以授與任何 AWS 帳戶中 VPC 端點的存取權。

您的資源策略應包含`aws:SourceVpc`或限制存取的`aws:SourceVpce`條件。建議您識別特定的 VPC 和 VPC 端點，並且不要建立允許存取所有 VPC 和 VPC 端點的資源策略。

下列程序顯示如何將資源策略附加至您的 API。

AWS Management Console

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. 選擇選擇一個模板，然後選擇源 VPC。
6. 將 `{{vpceID}}` (包括大括號) 取代為您的 VPC 端點識別碼。
7. 選擇儲存變更。

AWS CLI

以下update-rest-api命令顯示如何將資源策略附加到現有的 API：

```
aws apigateway update-rest-api \
    --rest-api-id a1b2c3 \
    --patch-operations op=replace,path=/
policy,value='"{\"jsonEscapedPolicyDocument\"}"'
```

您可能還想要控制哪些資源可以存取您的 VPC 端點。若要控制哪些資源可存取您的 VPC 端點，請將端點政策附加到您的 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”。

(選擇性) 將 VPC 端點與私有 API 建立關聯或取消關聯

當您將 VPC 端點與私有 API 建立關聯時，API Gateway 會產生新的 Route 53 別名 DNS 記錄。您可以使用此記錄來叫用私有 API，就像執行公用 API 一樣，而不會覆寫`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 Management Console)

您可以在建立 VPC 端點時或建立私有 API 後，將其與私有 API 建立關聯。下列程序顯示如何將 VPC 端點與先前建立的 API 建立關聯。

將 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 Management Console)

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命令顯示如何在建立 API 時關聯 VPC 端點：

```
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命令顯示如何將 VPC 端點與您已建立的 API 相關聯：

```
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)

下列update-rest-api命令顯示如何取消 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 API，您需要建立 API 部署並將其與階段產生關聯。下列程序顯示如何部署您的私有 API。

AWS Management Console

若要部署私有 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

以下創建部署命令顯示了如何部署私有 API：

```
aws apigateway create-deployment --rest-api-id a1b2c3 \
  --stage-name test \
  --stage-description 'Private API test stage' \
  --description 'First deployment'
```

疑難排解您的私有 API

以下提供建立私有 API 時可能遇到的錯誤和問題的疑難排解建議。

問題：我無法從 API Gateway VPC 端點連接到我的公用 API

建立 VPC 時，您可以設定 DNS 設定。我們建議您為虛擬私人雲端開啟私有 DNS。如果您選擇關閉私有 DNS，您只能透過公用 DNS 存取您的 API。

如果啟用私有 DNS，則無法從 VPC 端點存取公用 API Gateway API 的預設端點。您可以使用自訂網域名稱存取 API。

如果您建立區域自訂網域名稱，請使用 A 類型別名記錄，如果您建立邊緣最佳化的自訂網域名稱，您的記錄類型沒有任何限制。您可以在啟用私有 DNS 的情況下存取這些公用 API。如需詳細資訊，請參閱問題：我從 API Gateway VPC 端點連線到我的公用 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:********/****/****/"}**

在您的資源策略中，如果您將主參與者設定為 AWS 主參與者，如下所示：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            ""Principal": {
                "AWS": [
                    "arn:aws:iam::account-id:role/developer",
                    "arn:aws:iam::account-id:role/Admin"
                ]
            },
            "Action": "execute-api:Invoke",
            "Resource": [
                "execute-api:/*"
            ]
        },
        ...
        }
```

您必須對 API 中的每個方法使用AWS_IAM授權，否則您的 API 會返回先前的錯誤消息。如需如何開啟方法AWS_IAM授權的更多指示，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

如果您將 VPC 端點與私有 API 建立關聯或取消關聯，則需要重新部署 API。由於 DNS 傳播，更新作業可能需要幾分鐘才能完成。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 無法在 DNS 中解析，建議您重新部署 API。

調用一個私有的 API

您只能從 VPC 內叫用私有 API。您的私有 API 必須具有允許特定 VPC 和 VPC 端點叫用您的 API 的資源策略。

您可以通過以下方式調用您的私有 API：

- 使用路由 53 別名叫用您的 API。只有在您將 VPC 端點與 API 相關聯時，才能使用此功能。如需詳細資訊，請參閱 the section called “(選擇性) 將 VPC 端點與私有 API 建立關聯或取消關聯”。
- 使用私有 DNS 呼叫您的 API。只有在您的 VPC 啟用私有 DNS 時，才能使用此功能。
- 使用調用您的 API AWS Direct Connect。
- 使用端點特定的公有 DNS 主機名稱叫用您的 API。

要使用 DNS 名稱調用您的私有 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 Management Console

若要尋找 DNS 名稱

1. 登入 AWS Management Console 並開啟 Amazon VPC 主控台，網址為 https://console.aws.amazon.com/vpc/。
2. 在主導覽窗格中，選擇「端點」，然後為 API Gateway 選擇介面 VPC 端點。
3. 在 [詳細資料] 窗格中，您會在 [DNS 名稱] 欄位中看到五個值。前三個是 API 的公開 DNS 名稱。另外兩個是它的私人 DNS 名稱。

AWS CLI

使用以下describe-vpc-endpoints命令列出您的 DNS 值。

```
aws ec2 describe-vpc-endpoints --filters vpc-endpoint-id=vpce-01234567abcdef012
```

前三個是 API 的公開 DNS 名稱。另外兩個是它的私人 DNS 名稱。

使用路由 53 別名調用私有 API

您可以將 VPC 端點與私有 API 建立關聯或取消關聯。如需詳細資訊，請參閱 the section called “(選擇性) 將 VPC 端點與私有 API 建立關聯或取消關聯”。

將 VPC 端點與私有 API 建立關聯後，您可以使用下列基本 URL 來叫用 API：

```
https://{rest-api-id}-{vpce-id}.execute-api.{region}.amazonaws.com/{stage}
```

例如，如果您為test階段設置了GET /pets方法，並且您的 REST API ID 是01234567ab，並且您的 VPC 端點 ID 是vpce-01234567abcdef012，而您的區域是us-west-2，則可以調用您的 API 為：

```
curl -v https://01234567ab-vpce-01234567abcdef012.execute-api.us-west-2.amazonaws.com/
test/pets
```

使用私有 DNS 名稱叫用私有 API

如果您已啟用私有 DNS，則可以使用以下私人 DNS 名稱存取您的私人 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，如果您為test階段設定了GET /pets方法，而您的 REST API ID 是01234567ab且您的區域為us-west-2，則可以在瀏覽器中輸入以下 URL 來叫用私有 API：

```
https://01234567ab.execute-api.us-west-2.amazonaws.com/test/pets
```

或者，您可以使用以下 cURL 命令調用您的私有 API：

```
curl -X GET https://01234567ab.execute-api.us-west-2.amazonaws.com/test/pets
```

> **Warning**
>
> 如果您為 VPC 端點啟用私有 DNS，則可以存取公用 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？。

使用調用私有 API AWS Direct Connect

您可 AWS Direct Connect 以使用建立從現場部署網路到 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。

您也可以設定 Amazon Route 53 Resolver 輸入端點，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您的私人 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 主機名稱叫用私有 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，如果您為`test`階段設定`GET /pets`方法，而您的 REST API ID 是`abc1234`，其公用 DNS 主機名稱為`vpce-def-01234567`，而您的地區是`us-west-2`，則您可以使用 cURL 命令中的`Host`標頭使用其 vPCE ID 來叫用私有 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，您可以在 cURL 命令中使用以下格式的`x-apigw-api-id`標頭，透過其 API ID 叫用私有 API：

```
curl -v https://{public-dns-hostname}.execute-api.{region}.vpce.amazonaws.com/{stage} -
H 'x-apigw-api-id:{api-id}'
```

監控 REST API

在本節中，您可以了解如何使用 CloudWatch 指標、 CloudWatch 日誌、Firehose 和 AWS X-Ray. 通過結合 CloudWatch 執行日誌和 CloudWatch 指標，您可以記錄錯誤和執行跟踪，並監視 API 的性能。您也可能想要將 API 呼叫記錄到 Firehose。您也可以使 AWS X-Ray 用透過構成 API 的下游服務來追蹤呼叫。

> **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
> - 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。 CloudWatch 條目是模擬的。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。”。

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
- 記錄對 Amazon 資料 Firehose 的 API 呼叫
- 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「API Gateway」收集原始資料並將其處理為可讀的 near-real-time 指標來監控 API 執行。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 CloudWatch 在一分鐘內自動傳送至。有關更多信息，請參閱什麼是 Amazon CloudWatch？ 在 Amazon 用 CloudWatch 戶指南。

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：

- 監視指IntegrationLatency標以衡量後端的響應能力。
- 監控 Latency 指標，以測量您 API 呼叫的整體回應能力。
- 監控CacheHitCount和指CacheMissCount標以最佳化快取容量，以達到所需的效能。

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 記錄事件
- 中的監控工具 AWS

Amazon API Gateway 維度和指標

API Gateway 傳送給 Amazon 的指標和維度 CloudWatch 如下所示。如需詳細資訊，請參閱 使用 Amazon CloudWatch 指標監控 REST API 執行。

API Gateway 指標

Amazon API Gateway 會將指標資料傳送到 CloudWatch 每分鐘一次。

`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。 API Gateway 會將修改後的閘道回應狀態碼計為 4xxError 錯誤。 `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)。 Unit: Count
5XXError	在指定期間內擷取的伺服器端錯誤數目。 `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)。 Unit: Count
CacheHitCount	在指定期間內從 API 快取提供的要求數目。 `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)。

指標	描述
	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。 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)。 Unit: Count
Count	指定期間內的 API 要求總數。 SampleCount 統計資訊代表此指標。 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。 Unit: Millisecond
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。 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**
>
> API Gateway 會先從維度中移除非 ASCII 字元，然後再將量ApiName度傳送至 CloudWatch。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。 除非您已明確啟用詳細指標，否則 API Gateway 不會傳送這些 CloudWatch 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，您可以呼叫更新階段 AWS CLI 命令，將`metricsEnabled` 屬性更新為。`true` 啟用這些指標會產生您帳戶的額外費用。如需定價資訊，請參閱 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 主控台中的 API 儀表板，在 API Gateway 中顯示已部署 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

- 必要條件
- 檢查儀表板中的 API 活動

必要條件

1. 您必須擁有在 API Gateway 中建立的 API。請遵循中的說明進行在 API 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 Amazon 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 CloudWatch 度量，請確定您已在方法層級啟用 CloudWatch 記錄檔。如需設定方法層級記錄的詳細資訊，請參閱使用 API Gateway 主控台更新階段設定。

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。若要檢視 API 指標層級的指標，請開啟詳細指標。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. 開啟主 CloudWatch 控台，網址為 https://console.aws.amazon.com/cloudwatch/。
2. 如有需要，請變更 AWS 區域。在導覽列中，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。

若要使用 AWS CLI 檢視測量結果

1. 在命令提示字元中，使用下列命令來列出指標：

```
aws cloudwatch list-metrics --namespace "AWS/ApiGateway"
```

建立量度後，最多需要 15 分鐘的時間讓量度顯示。若要更快查看測量結果統計值，請使用get-metric-data或get-metric-statistics。

2. 若要每隔 5 分鐘檢視一次特定統計資料 (例如 Average)，請呼叫下列命令：

```
aws cloudwatch get-metric-statistics --namespace AWS/ApiGateway --metric-name Count
 --start-time 2011-10-03T23:00:00Z --end-time 2017-10-05T23:00:00Z --period 300 --
statistics Average
```

在 CloudWatch 主控台中檢視 API Gateway 記錄事件

必要條件

1. 您必須擁有在 API Gateway 中建立的 API。請遵循中的說明進行在 API 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 Amazon API Gateway 中部署 REST API和在 Amazon API Gateway 中叫用 REST API中的說明進行。
3. 您必須啟用階段的 CloudWatch 記錄檔。請遵循中的說明進行在 API Gateway 中設定 REST API 的 CloudWatch 記錄

使用 CloudWatch 主控台檢視記錄的 API 要求和回應

1. 開啟主 CloudWatch 控台，網址為 https://console.aws.amazon.com/cloudwatch/。
2. 如有需要，請變更 AWS 區域。在導覽列中，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「日誌群組」表格下，選擇 API 閘道執行日誌 _ {}/{階段名稱} 名稱的日誌群組。rest-api-id
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 可讓您刪除記錄群組或串流。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。

中的監控工具 AWS

AWS 提供各種可用來監視 API Gateway 的工具。您可以設定其中一些工具為您自動監控，但其他工具則需要手動介入。建議您盡可能自動化監控任務。

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 A CloudWatch lar ps — 觀看您指定期間內的單一指標，並根據指定臨界值在多個時段內相對於指定閾值的指標值執行一或多個動作。動作是傳送至亞馬遜簡單通知服務 (Amazon SNS) 主題或 Amazon EC2 Auto Scaling 政策的通知。 CloudWatch 警示不會僅因為處於特定狀態而叫用動作；狀態必須已變更並維持指定數目的期間。如需詳細資訊，請參閱 使用 Amazon CloudWatch 指標監控 REST API 執行。
- Amazon CloudWatch 日誌 — 監控、存放和存取來自 AWS CloudTrail 或其他來源的日誌檔。如需詳細資訊，請參閱何謂 CloudWatch 記錄？ 在 Amazon 用 CloudWatch 戶指南。
- Amazon EventBridge (以前稱為 CloudWatch 事件) — 匹配事件並將其路由到一個或多個目標函數或串流，以進行變更、擷取狀態資訊並採取糾正措施。有關更多信息，請參閱什麼是 Amazon EventBridge？ 在《EventBridge 使用者指南》中。
- AWS CloudTrail 防護記錄監控 — 在帳戶之間共用記 CloudTrail 錄檔、即時監控記錄檔案，方法是將 CloudWatch 記錄檔傳送至記錄檔、以 Java 撰寫記錄處理應用程式，以及驗證記錄檔在傳送之後是否未變更 CloudTrail。若要取得更多資訊，請參閱《使用指南》中的〈AWS CloudTrail 使用 CloudTrail 記錄檔〉。

手動監控工具

監視 API Gateway 的另一個重要部分是手動監視 CloudWatch 警報未涵蓋的項目。API Gateway 和其他 AWS 主控台儀表板可提供您 AWS 環境狀態的 at-a-glance 檢視。 CloudWatch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，您可以使用執行 CloudWatch 以下操作：
  - 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。
  - 搜尋並瀏覽所有資 AWS 源指標
  - 建立與編輯要通知發生問題的警示

建立 CloudWatch 警示以監控 API Gateway

您可以建立 CloudWatch 警示，在警示狀態變更時傳送 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。警示只會呼叫持續狀態變更的動作。 CloudWatch 警示不會僅因為處於特定狀態而叫用動作；狀態必須已變更並維持指定數目的期間。

在 API Gateway 中設定 REST API 的 CloudWatch 記錄

若要協助偵錯與 API 的請求執行或用戶端存取相關的問題，您可以啟用 Amazon CloudWatch Logs 記錄 API 呼叫。如需有關的更多資訊 CloudWatch，請參閱the section called “CloudWatch 度量”。

CloudWatch API Gateway 的記錄檔格式

API 記錄有兩種類型 CloudWatch：執行記錄和存取記錄。在執行記錄中，API Gateway 會管理記 CloudWatch 錄檔。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。

記錄的資料包括錯誤或執行追蹤 (例如要求或回應參數值或承載)、Lambda 授權者使用的資料 (以前稱為自訂授權者)、是否需要 API 金鑰、是否啟用使用計畫，以及其他資訊。API Gateway 會從記錄的資料中編輯授權標頭、API 金鑰值和類似的敏感要求參數。

當您部署 API 時，API Gateway 會在日誌群組下方建立日誌群組和日誌串流。日誌群組的命名是遵循 `API-Gateway-Execution-Logs_{rest-api-id}/{stage_name}` 格式。在每個日誌群組內，日誌會進一步分割為日誌串流，而其在報告所記錄的資料時會依 Last Event Time (上次事件時間) 排序。

在存取記錄中，您身為 API 開發人員且想要記錄誰已存取您的 API 以及發起人存取 API 的方式。您可以建立自己的日誌群組，或選擇由 API Gateway 管理的現有日誌群組。若要指定存取詳細資訊，請選取`$context`變數、記錄格式和記錄群組目的地。

存取日誌格式必須至少包含 `$context.requestId` 或 `$context.extendedRequestId`。最佳做法是在記錄格式`$context.extendedRequestId`中加入`$context.requestId`和。

`$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 會在您的存取記錄中取代不是 UUID 格式的已覆*`UUID`*`_REPLACED_INVALID_REQUEST_ID`寫要求 ID。

`$context.extendedRequestId`

延伸重新申請是 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 Sup AWS port 部門，以協助疑難排解您的 API。如需詳細資訊，請參閱 the section called “$context資料模型、授權者、對應範本和 CloudWatch 存取記錄的變數”。

> **Note**
>
> 僅支援`$context`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。若要定義記錄格式，請在舞台上的 accessLogSettings/destinationARN 屬性上設定記錄群組 ARN。您可以在 CloudWatch 主控台中取得記錄群組 ARN。若要定義存取記錄格式，請在舞台上的 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
```

CloudWatch 記錄的權限

若要啟用 CloudWatch 記錄，您必須授與 API Gateway 權限，才能讀取和寫入帳戶 CloudWatch 的記錄。AmazonAPIGatewayPushToCloudWatchLogs 受管政策 (ARN 為 arn:aws:iam::aws:policy/service-role/AmazonAPIGatewayPushToCloudWatchLogs) 具備所有必要許可：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:DescribeLogGroups",
                "logs:DescribeLogStreams",
                "logs:PutLogEvents",
                "logs:GetLogEvents",
                "logs:FilterLogEvents"
            ],
            "Resource": "*"
        }
    ]
}
```

> **Note**
>
> API Gateway 呼叫 AWS Security Token Service 以承擔 IAM 角色，因此請確定該區域 AWS STS 已啟用該角色。如需詳細資訊，請參閱管理 AWS 區域中的 AWS STS。

若要將這些許可授與您的帳戶，請建立 IAM 角色apigateway.amazonaws.com做為其受信任的實體，將上述政策附加到 IAM 角色，然後在帳戶的 Arn 屬性上設定 IAM 角色 cloudWatchRoleARN。您必須針對要在其中啟用 CloudWatch 記錄的每個 AWS 區域個別設定 cloudWatchRoleArn 屬性。

如果您在設定 IAM 角色 ARN 時收到錯誤訊息，請檢查您的 AWS Security Token Service 帳戶設定，確認 AWS STS 已在您使用的區域中啟用該功能。如需啟用的詳細資訊 AWS STS，請參閱 IAM 使用者指南中的管理 AWS 區域中的 AWS ST S。

使用 CloudWatch API Gateway 主控台設定 API 記錄

若要設定 CloudWatch API 記錄，您必須將 API 部署到階段。您還必須為您的帳戶配置了適當的 CloudWatch 日誌角色 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. 對於CloudWatch 記錄角色 ARN，請輸入具有適當許可的 IAM 角色的 ARN。您需要為每個使用 API 網關創建 API Gateway AWS 帳戶 的執行此操作一次。
4. 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。
5. 在主導覽窗格中，選擇階段。
6. 在日誌和追蹤區段中，選擇編輯。
7. 若要啟用執行記錄：
   a. 從 [記錄CloudWatch 檔] 下拉式功能表中選取記錄層級。記錄層級如下：
      - 關閉 — 此階段未開啟記錄功能。
      - 僅限錯誤 — 僅針對錯誤啟用記錄功能。
      - 錯誤和資訊記錄 — 已啟用所有事件的記錄功能。
      - 完整的請求和回應記錄 — 已啟用所有事件的詳細記錄。這對於故障排除 API 很有用，但可能會導致記錄機密資料。

      > **Note**
      > 我們建議您不要對生產 API 使用完整的請求和回應日誌。

   b. 如有需要，請選取詳細量度以開啟詳細 CloudWatch 量度。

   如需 CloudWatch 測量結果的詳細資訊，請參閱the section called “CloudWatch 度量”。
8. 若要啟用存取記錄：
   a. 開啟自訂存取記錄。
   b. 在存取日誌目的地 ARN 中輸入日誌群組的 ARN。ARN 的格式為 `arn:aws:logs:{region}:{account-id}:log-group:log-group-name`。
   c. 在日誌格式中，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 “CloudWatch API Gateway 的記錄檔格式”。
9. 選擇儲存變更。

> **Note**
>
> 您可以個別啟用執行記錄和存取記錄。

API Gateway 現在已準備好將請求記錄至 API。當您更新階段設定、日誌或階段變數時，不需要重新部署 API。

使用以下方 CloudWatch 式設定 API 記錄 AWS CloudFormation

使用下列範例 AWS CloudFormation 範本建立 Amazon Lo CloudWatch gs 日誌群組，並設定階段的執行和存取記錄。若要啟用 CloudWatch 記錄，您必須授與 API Gateway 權限，才能讀取和寫入帳戶 CloudWatch 的記錄。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
     Description: "test stage description"
     MethodSettings:
       - ResourcePath: "/*"
         HttpMethod: "*"
         LoggingLevel: INFO
     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
```

記錄對 Amazon 資料 Firehose 的 API 呼叫

若要協助偵錯與用戶端存取 API 相關的問題，您可以將 API 呼叫記錄到 Amazon 資料 Firehose。如需有關 Firehose 的詳細資訊，請參閱什麼是 Amazon 資料 Fire hose？ 。

對於存取記錄，您只能啟用 CloudWatch 或 Firehose st — 您無法同時啟用兩者。但是，您可以啟用 CloudWatch 執行日誌記錄，並啟用 Firehose 進行訪問日誌記錄。

主題

- API Gateway 的 Firehose 記錄格式
- Firehose 記錄的權限
- 使用 API Gateway 主控台設定 Firehose 存取記錄

API Gateway 的 Firehose 記錄格式

Firehose 記錄使用與記CloudWatch 錄相同的格式。

Firehose 記錄的權限

在階段啟用 Firehose 存取記錄時，如果角色尚未存在，API Gateway 會在您的帳戶中建立服務連結角色。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**
>
> 您的「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

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 串流的存取記錄功能：
   a. 開啟自訂存取記錄。
   b. 對於存取記錄目的地 ARN，請輸入 Firehose 串流的 ARN。ARN 的格式為 `arn:aws:firehose:{region}:{account-id}:deliverystream/amazon-apigateway-{your-stream-name}`。

      > **Note**
      > 您的「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 “CloudWatch API Gateway 的記錄檔格式”。
6. 選擇儲存變更。

API Gateway 現在已準備就緒，可將要求記錄到您的 API 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。

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。您可以在提供 X-Ray 功能的所有 AWS 區域使用 X-Ray 搭配 Amazon API Gateway。

由於 X-Ray 可讓您 end-to-end 檢視整個要求，因此您可以分析 API 及其後端服務中的延遲情況。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 API 進行設定

- 搭配 API Gateway 使用 AWS X-Ray 服務對應和追蹤檢視
- 設定 API Gateway API 的 AWS X-Ray 取樣規則
- 了解 Amazon API Gateway API 的 AWS X-Ray 追蹤

AWS X-Ray 使用 API Gateway REST API 進行設定

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

若要在建立階段時使用啟用 API 階段的使用中 X-Ray 追蹤，請呼叫命`create-stage`令，如下列範例所示： AWS CLI

```
aws apigateway create-stage \
    --rest-api-id {rest-api-id} \
    --stage-name {stage-name} \
    --deployment-id {deployment-id} \
    --region {region} \
    --tracing-enabled=true
```

以下是成功叫用的範例輸出：

```
{
    "tracingEnabled": true,
    "stageName": {stage-name},
```

```
    "cacheClusterEnabled": false,
    "cacheClusterStatus": "NOT_AVAILABLE",
    "deploymentId": {deployment-id},
    "lastUpdatedDate": 1533849811,
    "createdDate": 1533849811,
    "methodSettings": {}
}
```

若要在建立階段時使用停用 API 階段的使用中 X-Ray 追蹤，請如下列範例所示呼叫create-stage命令： AWS CLI

```
aws apigateway create-stage \
    --rest-api-id {rest-api-id} \
    --stage-name {stage-name} \
    --deployment-id {deployment-id} \
    --region {region} \
    --tracing-enabled=false
```

以下是成功叫用的範例輸出：

```
{
    "tracingEnabled": false,
    "stageName": {stage-name},
    "cacheClusterEnabled": false,
    "cacheClusterStatus": "NOT_AVAILABLE",
    "deploymentId": {deployment-id},
    "lastUpdatedDate": 1533849811,
    "createdDate": 1533849811,
    "methodSettings": {}
}
```

若要針對 AWS CLI 已部署的 API 啟用主動式 X-Ray 追蹤，請依照下列方式呼叫update-stage命令：

```
aws apigateway update-stage \
    --rest-api-id {rest-api-id} \
    --stage-name {stage-name} \
    --patch-operations op=replace,path=/tracingEnabled,value=true
```

若要針對 AWS CLI 已部署的 API 使用停用作用中的 X-Ray 追蹤，請呼叫命update-stage令，如下列範例所示：

```
aws apigateway update-stage \
    --rest-api-id {rest-api-id} \
    --stage-name {stage-name} \
    --region {region} \
    --patch-operations op=replace,path=/tracingEnabled,value=false
```

以下是成功叫用的範例輸出：

```
{
    "tracingEnabled": false,
    "stageName": {stage-name},
    "cacheClusterEnabled": false,
    "cacheClusterStatus": "NOT_AVAILABLE",
    "deploymentId": {deployment-id},
    "lastUpdatedDate": 1533850033,
    "createdDate": 1533849811,
    "methodSettings": {}
}
```

為 API 階段啟用 X-Ray 之後，請使用 X-Ray CLI 來擷取追蹤資訊。如需詳細資訊，請參閱搭配 AWS CLI 使用 AWS X-Ray API。

搭配 API Gateway 使用 AWS X-Ray 服務對應和追蹤檢視

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。

如需服務映射和追蹤檢視的詳細資訊，以及如何解譯它們的詳細資訊，請參閱 AWS X-Ray 主控台。

主題

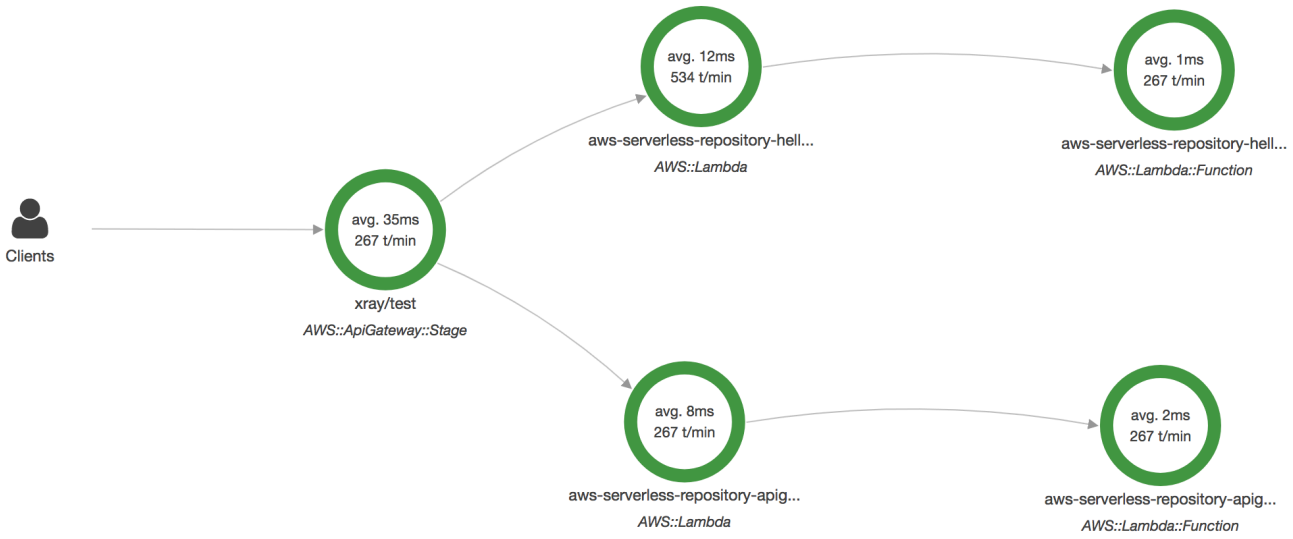
- 範例 X-Ray 服務地圖
- 範例 X-Ray 追蹤檢視

範例 X-Ray 服務地圖

AWS X-Ray 服務對應會顯示 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 服務 (例如 AWS Lambda 和 Amazon DynamoDB) 整合的 API，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。這個 API 已與 Lambda 授權方函數和 Lambda 後端函數進行 Lambda 整合。這些節點代表 API Gateway 服務、Lambda 服務，以及兩個 Lambda 函數。

如需服務對應結構的詳細說明，請參閱檢視服務對應。



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，請參閱在 AWS X-Ray 主控台中使用延遲長條圖。

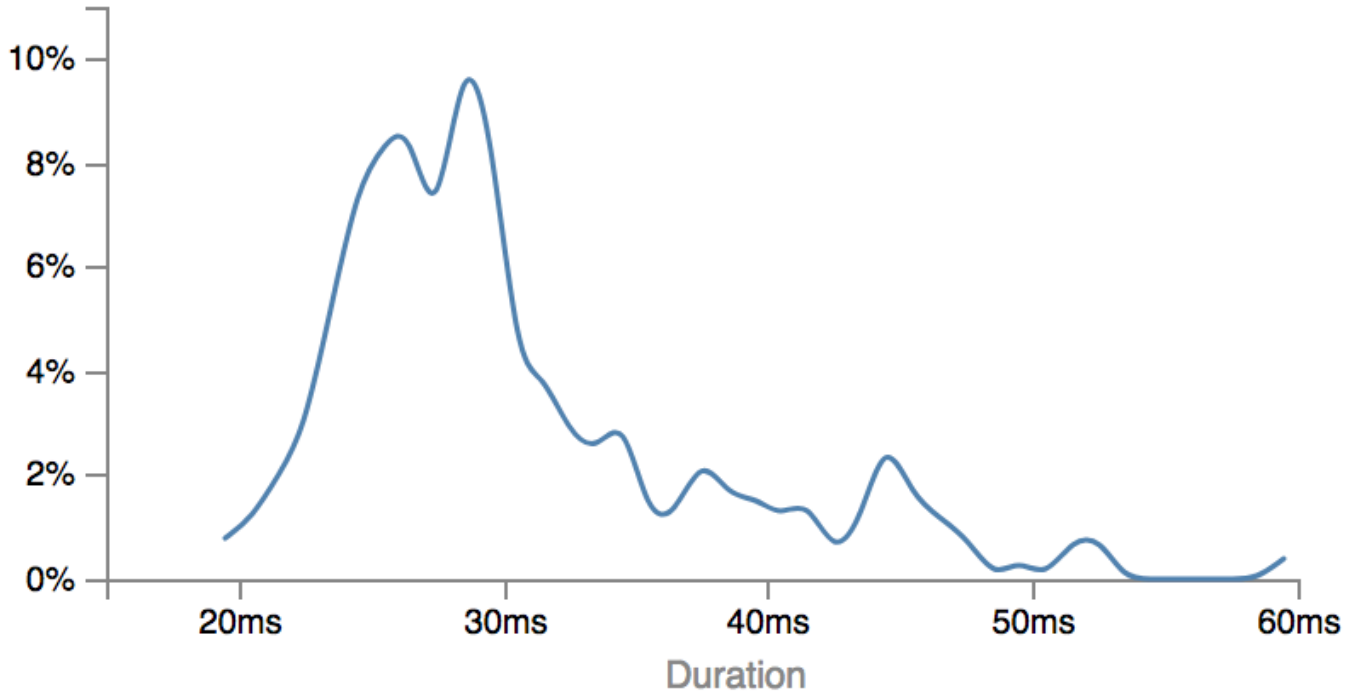
Service details

Name: xray/test

Type: AWS::ApiGateway::Stage

Response distribution

Click and drag to select an area to zoom in on or use as a latency filter when viewing traces.



Response status

Choose response statuses to add to the filter when viewing traces.

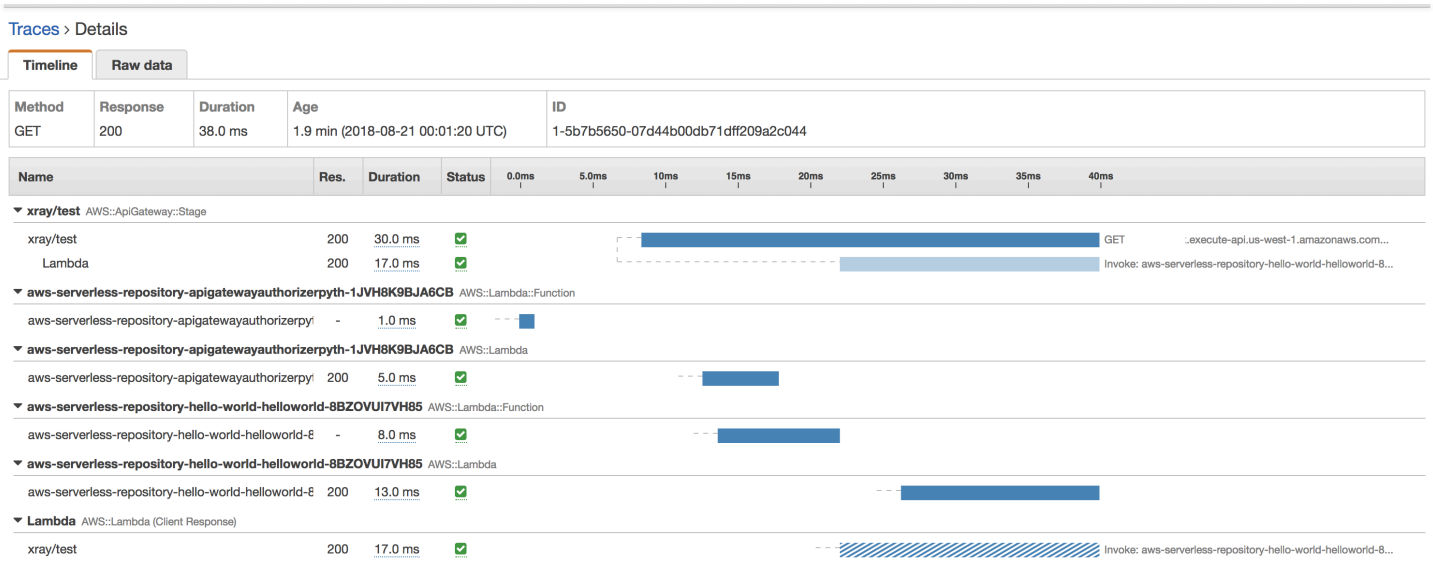
OK: 100% Error: 0%

Fault: 0% Throttle: 0%

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 後端函數和 Lambda 授權方函數。其中顯示成功的 API 方法請求，以及回應碼 200。

如需追蹤檢視的詳細說明，請參閱檢視追蹤。



設定 API Gateway API 的 AWS X-Ray 取樣規則

您可以使用 AWS X-Ray 主控台或開發套件來設定 Amazon API Gateway 的取樣規則。取樣規則指定 X-Ray 應該為您的 API 記錄哪些請求。透過自訂取樣規則，您可以控制記錄的資料量，以及迅速修改取樣行為，而無需修改或重新部署程式碼。

指定您的 X-Ray 取樣規則之前，請先參閱《X-Ray 開發人員指南》中的下列主題：

- 在 AWS X-Ray 主控台中設定抽樣規則
- 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 (*)。有關更多詳細信息，包括如何使用存儲器和速率設置的詳細說明，請在 AWS X-Ray 控制台中配置採樣規則。

- 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，如果您要將PetStore範例 API 部署到名為的階段`test`，則要在取樣規則中指定的 Service name 值為**`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 — **{}**

了解 Amazon API Gateway API 的 AWS X-Ray 追蹤

本節討論 Amazon API Gateway 的 AWS X-Ray 追蹤區段、子區段和其他追蹤欄位。

在閱讀本節之前，請先參閱《X-Ray 開發人員指南》中的下列主題：

- AWS X-Ray 主控台
- AWS X-Ray 區段文件
- X-Ray 概念

主題

- 為 API Gateway API 追蹤物件的範例
- 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，請參閱在 AWS X-Ray 主控台中設定取樣規則。

以下是 `annotations` 物件範例，其中 API 階段由 API ID 和 API 階段名稱識別：

```
"annotations": {
    "aws:api_id": "a1b2c3d4e5",
    "aws:api_stage": "dev"
}
```

AWS 資源資料

`aws` 物件僅顯示在區段中。以下是符合預設取樣規則的 `aws` 物件範例。如需深入了解取樣規則，請參閱在 AWS X-Ray 主控台中設定取樣規則。

```
"aws": {
    "xray": {
        "sampling_rule_name": "Default"
    },
    "api_gateway": {
        "account_id": "123412341234",
        "rest_api_id": "a1b2c3d4e5",
        "stage": "dev",
        "request_id": "a1b2c3d4-a1b2-a1b2-a1b2-a1b2c3d4e5f6"
    }
}
```

了解追蹤

以下是 API Gateway 階段的追蹤區段。如需組成追蹤區段之欄位的詳細說明，請參閱 AWS X-Ray 開發人員指南中的AWS X-Ray 區段文件。

```
        {
            "Document": {
                "id": "a1b2c3d4a1b2c3d4",
                "name": "testxray/dev",
                "start_time": 1533928226.229,
                "end_time": 1533928226.614,
                "metadata": {
                    "default": {
                        "extended_request_id": "abcde12345abcde=",
                        "request_id": "a1b2c3d4-a1b2-a1b2-a1b2-a1b2c3d4e5f6"
                    }
                },
                "http": {
                    "request": {
                        "url": "https://example.com/dev?
username=demo&message=hellofromdemo/",
                        "method": "GET",
                        "client_ip": "192.0.2.0",
                        "x_forwarded_for": true
                    },
                    "response": {
                        "status": 200,
                        "content_length": 0
                    }
                },
                "aws": {
                    "xray": {
                        "sampling_rule_name": "Default"
                    },
                    "api_gateway": {
                        "account_id": "123412341234",
                        "rest_api_id": "a1b2c3d4e5",
                        "stage": "dev",
                        "request_id": "a1b2c3d4-a1b2-a1b2-a1b2-a1b2c3d4e5f6"
                    }
                },
                "annotations": {
                    "aws:api_id": "a1b2c3d4e5",
                    "aws:api_stage": "dev"
                },
                "trace_id": "1-a1b2c3d4-a1b2c3d4a1b2c3d4a1b2c3d4",
                "origin": "AWS::ApiGateway::Stage",
```

```
                "resource_arn": "arn:aws:apigateway:us-east-1::/restapis/a1b2c3d4e5/
stages/dev",
                "subsegments": [
                    {
                        "id": "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"
        }
```

使用 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。如需詳細資訊，請參閱 the section called “在 REST API 與 HTTP API 之間進行選擇 ”。

您可以使用 HTTP API 將請求發送到 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 或開發套件來建立 HTTP API。 AWS CloudFormation

主題

- 在 API Gateway 中開發 HTTP API
- 發佈 HTTP API 以供客戶叫用
- 保護 HTTP API
- 監控您的 HTTP API
- 對 HTTP API 的問題進行疑難排解

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
- 使用 HTTP API 的路由
- 在 API Gateway 中控制和管理 HTTP API 的存取
- 設定 HTTP API 的整合

- 為 HTTP API 設定 CORS
- 轉換 API 請求和回應
- 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。

下列範例說明如何建立具有 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，請參閱入門。

主題

- 建立一個 HTTP API，方法是使用 AWS Management Console
- 透過使用 AWS CLI 建立一個 HTTP API

建立一個 HTTP API，方法是使用 AWS Management Console

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叫用 URL 或使用 Curl 來測試您的 API。

```
curl https://api-id.execute-api.us-east-2.amazonaws.com
```

透過使用 AWS CLI 建立一個 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。以下命令使用快速建立來建立與後端的 Lambda 函數整合的 API。

> **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。若要深入了解，請參閱 AWS Lambda 開發人員指南中的AWS Lambda 權限。

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叫用 URL 或使用 Curl 來測試您的 API。

```
curl https://api-id.execute-api.us-east-2.amazonaws.com
```

使用 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，請參閱使用 HTTP API 的 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 記錄檔。

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。

在 API Gateway 中控制和管理 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 授權者取得 HTTP API。
- 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 JWT 授權方控制對 HTTP API 的存取權。

- 標準 AWS IAM 角色和政策提供彈性且強大的存取控制。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。如需更多詳細資訊，請參閱 使用 IAM 授權。

使用 AWS Lambda 授權者取得 HTTP API

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 Management Console 會使用最新版本。如果您使用 AWS CLI、 AWS CloudFormation或 SDK 建立 Lambda 授權者，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。如需進一步了解，請參閱the section called “記錄變數”。

Example

```
{
  "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|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", // The principal user identification associated with the
 token sent by the client.
  "policyDocument": {
    "Version": "2012-10-17",
    "Statement": [
      {
        "Action": "execute-api:Invoke",
        "Effect": "Allow|Deny",
        "Resource": "arn:aws:execute-api:{regionId}:{accountId}:{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。支援下列身分來源：

選擇表達式

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。

快取授權方回應

您可以 Lambda 過指定 authorizerResultTtlInSeconds. 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。

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。在此範例中，我們會更新函數的資源政策，以便授予 API Gateway 叫用 Lambda 函數的許可。

```
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
```

下列命令可授予 API Gateway 叫用 Lambda 函數的許可。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。

```
aws apigatewayv2 update-route \
    --api-id abcdef123 \
    --route-id acd123 \
    --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 JWT 授權方控制對 HTTP API 的存取權

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 授權者授權 API 請求

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 可以快取公有金鑰兩小時。最佳作法是，當您輪換金鑰時，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。只有在不存在的情況下`client_id`才`aud`會驗證 API Gateway。當`aud`和`client_id`都存在時，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。

使用控制台創建 JWT 授權者

下面的步驟演示了如何使用控制台創建 JWT 授權者。

若要使用主控台建立 JWT 授權者

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. 在主導覽窗格中，選擇 [授權]。
4. 選擇 [管理授權者] 索引標籤。
5. 選擇建立。
6. 針對「授權者」類型，選擇「J WT」。
7. 設定您的 JWT 授權者，並指定定義權杖來源的身分識別來源。
8. 選擇建立。

使用建立 JWT 授權者 AWS CLI

下面的 AWS CLI 命令創建一個 JWT 授權者。對於 `jwt-configuration`，為您的身分提供者指定 `Audience` 和 `Issuer`。如果您使用 Amazon Cognito 做為身分識別提供者，請使用`IssuerUrl`. `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
```

使用創建一個 JWT 授權者 AWS CloudFormation

下列 AWS CloudFormation 範本會使用 Amazon Cognito 做為身分提供者的 JWT 授權者建立 HTTP API。

AWS CloudFormation 範本的輸出是 Amazon Cognito 託管使用者介面的 URL，用戶端可以在其中註冊並登入以接收 JWT。客戶端登錄後，客戶端將使用 URL 中的訪問令牌重定向到您的 HTTP API。要使用訪問令牌調用 API，請將 URL # 中的更改為 a 以使?用令牌作為查詢字符串參數。

範例 AWS CloudFormation 範本

```
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 授權者

您可以使用控制台 AWS CLI、或 AWS SDK 更新路由以使用 JWT 授權者。

使用控制台更新路由以使用 JWT 授權者

下面的步驟演示了如何更新路由使用控制台使用 JWT 授權者。

若要使用主控台建立 JWT 授權者

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. 在主導覽窗格中，選擇 [授權]。
4. 選擇方法，然後從下拉式功能表中選取您的授權者，然後選擇 [附加授權者]。

更新路由以使用 JWT 授權者 AWS CLI

下面的命令更新使用 JWT 授權者的路由。 AWS CLI

```
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 IAM 授權

您可以為 HTTP API 路由啟用 IAM 授權。啟用 IAM 授權後，用戶端必須使用簽名版本 4 (SIGv4) 來使用登入 AWS 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

下列 AWS CLI 命令會針對 HTTP API 路由啟用 IAM 授權。

```
aws apigatewayv2 update-route \
    --api-id abc123 \
    --route-id abcdef \
    --authorization-type AWS_IAM
```

設定 HTTP API 的整合

「整合」會將路由連接到後端資源。HTTP API 支援 Lambda 代理伺服器、 AWS 服務和 HTTP 代理伺服器整合。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

- 使用 HTTP API 的 AWS Lambda 代理整合
- 使用 HTTP API 的 HTTP 代理整合
- 使用 HTTP API 的 AWS 服務整合
- 使用 HTTP API 的私有整合

使用 HTTP API 的 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 傳送至 Lambda 整合的事件格式，以及 API Gateway 如何解譯 Lambda 的回應。如果您未指定承載格式版本，預設 AWS Management Console 會使用最

新版本。如果您使用 AWS CLI、 AWS CloudFormation或 SDK 建立 Lambda 整合，您必須指定payloadFormatVersion. 支援的值為 1.0 和 2.0。

如需如何設定的詳細資訊payloadFormatVersion，請參閱建立整合。如需如何判斷現有整合payloadFormatVersion的詳細資訊，請參閱取得整合

裝載格式差異

下列清單顯示1.0和2.0裝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有rawPath. 如果您使用 API 映射將您的階段連接到自定義域名，則rawPath不會提供 API 映射值。使用格式1.0和存path取自訂網域名稱的 API 對應。
- 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。

裝載格式結構

下列範例顯示每個裝載格式版本的結構。所有標題名稱都是小寫字母。

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

使用`1.0`格式版本時，Lambda 整合必須以下列 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,`

Lambda 函數輸出	API Gateway 解譯

	<pre>"statusCode": 200,
	"body": "Hello from Lambda!",
	"headers": {
	"content-type": "application/
	json"
	}
	}</pre>

<pre>{ "message": "Hello from Lambda!" }</pre>	<pre>{
	"isBase64Encoded": false,
	"statusCode": 200,
	"body": "{ \"message\": \"Hello from
	Lambda!\" }",
	"headers": {
	"content-type": "application/
	json"
	}
	}</pre>

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!"
}
```

使用 HTTP API 的 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。

使用 HTTP API 的 AWS 服務整合

您可以通過使用一流的集成將 HTTP API 與 AWS 服務集成。一級整合會將 HTTP API 路由連接到 AWS 服務 API。當用戶端叫用由一流整合支援的路由時，API Gateway 會為您叫用 AWS 服務 API。例如，您可以使用一流的整合將訊息傳送到 Amazon 簡單佇列服務佇列，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”。

參數映射

類型	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 "querystring1": "Value1,Value2" 。
路徑參數	$request.path.*name*	請求中的路徑參數值。例如，如果路由是 /pets/

類型	範例	備註
		{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*))。 **Note** 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

在建立一流的整合之前，您必須先建立 IAM 角色，以授與 API Gateway 許可，才能叫用要整合的 AWS 服務動作。如需進一步了解，請參閱為 AWS 服務建立角色。

若要建立一流的整合，請選擇支援的 AWS 服務動作SQS-SendMessage，例如設定要求參數，並提供授與 API Gateway 權限以呼叫整合式 AWS 服務 API 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”。

下列 AWS CLI 命令會建立傳送 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"}'
```

使用建立一流的整合 AWS CloudFormation

以下示例顯示了一個 AWS CloudFormation 代碼片段，該代碼段創建了與 Amazon 的一流集成的/{source}/{detailType}路由 EventBridge。

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
- SQS-SendMessage
- SQS-ReceiveMessage
- SQS-DeleteMessage
- SQS-PurgeQueue
- AppConfig-GetConfiguration
- Kinesis PutRecord
- StepFunctions-StartExecution
- StepFunctions-StartSyncExecution
- StepFunctions-StopExecution

EventBridge-PutEvents

將自訂事件傳送至 Amazon， EventBridge 以便符合規則。

EventBridge-PutEvents 1.0

參數	必要
Detail	True
DetailType	True
Source	True
Time	False

參數	必要
EventBusName	False
Resources	False
Region	False
TraceHeader	False

若要進一步了解，請PutEvents參閱 Amazon EventBridge API 參考中的。

SQS-SendMessage

將訊息傳遞到指定的佇列。

SQS-1.0 SendMessage

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 簡單佇列服務 API 參考SendMessage中的。

SQS-ReceiveMessage

您從指定的佇列擷取一個或多個訊息 (最多可達 10 個)。

SQS-1.0 ReceiveMessage

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 簡單佇列服務 API 參考ReceiveMessage中的。

SQS-DeleteMessage

從指定的佇列刪除指定的訊息。

SQS-1.0 DeleteMessage

參數	必要
ReceiptHandle	True
QueueUrl	True
Region	False

若要進一步了解，請參閱 Amazon 簡單佇列服務 API 參考DeleteMessage中的。

SQS-PurgeQueue

刪除指定佇列中的所有訊息。

SQS-1.0 PurgeQueue

參數	必要
QueueUrl	True
Region	False

若要進一步了解，請參閱 Amazon 簡單佇列服務 API 參考PurgeQueue中的。

AppConfig-GetConfiguration

接收組態的相關資訊。

AppConfig-GetConfiguration 1.0

參數	必要
Application	True
Environment	True
Configuration	True
ClientId	True
ClientConfigurationVersion	False
Region	False

要了解更多信息，請GetConfiguration參閱 AWS AppConfig API 參考中的。

Kinesis PutRecord

將單一資料記錄寫入 Amazon Kinesis 資料串流。

Kinesis-1.0 PutRecord

參數	必要
StreamName	True

參數	必要
Data	True
PartitionKey	True
SequenceNumberForOrdering	False
ExplicitHashKey	False
Region	False

若要進一步了解，請參閱 Amazon Kinesis 資料串流 API 參考資料PutRecord中的一文。

StepFunctions-StartExecution

啟動狀態機器的執行。

StepFunctions-StartExecution 1.0

參數	必要
StateMachineArn	True
Name	False
Input	False
Region	False

要了解更多信息，請StartExecution參閱 AWS Step Functions API 參考中的。

StepFunctions-StartSyncExecution

啟動同步狀態機器執行。

StepFunctions-StartSyncExecution 1.0

參數	必要
StateMachineArn	True

參數	必要
Name	False
Input	False
Region	False
TraceHeader	False

要了解更多信息，請StartSyncExecution參閱 AWS Step Functions API 參考中的。

StepFunctions-StopExecution

停止執行。

StepFunctions-StopExecution 1.0

參數	必要
ExecutionArn	True
Cause	False
Error	False
Region	False

要了解更多信息，請StopExecution參閱 AWS Step Functions API 參考中的。

使用 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。

若要建立私有整合，您必須先建立 VPC 連結。若要深入了解 VPC 連結，請參閱使用 HTTP API 的 VPC 連結。

建立 VPC 連結之後，您可以設定私人整合，以連線至 Application Load Balancer、Network Load Balancer 或使用 AWS Cloud Map 服務註冊的資源。

若要建立私有整合，所有資源都必須由相同 AWS 帳戶擁有 (包括負載平衡器或 AWS Cloud Map 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。如果您要求私有整合使用 HTTPS，您可以指定 `tlsConfig`。

> **Note**
>
> 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

在建立私有整合之前，您必須建立 VPC 連結。若要深入了解 VPC 連結，請參閱使用 HTTP API 的 VPC 連結。

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。

使用下列命令，建立使用 VPC 連結來連線至負載平衡器的私有整合。

```
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: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

在建立私有整合之前，您必須建立 VPC 連結。若要深入了解 VPC 連結，請參閱使用 HTTP API 的 VPC 連結。

對於與整合 AWS Cloud Map，API Gateway 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。要了解更多信息，請DiscoverInstances參閱 AWS Cloud Map API 參考中的。

> **Note**
>
> 如果您使用 Amazon ECS 在中填入項目 AWS Cloud Map，則必須設定 Amazon ECS 任務，將 SRV 記錄與 Amazon ECS 服務探索搭配使用，或開啟 Amazon ECS 服務 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，然後提供服務的 AWS Cloud Map ARN。

使用下列命令建立使用 AWS Cloud Map 服務探索來識別資源的私人整合。

```
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
```

使用 HTTP API 的 VPC 連結

VPC 連結可讓您建立私有整合，將 HTTP API 路由連接到 VPC 中的私有資源，例如 Application Load Balancers 或 Amazon ECS 容器型應用程式。若要深入了解如何建立私有整合，請參閱使用 HTTP API 的私有整合。

私有整合使用 VPC 連結，封裝 API Gateway 與目標 VPC 資源之間的連線。您可以在不同的路由和 API 中重複使用 VPC 連結。

當您建立 VPC 連結時，API Gateway 會為帳戶中的 VPC 連結建立和管理彈性網路介面。此程序需要幾分鐘的時間。當 VPC 連結已準備好可供使用時，其狀態會從 PENDING 轉換為 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 AWS CLI

請使用以下命令來建立 VPC 連結。若要建立 VPC 連結，涉及的所有資源都必須由同一 AWS 帳戶擁有。

```
aws apigatewayv2 create-vpc-link --name MyVpcLink \
    --subnet-ids subnet-aaaa subnet-bbbb \
    --security-group-ids sg1234 sg5678
```

Note

VPC 連結是不可變的。建立 VPC 連結之後，您無法變更其子網路或安全性群組。

使用刪除 VPC 連結 AWS CLI

使用下列命令刪除 VPC 連結。

```
aws apigatewayv2 delete-vpc-link --vpc-link-id abcd123
```

各個區域的供應情形

以下區域和可用區域支援 HTTP API 的 VPC 連結：

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
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

區域名稱	區域	支援的可用區域
美國西部 (奧勒岡)	us-west-2	usw2-az1、usw2-az2、usw2-az3、usw2-az4
亞太區域 (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
亞太區域 (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
歐洲 (倫敦)	eu-west-2	euw2-az1、euw2-az2、euw2-az3
歐洲 (巴黎)	eu-west-3	euw3-az1、euw3-az3
歐洲 (斯德哥爾摩)	eu-north-1	eun1-az1、eun1-az2、eun1-az3
中東 (巴林)	me-south-1	mes1-az1、mes1-az2、mes1-az3

區域名稱	區域	支援的可用區域
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

為 HTTP API 設定 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。

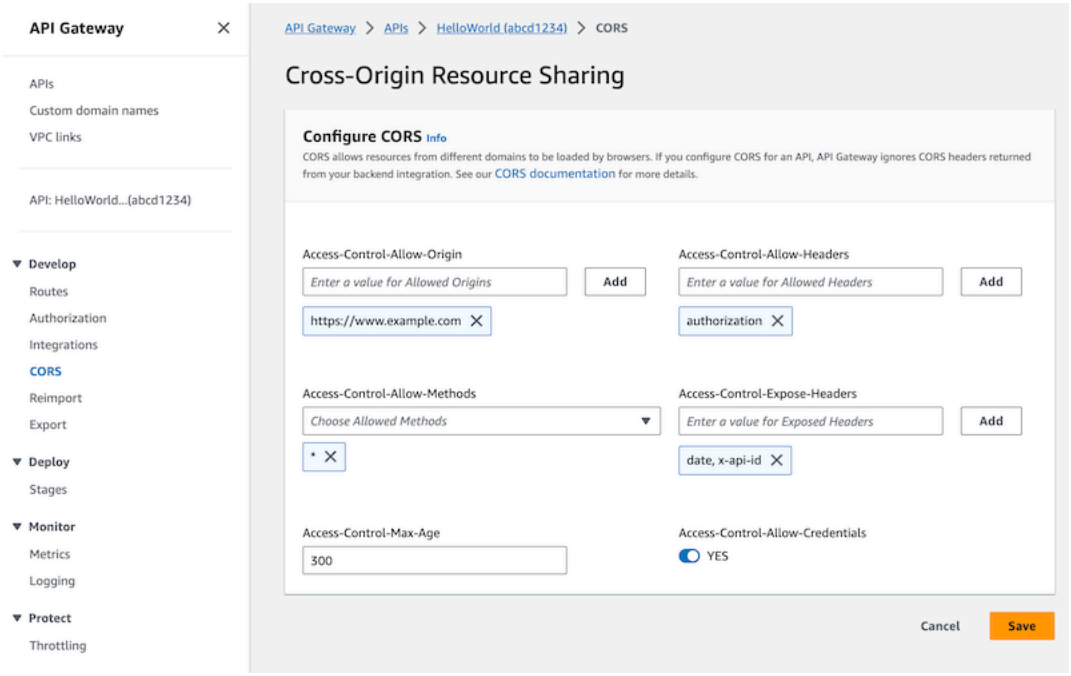
您可以在 CORS 組態中指定下列參數。若要使用 API Gateway HTTP API 主控台新增這些參數，請在輸入值之後選擇新增。

CORS 標頭	CORS 組態屬性	範例值
Access-Control-Allow-Origin	allowOrigins	• `https://www.example.com` • `*` (允許所有來源)

CORS 標頭	CORS 組態屬性	範例值
		• `https://*` (允許以 https:// 開頭的任何來源) • `http://*` (允許以 http:// 開頭的任何來源)
Access-Control-Allow-Credentials	allowCredentials	true
Access-Control-Expose-Headers	exposeHeaders	日期， 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。

您的 CORS 組態可能看起來如下：



使用$default路由和授權者為 HTTP API 配置 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 AWS CLI 為 HTTP API 設定 CORS

您可以使用下面的更新 API 命令來啟用 CORS 請求。https://www.example.com

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。

轉換 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。

參數映射金鑰

類型	語法
標頭	append|overwrite|remove:header. *headername*

類型	語法
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。

請求參數映射值

類型	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”。
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)`)。 **Note** 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

類型	語法	備註
		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 ${request.path.*name*}	請求中的路徑參數值。例如，如果路由為 `/pets/{petId}` ，您可以透過 `$request.path.petId` 從請求映射 `petId` 參數。
環境變數	$context.*variableName* 或 ${context.*variableName* }	內容變數的值。 **Note** 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。

回應參數映射金鑰

類型	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。

回應參數對映值

類型	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)`)。

類型	語法	備註
		Note 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 範例設定參數對映。如需範 AWS CloudFormation 本範本，請參閱GitHub。

將標頭新增至 API 請求

下列範例會在 API 請求達到後端整合之前，新增名稱為 `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

下列範例會將請求標頭從 header1 重新命名為 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

下列範例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

下列範例命令會移除之前設定的請求參數 append:header.header1。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" : {}}'
```

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 API 支援與其他 API 相同的 AWS 變數。如需進一步了解，請參閱AWS 匯入應用程式介面的變數。

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

下列命令會將 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 API 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 定義 AWS

下列命令會將名為 `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 令列 AWS 匯出 API 最新變更的 OpenAPI 3.0 定義

下列命令會將 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
```

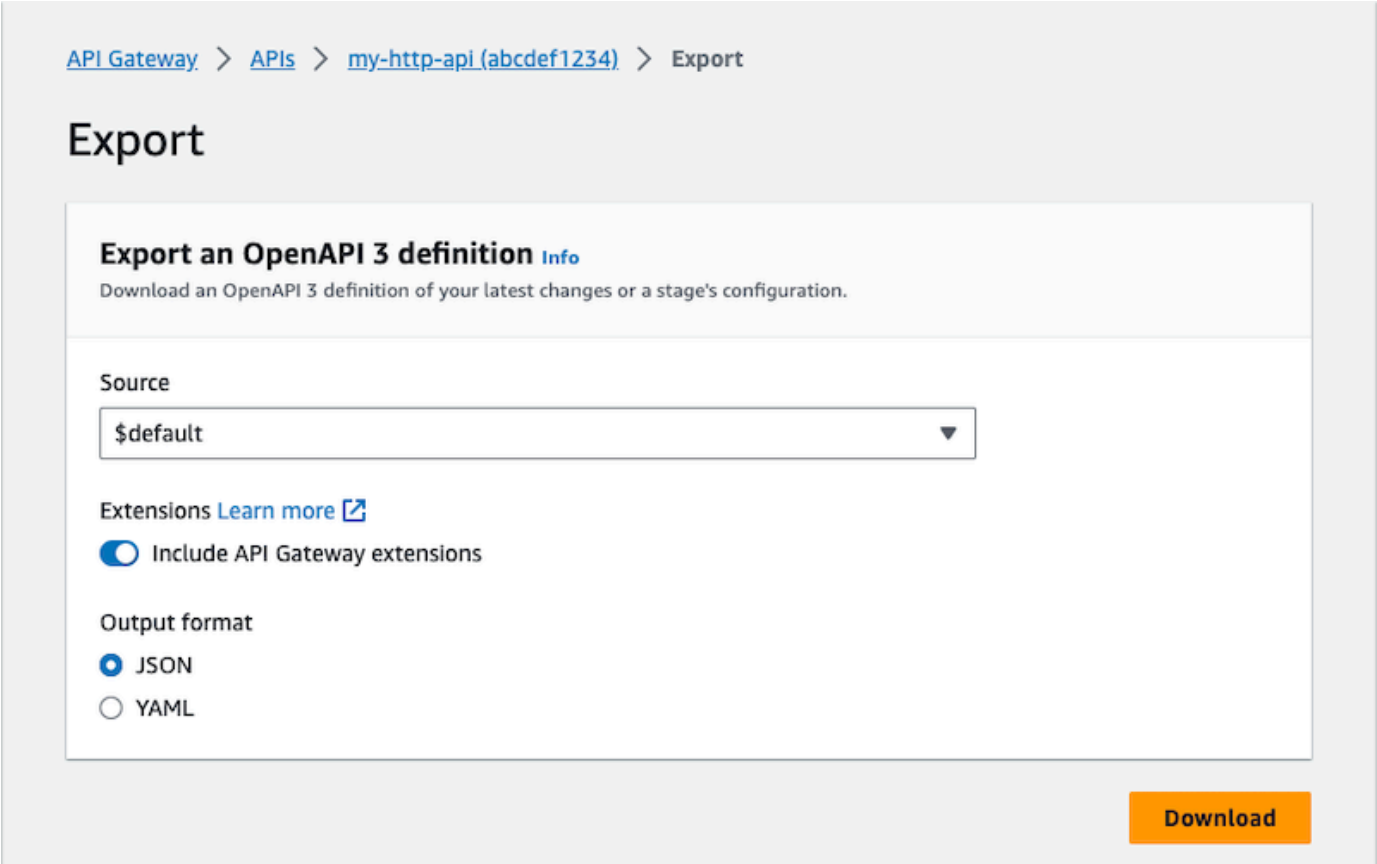
如需詳細資訊，請參閱《Amazon API Gateway 第 2 版 API 參考》中的 ExportAPI。

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)。

發佈 HTTP API 以供客戶叫用

您可以使用階段和自訂網域名稱來發佈 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

- 使用 HTTP API 的階段
- 適用於 API 的安全性原則
- 設定 HTTP API 的自訂網域名稱

使用 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。

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。

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給整合，請使用 AWS Lambda 授權者。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。您可以使用 AWS Command Line Interface (AWS CLI) 來執行此操作。

```
aws lambda add-permission --function-name arn:aws:lambda:XXXXXX:your-lambda-function-
name --source-arn arn:aws:execute-api:us-east-1:YOUR_ACCOUNT_ID:api_id/*/HTTP_METHOD/
resource --principal apigateway.amazonaws.com --statement-id apigateway-access --action
 lambda:InvokeFunction
```

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 整合認證

您可以使用階段變數做為使用 AWS 者或角色認證 ARN 的一部分，如下列範例所示。

- `arn:aws:iam::<account_id>:${stageVariables.<variable_name>}`

適用於 API 的安全性原則

API Gateway 會TLS_1_2針對所有 HTTP API 端點強制執行的安全性政策。

安全政策是 Amazon API Gateway 所提供的最低 TLS 版本和加密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。此安全性原則接受 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 通訊協定和密碼。

安全政策	TLS_1
TLS 通訊協定	
TLSv1.3	♦
TLSv1.2	♦
TLS 密碼	
TLS-A-128 克公分-沙 256	♦
TLS-AES-256 克公分-沙 384	♦
塔爾斯查查 20-聚 1305-SHA256	♦
ECDHE-ECDSA-AES128-GCM-SHA256	♦
ECDHE-RSA-AES128-GCM-SHA256	♦
ECDHE-ECDSA-AES128-SHA256	♦
ECDHE-RSA-AES128-SHA256	♦
ECDHE-ECDSA-AES256-GCM-SHA384	♦
ECDHE-RSA-AES256-GCM-SHA384	♦
ECDHE-ECDSA-AES256-SHA384	♦
ECDHE-RSA-AES256-SHA384	♦
AES128-GCM-SHA256	♦
AES128-SHA256	♦
AES256-GCM-SHA384	♦
AES256-SHA256	♦

OpenSSL 和 RFC 密碼名稱

對於相同的密碼，OpenSSL 和 IETF RFC 5246 使用不同的名稱。如需密碼名稱的清單，請參閱the section called “OpenSSL 和 RFC 密碼名稱”。

關於其他 API 和 WebSocket API 的資訊

如需有關其他 API 和 WebSocket API 的詳細資訊，請參閱the section called “選擇安全性原則”和the section called “ WebSocket API 的安全性原則”。

設定 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`api-id`中由 API Gateway 產生，`region`(AWS 區域) 由您在建立 API 時指定，並`stage`由您在部署 API 時指定。

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 端點很難取回，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 對應至您的 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**
>
> 自訂網域可與 REST API 和 HTTP API 相關聯。您可以使用 API Gateway 第 2 版 API 建立及管理 REST API 和 HTTP API 的區域性自訂網域名稱。
> 對於 HTTP API，TLS 1.2 是唯一支援的 TLS 版本。

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。您的網域名稱必須遵循 RFC 1035 規範，且每個標籤最多可有 63 個八位元組，總共 255 個八位元組。需要時，您可以使用

Amazon Route 53 或使用您選擇的第三方網域註冊機構來註冊網際網路網域。API 的自訂網域名稱可以是已註冊網際網路網域的子網域或根網域 (也稱為 "zone apex") 的名稱。

在 API Gateway 中建立自訂網域名稱，您必須建立或更新 DNS 提供者的資源記錄，以對應至您的 API 端點。如果沒有這種對應，送往自訂網域名稱的 API 請求無法到達 API Gateway。

區域性自訂網域名稱

當您建立區域 API 的自訂網域名稱時，API Gateway 會建立 API 的區域網域名稱。您必須設定 DNS 記錄，將自訂網域名稱對應至區域性網域名稱。您也必須提供自訂網域名稱的憑證。

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，在單一 AWS 帳戶中，您可以設定*.example.com和行a.example.com為不同。

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 帳戶建立了與萬用字元自訂網域名稱衝突的自訂網域名稱，您就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。

自訂網域名稱的憑證

> **Important**
>
> 您可以指定自訂網域名稱所用的憑證。如果您的應用程式使用憑證釘選 (有時稱為 SSL 釘選) 來釘選 ACM 憑證，則應用程式在續訂憑證後 AWS 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。

若要在支援 ACM 的區域中提供自訂網域名稱的憑證，您必須向 ACM 請求憑證。若要在不支援 ACM 的區域中提供區域自訂網域名稱的憑證，您必須將憑證匯入至該區域中的 API Gateway。

若要匯入 SSL/TLS 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。若要將受 AWS 管理憑證用於網域名稱，您只需參考其 ARN 即可。

ACM 可讓您直覺式地設定和使用 API 的自訂網域名稱。您可以建立所指定網域名稱的憑證 (或匯入憑證)、使用 ACM 所提供的憑證 ARN 在 API Gateway 中設定網域名稱，以及將自訂網域名稱下的基本路徑對應至 API 的已部署階段。使用 ACM 所發出的憑證，就不需要擔心公開任何敏感的憑證詳細資訊，例如私有金鑰。

如需設定自訂網域名稱的詳細資訊，請參閱 在 中備妥憑證 AWS Certificate Manager 和 在 API Gateway 中設定區域性自訂網域名稱。

使用適用於 HTTP API 的 API 映射

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，以選取要叫用的 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。

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。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 對應中，自訂網域名稱和對應的 API 必須位於相同的 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。
- 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 對應至與 HTTP API 或 REST API 相同的自訂網域名稱。

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”。

如需建立所有資源的 AWS Serverless Application Model 範例範本，請參閱開啟 SAM 的工作階段 GitHub。

AWS Management Console

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 對應)。
6. 選擇 Add new mapping (新增對應)。
7. 輸入 API、Stage (階段)，以及選擇性地輸入 Path (路徑)。
8. 選擇 Save (儲存)。

AWS CLI

下列 AWS CLI 命令會建立 API 對應。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
```

AWS CloudFormation

下列 AWS CloudFormation 範例會建立 API 對應。

```
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` 端點來叫用 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。

> **Note**
>
> 當您停用預設端點時，它會影響 API 的所有階段。

下列 AWS CLI 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 AWS CLI 指令會建立部署。

```
aws apigatewayv2 create-deployment \
    --api-id abcdef123 \
    --stage-name dev
```

保護 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

- 將請求調節到您的 HTTP API
- 為 HTTP API 配置交互 TLS 身分驗證

將請求調節到您的 HTTP API

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 AWS 節流限制。

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。下列命令會針對指定的 API 階段和路由設定自訂調節。

```
aws apigatewayv2 update-stage \
    --api-id a1b2c3d4 \
    --stage-name dev \
    --route-settings '{"GET /pets":
{"ThrottlingBurstLimit":100,"ThrottlingRateLimit":2000}}'
```

為 HTTP API 配置交互 TLS 身分驗證

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。相互 TLS 是物聯網 (IoT) 和 business-to-business 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
- 至少 AWS Certificate Manager 為您的自訂網域名稱設定一個憑證
- 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 核發的憑證

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 Private Certificate Authority 憑證

若要使用匯入 ACM 的憑證或來自 AWS Private Certificate Authority 相互 TLS 的憑證，API Gateway 需要 ACM 所`ownershipVerificationCertificate`核發的憑證。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，請建立信任存取 API 的 X.509 憑證的信任庫。

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 AWS CLI 命令會上傳`certificates.pem`到您的 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。將以下內容 (包含斜線) 貼到終端機中：

```
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

若要叫用啟用交互 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，才會叫用您的 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。

下列 AWS CLI 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。

```
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=''
```

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 AWS Support 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。

監控您的 HTTP API

您可以使用 CloudWatch 指標和 CloudWatch 記錄來監視 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

- 使用 HTTP API 的指標
- 設定 HTTP API 的記錄功能

使用 HTTP API 的指標

您可以使用 CloudWatch「API Gateway」收集原始資料並將其處理為可讀的 near-real-time 指標來監控 API 執行。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 CloudWatch 在一分鐘內自動傳送至。若要監控您的指標，請為 API 建立 CloudWatch 儀表板。有關如何建立 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。有關更多信息，請參閱什麼是 Amazon CloudWatch？ 在 Amazon 用 CloudWatch 戶指南。

HTTP API 支援下列指標。您也可以啟用詳細指標，將路由層級指標寫入 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。

指標	描述
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, 舞台	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	針對具有指定 API ID、階段 ID、資源路徑和路由 ID 的 API 方法篩選 API Gateway 指標。 除非您已明確啟用詳細指標，否則 API Gateway 不會傳送這些 CloudWatch 指標。您可以呼叫 API Gateway V2 REST API 的UpdateStage動作，將內detailedMetricsEna

維度	描述
	bled 容更新為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，請參閱 Amazon CloudWatch 定價。

設定 HTTP API 的記錄功能

您可以開啟記錄功能，將記錄寫入記 CloudWatch 錄檔。您可以使用記錄變數來自訂日誌的內容。

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 記錄檔記錄群組。
3. 針對 API 的某個階段提供 CloudWatch 記錄檔記錄群組的 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

```
{
    "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 Management Console 或建立記錄群組並啟動存取記錄 AWS CLI。

AWS Management Console

1. 建立 日誌群組

   若要了解如何使用主控台建立日誌群組，請參閱 Amazon CloudWatch 日誌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. 開啟存取記錄，輸入記 CloudWatch 錄，然後選取記錄格式。
8. 選擇儲存。

AWS CLI

下列 AWS CLI 命令會建立記錄群組。

```
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 ### ARN: AW: ##:# #:## ID: ###:. log-group-name*

下面的 AWS CLI 命令打開一個 HTTP API 的$default階段日誌記錄。

```
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，請參閱設定 HTTP API 的記錄功能。

參數	描述
`$context.accountId`	API 擁有者的 AWS 帳戶識別碼。
`$context.apiId`	API Gateway 指派給您 API 的識別碼。
`$context.authorizer.claims.` *property*	方法呼叫者成功驗證後，從 JSON Web Token (JWT) 傳回的宣告屬性，例如`$context.authorizer.claims.username`。如需詳細資訊，請參閱 使用 JWT 授權方控制對 HTTP API 的存取權。 **Note** 呼叫 `$context.authorizer.claims` 會傳回 null。
`$context.authorizer.error`	從授權方傳回的錯誤訊息。
`$context.authorizer.principalId`	Lambda 授權方傳回的主要使用者識別。
`$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	描述
	`"context" : { "key": "value", "numKey": 1, "boolKey": true }` 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`	AWS 端點來自`x-amz-request-id` 或`x-amzn-requestId` 標頭的要求 ID。
`$context.awsEndpointRequestId2`	AWS 端點來自`x-amz-id-2` 標頭的請求 ID。
`$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	描述
`$context.error.responseType`	一種 `GatewayResponse` 的類型。如需詳細資訊，請參閱 the section called “指標” 和 the section called “設定閘道回應以自訂錯誤回應”。
`$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。
`$context.identity.cognitoAuthenticationProvider`	提出請求的發起人所使用的 Amazon Cognito 身分驗證提供者清單 (以逗號分隔)。僅在使用 Amazon Cognito 登入資料簽署請求時才可使用。 例如，適用於 Amazon Cognito 使用者集區的身分，`cognito-idp.`*`region`*`.amazonaws.com/`*`user_pool_id`*`,cognito-idp.`*`region`*`.amazonaws.com/`*`user_pool_id`*`:CognitoSignIn:`*`token subject claim`* 如需詳細資訊，請參閱 Amazon Cognito 開發人員指南中的使用聯合身分。
`$context.identity.cognitoAuthenticationType`	提出請求的發起人的 Amazon Cognito 身分驗證類型。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` 。

參數	描述
$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。
$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。

參數	描述
$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，狀態碼會從後端 Lambda 函數程式碼傳回 AWS Lambda，而不是從後端 Lambda 函數
$context.integration.latency	整合延遲 (以毫秒為單位)。等同於 $context.integrationLatency 。
$context.integration.requestId	AWS 端點的要求識別碼。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)。
$context.integrationStatus	對於 Lambda 代理整合，此參數代表從後端 Lambda 函數傳回的狀態碼 AWS Lambda，而非從後端 Lambda 函數傳回。
$context.path	請求路徑。例如，/{stage}/root/child 。

參數	描述
$context.protocol	請求通訊協定，例如 HTTP/1.1。 Note 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。

對 HTTP API 的問題進行疑難排解

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：

若要使用建立日誌群組 AWS Management Console

1. 請在以下位置開啟 CloudWatch 主控台。 https://console.aws.amazon.com/cloudwatch/
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 ### ARN: AW: ##:##:## ID: ## #:. 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

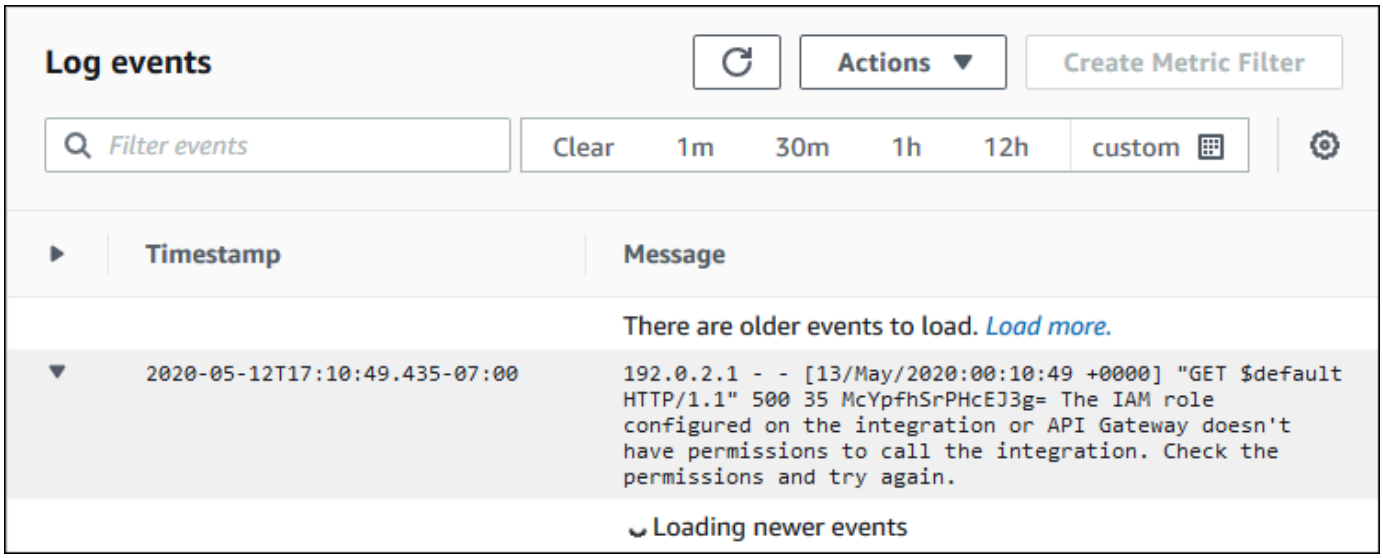
1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 選擇 HTTP API。
3. 在 Monitor (監控) 下，選擇 Logging (記錄)。
4. 選取 API 的階段。
5. 選擇 Edit (編輯)，然後存取記錄。
6. 針對 Log destination (日誌目的地)，輸入您在上一個步驟中建立之日誌群組的 ARN。
7. 在 Log format (日誌格式) 中，選擇 CLF。API Gateway 會建立範例記錄格式。
8. 將 `$context.integrationErrorMessage` 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 [檢視登入] CloudWatch。
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、 AWS CloudFormation或 SDK 建立 Lambda 整合時，您必須授與 API Gateway 叫用函數的權限。下列範例 AWS CLI 命令會授與不同 HTTP API 路由的權限，以叫用 Lambda 函數。

Example 範例 — 針對 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 — 針對 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。只有在不存在的情況下client_id才aud會驗證 API Gateway。當aud和client_id都存在時，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 JWT 授權方控制對 HTTP API 的存取權。

使用 WebSocket API

WebSocket API Gateway 中的 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。這種雙向行為可以實現更豐富的客戶端/服務互動，因為服務可以將數據推送到客戶端，而無需客戶端發出明確請求。 WebSocket API 通常用於實時應用程序，例如聊天應用程序，協作平台，多人遊戲和金融交易平台。

如需開始使用的應用程式範例，請參閱 教學課程：使用 WebSocket API、Lambda 和 DynamoDB 建置無伺服器聊天應用程式。

在本節中，您可以了解如何使用 API Gateway 開發、發佈、保護和監視 WebSocket API。

主題

- 關於 WebSocket API Gateway 中的 API
- 在 WebSocket API Gateway 中開發 API
- 發佈 WebSocket API 供客戶叫用
- 保護您的 WebSocket API
- 監控 WebSocket API

關於 WebSocket API Gateway 中的 API

在 API Gateway 中，您可以建立 WebSocket API 做為 AWS 服務 (例如 Lambda 或 DynamoDB) 或 HTTP 端點的可設定狀態前端。 WebSocket API 會根據從用戶端應用程式收到的訊息內容叫用後端。

與接收和回應請求的 REST API 不同， WebSocket API 支援用戶端應用程式與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。

在您的 WebSocket API 中，傳入的 JSON 訊息會根據您設定的路由導向至後端整合。(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。

- 當用戶端與 API 之間的持續連`$connect`線正在起始時， WebSocket API Gateway 會呼叫路由。
- 當用戶端或伺服器中斷與 API 的連線時，API Gateway 會呼叫 `$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 “呼叫後端整合”。

後端服務可將資料傳送到連線的用戶端應用程式。如需更多詳細資訊，請參閱 the section called “將資料從後端服務傳送到連線的用戶端”。

管理連線使用者和用戶端應用程式：**$connect** 和 **$disconnect** 路由

主題

- $connect 路由
- 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 升級要求來連線至您的 WebSocket API。如果請求成功，會在建立連線的同時執行 `$connect` 路由。

因為 WebSocket 連線是可設定狀態的連線，因此您只能在`$connect`路由上設定授權。AuthN/AuthZ將僅在連接時間進行。

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，請參閱設置需要子協議的$connect WebSocket 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。
- 您想將每個連線 ID 和其他資訊放到資料庫 (例如，Amazon DynamoDB)。

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 函數發佈到連接。

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。如需更多詳細資訊，請參閱 the section called “在後端服務使用 @connections 命令”。

呼叫您的後端整合：$default 路由和自訂路由

主題

- 使用路由來處理訊息
- $default 路由
- 自訂路由
- 使用 API Gateway WebSocket API 整合連接到您的業務邏輯
- WebSocket API 和其餘 API 之間的重要差異

使用路由來處理訊息

在 API Gateway WebSocket API 中，訊息可以從用戶端傳送至後端服務，反之亦然。與 HTTP 的請求/響應模型不同，在 WebSocket 後端可以將消息發送到客戶端，而無需客戶端採取任何操作。

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”。

WebSocket 透過 API Gateway 中的 API，可以路由 JSON 訊息，以根據訊息內容執行特定的後端服務。當客戶端通過其 WebSocket 連接發送消息時，這會導致向 WebSocket API 的路由請求。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、或使用 AWS SDK，在 WebSocket API Gateway 主控台中設定 AWS CLI API 的路由要求。

> **Note**
>
> 在 AWS CLI 和 AWS SDK 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 會在不呼叫後端的情況下失敗要求，將類似下列內容的`"Bad request body"`閘道回應傳送給用戶端，然後在 CloudWatch 記錄檔中發佈驗證結果：

```
{"message" : "Bad request body", "connectionId": "{connectionId}", "messageId":
 "{messageId}"}
```

這可減少對後端不必要的呼叫，讓您專注在 API 的其他要求。

您也可以為 API 路由定義路由回應，以啟用雙向通訊。路由回應說明會在特定路由整合完成時，將哪些資料傳送到用戶端。例如，如果您希望用戶端將訊息傳送到後端，而不接收回應 (單向通訊)，您不需要定義路由的回應。不過，如果您不提供路由回應，API Gateway 不會將任何與您整合結果相關的資訊傳送到用戶端。

`$default` 路由

每個 API Gateway WebSocket API 都可以有`$default`路由。這是一個特殊路由值，使用方式如下：

- 您可以使用該值搭配定義的路由金鑰，來為不符合任何定義路由金鑰的內送訊息指定「備用」路由 (例如，一般偽裝整合，會傳回特定的錯誤訊息)。
- 您可以使用該值與任何定義路由金鑰，來指定將路由委派到後端元件的 Proxy 模型。
- 您可以使用該值來為非 JSON 承載指定路由。

自訂路由

如果您想要根據訊息內容來叫用特定的整合，則可透過建立自訂路由來這麼做。

自訂路由會使用您指定的路由金鑰和整合。當內送訊息包含 JSON 屬性，而且該屬性的判斷值符合路由金鑰值時，API Gateway 會叫用整合。(如需詳細資訊，請參閱「the section called “關於 WebSocket API”」。)

例如，假設您想要建立聊天空間應用程式。您可以先創建一個 WebSocket API，其路由選擇表達式是`$request.body.action`。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 整合連接到您的業務邏輯

設定 API Gateway WebSocket API 的路由之後，您必須指定要使用的整合。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 和其餘 API 之間的重要差異

除了 WebSocket 以下差異之外，API 的整合類似於 REST API 的整合：

- 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 SDK 中，您可以將路由的目標設定為值來重複使用整合`"integrations/{integration-id}"`，其中`{integration-id}`"是要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 協議中，通信是雙向的。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。

將資料從後端服務傳送到連線的用戶端

API Gateway WebSocket API 提供下列方式，讓您將資料從後端服務傳送至連線的用戶端：

- 整合可以傳送回應，其回應會由您已定義的路由來傳回用戶端。

- 您可以使用 @connections API 來傳送 POST 請求。如需更多詳細資訊，請參閱 the section called "在後端服務使用 @connections 命令"。

WebSocket API Gateway 中的選取項運算式

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選取表示式摘要

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*。

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。對於 WebSocket API，路由響應是可選的。定義後，它會向 API Gateway 發出信號，它應該在收到 WebSocket 消息時向客戶端返回響應。

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選取表示式摘要

下表摘要說明 WebSocket API 中選取項運算式的使用案例：

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。	根據用戶端要求的內容路由 WebSocket 郵件。
Route.ModelSelectionExpression	Route.RequestModels 的金鑰	選用。 若提供給非代理整合，將出現模型驗證。 所支援的 $defaul 為全部截獲。	在相同路由內動態執行請求驗證。
Integration.Templa	Integration.RequestTemplates 的金鑰	選用。	依據請求的動

選擇表達式	判斷值為下列的金鑰	備註	範例使用案例
teSelectionExpression		可供非代理整合使用，藉此操控傳入承載。 支援 ${reque .body.j nPath} 和靜態值。 所支援的 $defaul 為全部截獲。	態屬性操控發起人的請求。

選擇表達式	判斷值為下列的金鑰	備註	範例使用案例
`Integration.IntegrationResponseSelectionExpression`	`IntegrationResponse.IntegrationResponseKey`	選用。可供非代理整合使用。 做為錯誤訊息 (來自 Lambda) 或狀態碼 (來自 HTTP 整合) 的模式比對。 非代理整合必須有 `$default` 來全部截獲成功回應。	從後端操控回應。 依據後端的動態回應來選擇欲採取的動作 (如明確處理特定錯誤)。

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。 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。 支援 ${request.body.jsonPath} 、${integration.response.statuscode} 、${integration.response.header.headerName} 、${integration.response.multivalueheader.headerName} 和靜態值。 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	應提供以啟動 WebSoc 路由的雙向通信。 目前此值限制為 $defau	
RouteResponse.ModelSelectionExpression	RouteResponse.RequestModels 的金鑰	目前不支援。	

在 WebSocket API Gateway 中開發 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 WebSocket API Gateway 中建立 API
- 使用 WebSocket API 的路由
- 在 API Gateway 中控制和管理 WebSocket API 的存取
- 設定 WebSocket API 整合功能

- 請求驗證
- 設定 API 的資料轉換 WebSocket
- 使用 WebSocket API 的二進位媒體類型
- 調用一個 API WebSocket

在 WebSocket API Gateway 中建立 API

您可以使用 WebSocket AWS CLI create-api 命令或在 SDK 中使用命令，在 API Gate way 主控台中建立 API。`CreateApi` AWS 下列程序顯示如何建立新 WebSocket API。

> **Note**
>
> WebSocket API 只支援 TLS 1.2。不支援舊版 TLS。

使用 AWS CLI 命令建立 WebSocket API

使用 AWS CLI 需要呼叫 create- WebSocket api 命令來建立 API，如下列範例所示，這會使用`$request.body.action`路由選取運算式建立 API：

```
aws apigatewayv2 --region us-east-1 create-api --name "myWebSocketApi3" --protocol-type
 WEBSOCKET --route-selection-expression '$request.body.action'
```

輸出範例：

```
{
    "ApiKeySelectionExpression": "$request.header.x-api-key",
    "Name": "myWebSocketApi3",
    "CreatedDate": "2018-11-15T06:23:51Z",
    "ProtocolType": "WEBSOCKET",
    "RouteSelectionExpression": "'$request.body.action'",
    "ApiId": "aabbccddee"
}
```

使用 WebSocket API Gateway 主控台建立 API

您可以通過選擇 WebSocket 協議並為 WebSocket API 命名來在控制台中創建 API。

> **Important**
>
> 一旦建立 API 後，您無法更改您選擇的通訊協定。沒有辦法將 WebSocket API 轉換為 REST API，反之亦然。

若要使用 WebSocket API Gateway 主控台建立 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 [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。

使用 WebSocket API 的路由

在您的 WebSocket API 中，傳入的 JSON 訊息會根據您設定的路由導向至後端整合。(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。

- 當用戶端與 API 之間的持續連`$connect`線正在起始時， WebSocket API Gateway 會呼叫路由。

- 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。對 WebSocket於基於 API，表達式應該是形式`$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

表達式	評估結果	描述
`${request.body.action}`	`join`	已包裝的變數
`${request.body.service}/${request.body.action}`	`chat/join`	具備靜態值的多個變數
`${request.body.action}-${request.body.invalidPath}`	`join-`	若未找到 JSONPath，則變數將解析為 ""。
`action`	`action`	靜態值
`\$default`	`$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

當您第一次建立新的 WebSocket API 時，會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 “關於 WebSocket API”。

> **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 WebSocket API Gateway 主控台設定 API 整合要求”。

使用建立路線 AWS CLI

若要使用 AWS CLI，呼叫`create-route`建立路由，如下列範例所示：

```
aws apigatewayv2 --region us-east-1 create-route --api-id aabbccddee --route-key
 $default
```

輸出範例：

```
{
    "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。授權者可以居住在您自己的 AWS 帳戶或不同的 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 授權者 (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 主控台設定 WebSocket API 的$connect路由要求：

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 路由可以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 主控台或或 AWS SDK 來設定路由回應和回應選取運算式。 AWS CLI

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，選擇`$default`路由上具有代理 Lambda 函數整合的 WebSocket API。
2. 在 Routes (路由) 下選擇 `$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 “用`wscat`於連接到 WebSocket API 並向其發送消息”。

```
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

若要使用設定 WebSocket API 的路由回應 AWS CLI，請呼叫命`create-route-response`令，如下列範例所示。您可以透過呼叫 `get-apis` 和 `get-routes` 來識別 API ID 和路由 ID。

```
aws apigatewayv2 create-route-response \
    --api-id aabbccddee \
    --route-id 1122334  \
    --route-response-key '$default'
```

輸出範例：

```
{
    "RouteResponseId": "abcdef",
    "RouteResponseKey": "$default"
}
```

設置需要子協議的$connect WebSocket 路由

客戶端可以使用該Sec-WebSocket-Protocol字段在連接到您的 WebSocket API 期間請求WebSocket 子協議。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 AWS CloudFormation 範例 API 和 Lambda 代理整合的範本，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)
```

若要進一步瞭解呼叫 WebSocket API，請參閱調用一個 API WebSocket 。

在 API Gateway 中控制和管理 WebSocket API 的存取

API Gateway 支援控制和管理 WebSocket API 存取的多種機制。

您可以使用下列機制進行身分驗證和授權：

- 標準 AWS IAM 角色和政策提供彈性且強大的存取控制。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。
- 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。如需更多詳細資訊，請參閱 建立 Lambda REQUEST 授權方函數。

主題

- 使用 IAM 授權
- 建立 Lambda REQUEST 授權方函數

使用 IAM 授權

WebSocket API 中的 IAM 授權與 REST API 的授權類似，但下列情況例外：

- 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 佈線使用不同的 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`)。

```
{
```

```
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "execute-api:Invoke"
      ],
      "Resource": [
        "arn:aws:execute-api:us-east-1:account-id:api-id/prod/*"
      ]
    },
    {
      "Effect": "Deny",
      "Action": [
        "execute-api:Invoke"
      ],
      "Resource": [
        "arn:aws:execute-api:us-east-1:account-id:api-id/prod/secret"
      ]
    },
    {
      "Effect": "Deny",
      "Action": [
        "execute-api:ManageConnections"
      ],
      "Resource": [
        "arn:aws:execute-api:us-east-1:account-id:api-id/*"
      ]
    }
  ]
}
```

建立 Lambda **REQUEST** 授權方函數

API 中的 Lambda 授權器函數與 REST WebSocket API 的授權程式函數類似，但下列情況例外：

- 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。

下列範例會顯示 WebSocket API REQUEST 授權者的輸入：

```
{
    "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 授權者函數是適用於其餘 API 的 Lambda 授權器函數 WebSocket 版本，位於：the section called “Lambda 授權者函數的其他範例”

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("Unauthorized");
   }
```

```
}

// 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:
        print('unauthorized')
        return 'unauthorized'

    # 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)
```

若要將上述 Lambda 函數設定為 WebSocket API 的REQUEST授權者函數，請遵循與 REST API 相同的程序。

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。當您連線到 WebSocket API 時，您需要為任何已設定的身分識別來源提供值。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
```

設定 WebSocket API 整合功能

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 AWS 服務動作。API 整合有整合請求和整合回應。

在本節中，您可以瞭解如何設定 WebSocket API 的整合要求和整合回應。

主題

- 在 API Gateway 中設定 WebSocket API 整合要求
- 在 API Gateway 中設定 WebSocket API 整合回應

在 API Gateway 中設定 WebSocket API 整合要求

設定整合請求包含下列事項：

- 選擇路由金鑰來整合後端。
- 指定要叫用的後端端點。 WebSocket API 支援下列整合類型：
  - AWS_PROXY
  - AWS
  - HTTP_PROXY
  - HTTP
  - MOCK

  如需有關整合類型的詳細資訊，請參閱 API Gateway V2 REST API IntegrationType中的。
- 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 WebSocket API Gateway 主控台設定 API 整合要求

使用 API Gateway 主控台將整合要求新增至 WebSocket API 中的路由

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：

   - 只有在您的 API 將與您在此帳戶或其他帳戶中建立的 AWS Lambda 函數整合時，才選擇 Lambda 函數。

     若要在中建立新的 Lambda 函數 AWS Lambda、設定 Lambda 函數的資源權限，或是要執行任何其他 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 在 API Gateway 中設定 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 在 API Gateway 中設定模擬整合。
   - 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 API Gateway 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"。

使用設定整合要求 AWS CLI

您可以使用如下列範例所示，建立模擬整合 AWS CLI ，在 WebSocket API 中設定路由的整合要求：

1. 建立名為 integration-params.json 的檔案，其中內容如下：

```
{"PassthroughBehavior": "WHEN_NO_MATCH", "TimeoutInMillis": 29000,
 "ConnectionType": "INTERNET", "RequestTemplates": {"application/json":
 "{\"statusCode\":200}"}, "IntegrationType": "MOCK"}
```

2. 執行建立整合命令，如下列範例所示：

```
aws apigatewayv2 --region us-east-1 create-integration --api-id aabbccddee --cli-
input-json file://integration-params.json
```

此範例的範例輸出如下：

```
{
    "PassthroughBehavior": "WHEN_NO_MATCH",
    "TimeoutInMillis": 29000,
    "ConnectionType": "INTERNET",
    "IntegrationResponseSelectionExpression": "${response.statuscode}",
    "RequestTemplates": {
        "application/json": "{\"statusCode\":200}"
    },
    "IntegrationId": "0abcdef",
    "IntegrationType": "MOCK"
}
```

或者，您也可以使用如下列範例所示，設定 Proxy 整合 AWS CLI 的整合要求：

1. 在 Lambda 主控台中建立 Lambda 函數，並賦予基本的 Lambda 執行角色。
2. 執行建立整合命令命令，如下列範例所示：

```
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
```

此範例的範例輸出如下：

```
{
    "PassthroughBehavior": "WHEN_NO_MATCH",
    "IntegrationMethod": "POST",
    "TimeoutInMillis": 29000,
```

```
    "ConnectionType": "INTERNET",
    "IntegrationUri": "arn:aws:apigateway:us-east-1:lambda:path/2015-03-31/functions/
arn:aws:lambda:us-east-1:123412341234:function:simpleproxy-echo-e2e/invocations",
    "IntegrationId": "abcdefg",
    "IntegrationType": "AWS_PROXY"
}
```

Lambda 函數的輸入格式，用於 WebSocket API 的代理整合

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`$connect`路由中輸入事件的結構，以及 API Gateway 傳送至 Lambda 代理整合`$disconnect`之路由的輸入事件。

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

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 集成響應的設置有一些差異，但在概念上，行為是相同的。

WebSocket 路由可以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。
- 如果將路由設定為單向通訊，則無論任何整合回應組態為何，在處理訊息之後，都不會透過通 WebSocket 道傳回任何回應。

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

若要使用 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 並選擇您的路線。
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 "用wscat於連接到 WebSocket API 並向其發送消息"。

```
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

若要使用 AWS CLI 呼叫create-integration-response命令設定 WebSocket API 的整合回應。下列 CLI 命令顯示建立 $default 整合回應的範例：

```
aws apigatewayv2 create-integration-response \
    --api-id vaz7da96z6 \
    --integration-id a1b2c3 \
    --integration-response-key '$default'
```

請求驗證

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 會在不呼叫後端的情況下失敗要求，傳送「錯誤的要求主體」閘道回應給用戶端，然後在 CloudWatch 記錄檔中發佈驗證結果。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。

定義 WebSocket API 的路由時，您可以選擇性地指定模型選取運算式。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 "了解資料模型"。

使用 API Gateway 主控台設定請求驗證

下列範例說明如何在路由上設定要求驗證。

首先，建立模型，然後建立路線。接下來，您可以在剛建立的路由上設定要求驗證。最後，您部署和測試您的 API。若要完成本教學課程，您需要一個 WebSocket API `$request.body.action` 作為路由選取運算式，以及新路由的整合端點。

您還需要 `wscat` 來連線到 API。如需詳細資訊，請參閱 the section called “用wscat於連接到 WebSocket API 並向其發送消息”。

建立裝置

1. 在以下網址登入 API Gateway 主控台：https://console.aws.amazon.com/apigateway。
2. 選擇一個 WebSocket API。
3. 在主導覽窗格中，選擇模型。
4. 選擇建立模型。
5. 針對名稱，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"
        }
    }
}
```

   此模型要求請求包含電子郵件地址。
8. 選擇儲存。

在此步驟中，您會為 WebSocket API 建立路由。

建立路由

1. 在主導覽窗格中，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，您可以設定sendMessage路由的要求驗證。

若要設定請求驗證

1. 在 [路由要求] 索引標籤的 [路由要求設定] 下，選擇 [編輯]。
2. 對於「模型」選取表示式，輸入**${request.body.messageType}**。

   API Gateway 會使用messageType屬性來驗證傳入的要求。
3. 選擇 [新增請求模型]。
4. 對於「模型金鑰」，輸入**email**。
5. 對於「型號」，選擇「電子郵件模

   API Gateway 會email針對此模型驗證內送訊息，並將messageType屬性設定為。

   > **Note**
   >
   > 如果 API Gateway 無法將模型選取運算式與模型金鑰相符，則會選取$default模型。如果沒有$default模型，則驗證失敗。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，您將部署並測試您的 API。

若要部署和測試您的 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. 複製您的 API WebSocket 網址。URL 看起來應該會像這樣：wss://*abcdef123*.execute-api.*us-east-2*.amazonaws.com/production。
6. 打開一個新的終端並使用以下參數運行wscat命令。

```
wscat -c wss://abcdef123.execute-api.us-west-2.amazonaws.com/production
```

```
Connected (press CTRL+C to quit)
```

7. 使用以下命令來測試您的 API。

```
{"action": "sendMessage", "messageType": "email"}
```

```
{"message": "Invalid request body", "connectionId":"ABCD1=234",
 "requestId":"EFGH="}
```

API Gateway 將失敗請求。

使用下一個命令將有效請求發送到您的 API。

```
{"action": "sendMessage", "messageType": "email", "address":
 "mary_major@example.com"}
```

設定 API 的資料轉換 WebSocket

在 API Gateway 中， WebSocket API 的方法請求可以根據後端的要求，以與對應的整合請求有效負載不同的格式獲取有效負載。同樣地，後端可能會依照前端的預期，傳回與方法回應承載不同的整合回應承載。

API Gateway 可讓您使用對應範本，將承載從方法請求對應到對應的整合請求，以及從整合回應對應到對應的方法回應。您可以指定範本選取表示式，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 了解對應範本。

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 了解資料模型。

範本選擇表達式

若要使用對應範本轉換承載，您可以在整合要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

當您設定 WebSocket API 的整合回應時，您可以選擇性地指定整合回應選取項運算式。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 WebSocket API 的資料對應

資料映射可讓您將路由請求中的資料映射至後端整合。

> **Note**
>
> 中不支援 WebSocket API 的資料對應 AWS Management Console。您必須使用 AWS CLI AWS CloudFormation、或 SDK 來配置資料對應。

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。

在下表中，*PARAM_NAME* 是指定參數類型的方法請求參數名稱。它必須符合規則表達式 '^[a-zA-Z0-9._$-]+$]'。*JSONPath_EXPRESSION* 是請求內文的 JSON 欄位的 JSONPath 表達式。

整合請求資料對應表達式

對應的資料來源	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。

範例

下列 AWS CLI 範例設定資料對映。如需範例 AWS 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

下列範例命令會將用戶端的 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

下列範例命令會將 authToken 查詢字串參數映射至整合請求中的 authToken 標頭。

首先，將 authToken 查詢字符串參數新增到路由的請求參數。

```
aws apigatewayv2 update-route --route-id 0abcdef \
    --api-id a1b2c3d4 \
    --request-parameters '{"route.request.querystring.authToken": {"Required": false}}'
```

接下來，將查詢字串參數映射至請求中的 authToken 標頭到後端整合。

```
aws apigatewayv2 update-integration \
    --integration-id abc123 \
    --api-id a1b2c3d4 \
    --request-parameters
 'integration.request.header.authToken'='route.request.querystring.authToken'
```

如有必要，請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'
```

API Gateway WebSocket API 對應範本參考

本節總結了 API Gateway 中目前支援 WebSocket API 的一組變數。

參數	描述
$context.connectionId	連線的唯一 ID，可用來回呼用戶端。
$context.connectedAt	Epoch 格式化連線時間。
$context.domainName	WebSocket API 的網域名稱。此可用於回呼用戶端 (非硬式編碼的值)。
$context.eventType	事件類型：CONNECT、MESSAGE 或 DISCONNECT 。
$context.messageId	訊息的伺服器端唯一 ID。僅在 $context.eventType 為 MESSAGE 時可用。
$context.routeKey	所選路由金鑰。
$context.requestId	與 $context.extendedRequestId 相同。
$context.extendedRequestId	API 呼叫的自動產生的 ID，其中包含更多除錯/故障排除的有用的資訊。
$context.apiId	API Gateway 指派給您 API 的識別碼。
$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。

$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：
	`"context" : {`
	`  "key": "value",`
	`  "numKey": 1,`
	`  "boolKey": true`
	`}`

參數	描述
	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。
$context.identity.cognitoAuthenticationProvider	提出請求的發起人所使用的 Amazon Cognito 身分驗證提供者清單 (以逗號分隔)。僅在使用 Amazon Cognito 登入資料簽署請求時才可使用。 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* 如需詳細資訊，請參閱 Amazon Cognito 開發人員指南中的使用聯合身分。

參數	描述
`$context.identity.cognitoAuthenticationType`	提出請求的發起人的 Amazon Cognito 身分驗證類型。僅在使用 Amazon Cognito 登入資料簽署請求時才可使用。可能的值包括用於已驗證身分的 `authenticated` 和未經驗證身分的 `unauthenticated` 。
`$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。
`$context.identity.userArn`	身分驗證之後識別之有效使用者的 Amazon Resource Name (ARN)。
`$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。

參數	描述
`$input.json(x)`	此函數會評估 JSONPath 表達式，並傳回結果作為 JSON 字串。 例如，`$input.json('$.pets')` 會傳回代表 pets 結構的 JSON 字串。 如需 JSONPath 的詳細資訊，請參閱 JSONPath 或適用於 Java 的 JSONPath。

參數	描述
`$input.path(x)`	採用 JSONPath 表達式字串 (x)，並傳回結果的 JSON 物件呈現。這可讓您存取和運用 Apache Velocity 範本語言 (VTL) 中原生承載的元素。 例如，如果表達式 `$input.path('$.pets')` 傳回如下物件： `[ { "id": 1, "type": "dog", "price": 249.99 }, { "id": 2, "type": "cat", "price": 124.99 }, { "id": 3, "type": "fish", "price": 0.99 } ]` `$input.path('$.pets').count()` 會傳回 "3"。 如需 JSONPath 的詳細資訊，請參閱 JSONPath 或適用於 Java 的 JSONPath。
`$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	描述
`$util.escapeJavaScript()`	轉義使用字符串規則的字符串中的 JavaScript 字符。 **Note** 此函數會將任何一般單引號 (') 轉換為逸出單引號 (\')。不過，逸出單引號不適用於 JSON。因此，將此函數的輸出用於 JSON 屬性時，您必須將任何逸出單引號 (\') 轉換為一般單引號 (')。下列範例顯示這種情況： `$util.escapeJavaScript( data).replaceAll("\\'",'')`

參數	描述

$util.parseJson()	採用「字串化」JSON，並傳回結果的物件呈現。您可以使用此函數的結果，來存取和運用 Apache Velocity 範本語言 (VTL) 中原生承載的元素。例如，如果您有下列承載：
	`{"errorMessage":"{\"key1\":\"var1\", \"key2\":{\"arr\":[1,2,3]}}"}`
	並使用下列映射範本
	`#set ($errorMessageObj = $util.parseJson($input.path('$.errorMessage')))`
	`{`
	`  "errorMessageObjKey2ArrVal" : $errorMessageObj.key2.arr[0]`
	`}`
	您將會收到下列輸出：
	`{`
	`  "errorMessageObjKey2ArrVal" : 1`
	`}`

$util.urlEncode()	將字符串轉換為「應用程序/x-www-form-urlencoded」格式。
$util.urlDecode()	解碼「應用程序/x-www-form-urlencoded」字符串。
$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。

使用 WebSocket API 的二進位媒體類型

API Gateway WebSocket API 目前不支援傳入訊息承載中的二進位框架。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。

調用一個 API WebSocket

部署 WebSocket API 之後，用戶端應用程式就可以連線到它並傳送訊息給它，而且您的後端服務可以將訊息傳送至連線的用戶端應用程式：

- 您可以使用連接`wscat`到 WebSocket API 並向其發送消息以模擬客戶端行為。請參閱the section called “用`wscat`於連接到 WebSocket API 並向其發送消息”。
- 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”。
- 客戶端應用程序可以使用自己的 WebSocket 庫來調用您的 WebSocket API。

用`wscat`於連接到 WebSocket API 並向其發送消息

該`wscat`實用程序是一個方便的工具，用於測試您在 WebSocket API Gateway 中創建和部署的 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

您的後端服務可以使用下列 WebSocket 連線 HTTP 要求，將回呼訊息傳送至連線的用戶端、取得連線資訊或中斷用戶端的連線。

> **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。

在以下命令中，您需要替換為實際*{api-id}*的 API ID，該 ID 顯示在 API Gateway 控制台中或由 AWS CLI create- api 命令返回。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,
  };
};
```

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。

發佈 WebSocket API 供客戶叫用

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

- 使用 WebSocket API 的階段
- 在 WebSocket API Gateway 中部署 API
- WebSocket API 的安全性原則
- 為 WebSocket API 設定自訂網域名稱

使用 WebSocket 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。

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

階段變數是您可以為 WebSocket API 階段定義的索引鍵值配對。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給整合，請使用 AWS Lambda 授權者。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。您可以使用 AWS Command Line Interface (AWS CLI) 來執行此操作。

```
aws lambda add-permission --function-name arn:aws:lambda:XXXXXX:your-lambda-function-
name --source-arn arn:aws:execute-api:us-east-1:YOUR_ACCOUNT_ID:api_id/*/HTTP_METHOD/
```

```
resource --principal apigateway.amazonaws.com --statement-id apigateway-access --action
 lambda:InvokeFunction
```

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 整合認證

您可以使用階段變數做為使用 AWS 者或角色認證 ARN 的一部分，如下列範例所示。

- `arn:aws:iam::<account_id>:${stageVariables.<variable_name>}`

在 WebSocket API Gateway 中部署 API

建立 WebSocket API 之後，您必須部署它，使其可供使用者叫用。

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
> - Authorizers
>
> 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至 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，指定階段 (*`{stageName}`*) 中 WebSocket API 的 URL 格式如下：

```
wss://{api-id}.execute-api.{region}.amazonaws.com/{stageName}
```

為了使 WebSocket API 的 URL 更易於使用，您可以創建自定義域名（例如`api.example.com`）以替換 API 的默認主機名。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 WebSocket API Gateway 主控台建立 API 部署

使用建立 WebSocket API 部署 AWS CLI

若要用 AWS CLI 來建立部署，請使用建立部署指令，如下列範例所示：

```
aws apigatewayv2 --region us-east-1 create-deployment --api-id aabbccddee
```

輸出範例：

```
{
    "DeploymentId": "fedcba",
    "DeploymentStatus": "DEPLOYED",
    "CreatedDate": "2018-11-15T06:49:09Z"
}
```

在您將此部署與階段建立關聯前，都無法呼叫已部署的 API。您可以建立新階段或重複使用您之前建立的階段。

若要建立新階段並將其與部署產生關聯，請使用 create-stage 命令，如下列範例所示：

```
aws apigatewayv2 --region us-east-1 create-stage --api-id aabbccddee --deployment-id
 fedcba --stage-name test
```

輸出範例：

```
{
    "StageName": "test",
    "CreatedDate": "2018-11-15T06:50:28Z",
    "DeploymentId": "fedcba",
    "DefaultRouteSettings": {
        "MetricsEnabled": false,
        "ThrottlingBurstLimit": 5000,
        "DataTraceEnabled": false,
```

```
        "ThrottlingRateLimit": 10000.0
    },
    "LastUpdatedDate": "2018-11-15T06:50:28Z",
    "StageVariables": {},
    "RouteSettings": {}
}
```

若要重複使用現有的階段，請使用 update stage 命令，以新建立的部署 ID (*{deployment-id}*) 更新階段的deploymentId屬性。

```
aws apigatewayv2 update-stage --region {region} \
   --api-id {api-id} \
   --stage-name {stage-name} \
   --deployment-id {deployment-id}
```

使用 WebSocket API Gateway 主控台建立 API 部署

若要使用 API Gateway 主控台建立 WebSocket API 的部署：

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。

WebSocket API 的安全性原則

API Gateway 會TLS_1_2針對所有 WebSocket API 端點強制執行的安全性政策。

安全政策是 Amazon API Gateway 所提供的最低 TLS 版本和加密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。此安全性原則接受 TLS 1.2 和 TLS 1.3 流量，並拒絕 TLS 1.0 流量。

API 支援的 TLS 通訊協定和密碼 WebSocket

下表說明 API 支援的 TLS 通訊協定和密碼。 WebSocket

安全政策	TLS_1
TLS 通訊協定	

安全政策	TLS_1
TLSv1.3	♦
TLSv1.2	♦
TLS 密碼	
TLS_AES_128_GCM_SHA256	♦
TLS_AES_256_GCM_SHA384	♦
TLS_CHACHA20_POLY1305_SHA256	♦
ECDHE-ECDSA-AES128-GCM-SHA256	♦
ECDHE-RSA-AES128-GCM-SHA256	♦
ECDHE-ECDSA-AES128-SHA256	♦
ECDHE-RSA-AES128-SHA256	♦
ECDHE-ECDSA-AES256-GCM-SHA384	♦
ECDHE-RSA-AES256-GCM-SHA384	♦
ECDHE-ECDSA-AES256-SHA384	♦
ECDHE-RSA-AES256-SHA384	♦
AES128-GCM-SHA256	♦
AES128-SHA256	♦
AES256-GCM-SHA384	♦
AES256-SHA256	♦

OpenSSL 和 RFC 密碼名稱

OpenSSL 和 IETF RFC 5246，對於相同的密碼使用不同的名稱。如需密碼名稱的清單，請參閱the section called “OpenSSL 和 RFC 密碼名稱”。

關於其餘 API 和 HTTP API 的資訊

如需有關其他 API 和 HTTP API 的詳細資訊，請參閱the section called “選擇安全性原則”和the section called “適用於 API 的安全性原則”。

為 WebSocket API 設定自訂網域名稱

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`api-id`中由 API Gateway 產生，`region`(AWS 區域) 由您在建立 API 時指定，並`stage`由您在部署 API 時指定。

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 端點很難取回，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 對應至您的 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**
>
> WebSocket API 的自訂網域名稱無法對應至其他 API 或 HTTP API。
> 對於 WebSocket API，支援地區自訂網域名稱。
> 對於 WebSocket API，TLS 1.2 是唯一受支援的 TLS 版本。

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。您的網域名稱必須遵循 RFC 1035 規範，且每個標籤最多可有 63 個八位元組，總共 255 個八位元組。需要時，您可以使用 Amazon Route 53 或使用您選擇的第三方網域註冊機構來註冊網際網路網域。API 的自訂網域名稱可以是已註冊網際網路網域的子網域或根網域 (也稱為 "zone apex") 的名稱。

在 API Gateway 中建立自訂網域名稱，您必須建立或更新 DNS 提供者的資源記錄，以對應至您的 API 端點。如果沒有這種對應，送往自訂網域名稱的 API 請求無法到達 API Gateway。

區域性自訂網域名稱

當您建立區域 API 的自訂網域名稱時，API Gateway 會建立 API 的區域網域名稱。您必須設定 DNS 記錄，將自訂網域名稱對應至區域性網域名稱。您也必須提供自訂網域名稱的憑證。

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，在單一 AWS 帳戶中，您可以設定*.example.com和行a.example.com為不同。

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 API Gateway 映射範本和存取記錄變數參考。

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 帳戶建立了與萬用字元自訂網域名稱衝突的自訂網域名稱，您就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。

自訂網域名稱的憑證

> **Important**
>
> 您可以指定自訂網域名稱所用的憑證。如果您的應用程式使用憑證釘選 (有時稱為 SSL 釘選) 來釘選 ACM 憑證，則應用程式在續訂憑證後 AWS 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。

若要在支援 ACM 的區域中提供自訂網域名稱的憑證，您必須向 ACM 請求憑證。若要在不支援 ACM 的區域中提供區域自訂網域名稱的憑證，您必須將憑證匯入至該區域中的 API Gateway。

若要匯入 SSL/TLS 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。若要將受 AWS 管理憑證用於網域名稱，您只需參考其 ARN 即可。

ACM 可讓您直覺式地設定和使用 API 的自訂網域名稱。您可以建立所指定網域名稱的憑證 (或匯入憑證)、使用 ACM 所提供的憑證 ARN 在 API Gateway 中設定網域名稱，以及將自訂網域名稱下的基本路徑對應至 API 的已部署階段。使用 ACM 所發出的憑證，就不需要擔心公開任何敏感的憑證詳細資訊，例如私有金鑰。

設定自訂網域名稱

如需設定自訂網域名稱的詳細資訊，請參閱 在 中備妥憑證 AWS Certificate Manager 和 在 API Gateway 中設定區域性自訂網域名稱。

使用 API 的 WebSocket API 對應

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 對應中，自訂網域名稱和對應的 API 必須位於相同的 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。
- 您無法將 WebSocket API 對應至與 HTTP API 或 REST API 相同的自訂網域名稱。

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 Management Console

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 對應)。
6. 選擇 Add new mapping (新增對應)。
7. 輸入 API、Stage (階段)，以及選擇性地輸入 Path (路徑)。
8. 選擇 Save (儲存)。

AWS CLI

下列 AWS CLI 命令會建立 API 對應。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
```

AWS CloudFormation

下列 AWS CloudFormation 範例會建立 API 對應。

```
MyApiMapping:
  Type: 'AWS::ApiGatewayV2::ApiMapping'
  Properties:
    DomainName: api.example.com
    ApiMappingKey: 'v1'
    ApiId: !Ref MyApi
    Stage: !Ref MyStage
```

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` 端點來叫用 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。

> **Note**
>
> 當您停用預設端點時，它會影響 API 的所有階段。

下列 AWS CLI 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 指令會建立部署。

```
aws apigatewayv2 create-deployment \
    --api-id abcdef123 \
    --stage-name dev
```

保護您的 WebSocket API

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 AWS 節流限制。

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。下列命令會針對指定的 API 階段和路由設定自訂調節。

```
aws apigatewayv2 update-stage \
    --api-id a1b2c3d4 \
    --stage-name dev \
    --route-settings '{"messages":
{"ThrottlingBurstLimit":100,"ThrottlingRateLimit":2000}}'
```

監控 WebSocket API

您可以使用 CloudWatch 指標和 CloudWatch 日誌來監視 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
- 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。如需詳細資訊，請參閱 使用 Amazon CloudWatch 指標監控 REST API 執行。

WebSocket API 支援下列指標：

指標	描述
ConnectCount	傳送到 `$connect` 路由整合的訊息數。
MessageCount	從用戶端或傳送至用戶端的訊息數目。 WebSocket
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, 舞台	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	針對具有指定 API ID、階段 ID、資源路徑和路由 ID 的 API 方法篩選 API Gateway 指標。 除非您已明確啟用詳細指標，否則 API Gateway 不會傳送這些 CloudWatch 指標。您可以呼叫 API Gateway V2 REST API 的UpdateStage動作，將內`detailedMetricsEnabled` 容更新為`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，請參閱 Amazon CloudWatch 定價。

設定 WebSocket API 的記錄

您可以啟用記錄功能，將記錄寫入記 CloudWatch 錄檔。API 記錄有兩種類型 CloudWatch：執行記錄和存取記錄。在執行記錄中，API Gateway 會管理記 CloudWatch 錄檔。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。

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。

如需如何設定 CloudWatch 記錄的指示，請參閱the section called “使用 CloudWatch API Gateway 主控台設定 API 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)。
`$context.authorizer.requestId`	AWS 端點的要求識別碼。
`$context.authorizer.status`	從授權方傳回的狀態碼。
`$context.authorizer.principalId`	與用戶端所傳送並從 API Gateway Lambda 授權方 Lambda 函數所傳回之權杖建立關聯的主要使用者識別。(Lambda 授權方之前稱為自訂授權方。)
`$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： `"context" : { "key": "value",`

參數	描述

	<pre>"numKey": 1,
	"boolKey": true
	}</pre>
	呼叫 $context.authorizer.key 會傳回 "value" 字串、呼叫 $context.authorizer.numKey 會傳回 "1" 字串，而呼叫 $context.authorizer.boolKey 會傳回 "true" 字串。

$context.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。
$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 。

參數	描述
`$context.identity.accountId`	與請求相關聯的 AWS 帳戶 ID。
`$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。
`$context.identity.cognitoAuthenticationProvider`	提出請求的發起人所使用的 Amazon Cognito 身分驗證提供者清單 (以逗號分隔)。僅在使用 Amazon Cognito 登入資料簽署請求時才可使用。 例如，適用於 Amazon Cognito 使用者集區的身分，`cognito-idp.`*`region`*`.amazonaws.com/`*`user_pool_id`*`,cognito-idp.`*`region`*`.amazonaws.com/`*`user_pool_id`*`:CognitoSignIn:`*`token subject claim`* 如需詳細資訊，請參閱 Amazon Cognito 開發人員指南中的使用聯合身分。
`$context.identity.cognitoAuthenticationType`	提出請求的發起人的 Amazon Cognito 身分驗證類型。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。

參數	描述
$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，狀態碼會從後端 Lambda 函數程式碼傳回 AWS Lambda，而不是從後端 Lambda 函數
$context.integration.latency	整合延遲 (以毫秒為單位)。等同於 $context.integrationLatency 。
$context.integration.requestId	AWS 端點的要求識別碼。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。

參數	描述
$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 策略中使用 ARN，請參閱Amazon API Gateway 與 IAM 搭配運作的方式和使用 IAM 許可控制 API 的存取。

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
ApiMapping	arn:*partition* :apigateway: *region*::/domainnames/ *domain-name* /apimappings/ *id*
ApiMappings	arn:*partition* :apigateway: *region*::/domainnames/ *domain-name* /apimappings
授權方	arn:*partition* :apigateway: *region*::/apis/*api-id*/authorizers/ *id*
Authorizers	arn:*partition* :apigateway: *region*::/apis/*api-id*/authorizers

資源	ARN
Cors	arn:*partition* :apigatew ay: *region*::/apis/*api-id*/cors
部署	arn:*partition* :apigatew ay: *region*::/apis/*api-id*/deployme nts/ *id*
部署	arn:*partition* :apigatew ay: *region*::/apis/*api-id*/deployme nts
DomainName	arn:*partition* :apigatew ay: *region*::/domainnames/ *domain-na me*
DomainNames	arn:*partition* :apigatew ay: *region*::/domainnames
ExportedAPI	arn:*partition* :apigatew ay: *region*::/apis/*api-id*/exports/ *specification*
整合	arn:*partition* :apigatew ay: *region*::/apis/*api-id*/integrat ions/ *integration-id*
整合	arn:*partition* :apigatew ay: *region*::/apis/*api-id*/integrat ions
IntegrationResponse	arn:*partition* :apigatew ay: *region*::/apis/*api-id*/integrat ionresponses/ *integration-respon se*

資源	ARN
IntegrationResponses	arn:*partition* :apigatew ay: *region*::/apis/*api-id*/integrat ionresponses
模型	arn:*partition* :apigatew ay: *region*::/apis/*api-id*/models/*id*
模型	arn:*partition* :apigatew ay: *region*::/apis/*api-id*/models
ModelTemplate	arn:*partition* :apigatew ay: *region*::/apis/*api-id*/models/*id*/ template
路由	arn:*partition* :apigatew ay: *region*::/apis/*api-id*/routes/*id*
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*

資源	ARN
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*
Authorizers	arn:*partition* :apigatew ay: *region*::/restapis/ *api-id*/ authorizers

資源	ARN
BasePathMapping	arn:*partition* :apigatew ay: *region*::/domainnames/ *domain-na me* /basepathmappings/ *basepath*
BasePathMappings	arn:*partition* :apigatew ay: *region*::/domainnames/ *domain-na me* /basepathmappings
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
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*

資源	ARN
DocumentationVersions	arn:*partition* :apigatew ay: *region*::/restapis/ *api-id*/ documentation/versions
DomainName	arn:*partition* :apigatew ay: *region*::/domainnames/ *domain-na me*
DomainNames	arn:*partition* :apigatew ay: *region*::/domainnames
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*
方法	arn:*partition* :apigatew ay: *region*::/restapis/ *api-id*/ resources/ *resource-id* /methods/ *http-method*

資源	ARN
MethodResponse	arn:*partition* :apigateway: *region*::/restapis/ *api-id*/resources/ *resource-id* /methods/ *http-method* /responses/ *status-code*
模型	arn:*partition* :apigateway: *region*::/restapis/ *api-id*/models/*model-name*
模型	arn:*partition* :apigateway: *region*::/restapis/ *api-id*/models
RequestValidator	arn:*partition* :apigateway: *region*::/restapis/ *api-id*/requestvalidators/ *id*
RequestValidators	arn:*partition* :apigateway: *region*::/restapis/ *api-id*/requestvalidators
資源	arn:*partition* :apigateway: *region*::/restapis/ *api-id*/resources/ *id*
資源	arn:*partition* :apigateway: *region*::/restapis/ *api-id*/resources
RestApi	arn:*partition* :apigateway: *region*::/restapis/ *api-id*
RestApis	arn:*partition* :apigateway: *region*::/restapis

資源	ARN
階段	arn:*partition* :apigatew ay: *region*::/restapis/ *api-id*/ stages/*stage-name*
階段	arn:*partition* :apigatew ay: *region*::/restapis/ *api-id*/ stages
標籤	arn:*partition* :apigatew ay: *region*::/tags/*url-encoded-resource-arn*
範本	arn:*partition* :apigatew ay: *region*::/restapis/models / *model-name* /template
UsagePlan	arn:*partition* :apigatew ay: *region*::/usageplans/ *usageplan-id*
UsagePlans	arn:*partition* :apigatew ay: *region*::/usageplans
UsagePlanKey	arn:*partition* :apigatew ay: *region*::/usageplans/ *usageplan-id* /keys/*id*
UsagePlanKeys	arn:*partition* :apigatew ay: *region*::/usageplans/ *usageplan-id* /keys
VpcLink	arn:*partition* :apigatew ay: *region*::/vpclinks/ *vpclink-id*
VpcLinks	arn:*partition* :apigatew ay: *region*::/vpclinks

execute-api（HTTP API，WebSocket API 和其餘 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”。

使用 OpenAPI 的 API Gateway 延伸

API Gateway 延伸模組支援 REST API 和 HTTP API 的特定授權和 API 閘道特定 API 整合。 AWS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 API Gateway 匯出 HTTP API。

主題

- x-amazon-apigateway-any-方法對象
- x-amazon-apigateway-cors 物件
- x-amazon-apigateway-api-鍵源屬性
- x-amazon-apigateway-auth 物件
- x-amazon-apigateway-authorizer 物件
- x-amazon-apigateway-authtype 財產
- x-amazon-apigateway-binary-媒體類型屬性
- x-amazon-apigateway-documentation 物件
- x-amazon-apigateway-endpoint-配置對象
- x-amazon-apigateway-gateway-響應對象
- x-amazon-apigateway-gateway-響應. 網關響應對象
- x-amazon-apigateway-gateway-回應. 回應參數物件
- x-amazon-apigateway-gateway-響應. 響應模板對象
- x-amazon-apigateway-importexport-版本
- x-amazon-apigateway-integration 物件
- x-amazon-apigateway-integrations 物件
- x-amazon-apigateway-integration。requestTemplates 對象
- x-amazon-apigateway-integrationrequestParameters 對象

- x-amazon-apigateway-integration. 回應物件
- x-amazon-apigateway-integration. 回應物件
- x-amazon-apigateway-integration. responseTemplates 物件
- x-amazon-apigateway-integration. responseParameters 物件
- x-amazon-apigateway-integration.tlsconfig 物件
- x-amazon-apigateway-minimum-壓縮大小
- x-amazon-apigateway-policy
- x-amazon-apigateway-request-驗證器屬性
- x-amazon-apigateway-request-驗證對象
- x-amazon-apigateway-request-驗證器. 請求驗證器對象
- x-amazon-apigateway-tag-值屬性

x-amazon-apigateway-any-方法對象

在 OpenAPI 路徑項目物件中指定 API Gateway 全部截獲 ANY 方法的 OpenAPI 操作物件。此物件會與其他操作物件並存，並會截獲未明確宣告的任何 HTTP 方法。

下表列有 API Gateway 延伸的屬性。如需其他 OpenAPI 操作屬性，請參閱 OpenAPI 規定。

屬性

屬性名稱	類型	描述
`isDefaultRoute`	`Boolean`	指定路由是否為 `$default` 路由。僅支援 HTTP API。如需進一步了解，請參閱使用 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`、`HTTP`、`HTTP_PROXY` 或 `MOCK`。

x-amazon-apigateway-any-方法示例

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，請參閱為 HTTP API 設定 CORS。

屬性

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 例子

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-鍵源屬性

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。如需設定方法以需要 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 金鑰的方法”。

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-關鍵源示例

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

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 例子

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 和 OpenAPI 3 中的安全性定義。

屬性

屬性名稱	類型	描述
`type`	`string`	授權方的類型。這是必要屬性。 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 使用者集區進行 API 存取控制的授權方指定 `cognito_user_pools`。

屬性名稱	類型	描述
		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： `"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授權者。
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。

x-amazon-apigateway-authorizer 其餘 API 的範例

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" : "oauth2", // Specifies the authorization
mechanism for the client.
     "x-amazon-apigateway-authorizer" : {       // An API Gateway Lambda authorizer
definition
       "type" : "token",                         // Required property and the value
must "token"
       "authorizerUri" : "arn:aws:apigateway:us-east-1:lambda:path/2015-03-31/
functions/arn:aws:lambda:us-east-1:account-id:function:function-name/invocations",
       "authorizerCredentials" : "arn:aws:iam::account-id:role",
       "identityValidationExpression" : "^x-[a-z]+",
       "authorizerResultTtlInSeconds" : 60
     }
   }
 }
```

下列 OpenAPI 操作物件程式碼片段會設定 `GET /http` 使用上述的 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"
       }
     }
   }
```

x-amazon-apigateway-authorizer 適用於 HTTP API 的範例

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 財產

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 例子

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-媒體類型屬性

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-媒體類型的例子

下列範例示範 API 的編碼查詢順序。

```
"x-amazon-apigateway-binary-media-types": [ "application/octet", "image/jpeg" ]
```

x-amazon-apigateway-documentation 物件

定義要匯入 API Gateway 的文件部分。這個物件是包含 `DocumentationPart` 執行個體陣列的 JSON 物件。

屬性

屬性名稱	類型	描述
`documentationParts`	`Array`	匯出或匯入的 `DocumentationPart` 執行個體陣列。
`version`	`String`	匯出文件部分快照的版本識別符。

x-amazon-apigateway-documentation 例子

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-配置對象

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

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。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	要針對其建立 Route 53 別名記錄的 VpcEndpoint 識別碼清單。僅支援 PRIVATE 端點類型的 REST API。

x-amazon-apigateway-endpoint-配置示例

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
}
```

x-amazon-apigateway-gateway-響應對象

將 API 的閘道回應定義為索引鍵值配對的字串對GatewayResponse映。該擴展適用於根層級 OpenAPI 結構。

屬性

屬性名稱	類型	描述
responseType	x-amazon-apigateway-gateway-響應。網關響應	指定之 *responseType* 的 GatewayResponse 。

x-amazon-apigateway-gateway-響應示例

下列 OpenAPI 的 API Gateway 延伸模組範例會定義包含兩個GatewayResponse執行個體的GatewayResponses對應，一個用於類型，另一個用於DEFAULT_4XX類型。INVALID_API_KEY

```
{
  "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-響應. 網關響應對象

定義指定回應類型的閘道回應，包括狀態碼、任何適用的回應參數，或回應範本。

屬性

屬性名稱	類型	描述
responseParameters	x-amazon-apigateway-gateway-回應. 回應參數	指定GatewayResponse參數，即標題參數。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-響應. 響應模板	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-響應。網關響應示例

下列 OpenAPI Gateway 延伸模組範例會GatewayResponse定義自訂回INVALID_API_KEY應以傳回狀態碼456、傳入要求的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-回應. 回應參數物件

定義索引鍵值配對的對 string-to-string 映，以從傳入的要求參數或使用常值字串產生閘道回應參數。僅支援 REST API。

屬性

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，請參閱將方法請求資料對應到整合請求參數。

x-amazon-apigateway-gateway-回應. 回應參數範例

下列 OpenAPI 擴充功能範例顯示GatewayResponse回應參數對應運算式，以啟用對網域資源的 CORS 支援。*.example.domain

```
    "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-響應. 響應模板對象

將對應範本定義為指定閘道回 string-to-string 應的索引鍵值配對對GatewayResponse映。每個鍵值對的鍵都是內容類型。例如 "application/json"，而值則是簡單變數替換的字串化對應範本。GatewayResponse 對應範本不由 Velocity 範本語言 (VTL) 引擎處理。

屬性

屬性名稱	類型	描述
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-響應。響應模板示例

以下 OpenAPI 擴展示了一個GatewayResponse映射模板，用於將 API Gateway 生成的錯誤響應自定義為特定於應用程序的格式。

```
      "responseTemplates": {
        "application/json": "{ \"message\": $context.error.messageString, \"type\":
$context.error.responseType, \"statusCode\": '488' }"
      }
```

以下 OpenAPI 擴展示例顯示了一個GatewayResponse映射模板，用於覆蓋 API Gateway 生成的錯誤響應，並帶有靜態錯誤消息。

```
      "responseTemplates": {
        "application/json": "{ \"message\": 'API-specific errors' }"
      }
```

x-amazon-apigateway-importexport-版本

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-版本示例

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

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VpcLink的識別碼。
connectionType	string	整合連線類型。有效值是私有整合的 "VPC_LINK" 或 "INTERNET" 。
credentials	string	對於以 AWS IAM 角色為基礎的登入資料，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。 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_T

屬性名稱	類型	描述
		O_TEXT ，將二進位承載轉換為 base64 編碼的字串或將文字承載轉換為 utf-8 編碼的字串，或以原生方式傳遞文字承載，不予修改；和 2) CONVERT_TO_BINARY ，將文字承載轉換為 Base64 編碼的 blob，或以原生方式傳遞二進位承載，不予修改。
httpMethod	string	用於整合請求中的 HTTP 方法。若要呼叫 Lambda 函數，該值必須為 POST。
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”。
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。

屬性名稱	類型	描述
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”。
requestParameters	x-amazon-apigateway-integrationrequestParameters 對象	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。 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 AWS_PROXY 的 integrationSubtype 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 對象	指定 MIME 類型的請求承載對應範本。
responses	x-amazon-apigateway-integration. 回應物件	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。

屬性名稱	類型	描述
timeoutInMillis	integer	整合逾時，介於 50 毫秒到 29,000 毫秒之間。
type	string	與指定後端整合的類型。有效值為： • http 或 http_proxy ：用於與 HTTP 後端整合。 • aws_proxy ，以便與 AWS Lambda 函數整合。 • aws，用於整合 AWS Lambda 函數或其他 AWS 服務，例如 Amazon DynamoDB、Amazon 簡單通知服務或 Amazon 簡單佇列服務。 • mock：用於與 API Gateway 整合，但不呼叫任何後端。 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 例子

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

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 例子

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 對象

為指定 MIME 類型的請求承載指定對應範本。

屬性

屬性名稱	類型	描述
MIME type	string	MIME 類型範例為 application/json 。如需建立映射範本的相關資訊，請參閱 PetStore 對映樣板。

x-amazon-apigateway-integration。requestTemplates 示例

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-integrationrequestParameters 對象

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

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>*

屬性名稱	類型	描述
		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，$context. *VARIABLE_NAME* 、$stageVariables. *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**
>
> 如果您通過 OpenAPI 創建資源 AWS CloudFormation，或者靜態值應該用單引號括起來。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. 回應物件

定義方法的回應，並指定從整合回應到方法回應的參數對應或承載對應。

屬性

屬性名稱	類型	描述
######	x-amazon-apigateway-integration. 回應物件	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 叫用，當 Lambda 函數執行擲回例外狀況時，正則運算式會套用至 AWS Lambda 作為失敗回應主體所傳回之錯誤資訊物件的`errorMessage` 欄位。 **Note** ######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。它不對應於 API Gateway REST API

屬性名稱	類型	描述
		中IntegrationResponse資源的任何識別碼。

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. 回應物件

定義回應，並指定從整合回應到方法回應的參數對應或承載對應。

屬性

屬性名稱	類型	描述
`statusCode`	`string`	方法回應的 HTTP 狀態碼，例如 `"200"`。這必須對應到 OpenAPI 操作 `responses` 欄位中的相符回應。
`responseTemplates`	x-amazon-apigateway-integration. responseTemplates 物件	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。
`contentHandling`	`string`	回應承載編碼轉換類型。有效值為 1) `CONVERT_TO_TEXT` ，將二進位承載轉換為 base64 編碼的字串或將文字承載轉換為 `utf-8` 編碼的字串，或以原生方式傳遞文字承載，不予修改；和 2) `CONVERT_TO_BINARY` ，將文字承載轉換為 Base64 編碼的 blob，或以原生方式傳遞二進位承載，不予修改。

x-amazon-apigateway-integration.response 範例

下列範例會為從後端產生 `302` 或 `application/json` MIME 類型之承載的方法，定義 `application/xml` 回應。該回應使用提供的對應範本，並會傳回出自於方法之 `Location` 標頭中整合回應的重新導向 URL。

```
{
    "statusCode" : "302",
```

```
    "responseTemplates" : {
         "application/json" : "#set ($root=$input.path('$')) { \"stage\": \"$root.name
\", \"user-id\": \"$root.key\" }",
         "application/xml" : "#set ($root=$input.path('$')) <stage>$root.name</stage> "
    },
    "responseParameters" : {
        "method.response.header.Location": "integration.response.body.redirect.url"
    }
}
```

x-amazon-apigateway-integration. responseTemplates 物件

為指定 MIME 類型的回應承載指定對應範本。

屬性

屬性名稱	類型	描述
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PetStore 對映樣板。*MIME ##*範例為 application/json。

x-amazon-apigateway-integration. 回應範本範例

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

屬性名稱	類型	描述
`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

下列範列會將整合回應的 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 物件

指定整合的 TLS 組態。

屬性

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，

屬性名稱	類型	描述
		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，其中包括檢查憑證的到期日、主機名稱以及是否存在根憑證授權單位。屬於私人授權單位的根憑證必須滿足下列限制： • x509 延伸 keyUsage 必須有 keyCertSign 。 • x509 延伸 basicConstraints 必須有 CA:TRUE。 僅支援 HTTP 與 HTTP_PROXY 整合。 ⚠ Warning 不建議啟用 insecureSkipVerification ，特別是對於與公有 HTTPS 端點的整合。如果啟用insecureSkipVerification ，就會增加 man-in-the-middle 攻擊的風險。

屬性名稱	類型	描述
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 範例

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-壓縮大小

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，請參閱啟用 API 的承載壓縮功能。

x-amazon-apigateway-minimum-壓縮大小示例

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 資源政策控制對 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，`"execute-api:/*"`會使用目前的區域、您的 AWS 帳戶 ID 和目前的 REST API ID 轉換為。`arn:aws:execute-api:`*`region`*`:`*`account-id`*`:`*`api-id`*`/*`

```
"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-驗證器屬性

指定請求驗證程式，方法是參考 *request_validator_name* 對應的 x-amazon-apigateway-request-驗證對象，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-驗證對象

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

屬性名稱	類型	描述
request_validator_name	x-amazon-apigateway-request-驗證器. 請求驗證器對象	指定包含具名驗證程式的驗證規則。例如： `"basic" : { "validateRequestBody" : true, "validateRequestParameters" : true },` 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-驗證器屬性 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-驗證器. 請求驗證器對象

將請求驗證程式的驗證規則指定為 x-amazon-apigateway-request-驗證對象 對應定義的一部分。

屬性

屬性名稱	類型	描述
validateRequestBody	Boolean	指定驗證請求內文 (true) 或不驗證 (false)。
validateRequestParameters	Boolean	指定驗證必要的請求參數 (true) 或不驗證 (false)。

x-amazon-apigateway-request-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-tag-值屬性

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，以指定 HTTP API 的標 AWS 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

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

雲安全 AWS 是最高的優先級。身為 AWS 客戶，您可以從資料中心和網路架構中獲益，該架構專為滿足對安全性最敏感的組織的需求而打造。

安全是 AWS 與您之間共同承擔的責任。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 雲端中執行 AWS 服務的基礎架構。 AWS 還為您提供可以安全使用的服務。若要了解適用於 Amazon API Gateway 的合規計劃，請參閱合規計劃AWS 服務範圍內的合規計劃AWS的服務。
- 雲端中的安全性 — 您的責任取決於您使用的 AWS 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。您也會學到如何使用其他可 AWS 協助您監控和保護 API Gateway 資源的服務。

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

AWS 共同責任模型適用於 Amazon API Gateway 的資料保護。如此模型所述，AWS 負責保護執行所有 AWS 雲端 的全球基礎設施。您必須負責維護在此基礎設施上託管之內容的控制權。您也必須負責

您所使用的 AWS 服務 安全性設定和管理工作。如需有關資料隱私權的詳細資訊，請參閱資料隱私權常見問答集。如需有關歐洲資料保護的相關資訊，請參閱 AWS 安全性部落格上的 AWS 共同的責任模型和 GDPR 部落格文章。

基於資料保護目的，建議您使用 AWS IAM Identity Center 或 AWS Identity and Access Management (IAM) 保護 AWS 帳戶 憑證，並設定個人使用者。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 AWS CloudTrail 設定 API 和使用者活動日誌記錄。
- 使用 AWS 加密解決方案，以及 AWS 服務 內的所有預設安全控制項。
- 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。
- 如果您在透過命令列介面或 API 存取 AWS 時，需要 FIPS 140-2 驗證的加密模組，請使用 FIPS 端點。如需有關 FIPS 和 FIPS 端點的詳細資訊，請參閱聯邦資訊處理標準 (FIPS) 140-2 概觀。

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。這包含在您利用主控台、API、AWS CLI 或 AWS 軟體開發套件使用 API Gateway 或其他 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，請參閱啟用 API 快取以提升回應能力。

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。

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性原則。

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 AWS 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 “私人休息 API”。

適用於 Amazon API Gateway 的 Identity and Access Management

AWS Identity and Access Management (IAM) 是一種 AWS 服務，讓管理員能夠安全地控制對 AWS 資源的存取權限。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 是一種您可以免費使用的 AWS 服務。

主題

- 對象
- 使用身分來驗證
- 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

對象

根據您在 API Gateway 中所進行的工作而定，AWS Identity and Access Management (IAM) 的使用方式會不同。

服務使用者 – 如果您使用 API Gateway 執行任務，您的管理員會為您提供您需要的登入資料和許可。隨著您為了執行作業而使用的 API Gateway 功能數量變多，您可能會需要額外的許可。了解存取的管理方式可協助您向管理員請求正確的許可。如果您無法存取 API Gateway 中的某項功能，請參閱疑難排解 Amazon API Gateway 身分和存取。

服務管理員 – 若您在公司負責管理 API Gateway 資源，則應該具備 API Gateway 的完整存取權。您的任務是判斷服務使用者應存取的 API Gateway 功能和資源。您接著必須將請求提交給您的 IAM 管理員，來變更您服務使用者的許可。檢閱此頁面上的資訊，了解 IAM 的基本概念。若要進一步了解貴公司可搭配 API Gateway 使用 IAM 的方式，請參閱 Amazon API Gateway 與 IAM 搭配運作的方式。

IAM 管理員 – 如果您是 IAM 管理員，建議您掌握如何撰寫政策以管理 API Gateway 存取權的詳細資訊。若要檢視您可以在 IAM 中使用的範例 API Gateway 身分型政策，請參閱 Amazon API Gateway 身分型政策範例。

使用身分來驗證

身分驗證是使用身分憑證登入 AWS 的方式。您必須以 AWS 帳戶根使用者、IAM 使用者身分，或擔任 IAM 角色進行 驗證 (登入至 AWS)。

您可以使用透過身分來源提供的憑證，以聯合身分登入 AWS。AWS IAM Identity Center(IAM Identity Center) 使用者、貴公司的單一登入身分驗證和您的 Google 或 Facebook 憑證都是聯合身分的範例。當您以聯合身分登入時，您的管理員先前已設定使用 IAM 角色的聯合身分。當您 AWS 藉由使用聯合進行存取時，您會間接擔任角色。

根據您的使用者類型，您可以登入 AWS Management Console 或 AWS 存取入口網站。如需有關登入至 AWS 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。

如果您是以程式設計的方式存取 AWS，AWS 提供了軟體開發套件 (SDK) 和命令行介面 (CLI)，以便使用您的憑證透過密碼編譯方式簽署您的請求。如果您不使用 AWS 工具，您必須自行簽署請求。如需使用建議的方法自行簽署請求的詳細資訊，請參閱 IAM 使用者指南中的簽署 AWS API 請求。

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

AWS 帳戶 根使用者

如果是建立 AWS 帳戶，您會先有一個登入身分，可以完整存取帳戶中所有 AWS 服務 與資源。此身分稱為 AWS 帳戶 根使用者，使用建立帳戶時所使用的電子郵件地址和密碼即可登入並存取。強烈建議您不要以根使用者處理日常作業。保護您的根使用者憑證，並將其用來執行只能由根使用者執行的任務。如需這些任務的完整清單，了解需以根使用者登入的任務，請參閱 IAM 使用者指南中的需要根使用者憑證的任務。

IAM 使用者和群組

IAM 使用者是您 AWS 帳戶 中的一種身分，具備單一人員或應用程式的特定許可。建議您盡可能依賴暫時性憑證，而不是擁有建立長期憑證 (例如密碼和存取金鑰) 的 IAM 使用者。但是如果特定使用案例需要擁有長期憑證的 IAM 使用者，建議您輪換存取金鑰。如需詳細資訊，請參閱 IAM 使用者指南中的為需要長期憑證的使用案例定期輪換存取金鑰。

IAM 群組是一種指定 IAM 使用者集合的身分。您無法以群組身分登入。您可以使用群組來一次為多名使用者指定許可。群組可讓管理大量使用者許可的過程變得更為容易。例如，您可以擁有一個名為 IAMAdmins 的群組，並給予該群組管理 IAM 資源的許可。

使用者與角色不同。使用者只會與單一人員或應用程式建立關聯，但角色的目的是在由任何需要它的人員取得。使用者擁有永久的長期憑證，但角色僅提供臨時憑證。如需進一步了解，請參閱IAM 使用者指南中的建立 IAM 使用者 (而非角色) 的時機。

IAM 角色

IAM 角色是您 AWS 帳戶 中的一種身分，具備特定許可。它類似 IAM 使用者，但不與特定的人員相關聯。您可以在 AWS Management Console 中透過切換角色來暫時取得 IAM 角色。您可以透過呼叫 AWS CLI 或 AWS API 操作，或是使用自訂 URL 來取得角色。如需使用角色的方法詳細資訊，請參閱 IAM 使用者指南中的使用 IAM 角色。

使用臨時性憑證的 IAM 角色在下列情況中非常有用：

- 聯合身分使用者存取 — 如需向聯合身分指派許可，請建立角色，並為角色定義許可。當聯合身分進行身分驗證時，該身分會與角色建立關聯，並獲授予由角色定義的許可。如需有關聯合角色的詳細資訊，請參閱 IAM 使用者指南中的為第三方身分供應商建立角色。如果您使用 IAM Identity Center，則需要設定許可集。為控制身分驗證後可以存取的內容，IAM Identity Center 將許可集與 IAM 中的角色相關聯。如需有關許可集的資訊，請參閱 AWS IAM Identity Center 使用者指南中的許可集。

- 暫時 IAM 使用者許可：使用者可以擔任 IAM 角色或角色來暫時針對特定任務採用不同的許可。
- 跨帳戶存取權：您可以使用 IAM 角色，允許不同帳戶中的某人 (信任的主體) 存取您帳戶的資源。角色是授予跨帳戶存取權的主要方式。但是，針對某些 AWS 服務，您可以將政策直接連接到資源 (而非使用角色作為代理)。如需了解使用角色和資源型政策進行跨帳戶存取之間的差異，請參閱IAM 使用者指南中的 IAM 角色與資源類型政策的差異。
- 跨服務存取權：有些 AWS 服務 會使用其他 AWS 服務 中的功能。例如，當您在服務中進行呼叫時，該服務通常會在 Amazon EC2 中執行應用程式或將物件存放在 Amazon Simple Storage Service (Amazon S3) 中。服務可能會使用呼叫主體的許可、使用服務角色或使用服務連結角色來執行此作業。
  - 轉發存取工作階段 (FAS)：當您使用 IAM 使用者或角色在 AWS 中執行動作時，系統會將您視為主體。使用某些服務時，您可能會執行某個動作，進而在不同服務中啟動另一個動作。FAS 使用主體的許可呼叫 AWS 服務，搭配請求 AWS 服務以向下游服務發出請求。只有在服務收到需要與其他 AWS 服務或資源互動才能完成的請求之後，才會提出 FAS 請求。在此情況下，您必須具有執行這兩個動作的許可。如需提出 FAS 請求時的政策詳細資訊，請參閱《轉發存取工作階段》https://docs.aws.amazon.com/IAM/latest/UserGuide/access_forward_access_sessions.html。
  - 服務角色 – 服務角色是服務擔任的 IAM 角色，可代表您執行動作。IAM 管理員可以從 IAM 內建立、修改和刪除服務角色。如需詳細資訊，請參閱 IAM 使用者指南中的建立角色以委派許可給 AWS 服務 服務。
  - 服務連結角色：服務連結角色是一種連結到 AWS 服務 的服務角色類型。服務可以擔任代表您執行動作的角色。服務連結角色會顯示在您的 AWS 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。
- 在 Amazon EC2 上執行的應用程式：針對在 EC2 執行個體上執行並提出 AWS CLI 和 AWS API 請求的應用程式，您可以使用 IAM 角色來管理臨時性憑證。這是在 EC2 執行個體內存放存取金鑰的較好方式。如需指派 AWS 角色給 EC2 執行個體並提供其所有應用程式使用，您可以建立連接到執行個體的執行個體設定檔。執行個體設定檔包含該角色，並且可讓 EC2 執行個體上執行的程式取得臨時性憑證。如需詳細資訊，請參閱 IAM 使用者指南中的利用 IAM 角色來授予許可給 Amazon EC2 執行個體上執行的應用程式。

如需了解是否要使用 IAM 角色或 IAM 使用者，請參閱 IAM 使用者指南中的建立 IAM 角色 (而非使用者) 的時機。

使用政策管理存取權

您可以透過建立政策並將其連接到 AWS 身分或資源，在 AWS 中控制存取。政策是 AWS 中的一個物件，當其和身分或資源建立關聯時，便可定義其許可。AWS 會在主體 (使用者、根使用者或角色工作

階段) 發出請求時評估這些政策。政策中的許可決定是否允許或拒絕請求。大部分政策以 JSON 文件形式存放在 AWS 中。如需 JSON 政策文件結構和內容的詳細資訊，請參閱 IAM 使用者指南中的 JSON 政策概觀。

管理員可以使用 AWS JSON 政策來指定誰可以存取哪些內容。也就是說，哪個主體在什麼条件下可以對什麼資源執行哪些動作。

根據預設，使用者和角色沒有許可。若要授予使用者對其所需資源執行動作的許可，IAM 管理員可以建立 IAM 政策。然後，管理員可以將 IAM 政策新增至角色，使用者便能擔任這些角色。

IAM 政策定義該動作的許可，無論您使用何種方法來執行操作。例如，假設您有一個允許 `iam:GetRole` 動作的政策。具備該政策的使用者便可以從 AWS Management Console、AWS CLI 或 AWS API 取得角色資訊。

身分型政策

身分型政策是可以附加到身分 (例如 IAM 使用者、使用者群組或角色) 的 JSON 許可政策文件。這些政策可控制身分在何種條件下能對哪些資源執行哪些動作。若要了解如何建立身分類型政策，請參閱《IAM 使用者指南》中的建立 IAM 政策。

身分型政策可進一步分類成內嵌政策或受管政策。內嵌政策會直接內嵌到單一使用者、群組或角色。受管政策則是獨立的政策，您可以將這些政策連接到 AWS 帳戶 中的多個使用者、群組和角色。受管政策包含 AWS 管理政策和客戶管理政策。如需了解如何在受管政策及內嵌政策間選擇，請參閱 IAM 使用者指南中的在受管政策和內嵌政策間選擇。

資源型政策

資源型政策是連接到資源的 JSON 政策文件。資源型政策的最常見範例是 Amazon Simple Storage Service (Amazon S3) 儲存貯體政策和 IAM 角色信任政策。在支援資源型政策的服務中，服務管理員可以使用它們來控制對特定資源的存取權限。對於附加政策的資源，政策會定義指定的主體可以對該資源執行的動作以及在何種條件下執行的動作。您必須在資源型政策中指定主體。主體可以包括帳戶、使用者、角色、聯合身分使用者或 AWS 服務。

資源型政策是位於該服務中的內嵌政策。您無法在資源型政策中使用來自 IAM 的 AWS 受管政策。

存取控制清單 (ACL)

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

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

其他政策類型

AWS 支援其他較少見的政策類型。這些政策類型可設定較常見政策類型授予您的最大許可。

- 許可界限：許可界限是一種進階功能，可供您設定身分型政策能授予 IAM 實體 (IAM 使用者或角色) 的最大許可。您可以為實體設定許可界限。所產生的許可會是實體的身分型政策和其許可界限的交集。會在 `Principal` 欄位中指定使用者或角色的資源型政策則不會受到許可界限限制。所有這類政策中的明確拒絕都會覆寫該允許。如需許可界限的詳細資訊，請參閱 IAM 使用者指南中的 IAM 實體許可界限。
- 服務控制政策 (SCP)：SCP 是 JSON 政策，可指定 AWS Organizations 中組織或組織單位 (OU) 的最大許可。AWS Organizations 服務可用來分組和集中管理您企業所擁有的多個 AWS 帳戶。若您啟用組織中的所有功能，您可以將服務控制政策 (SCP) 套用到任何或所有帳戶。SCP 會限制成員帳戶中實體的許可，包括每個 AWS 帳戶根使用者。如需 Organizations 和 SCP 的詳細資訊，請參閱 AWS Organizations 使用者指南中的 SCP 運作方式。
- 工作階段政策：工作階段政策是一種進階政策，您可以在透過編寫程式的方式建立角色或聯合身分使用者的暫時工作階段時，作為參數傳遞。所產生工作階段的許可會是使用者或角色的身分型政策和工作階段政策的交集。許可也可以來自資源型政策。所有這類政策中的明確拒絕都會覆寫該允許。如需詳細資訊，請參閱 IAM 使用者指南中的工作階段政策。

多種政策類型

將多種政策類型套用到請求時，其結果形成的許可會更為複雜、更加難以理解。若要了解 AWS 在涉及多種政策類型時如何判斷是否允許一項請求，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。若要取得 API Gateway 和其他 AWS 服務如何使用 IAM 的高層級檢視，請參閱 IAM 使用者指南中的使用 IAM 的 AWS 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”。

```
{
  "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)。

如需更多詳細資訊，請參閱 the section called “身分型政策範例”。

資源

管理員可以使用 AWS JSON 政策來指定誰可以存取哪些內容。也就是說，哪個主體在什麼条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。陳述式必須包含 Resource 或 NotResource 元素。最佳實務是使用其 Amazon Resource Name (ARN) 來指定資源。您可以針對支援特定資源類型的動作 (稱為資源層級許可) 來這麼做。

對於不支援資源層級許可的動作 (例如列出作業)，請使用萬用字元 (*) 來表示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。

條件鍵

管理員可以使用 AWS JSON 政策來指定誰可以存取哪些內容。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 (或 Condition 區塊) 可讓您指定使陳述式生效的條件。Condition 元素是選用項目。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。

若您在陳述式中指定多個 Condition 元素，或是在單一 Condition 元素中指定多個索引鍵，AWS 會使用邏輯 AND 操作評估他們。若您為單一條件金鑰指定多個值，AWS 會使用邏輯 OR 操作評估條件。必須符合所有條件，才會授予陳述式的許可。

您也可以在指定條件時使用預留位置變數。例如，您可以只在使用者使用其 IAM 使用者名稱標記時，將存取資源的許可授予該 IAM 使用者。如需詳細資訊，請參閱《IAM 使用者指南》中的 IAM 政策元素：變數和標籤。

AWS 支援全域條件金鑰和服務特定的條件金鑰。若要查看 AWS 全域條件金鑰，請參閱 IAM 使用者指南中的 AWS 全域條件內容金鑰。

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。如需更多詳細資訊，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 帳戶中具備特定許可的實體。

搭配 API Gateway 使用暫時登入資料

您可以搭配聯合使用暫時登入資料登入、擔任 IAM 角色，或是擔任跨帳戶角色。您取得暫時安全登入資料的方式是透過呼叫 AWS STS API 操作 (例如，AssumeRole 或 GetFederationToken)。

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 Management Console、AWS CLI 或 AWS 軟體開發套件執行任務。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 操作的所需許可。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

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 受管政策並朝向最低權限許可的目標邁進 — 如需開始授予許可給使用者和工作負載，請使用 AWS 受管政策，這些政策會授予許可給許多常用案例。它們可在您的 AWS 帳戶 中使用。我們建議您定義特定於使用案例的 AWS 客戶管理政策，以便進一步減少許可。如需詳細資訊，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 套用許可的詳細資訊，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。您也可以使用條件來授予對服務動作的存取權，前提是透過特定 AWS 服務 (例如 AWS CloudFormation) 使用條件。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，可協助您編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 IAM Access Analyzer 政策驗證。
- 需要多重要素驗證 (MFA) – 如果存在需要 AWS 帳戶中 IAM 使用者或根使用者的情況，請開啟 MFA 提供額外的安全性。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 使用者指南中的設定 MFA 保護的 API 存取。

有關 IAM 中最佳實務的詳細資訊，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 使用者檢視連接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上，或是使用 AWS CLI 或 AWS API 透過編寫程式的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

此政策範例為使用者提供許可，以取得 HTTP 或 WebSocket API 所有資源的相關資訊，並在 us-east-1 的 AWS 區域中使用 a123456789 識別符。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。

```
{
  "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。

```
{
  "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。

```
{
  "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。

```
{
    "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。

```
{
  "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。

```
{
  "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/*"
      ]
    }
  ]
}
```

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 帳戶以外的人員存取我的 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
```

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。

如需任何協助，請聯絡您的 AWS 管理員。您的管理員提供您的登入憑證。

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 允許您傳遞現有的角色至該服務，而無須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 沒有將角色傳遞至該服務的許可。

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

在這種情況下，Mary 的政策必須更新，允許她執行 iam:PassRole 動作。

如需任何協助，請聯絡您的 AWS 管理員。您的管理員提供您的登入憑證。

我想要允許 AWS 帳戶以外的人員存取我的 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對象取得該角色。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。
- 若要了解如何存取您擁有的所有 AWS 帳戶 所提供的資源，請參閱 IAM 使用者指南中的將存取權提供給您所擁有的另一個 AWS 帳戶 中的 IAM 使用者。
- 如需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擁有的 AWS 帳戶。
- 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)。
- 若要了解使用角色和資源型政策進行跨帳戶存取之間的差異，請參閱《IAM 使用者指南》中的 IAM 角色與資源型政策的差異。

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，並包含服務代表您呼叫其他服 AWS 務所需的所有權限。

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 使用名為的服務連結角色 AWSServiceRoleForAPIGateway— 允許 API Gateway 代表您存取 Elastic Load Balancing、Amazon 資料 Firehose 和其他服務資源。

服 AWSServiceRoleForAPIGateway 務連結角色會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：

```
{
    "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

您不需要手動建立一個服務連結角色。當您在、或 API 中建立 API、自訂網域名稱或 VPC 連結時 AWS CLI， AWS API Gateway 會為您建立服務連結角色。 AWS Management Console

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。如需更多資訊，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。

若要刪除使用的 API Gateway 資源 AWSServiceRoleForAPIGateway

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 主控台或 AWS API 刪除 AWSServiceRoleForAPIGateway服務連結角色。 AWS CLI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 API Gateway AWS 受管政策更新的詳細資料，因為此服務開始追蹤這些變更。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

監控是維持 API Gateway 和 AWS 解決方案的可靠性、可用性和效能的重要組成部分。您應該從 AWS 解決方案的所有部分收集監視資料，以便在發生多點失敗時更輕鬆地偵錯。 AWS 提供數種工具來監控 API Gateway 資源並回應潛在事件：

Amazon CloudWatch 日誌

若要協助偵錯與要求執行或用戶端存取 API 相關的問題，您可以啟用 CloudWatch Logs 來記錄 API 呼叫。如需詳細資訊，請參閱 the section called “CloudWatch 日誌”。

Amazon CloudWatch 警報

您可以使用 CloudWatch 警示來監視指定期間內的單一量度。如果指標超過指定臨界值，則會向 Amazon 簡單通知服務主題或 AWS Auto Scaling 政策傳送通知。 CloudWatch 當測量結果處於特定狀態時，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 度量”。

通往 Firehose 洪的通道紀錄

若要協助偵錯與用戶端存取 API 相關的問題，您可以啟用 Firehose 記錄 API 呼叫。如需詳細資訊，請參閱 the section called “Firehose”。

AWS CloudTrail

CloudTrail 提供使用者、角色或 AWS 服務在 API Gateway 中採取的動作記錄。使用收集的資訊 CloudTrail，您可以判斷向 API Gateway 發出的要求、提出要求的來源 IP 位址、提出要求的人員、提出要求的時間，以及其他詳細資訊。如需詳細資訊，請參閱 the section called “使用 CloudTrail”。

AWS X-Ray

X-Ray 是一項 AWS 服務，可收集應用程式所提供之要求的相關資料，並使用它來建構服務對應，供您識別應用程式的問題，以及進行最佳化的機會。如需詳細資訊，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 來定義規則，以評估資源組態是否符合資料。 AWS Config 規則代表 API Gateway 資源的理想組態設定。如果資源違反規則且被標記為不合規， 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”。

使用記錄 Amazon API Gateway API 呼叫 AWS CloudTrail

Amazon API Gateway 已與服務整合 AWS CloudTrail，該服務可提供使用者、角色或角色所採取的動作記錄 AWS 服務。 CloudTrail 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 API Gateway 服務 API 的程式碼呼叫。使用收集的資訊 CloudTrail，您可以判斷對 API Gateway 發出的要求、提出要求的來源 IP 位址、提出要求的時間以及其他詳細資訊。

> **Note**
>
> TestInvokeAuthorizer並TestInvokeMethod且沒有登錄 CloudTrail。

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 身分中心使用者提出。
- 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。

CloudTrail 在您創建帳戶 AWS 帳戶 時處於活動狀態，並且您自動可以訪問 CloudTrail 事件歷史記錄。 CloudTrail 事件歷史記錄提供了過去 90 天中記錄的管理事件的可查看，可搜索，可下載和不可變的記錄。 AWS 區域若要取得更多資訊，請參閱《使用指南》中的〈AWS CloudTrail 使用 CloudTrail 事件歷程〉。查看活動歷史記錄不 CloudTrail收取任何費用。

如需過 AWS 帳戶 去 90 天內持續的事件記錄，請建立追蹤或 CloudTrailLake 事件資料存放區。

CloudTrail 小徑

追蹤可 CloudTrail 將日誌檔交付到 Amazon S3 儲存貯體。使用建立的所有系統線 AWS Management Console 都是多區域。您可以使用建立單一區域或多區域系統線。 AWS CLI建議您建立多區域追蹤，因為您會擷取帳戶 AWS 區域 中的所有活動。如果您建立單一區域追蹤，則只能檢視追蹤記錄中的 AWS 區域事件。如需有關追蹤的詳細資訊，請參閱《AWS CloudTrail 使用指南》中的「為您的建立追蹤」 AWS 帳戶和「為組織建立追蹤」。

您可以透 CloudTrail 過建立追蹤，免費將一份正在進行的管理事件副本傳遞到 Amazon S3 儲存貯體，但是需要支付 Amazon S3 儲存費用。如需有關 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 湖泊事件資料存放區

CloudTrail L@@ ake 可讓您針對事件執行 SQL 型查詢。 CloudTrail 湖將基於行的 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。若要取得有關 CloudTrail Lake 的更多資訊，請參閱使用指南中的〈AWS CloudTrail 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。如需有關 CloudTrail 定價的詳細資訊，請參閱AWS CloudTrail 定價。

中的 API Gateway 管理事件 CloudTrail

管理事件提供有關在您的資源上執行的管理作業的資訊 AWS 帳戶。這些也稱為控制平面操作。依預設，會 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，TestInvokeAuthorizer和除外TestInvokeMethod。如需 API Gateway 記錄到的 Amazon API Gateway 動作清單 CloudTrail，請參閱 Amazon API Gateway 參考資料。

API Gateway 事件範例

事件代表來自任何來源的單一請求，並包括有關請求的 API 操作，操作的日期和時間，請求參數等信息。 CloudTrail 日誌文件不是公共 API 調用的有序堆棧跟踪，因此事件不會以任何特定順序顯示。

下列範例顯示示範「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 ...
    ]
}
```

若要取得有關 CloudTrail 記錄內容的資訊，請參閱AWS CloudTrail 使用指南中的CloudTrail記錄內容。

使用 AWS Config 監控 API Gateway API 組態

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 可以追蹤對下列組態所做的變更：

- API 階段組態，例如：
  - 快取叢集設定
  - 調節設定
  - 存取日誌設定
  - 階段上使用中的部署設定
- API 組態，例如：
  - 端點組態
  - version
  - 通訊協定
  - tags

此外，AWS Config 規則 功能可讓您定義組態規則，以及自動偵測、追蹤和提醒這些規則的違規。透過追蹤這些資源組態屬性的變更，您也可以為 API Gateway 資源撰寫變更觸發的 AWS Config 規則，並根據最佳實務測試您的資源組態。

您可以使用 AWS Config 主控台或 AWS Config，在您的帳戶中啟用 AWS CLI。選取您要追蹤變更的資源類型。如果您之前已將 AWS Config 設定為記錄所有資源類型，則這些 API Gateway 資源會自動記錄在您的帳戶中。所有 AWS 公有區域和 AWS GovCloud (US) 都支援 AWS Config 中的 Amazon API Gateway。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 以記錄 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

以下 API Gateway 資源類型會與 AWS Config 整合，並記載於 AWS Config 支援的 AWS 資源類型和資源關係中：

- `AWS::ApiGatewayV2::Api` (WebSocket 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 AWS Config 的詳細資訊，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，您將會在 AWS Config 主控台中看到屬性變更，但先前的屬性設定仍然有效；API 的執行時間行為將保持不變。
>
> - **`AWS::ApiGateway::RestApi`** – `binaryMediaTypes`, `minimumCompressionSize`, `apiKeySource`
> - **`AWS::ApiGatewayV2::Api`** – `apiKeySelectionExpression`

設定 AWS Config

若要初始設定 AWS Config，請參閱下列在 AWS Config 開發人員指南中的主題。

- 使用主控台設定 AWS Config
- 使用 AWS CLI 設定 AWS Config

設定 AWS Config 以記錄 API Gateway 資源

在預設情況下，AWS Config 會記錄所有區域資源支援類型的組態變更，它會探索環境執行所在的區域。您可以自訂 AWS Config 以記錄變更，其僅適用於特定的資源類型，或全域資源的變化。

若要了解區域與全域資源，以及了解如何自訂您的 AWS Config 組態，請參閱選取 AWS Config 記錄的資源。

在 AWS Config 主控台中檢視 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 Config 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，以及其他有關您所選取資源的資訊。
7. 若要查看記錄組態的完整詳細資訊，請選擇 View Details (檢視詳細資訊)。

若要了解尋找資源和檢視此頁面資訊的更多方式，請參閱 AWS Config 開發人員指南中的檢視 AWS 資源組態和歷史紀錄。

使用 AWS Config 規則評估 API Gateway 資源

您可以建立 AWS Config 規則，其代表您的 API Gateway 資源的理想組態設定。您可以使用預先定義的 AWS Config 受管規則，或是定義自訂規則。AWS Config 會持續追蹤您資源組態的變更，以判斷變更是否會違反您的規則條件。AWS Config 主控台會顯示您規則和資源的合規狀態。

如果資源違反規則並標示為不合規，AWS Config 可以使用 Amazon Simple Notification Service 開發人員指南 (Amazon SNS) 主題來提醒您。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

若要瞭解 AWS 服務 是否屬於特定規範遵循方案的範圍內，請參閱AWS 服務 遵循規範計劃方案中的，並選擇您感興趣的合規方案。如需一般資訊，請參閱AWS 規範計劃AWS。

您可以使用下載第三方稽核報告 AWS Artifact。如需詳細資訊，請參閱下載中的報告中的 AWS Artifact。

您在使用時的合規責任取決 AWS 服務 於您資料的敏感性、公司的合規目標以及適用的法律和法規。AWS 提供下列資源以協助遵循法規：

- 安全性與合規性快速入門指南 — 這些部署指南討論架構考量，並提供部署以安全性和合規性 AWS 為重點的基準環境的步驟。
- 在 Amazon Web Services 上架構 HIPAA 安全性與合規性 — 本白皮書說明公司如何使用建立符合 HIPAA 資格的應 AWS 用程式。

> **Note**
>
> 並非所有人 AWS 服務 都符合 HIPAA 資格。如需詳細資訊，請參閱 HIPAA 資格服務參照。

- AWS 合規資源AWS — 此工作簿和指南集合可能適用於您的產業和所在地。
- AWS 客戶合規指南 — 透過合規的角度瞭解共同的責任模式。這份指南總結了在多個架構 (包括美國國家標準技術研究所 (NIST)、支付卡產業安全標準委員會 (PCI) 和國際標準化組織 (ISO)) 中，保 AWS 服務 護指引並對應至安全性控制的最佳實務。
- 使用AWS Config 開發人員指南中的規則評估資源 — 此 AWS Config 服務會評估您的資源組態符合內部實務、產業準則和法規的程度。
- AWS Security Hub— 這 AWS 服務 提供了內部安全狀態的全面視圖 AWS。Security Hub 使用安全控制，可評估您的 AWS 資源並檢查您的法規遵循是否符合安全業界標準和最佳實務。如需支援的服務和控制清單，請參閱 Security Hub controls reference。
- Amazon GuardDuty — 透過監控環境中的 AWS 帳戶可疑和惡意活動，藉此 AWS 服務 偵測您的工作負載、容器和資料的潛在威脅。 GuardDuty 可協助您滿足特定合規性架構所要求的入侵偵測需求，例如 PCI DSS 等各種合規性需求。
- AWS Audit Manager— 這 AWS 服務 有助於您持續稽核您的 AWS 使用情況，以簡化您管理風險的方式，以及遵守法規和業界標準的方式。

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、容錯能力和擴充能力，均較單一或多個資料中心的傳統基礎設施還高。

如需有關 AWS 區域與可用區域的詳細資訊，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。如需進一步了解，請參閱調節 API 請求以獲得較佳輸送。

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 是一項受管服務，受到 AWS 全球網路安全的保護。如需有關 AWS 安全服務以及 AWS 如何保護基礎設施的詳細資訊，請參閱 AWS 雲端安全。若要使用基礎設施安全性的最佳實務來設計您的 AWS 環境，請參閱安全性支柱 AWS 架構良好的框架中的基礎設施保護。

您可使用 AWS 發佈的 API 呼叫，透過網路存取 API Gateway。用戶端必須支援下列項目：

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

此外，請求必須使用存取索引鍵 ID 和與 IAM 主體相關聯的私密存取索引鍵來簽署。或者，您可以使用 AWS Security Token Service (AWS STS) 來產生暫時安全憑證來簽署請求。

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 AWS 網路內特定 VPC 對指定 API Gateway 資源的網路存取。

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 REST API 的存取、在 API Gateway 中控制和管理 WebSocket API 的存取和使用 JWT 授權方控制對 HTTP API 的存取權。

實作記錄

使用 CloudWatch 日誌或 Amazon 數據 Firehose 將請求記錄到您的 API。如需進一步了解，請參閱監控 REST API、設定 WebSocket API 的記錄和設定 HTTP API 的記錄功能。

實施 Amazon CloudWatch 警報

您可以使用 CloudWatch 警示來監視指定期間內的單一量度。如果指標超過指定臨界值，則會向 Amazon 簡單通知服務主題或 AWS Auto Scaling 政策傳送通知。 CloudWatch 當測量結果處於特定狀態時，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 度量”。

啟用 AWS CloudTrail

CloudTrail 提供使用者、角色或 AWS 服務在 API Gateway 中採取的動作記錄。使用收集的資訊 CloudTrail，您可以判斷向 API Gateway 發出的要求、提出要求的來源 IP 位址、提出要求的人員、提出要求的時間，以及其他詳細資訊。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 來定義規則，以評估資源組態是否符合資料。 AWS Config 規則代表 API Gateway 資源的理想組態設定。如果資源違反規則且被標記為不合規， 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

透過使用 AWS Security Hub 監視您 API Gateway 的使用狀況，因為它關係到安全最佳實務。Security Hub 會透過安全控制來評估資源組態和安全標準，協助您遵守各種合規架構。如需有關使用 Security Hub 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

標籤是您指派或指派給 AWS 資源的 AWS 中繼資料標籤。每個標籤有兩個部分：

- 標籤鍵 (例如，`CostCenter`、`Environment` 或 `Project`)。標籤鍵會區分大小寫。
- 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 Billing and Cost Management 儀表板上啟用這些標籤。 AWS 使用標籤來分類您的成本，並提供每月成本分配報告給您。如需詳細資訊，請參閱《AWS Billing 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 服務都支援標記，因此您可以將相同標籤指派給來自不同服務的資源，以指出資源是相關的。例如，您可以將相同的標籤指派給指派給 CloudWatch 事件規則的 API Gateway 階段。

有關使用標籤的提示，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 中的下列 HTTP WebSocket API 或 API 資源上設定標籤：

- `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
- 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
- 如果在 DomainName 上設定標籤，它之下的任何 BasePathMapping 資源會繼承該標籤。
- 如果在 UsagePlan 上設定標籤，它之下的任何 UsagePlanKey 資源會繼承該標籤。

> **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視中的成本 AWS Cost Explorer。當您呼叫資源時，API Gateway 不會傳回繼承GetTags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 資源權限的語法的一部分。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。

```
{
  "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。

```
{
  "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。

```
{
  "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。

```
{
  "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 提供用於建立和部署您自己的 HTTP 和 API 的 API 的 API 的 WebSocket API。此外，API Gateway API 也可在標準 AWS 開發套件中使用。

如果您使用的是 AWS SDK 存在的語言，您可能更喜歡使用 SDK，而不是直接使用 API Gateway REST API。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。

以下是可以找到 AWS 開發套件和 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 API 參考

Amazon API Gateway 配額和重要說明

主題

- 每個區域的 API Gateway 帳戶層級配額
- HTTP API 配額
- 用於設定和執行 API 的 WebSocket API Gateway 配額
- 設定和執行 REST API 的 API Gateway 配額
- 建立、部署和管理 API 的 API Gateway 配額
- Amazon API Gateway 重要說明

除非另有說明，否則可以在請求時提高配額。若要請求提高配額，您可以使用 Service Quotas 或與 AWS 支援中心聯絡。

在方法上啟用授權時，方法 ARN 的長度上限 (例如 `arn:aws:execute-api:{region-id}:{account-id}:{api-id}/{stage-id}/{method}/{resource}/{path}`) 是 1600 個位元組。路徑參數值 (其大小是在執行階段所決定) 可能會導致 ARN 長度超過限制。發生這種情況時，API 用戶端會收到 `414 Request URI too long` 回應。

> **Note**
>
> 這會在使用資源政策時會限制 URI 長度。在私有 API 需要資源政策的情況下，這會限制所有私有 API 的 URI 長度。

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
針對 HTTP API、REST API、API 和 WebSocket 回呼	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。	是

資源或操作	預設配額	可以提高
API 之間每個區域的每個帳戶進行限制配額 WebSocket	Note 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。	
區域 API	600	否
邊緣最佳化的 API	120	否

* 以下區域的預設節流配額為 2500 RPS，預設突發配額為 1250 RPS：非洲 (開普敦)、歐洲 (米蘭)、亞太區域 (雅加達)、中東 (阿聯酋)、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) 及加拿大西部 (卡加利)。

HTTP API 配額

下列配額適用於在 API Gateway 中設定和執行 HTTP API。

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	否
每個階段的標籤	50	否

資源或操作	預設配額	可以提高
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 日誌日誌條目	1 MB	否
每個 API 的授權方	10	是
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

資源或操作	預設配額	可以提高
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
階段變數中的數值長度 (以字元為單位)	512	否

用於設定和執行 API 的 WebSocket API Gateway 配額

下列配額適用於在 Amazon WebSocket API 閘道中設定和執行 API。

資源或操作	預設配額	可以提高
每個區域每個帳戶 (跨所有 WebSocket API) 的每秒新連線數	500	是
並行連線	不適用 *	不適用
AWS Lambda 每個 API 的授權者	10	是
AWS Lambda 授權者結果大小	8 KB	否
每 API 的路由	300	是

資源或操作	預設配額	可以提高
每 API 的整合	300	是
整合逾時	所有整合類型 (包括 Lambda、Lambda 代理、HTTP、HTTP 代理伺服器和 AWS 整合) 皆為 50 毫秒至 29 秒。	否
每個 API 階段	10	是
WebSocket 框架尺寸	32 KB	否
訊息承載大小	128 KB **	否
WebSocket API 的連線持續時間	2 小時	否
閒置連線逾時	10 分鐘	否
WebSocket API 的網址長度 (以字元為單位)	4096	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，因此必須將大於 32 KB 的訊息分割成多個框架，每個框架都必須小於 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。

設定和執行 REST API 的 API Gateway 配額

下列配額適用於在 Amazon API Gateway 中設定和執行 REST API。針對 restapi:import 或 restapi:put，API 定義檔案的大小上限為 6 MB。

每個 API 的所有限制只有在特定 API 上才能提高。

資源或操作	預設配額	可以提高
每個區域每個帳戶的自訂網域名稱	120	是
每個網域的多層 API 映射	200	否
邊緣最佳化 API 的 URL 長度 (以字元為單位)	8192	否
區域性 API 的 URL 長度 (以字元為單位)	10240	否
每個區域每個帳戶的私有 API	600	否
API Gateway 資源政策的長度 (以字元為單位)	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用戶端憑證	60	是
授權者每個 API (AWS Lambda 和 Amazon Cognito)	10	是
每個 API 的文件部分	2000	是
每個 API 資源	300	是

資源或操作	預設配額	可以提高
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)
快取的回應大小	1048576 個位元組。快取資料加密可能增加正在快取的項目大小。	否
整合逾時	所有整合類型 (包括 Lambda、Lambda 代理、HTTP、HTTP 代理伺服器和 AWS 整合) 皆為 50 毫秒至 29 秒。	是 *
所有標頭值的總合併大小	10240 個位元組	否

資源或操作	預設配額	可以提高
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
具有授權之方法的 ARN 長度	1600 個位元組	否
用量計劃中單一階段的方法層級調節設定	20	是
每個 API 的資料大小	400 KB	否
信任庫中的證書數量	1,000 個憑證，最多可達 1 MB 的物件總大小。	否

* 您無法將整合逾時設定為少於 50 毫秒。您可以將區域 API 和私有 API 的整合逾時提高到 29 秒以上，但這可能需要降低帳戶層級節流配額限制。

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 SDK 在 API Gateway 中建立、部署和管理 API。 AWS CLI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
CreateRestApi	區域或私有 API • 每個帳戶每 3 秒 1 個請求。 邊緣最佳化的 API • 每個帳戶每 30 秒 1 個請求。	否
CreateVpcLink(第二部分)	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(第二部分)	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
ImportRestApi	區域或私有 API • 每個帳戶每 3 秒 1 個請求。	否

動作	預設配額	可以提高
	邊緣最佳化的 API • 每個帳戶每 30 秒 1 個請求。	
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
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

Amazon API Gateway 重要說明

主題

- Amazon API Gateway 其他 API、HTTP API 和 WebSocket API 的重要注意事項
- Amazon API Gateway REST 和 WebSocket API 的重要注意事項
- Amazon API Gateway API 的 WebSocket 重要注意事項
- 適用於 REST API 的 Amazon API Gateway 重要說明

Amazon API Gateway 其他 API、HTTP API 和 WebSocket API 的重要注意事項

- 簽名版本 4A 不受 Amazon API Gateway 官方支援。

Amazon API Gateway REST 和 WebSocket API 的重要注意事項

- API Gateway 不支援在 REST 和 WebSocket API 之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。大於 1024 位元組的記錄事件 (例如要求和回應內文) 會在提交至 CloudWatch 記錄檔之前被 API Gateway 截斷。
- CloudWatch 量度目前將維度名稱和值限制為 255 個有效 XML 字元。(如需詳細資訊，請參閱《CloudWatch 使用指南》。) 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，請注意不要超過 CloudWatch 量度限制。
- 映射範本的大小上限為 300 KB。

Amazon API Gateway API 的 WebSocket 重要注意事項

- 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 (|)，而且必須對其進行 URL 編碼。
- 任何請求 URL 查詢字串不支援分號字元 (;) 且會 導致資料分割。
- 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 解碼參數，然後將它們作為 unicode 傳遞給後端集成。
- 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。
- 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。
- `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

標頭名稱	請求 (**http/http_proxy /lambda**)	回應 (**http/http_proxy /lambda**)
Connection	傳遞/傳遞/已捨棄	已重新對應
Content-Encoding	傳遞/已捨棄/傳遞	傳遞
Content-Length	傳遞 (依據內文產生)	傳遞
Content-MD5	已捨棄	已重新對應
Content-Type	傳遞	傳遞
Date	傳遞	已重新對應覆寫
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

標頭名稱	請求 (**http/http_proxy /lambda**)	回應 (**http/http_proxy /lambda**)
Server	已捨棄	已重新對應覆寫
TE	已捨棄	已捨棄
Transfer-Encoding	已捨棄/已捨棄/例外	已捨棄
Trailer	已捨棄	已捨棄
Upgrade	已捨棄	已捨棄
User-Agent	傳遞	已重新對應
Via	已捨棄/已捨棄/傳遞	傳遞/已捨棄/已捨棄
Warn	傳遞	傳遞
WWW-Authenticate	已捨棄	已重新對應

* 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 產生的 Java、Android 和 iOS JavaScript 開發套件不同，API Gateway 產生的 API 開發套件不支援 500 個層級錯誤的重試。
- 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。

- 最新文件更新：2024 年 2 月 15 日

變更	描述	日期
增加了對 TLS 1.3 的支持	API Gateway 現在支援地區性其餘 API、HTTP API 和 WebSocket API 上的 TLS 1.3。如需詳細資訊，請參閱在 API Gateway 中為您的自訂網域選擇安全性原則。	2024年2月15日
其餘 API 和 WebSocket API 主控台更新	已更新其他 API 和 WebSocket API 的主控台資訊。	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	新增支援使用 Amazon Route 53 運作狀態檢查，在次要區域中從主要 AWS 區域 到一個 API 的 API Gateway REST API 控制 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	2022 年 5 月 31 日

	要。如需詳細資訊，請參閱在 REST API 和 HTTP API 這兩種 API 之間選擇。	
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
API AWS Lambda 授權者	已新增對 HTTP API AWS Lambda 授權者的支援。如需詳細資訊，請參閱使用 HTTP API 的 AWS Lambda 授權者。	2020 年 9 月 9 日
API AWS 服務整合	已新增對 HTTP API AWS 服務整合的支援。如需詳細資訊，請參閱使用 HTTP API 的 AWS 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 匯入應用程式介面的變數	增加了對 OpenAPI 定義中的 AWS 變量的支持。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 數據火災日誌記錄	增加了對 Amazon 數據 Firehose 的支持，作為訪問日誌數據的目的地。如需詳細資訊，請參閱使用 Amazon 資料 Firehose 做為 API Gateway 存取記錄的目的地。	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
API 的基於標籤的訪問控制 WebSocket	增加了對 WebSocket API 的基於標籤的訪問控制的支持。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佈 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

Support AWS Config	增加了對 AWS Config. 如需詳細資訊，請參閱使用 AWS Config.	2019 年 3 月 20 日
Support AWS CloudFormation	已將 API Gateway V2 API 新增至 AWS 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
WebSocket API 的 Support	增加了對 WebSocket API 的支持。如需詳細資訊，請參閱在 Amazon WebSocket API 閘道中建立 API。	2018 年 12 月 18 日
無伺服器開發人員入口網站 AWS Serverless Application Repository	Amazon API Gateway 開發人員入口網站無伺服器應用程式現在可從 AWS Serverless Application Repository(除了 GitHub) 取得。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
Support AWS WAF	新增對 AWS WAF (Web 應用程式防火牆) 的支援。有關詳情，請參閱使用 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
積極 AWS X-Ray 整合	您現在可以用 AWS X-Ray 來追蹤和分析使用者要求中透過 API 傳送至基礎服務的延遲情況。如需詳細資訊，API Gateway 閱使用 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。先前發佈的更新仍可在此頁面中取得。使用頂部選單面板中的 RSS 按鈕來訂閱摘要。	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

變更	描述	變更日期
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 函數作為 Lambda 授權者函數或 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 “教學課程：建置具有跨帳戶 Lambda 代理整合的 API”及the section called “為 REST API 設定跨帳戶 Amazon Cognito 授權方”。	2018 年 4 月 2 日
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 “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	使用 API Gateway 私有整合建立 API，讓客戶能夠從 VPC 外部透過資源存取 Amazon VPC 中的 HTTP/HTTPS 資源。VpcLink如需詳細資訊，請參閱 the section called “教學：建置具有私有整合的 API” 及 the section called “私有整合”。	2017 年 11 月 30 日

變更	描述	變更日期
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。您可以設定初期測試版本的階段流量百分比，並啟用記錄在單獨的 CloudWatch 記錄檔記錄中的金絲雀特定執行和存取。如需詳細資訊，請參閱 the section called “設定 Canary Release 部署”。	2017 年 11 月 28 日
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 “產生 API 的 Ruby 開發套件” 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

變更	描述	變更日期
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
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 API 請求以獲得較佳輸送。	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 在 API Gateway 中使用請求驗證。	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 設定 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	在使用計劃中將您的 API 作為 SaaS 產品銷售 AWS Marketplace。使用 AWS Marketplace 推廣您的 API。依靠代表您 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

變更	描述	變更日期
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)，來發佈 API 文件。如需詳細資訊，請參閱 記載 REST API。	2016 年 12 月 1 日
更新了自訂授權方	客戶授權方 Lambda 函數現在會傳回發起人的主體識別符。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 授權者的輸出。	2016 年 12 月 1 日
支援二進位承載	binaryMediaTypes在您的 API 上設置以支持請求或響應的二進制有效載荷。在「整合」上設定contentHandling 屬性，或IntegrationResponse指定是否要將二進位裝載作為原生二進位 Blob、以 Base64 字串形式處理，或是未經修改的傳遞。如需詳細資訊，請參閱 使用 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 使用 API 金鑰建立及使用用量計劃。	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 使用者集區作為授權方來控制對 REST API 的存取。	2016 年 7 月 28 日

變更	描述	變更日期
在AWS/ApiGateway 命名空間下啟用 Amazon CloudWatch 指標和維度。	API Gateway 量度現在已在的 CloudWatch 命名空間下標準化AWS/ApiGateway 。您可以在 API Gateway 主控台和 Amazon CloudWatch 主控台中檢視它們。如需詳細資訊，請參閱 Amazon API Gateway 維度和指標。	2016 年 7 月 28 日
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 教學課程：建置具有非 HTTP 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「使用 OpenAPI 設定 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API Gateway 映射範本和存取記錄變數參考。	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 API 請求以獲得較佳輸送。	2016 年 3 月 25 日

變更	描述	變更日期
使用自訂授權啟用和呼叫 API Gateway API	建立並設定實作自訂授權的 AWS Lambda 函數。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
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 使用 OpenAPI 設定 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 Amazon API Gateway API 請求和回應資料映射參考。	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 設定 REST API 部署的階段變數。	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 為 REST API 資源啟用 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 產生和設定後端身分驗證的 SSL 憑證。	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

變更	描述	變更日期
Amazon Cognito Identity 支援	Amazon API Gateway 已擴展 `$context` 變數的範圍，因此當請求以 Amazon Cognito 登入資料簽署時，該變數現在會傳回 Amazon Cognito Identity 的相關資訊。此外，我們還添加了一個`$util`變量，用於轉義字符以及對 URL JavaScript 和字符串進行編碼。如需詳細資訊，請參閱 API Gateway 映射範本和存取記錄變數參考。	2015 年 8 月 28 日
Swagger 整合	使用施瓦格匯入工具將施瓦格 API 定義匯入 Amazon API Gateway。 GitHub進一步了解 使用 OpenAPI 的 API Gateway 延伸，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 API Gateway 映射範本和存取記錄變數參考 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

AWS 詞彙表

如需最新的 AWS 術語，請參閱《AWS 詞彙表 參考》中的 AWS 詞彙表。

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。